CURS 5

Context Engineering (nivel 2):

sisteme de regăsire semantică

CURS 5.

C1 LLM-uri, API-uri - ce construim și cum pornim
C2 Ecosistemul de modele - alegem modelul de bază
C3 Date și corpus - colectare, curățare, metadate
C4 Prompting, adnotare, context engineering
C5 Embeddings, retrieval, RAG
C6 RAG, agenți, LangChain, LangGraph
C7 Agenți orchestrați - LangGraph,metrici,etică
C8 Integrare aplicație - Gradio
C9 Demo final - prezentări și feedback

Tematică
- RAG: diferența dintre regăsire și generare
- embedding-uri ca reprezentări vectoriale ale textelor
- index vectorial: FAISS + metadate
- primele k rezultate și scoruri de similaritate
- verificare umană a rezultatelor recuperate

Activități practice
- Explorăm corpusul cu tipologii discursive
- Alegem un agent / o bulă discursivă
- Curățăm textele slabe și exportăm un JSONL
- Construim un index FAISS pentru bula aleasă
- Testăm retrieval-ul pe un input politic scurt

Livrabile
- notebooks/student_XX/C5_01_explore_typed_corpus.ipynb
- notebooks/student_XX/C5_02_build_vectorstore.ipynb
- data/bubbles/<agent_slug>.jsonl
- assets/vectorstores/<agent_slug>/index.faiss
- assets/vectorstores/<agent_slug>/index.pkl
- scripts/build_vectorstore.py

→ La finalul C5
- Avem un corpus curat pentru fiecare agent
- Avem un vector store FAISS pentru fiecare bulă
- Putem recupera top-k fragmente relevante

De ce nu ajunge căutarea cuvinte cheie?

Căutare după cuvinte-cheie

`(keyword search)`

Găsește texte care conțin exact termenii căutați.
Este bună pentru nume de instituții, partide, persoane sau expresii fixe.

Limită: nu recunoaște sinonime, reformulări, prescurtări, ironie sau formulări indirecte.

Căutare semantică

`(semantic search)`

Găsește texte apropiate ca sens, chiar dacă folosesc alte cuvinte.
Modelul transformă textele în vectori și compară apropierea lor în spațiul semantic.

Este **utilă** când întrebarea și documentul spun același lucru în forme diferite.

EXEMPLU

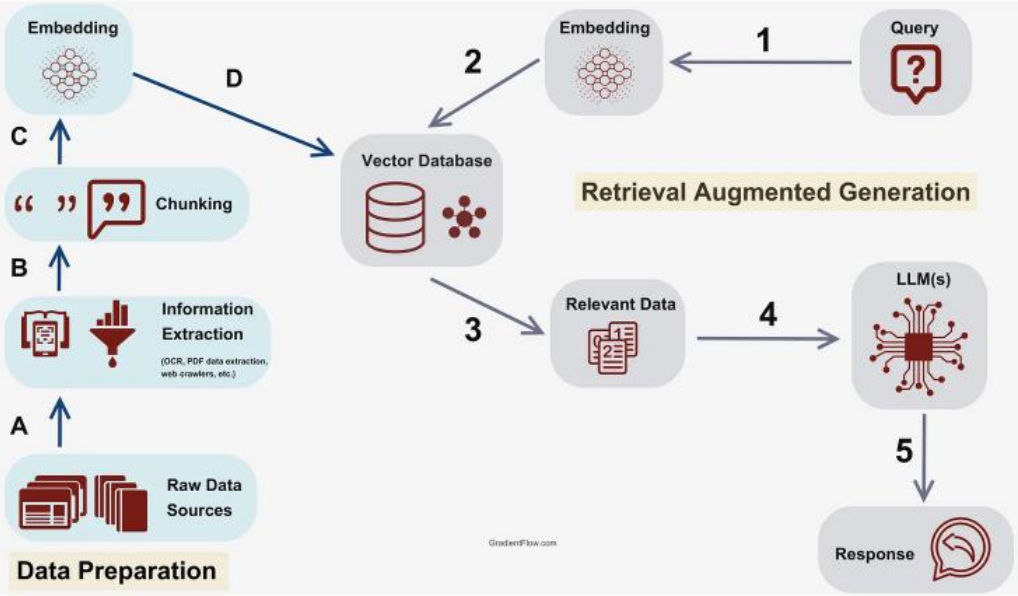
Căutare: *„Curtea Constituțională”*

- Căutarea după cuvinte-cheie găsește: *„Curtea Constituțională”*
- Căutarea semantică poate găsi și: *„CCR”, „judecătorii constituționali”, „decizia de anulare”*

În practică, cele mai bune sisteme folosesc adesea căutare hibridă: cuvinte-cheie + căutare semantică.

Retrieval-Augmented Generation (RAG)

Întrebare → Retriever → Context → LLM → Răspuns



- Modelul nu răspunde doar din ce a învățat la antrenare, ci primește fragmente relevante ca **context**.
- Corpusul este pregătit înainte: texte curate, fragmente, embeddings și vector store.
- La o întrebare nouă, sistemul caută fragmentele cele mai apropiate semantic.
- LLM-ul folosește aceste fragmente pentru a formula un răspuns mai ancorat în date.
- Calitatea răspunsului depinde de calitatea retrieval-ului: date, embeddings, top-k și verificare umană.

https://gradientflow.substack.com/p/best-practices-in-retrieval-augmented

RAG are două jumătăți

C5 - REGĂSIRE

Sistemul caută fragmente relevante în corpus.
Interogarea este transformată într-un embedding și comparată cu embedding-urile textelor indexate.

Rezultatul este o listă de top-k pasaje candidate, fiecare cu scor de similaritate și metadate.

Corpus → Embeddings → FAISS → Top-k → Context

C6 - GENERARE

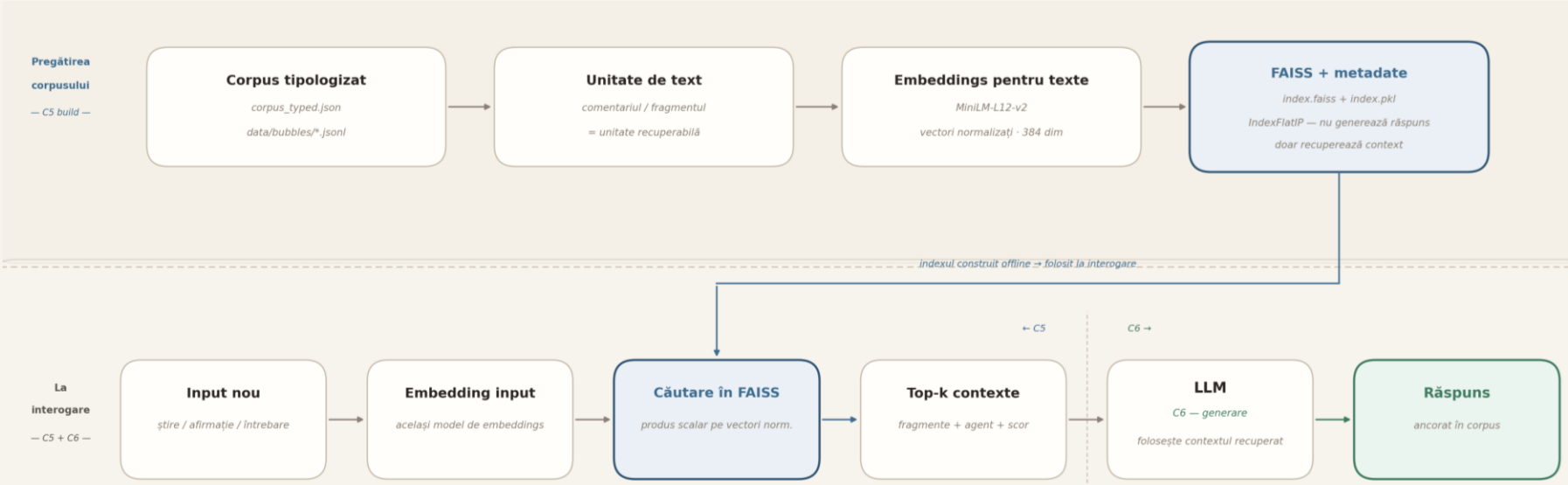
LLM-ul primește contextul recuperat în prompt.
Răspunsul este generat pe baza instrucțiunilor, rolului și fragmentelor selectate.

Aici apar riscurile de formulare, omisiune sau interpretare greșită.

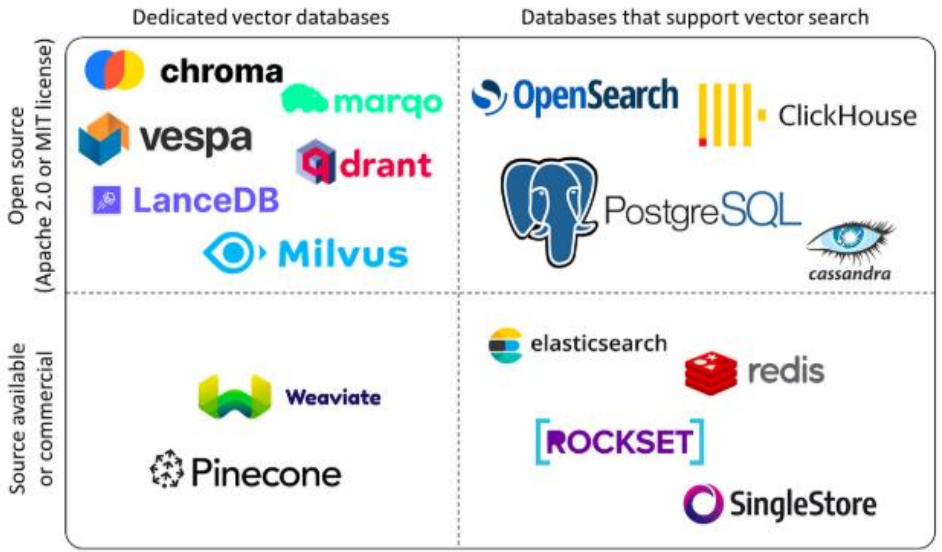
Context → Prompt → LLM → Răspuns

Astăzi construim prima jumătate: cum este ales contextul, nu cum este generat răspunsul.

EchoChamber - RAG



Depozit vectorial sau bază de date vectorială?



1. INDEX VECTORIAL
FAISS
- caută rapid în embeddings
- rulează local sau pe server
- metadatele se gestionează separat
- bun pentru laborator și prototip

2. DEPOZIT VECTORIAL
Chroma
- stochează documente + embeddings + metadate
- are persistență integrată
- permite filtrare mai simplă
- bun pentru RAG local și aplicații mici/medii

3. BAZĂ DE DATE VECTORIALĂ
Pinecone / Weaviate / Qdrant / Supabase pgvector
- gestionează vectori, metadate, filtre și actualizări
- suportă aplicații mai mari sau multi-user
- poate rula în cloud, pe server sau self-hosted, în funcție de soluție
- bună pentru producție

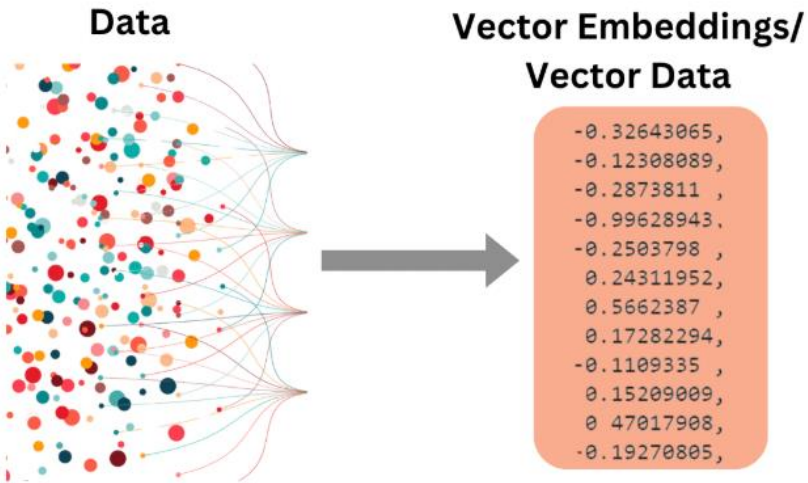
În C5 folosim FAISS pentru simplitate. În aplicații mai mari, aceeași logică poate fi mutată într-un depozit sau într-o bază de date vectorială.

Ce este un depozit vectorial (vector store) ?

Embedding-ul (reprezentarea vectorială) este forma numerică a unui text.

Modelul transformă fiecare comentariu într-o listă de numere, iar texte apropiate ca sens ajung aproape unele de altele în spațiul vectorial.

Depozitul vectorial folosește această geometrie: când primește un input nou, îi calculează embedding-ul și caută textele cu vectorii cei mai apropiați.



https://www.linkedin.com/pulse/complete-guide-creating-storing-vector-embeddings-pavan-belagatti-5fyfc/

Text curat → embedding → vector store
Input nou → embedding → căutare semantică → top-k fragmente relevante

Vector search: de la FAISS la baze de date vectoriale

Toate instrumentele caută în reprezentări vectoriale, dar nu gestionează datele la fel.

- FAISS este un index vectorial simplu: rapid, local/server, dar cu metadate gestionate separat.
- Chroma este un depozit vectorial persistent: păstrează documente, embeddings și metadate împreună.
- Pinecone, Weaviate și Supabase/pgvector sunt soluții mai apropiate de baze de date vectoriale, potrivite pentru aplicații mai mari.

Instrument	Ce este	Poate rula local?	Poate rula pe server/cloud?	Persistență	Metadate / filtre
FAISS	bibliotecă/index vectorial	da	da, în aplicația ta Python	manuală: index.faiss	manual: index.pkl + cod
Chroma	depozit vectorial	da	da	nativă: colecții salvate	integrate
Pinecone	bază vectorială administrată	arțial	da, cloud managed	nativă	integrate
Weaviate	bază vectorială / motor semantic	da, Docker	da, cloud/server	nativă	integrate + căutare hibridă
Supabase/pgvector	PostgreSQL + vectori	self-host posibil	da, cloud/server	nativă SQL	prin SQL

În C5 folosim FAISS ca index vectorial local. În aplicații mari, aceeași logică poate fi mutată într-o bază de date vectorială.

FAISS în EchoChamber

- FAISS (Facebook AI Similarity Search este o bibliotecă pentru căutare rapidă în colecții mari de vectori.
- FAISS nu „înțelege” singur textele și nu generează răspunsuri. El compară vectori și returnează fragmentele cele mai apropiate de inputul nou.
- În laborator folosim **IndexFlatIP**, varianta simplă: caută exact în toți vectorii folosind produsul scalar.

De aceea avem două fișiere separate:

- **index.faiss**
  — vectorii numerici (float32)
- **index.pkl**
  — textele originale și metadatele (sursă, bulă, dată)
- **core/retriever.py**
  — combină cele două și returnează fragmente interpretibile


```
# pipeline build
data/bubbles/<agent_slug>.jsonl
  → encode(normalize_embeddings=True)
  → faiss.IndexFlatIP(dim)   # produs scalar = cosine
  → index.faiss  +  index.pkl

# query
retriever.search(query, k=5)
  → [{text, agent, score}]
```

IndexFlatIP + normalize_embeddings=True: produsul scalar devine echivalentul similarității cosinus.

Embeddings

- Un embedding transformă un text într-un vector numeric de dimensiune fixă.
  Exemplu: MiniLM-L12-v2 produce 384 de valori float32 pentru fiecare text.
- Textele cu sens apropiat primesc vectori apropiați în spațiul semantic.
  Modelul nu caută doar aceleași cuvinte, ci relații de sens.
- SentenceTransformers oferă modele pre-antrenate care produc reprezentări dense ale textului.
- Reprezentările pot fi comparate numeric în pașii următori.

```
from sentence_transformers import
SentenceTransformer

model = SentenceTransformer('paraphrase-
multilingual-MiniLM-L12-v2’)

texts = ["Referendumul a fost anulat.", "Votul
CCR este constituțional."]

embeddings = model.encode(texts,
normalize_embeddings=True)

# embeddings.shape: (2, 384)
# fiecare text = un vector de 384 valori
float32
```

CE PRODUCE MODELUL DE EMBEDDINGS?

- $text_i \rightarrow embedding_i = v_i \in \mathbb{R}^{384}$
- un text = un vector dens
- dimensiunea depinde de model: 384 / 768 / 1024
- normalize_embeddings=True → vector cu lungime 1
- model multilingv ≠ performanță garantată în română
- calitatea trebuie verificată pe corpusul nostru

Embeddings: proximitate, nu adevăr

Embedding-ul transformă textul într-un vector numeric care păstrează relații semantice. Sentence Transformers recomandă normalizarea embeddings pentru dot-product/cosine scoring. Dar apropierea semantică nu garantează adevăr factual, ton corect sau interpretare sociologică validă.

PROBLEME FRECVENTE ÎN DISCURSUL POLITIC ROMÂNESC

- **sarcasm**

— texte ironice pot părea semantic relevante (embeddings nu disting tonul)

- **ambiguitate**

— sensuri multiple se comprimă în același spațiu vectorial

- **bias lingvistic**

— modelele multilingual pot performa diferit pe română; testăm rezultatele pe corpusul nostru și folosim HITL

- **false positives**

— texte semantic similare dar factual greșite sau din contexte diferite

Embedding-ul măsoară proximitate în spațiul vectorial, nu adevăr și nu calitate sociologică.

LLM-ul nu vede corpusul. Vede un subset.

CORPUSUL COMPLET

Sute sau mii de fragmente colectate în C4. Nu este eficient și nici metodologic util să trimitem tot corpusul în prompt - context window limitat, cost ridicat, pierdere de precizie.

CONTEXT SELECTAT

Câteva fragmente alese de retriever, ordonate printr-un scor de similaritate. Ce nu intră în context nu poate influența răspunsul - modelul nu știe că există.

Implicație metodologică: retriever-ul devine filtrul realității modelului. Un retriever biasat produce un model biasat indiferent de calitatea LLM-ului.

Fluxul de regăsire semantică

```
Corpus → Chunking → Embeddings → Index FAISS
Interogare → Embedding interogare → Căutare FAISS → Top-k pasaje → Context
```

- Regăsirea are pași separați, iar fiecare decizie schimbă ce fragmente ajung în context.
- În proiectul nostru, corpusul este deja împărțit în texte scurte; de aceea chunking-ul înseamnă mai ales verificarea unității de analiză.

FIECARE PAS = DECIZIE DE CERCETARE

- **chunking**

— care este unitatea de analiză: frază, paragraf, fragment?

- **embedding**

— alegerea spațiului semantic; modelul determină ce înseamnă 'similar'

- **top-k**

— câte fragmente din corpus intră în context?

- **metadata**

— filtrare după sursă, dată, bulă; audit și transparență

Laborator C5:

ce construim azi?

NOTEBOOK 1 — EXPLORARE + EXPORT

`notebooks/student_XX/C5_01_explore_typed_corpus.ipynb`

- explorăm corpus_typed.json și alegem agentul final
- verificăm textele, eliminăm exemplele slabe
- exportăm data/bubbles/<agent_slug>.jsonl

NOTEBOOK 2 — BUILD + MINI-TEST

`notebooks/student_XX/C5_02_build_vectorstore.ipynb`

- încărcăm bula curată, generăm embeddings (MiniLM-L12-v2, 384 dim)
- construim IndexFlatIP, salvăm index.faiss + index.pkl
- facem un mini-test de retrieval manual

TEAM SCRIPT

`scripts/build_vectorstore.py`

Construiește vectorstore-uri pentru toate bulele după ce fișierele .jsonl sunt curate.

La finalul C5

La finalul C5, sistemul nu generează încă răspunsuri. El primește un input - o știre, o afirmație sau un context politic - și returnează exemple discursive relevante cu scor. Aceste fragmente devin contextul pe care C6 îl va trimite către LLM.

INPUT

"CCR a decis anularea alegerilor după suspiciuni privind influențe externe."

OUTPUT

```
[
  { text: "...", agent: "Anti-sistem",     score: 0.82 },
  { text: "...", agent: "Conspirationist", score: 0.77 },
  { text: "...", agent: "Pro-european",    score: 0.71 }
]
```

C5 = caută contextul · C6 = generează cu context · C7 = controlează agenții