



AKADEMIA E FORCAVE
TË ARMATOSURA

Hyrje në Modele të Mëdha Gjuhësore

Bazat e të dhënave vektoriale (Vector Databases)

Dr. Fiorela Ciroku



Bazat e të dhënave vektoriale

- Embeddings (paraqitjet vektoriale) e kthejnë tekstin në vektorë ku “kuptimi” bëhet gjeometri.
- Por tani vjen problemi real i sistemeve:

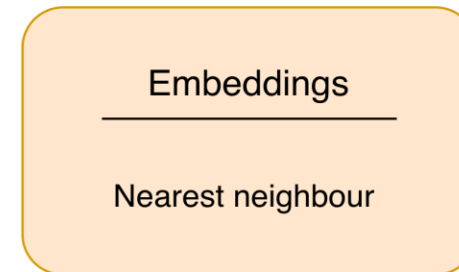
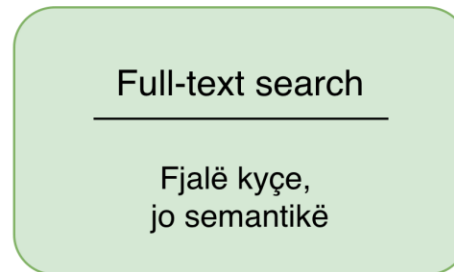
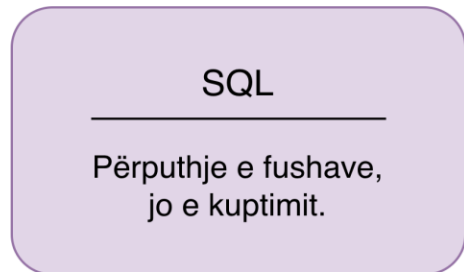
Nëse kam 2 milion chunks dokumentesh,
si i kërkoj shpejt “10 më të afërtat” me pyetjen time?

- Një kërkim i thjeshtë “krahasso me të gjithë” është shumë i ngadaltë.
- ***Bazat e të dhënave vektoriale të cilat janë sisteme të dizajnuara për të bërë kërkim nearest neighbor në hapësira me dimensione të larta*** (p.sh. 384, 768, 1024, 1536+).

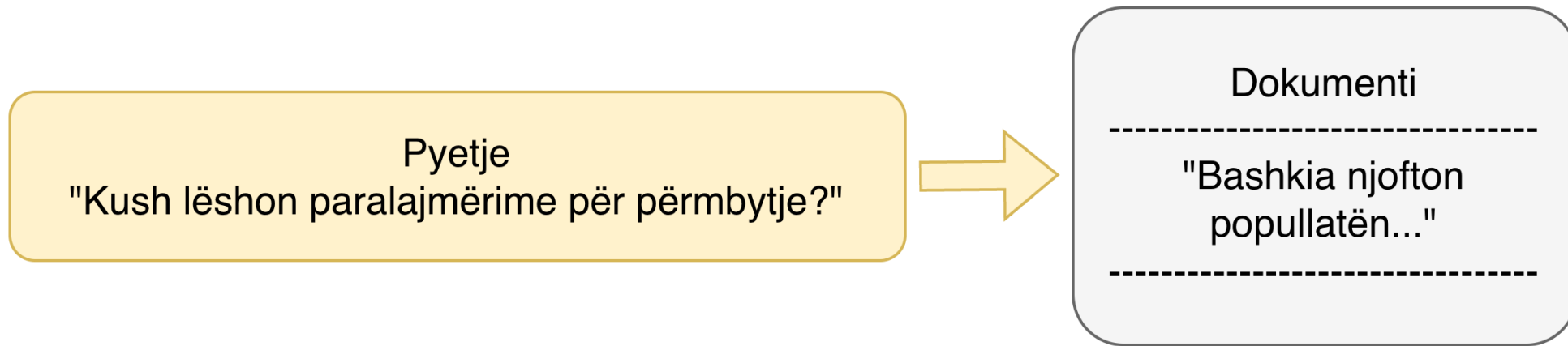


Pse nuk mjafton një DB tradicionale?

- Një database tradicionale (SQL/NoSQL) është perfekte kur e di çfarë kërkon (ID, fjalë kyçe, datë, autor), ose kur kërkon barazi ose intervale.
- Por në retrieval semantik, ne kërkojmë “gjej tekstin me kuptimin më të ngjashëm”.
- Nëse pyetja përdor sinonime ose parafrazim, keyword search dështon.



AI Mbrojtja civile

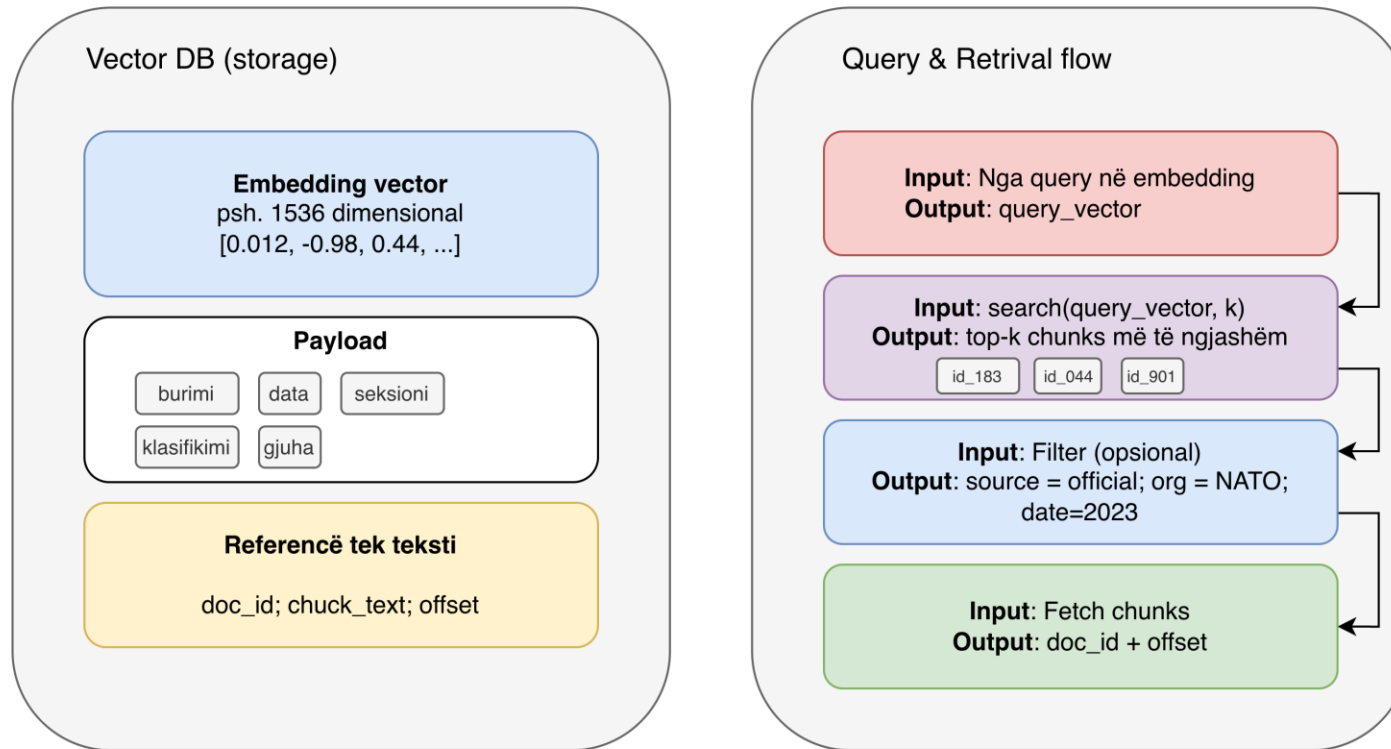


FAIL: Keyword search (paralajmërim vs njoftim, mungesë termash, etj)



SUCCESS: Vector DB (Kuptimi përputhet)

Çfarë është një baza e të dhënave vektoriale?



Në praktikë, “vector DB” mund të jetë: FAISS (library), Milvus, Qdrant, Weaviate (sisteme), Chroma (lightweight), Elasticsearch/OpenSearch me vector fields (hybrid).



Nearest neighbour

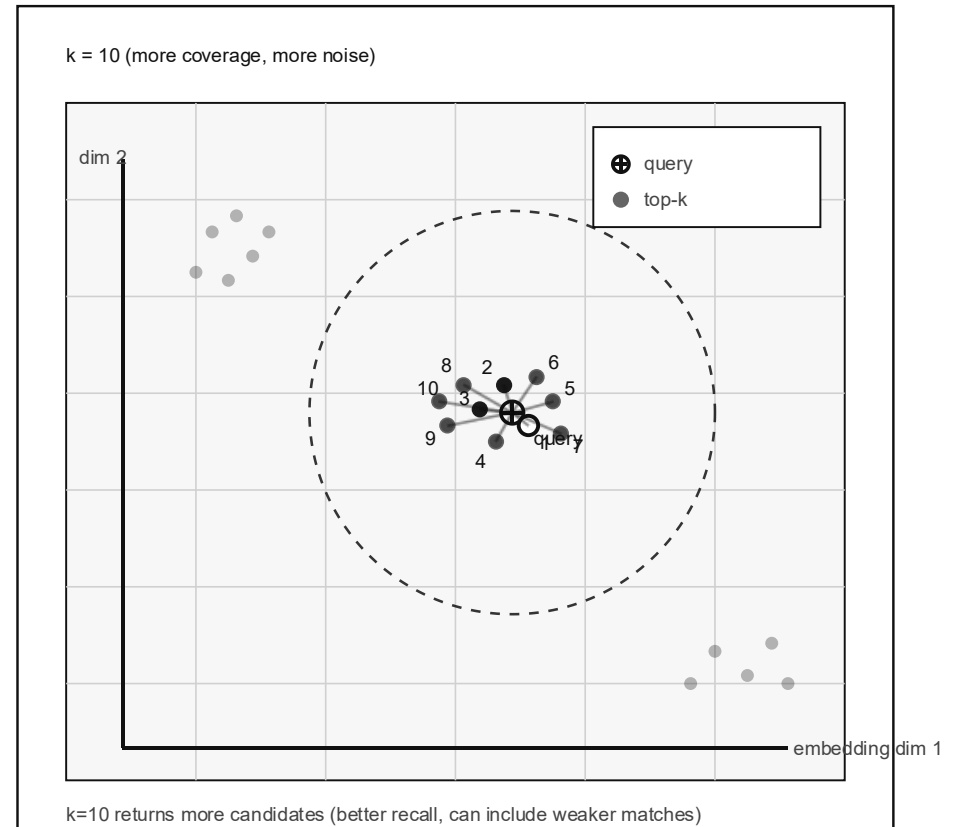
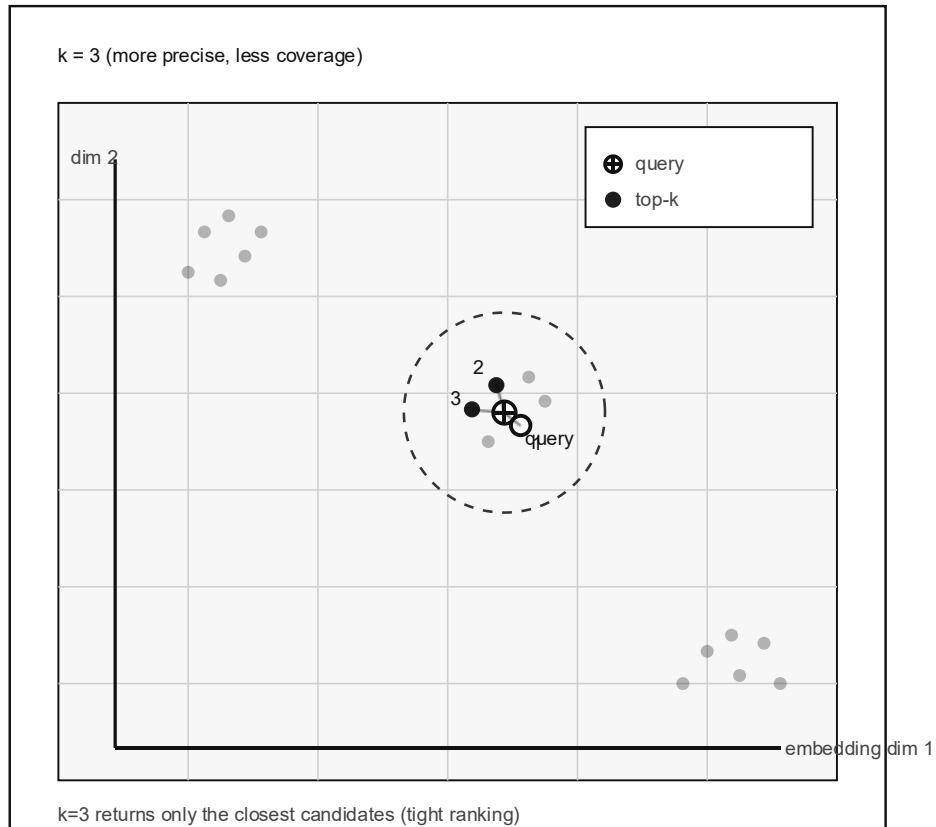
- “Nearest neighbor” nënkupton identifikimin e vektorëve që janë më të ngjashëm me vektorin e pyetjes brenda hapësirës së embedding.
- Në këtë hapësirë, **ngjashmëria** zakonisht vlerësohet përmes ngjashmërisë kosinus (më e përdorura), produktit skalar (dot product), distancës Euklidiane (*më rrallë në embeddings moderne*).
- Qëllimi kryesor nuk është gjetja e një përgjigjeje “të përkryer”, por **renditja e kandidatëve sipas rëndësisë**.
- Për këtë arsye, **RAG rikthen zakonisht k kandidatë**, dhe jo vetëm një të vetëm.



Nearest neighbour

Nearest Neighbor in Embedding Space (2D visualization)

Query point + Top-k nearest neighbors — compare k=3 vs k=10 (ranking focus)



Note: this diagram uses 2D distance for intuition; real systems often rank by cosine similarity in high-dimensional space.



Pse kërkimi i saktë është shumë i ngadaltë?

- **Në një qasje brute-force**, për çdo pyetje (query) llogaritet distanca me **çdo vektor** në hapësirën e embeddings.
- Kjo do të thotë se kostoja llogaritëse rritet afërsisht me $\mathbf{N} \times \mathbf{d}$, ku \mathbf{N} është numri total i vektorëve (chunks), \mathbf{d} është dimensionaliteti i embedding-ut.
- Për shembull, nëse kemi:

$\mathbf{N} = 5,000,000$ segmente dokumentesh

$\mathbf{d} = 1536$ dimensione

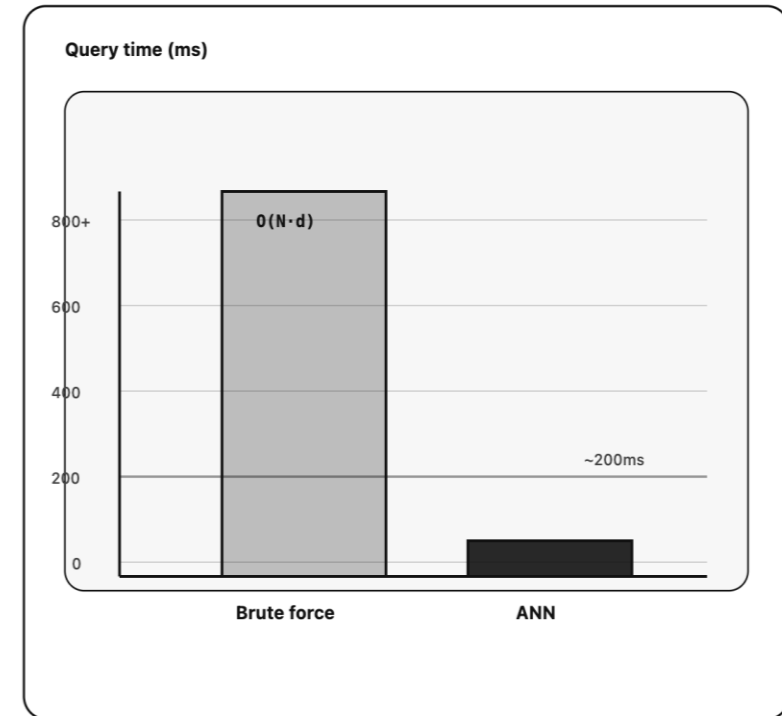
$\mathbf{N} \times \mathbf{d} = 7.680.000.000$

- Çdo query bëhet jashtëzakonisht i kushtueshëm, si në kohë përpunimi ashtu edhe në burime llogaritëse.



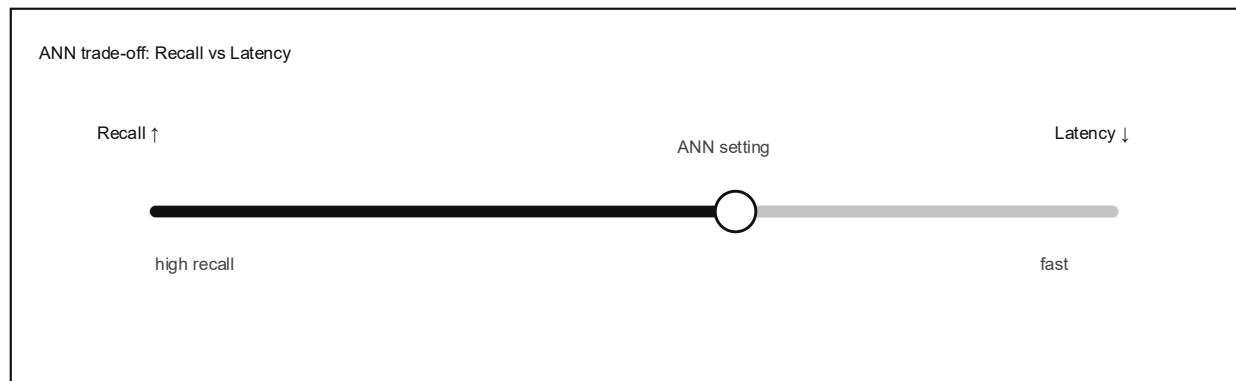
Pse kërkimi i saktë është shumë i ngadaltë?

- Sistemet reale nuk mund ta përballojnë këtë qasje, sepse kërkohet përgjigje pothuajse në kohë reale (nën 1 sekondë), shpesh madje nën 200 ms.
- Për këtë arsye, përdoren algoritme të Approximate Nearest Neighbor (ANN), të cilat:
 - shmangin krahasimin me të gjithë vektorët,
 - sakrifikojnë pak saktësi për shumë më tepër shpejtësi,
 - e bëjnë kërkimin semantik praktik në shkallë të madhe.



Approximate Nearest Neighbor (ANN)

- Algoritmet Approximate Nearest Neighbor (ANN) nuk garantojnë gjetjen e *vektorit absolutisht më të afërt* në çdo rast.
- Në vend të kësaj, ato ofrojnë një zgjidhje shumë më praktike: “***Do të gjej vektorë mjaftueshëm të afërt dhe do ta bëj këtë shumë shpejt.***”
- Kompromisi (trade-off): shpejtësia rritet ndjeshëm, por mund të humbet ndonjë kandidat i rëndësishëm.



Approximate Nearest Neighbor (ANN)

Pse kjo është e pranueshme në RAG?

- Në sistemet RAG nuk mjafton një rezultat i vetëm, zakonisht merren **top-k kandidatë**.
- Këta kandidatë **rirenditen (re-ranking)** më pas me modele më të sakta.
- Nuk kërkohet perfeksion matematikor në hapin e parë të kërkimit. Ajo që kërkohet realisht është **stabilitet** i sistemit, **kohë përgjigjeje e shpejtë**, performancë e qëndrueshme në shkallë të madhe.



Indeksimi i vektorëve

Indeksimi është mënyra se si i organizojmë vektorët paraprakisht, në mënyrë që kërkimi të mos bëhet i ngadalshëm dhe i shtrenjtë sa herë që vjen një query.

Pa indeksim:

- çdo query do të krahasohej me çdo vektor
- sistemi nuk do të ishte i përdorshëm në praktikë

Me indeksim:

- kërkimi kufizohet në zona “të mundshme”
- fitohen shpejtësi dhe stabilitet



Indeksimi: IVF, HNSW, PQ

Inverted File Index (IVF):

- Vektorët ndahen paraprakisht në klasterë. Çdo klaster përfaqëson një zonë të hapësirës semantike.
- Kur vjen një query ai nuk kërkon kudo, por viziton vetëm disa klasterë më të afërt.
- **Rezultati:** krahasohen shumë më pak vektorë; fitimi në shpejtësi është i madh.

Hierarchical Navigable Small World (HNSW):

- Ndërtohet një graf fqinjësie. Çdo vektor lidhet me disa fqinjë “të afërt”.
- Kërkimi: query “ecën” nëpër graf në mënyrë greedy, duke u afruar gjithnjë e më shumë te zona më rëndësi.
- **Rezultati:** kombinon cilësi të mirë dhe shpejtësi të lartë; shumë robust në praktikë; përdoret gjerësisht në sisteme reale (RAG, search, QA)



Indeksimi: IVF, HNSW, PQ

Product Quantization (PQ):

- Vektorët **kompresohen**, nuk ruhen më si numra floating-point të plotë.
- Efekti: përdor shumë më pak memorie dhe shpesh rrit edhe shpejtësinë e kërkimit.
- Kompromisi: humbet pak saktësi, por fiton shumë në shkallëzim.

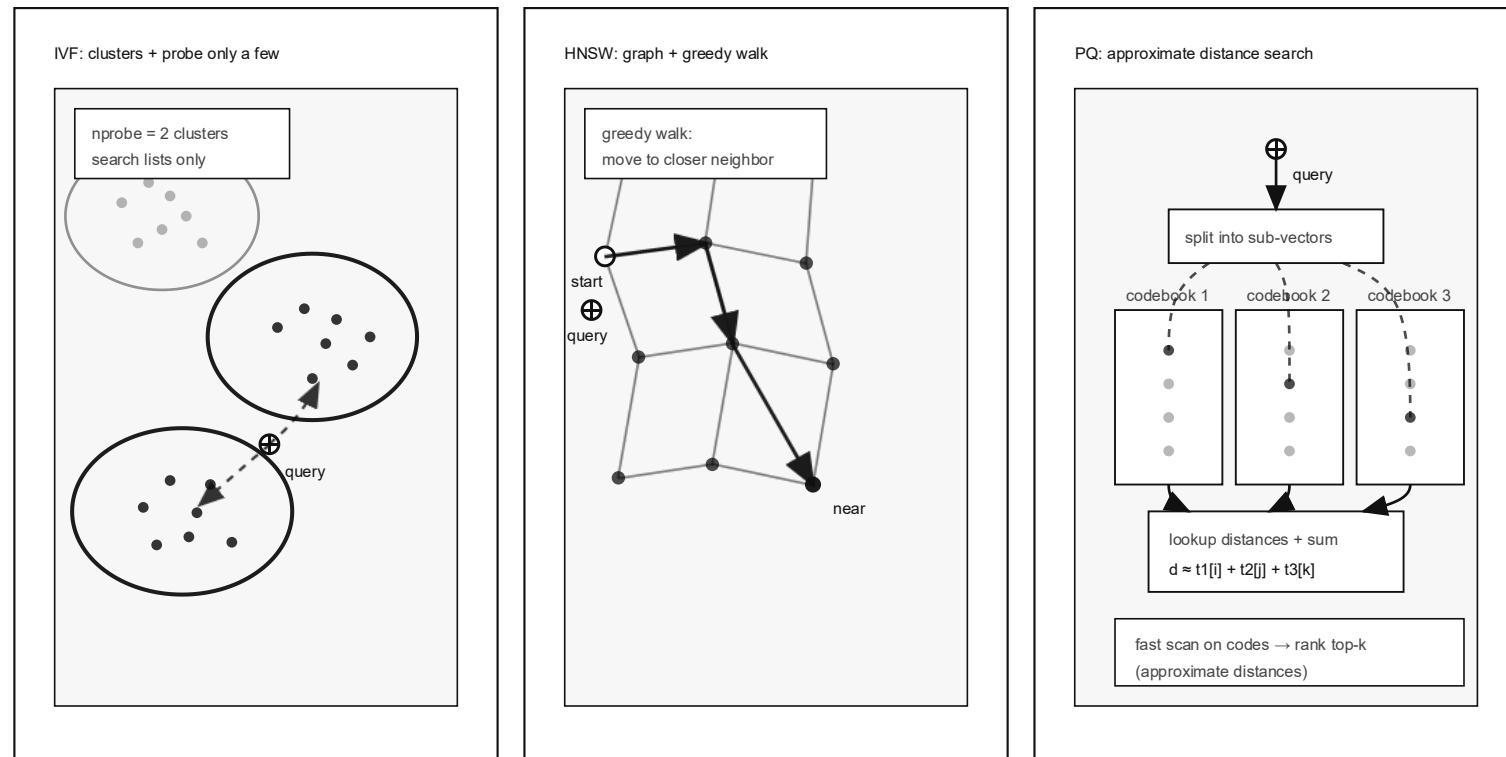
Çfarë ndodh në praktikë moderne?

- **HNSW** është zgjedhja më e shpeshtë për shumicën e rasteve
- **IVF + PQ** përdoret kur memoria është shumë e kufizuar ose dataset-et janë shumë të mëdha.
- Nuk ka “një zgjidhje perfekte”, pasi zgjedhja e llojit të indeksimit varet nga madhësia e të dhënave, kufizimet e memories dhe kërkesat për shpejtësi.



Indeksimi: IVF, HNSW, PQ

Three ANN idea families: IVF, HNSW, PQ (search intuition)



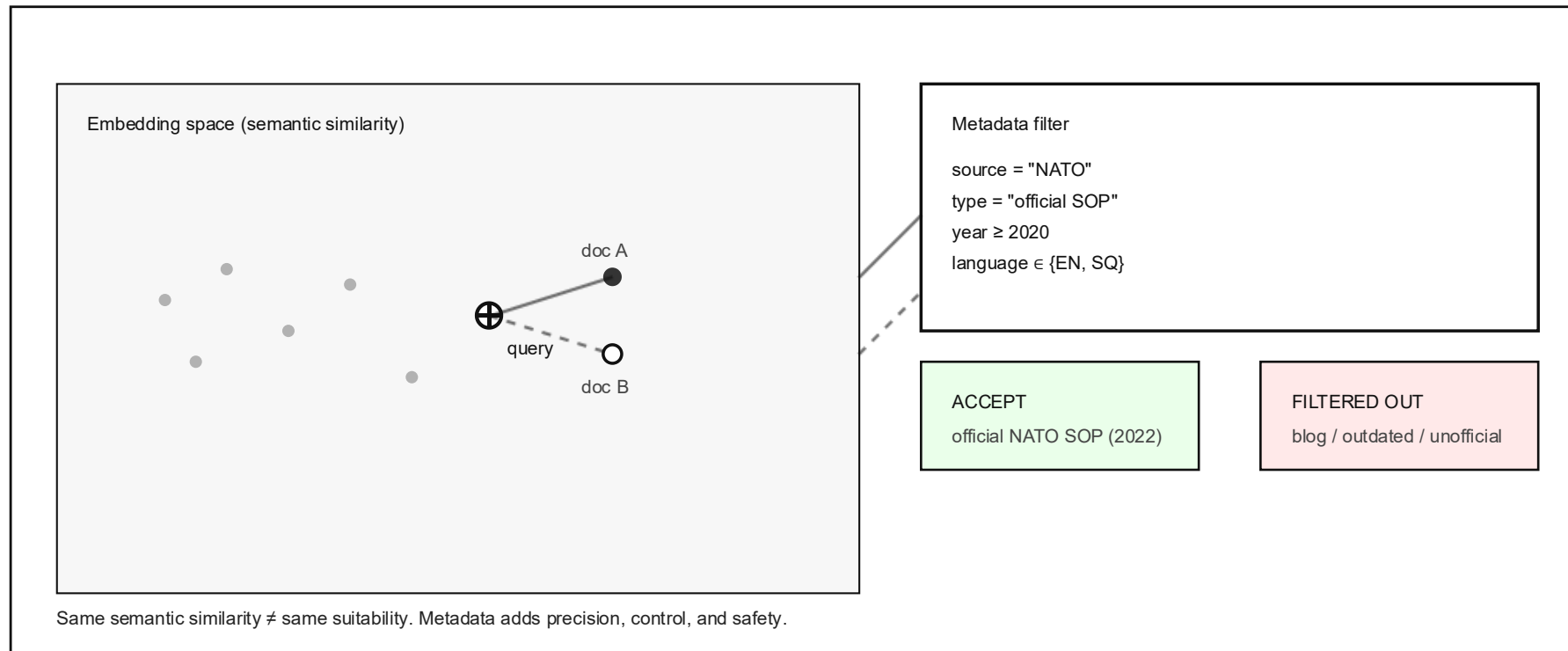
Metadata & Filtrimi

- Në domaine të ndjeshme, rikthimi i informacionit të gabuar përbën një rrezik serioz.
- Pa përdorimin e metadata-s, një sistem kërkimi mund të rikthejë dokumente të vjetruara, materiale jozyrtaren ose përmbajtje që nuk i përshtatet kontekstit operacional.
- Kjo mund të çojë në keqinformim, konfuzion ose vendime të pasakta.
- Metadata i lejon sistemit të kufizojë dhe kontrollojë kërkimin, psh duke rikthyer vetëm SOP zyrtare, procedura të NATO-s, dokumente të publikuara pas vitit 2020, materiale në shqip ose anglisht, etj.
- Përdorimi i metadata-s rrit ndjeshëm precizionin e rezultateve, kontrollin mbi burimet e informacionit dhe sigurinë e sistemit.



Metadata & Filtrimi

Why metadata filtering matters in sensitive domains



Segmentimi + Vector DB: pse ndahen dokumentet?

Një bazë të dhënash vektoriale zakonisht nuk ruan dokumente të plota, por i ndan ato në segmente më të vogla (chunks). Kjo bëhet për disa arsye thelbësore:

- një MMGj nuk mund të përfshijë qindra faqe dokumentesh në një prompt të vetëm.
- procesi i retrieval-it duhet të gjejë pikërisht pjesën më me rëndësi, jo gjithë dokumentin.

Vendosja e madhësisë së segmentit përfshin një trade-off të rëndësishëm:

- chunks shumë të vegjël → humbet konteksti dhe lidhja me pjesët përreth
- chunks shumë të mëdhenj → retrieval bëhet më i paqartë → prompt-i mbushet me informacion të panevojshëm.



Hybrid Search: Pse shpesh kombinojmë keyword + vector?

- Në dokumente procedurale ekzistojnë terma dhe elemente që duhet të identifikohen saktësisht, pa asnjë interpretim të lirë. Këtu përfshihen:
 - kode dhe identifikues specifikë,
 - emra njësisish ose strukturash organizative,
 - referenca ligjore dhe nene konkrete, etj.
- Për këto raste, kërkimi me fjalë kyçe (keyword search) është shumë i fortë, sepse:
 - kërkon përputhje të drejtpërdrejta,
 - nuk humbet informacion specifik,
 - është i besueshëm për terma strikt.



Hybrid Search: Pse shpesh kombinojmë keyword + vector?

Nga ana tjetër, kërkimi vektorial (vector search) shkëlqen në situata ku përdoruesi:

- përdor parafrazime
- përdor sinonime
- bën pyetje në gjuhë natyrore, jo teknike

Është kërkim më fleksibël, por mund të humbasë përputhje shumë precize.

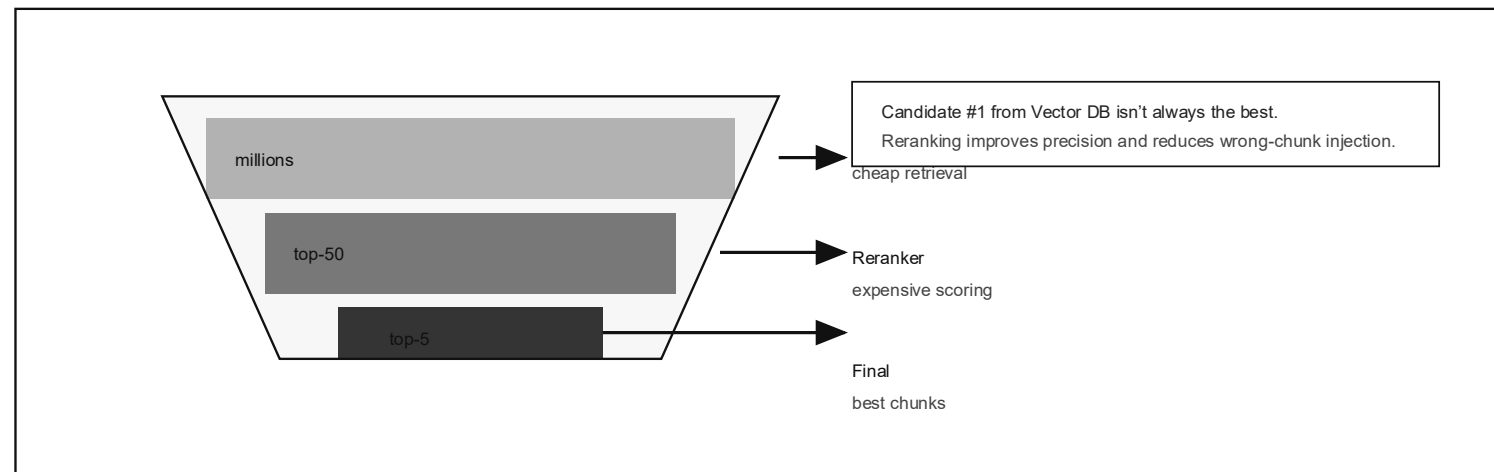
- Në sisteme serioze, nuk zgjidhet vetëm njëra qasje. Në vend të kësaj, përdoret hybrid retrieval (kombinim i keyword search + vector search)
- Re-ranking bëhet në një hap të dytë për të renditur rezultatet sipas rëndësisë reale.



Re-ranking: Si e rrisim cilësinë e top-k?

- Një Vector Database është shumë efikase për të gjetur shpejt kandidatë të mundshëm.
- Megjithatë, rezultati i renditur si #1 nga kërkimi vektorial nuk është gjithmonë më i miri ose më i përshtatshmi për kontekstin konkret.
- Arsyeja është se vector search përdor modele relativisht të lira dhe afërsore, të dizajnuara për shpejtësi, jo për gjykim të hollë semantik.

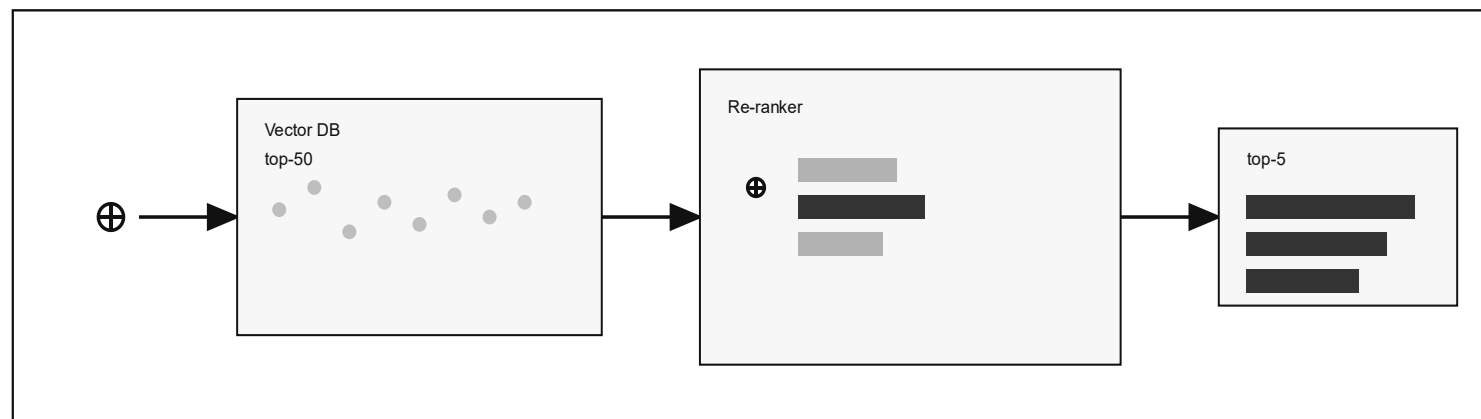
Two-stage retrieval: candidates shrink fast



Re-ranking: Si e rrisim cilësinë e top-k?

Për këtë arsye përdoret një hap i dytë, i quajtur re-ranking, i cili funksionon zakonisht kështu:

- Merret top-50 (ose top-100) kandidatë nga Vector DB.
- Këta kandidatë kalojnë në një model më të shtrenjtë dhe më të saktë.
- Modeli i rishikon kandidatët një për një, duke marrë parasysh plotësisht pyetjen, përmbajtjen esegmentit, dhe marrëdhënien e saktë mes tyre.
- Rezultatet rirenditen dhe përzgjidhen vetëm më të mirët (p.sh. top-5).



Praktika e mirë: çfarë duhet të log-osh?

- Në sisteme reale, veçanërisht në ato të bazuara në RAG, debugging pa logs është praktikisht i pamundur.
- Kur diçka shkon keq, pa të dhëna të regjistruara nuk është e mundur të kuptohet nëse problemi vjen nga kërkimi (retrieval) apo gjenerimi nga MMGj-ja.
- Çfarë duhet të regjistrohet (log-jet minimale)?
 - Teksti i pyetjes + një hash ose ID e vektorit të pyetjes
 - Cilat chunks u rikthyen nga Vector DB
 - Vlerat e similarity score-ve për secilin kandidat
 - Çfarë filtrash u aplikuan (*metadata, gjuhë, datë, autoritet, etj.*)
 - Nëse u përdor re-ranking dhe me cilin model

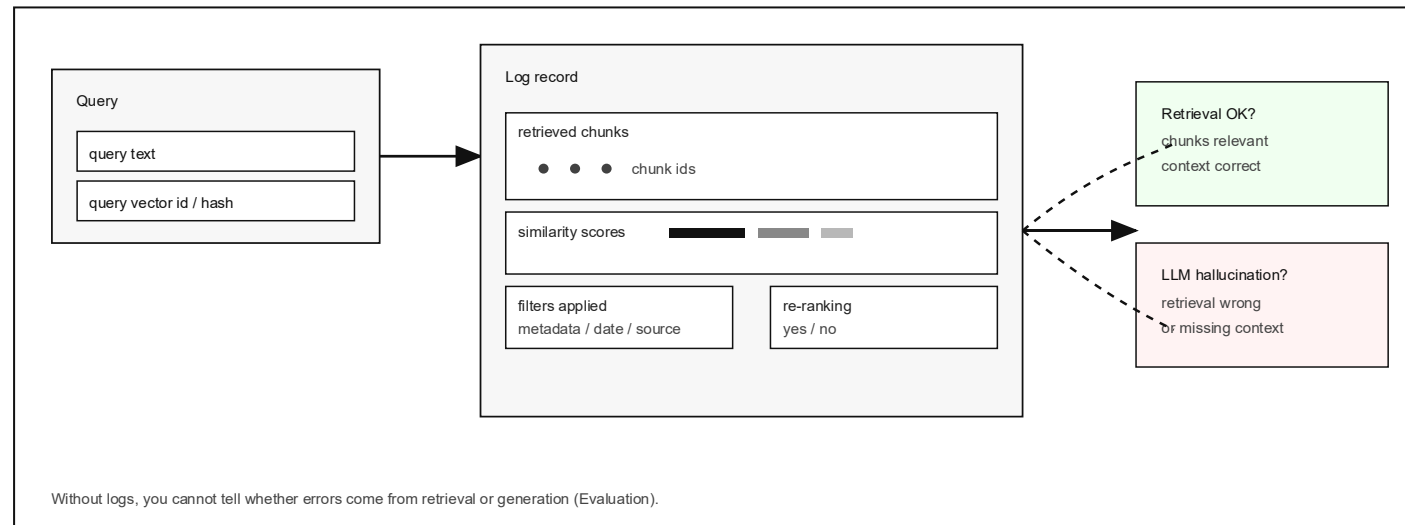


Praktika e mirë: çfarë duhet të log-osh?

Vetëm duke i analizuar këto të dhëna mund të përgjigjemi në pyetje kritike si:

- A është retrieval-i i saktë dhe i përshtatshëm?
- A po “halucinon” MMGJ-ja sepse chunk-et e rikthyera janë të gabuara ose të pamjaftueshme?

RAG debugging: log record structure



Gabimet më tipike

- **Model embedding i papërshtatshëm për domenin:** Nëse modeli i embeddings nuk është i përshtatur për domenin konkret, atëherë semantika kapet keq, terma teknikë interpretohen gabim, pyetjet duken “të ngjashme”, por nuk janë relevante.
- **Chunking i dobët:** Ndarja jo e duhur e dokumenteve çon në humbje konteksti (segmente shumë të vegjël), ose retrieval të paqartë dhe jo preciz (chunks shumë të gjatë).
- **Konfigurim i gabuar i indeksit:** Nëse ANN është shumë agresiv, atëherë kërkimi bëhet shumë i shpejtë por recall bie. Segmente të rëndësishme nuk rikthehen fare dhe kjo krijon një iluzion performance, por ul cilësinë reale.
- **Document drift:** Një rrezik shpesh i nënvlerësuar pasi dokumentet përditësohen por indeksi nuk rifreskohet. Si pasojë sistemi punon mbi informacion të vjetëruar. Përgjigjet duken të sakta, por janë faktikisht të gabuara.



Çfarë do të trajtojmë në vazhdimësi?

- ✓ Hyrje
- ✓ Tokenizimi & Embeddings
- ✓ Arkitektura Transformer
- ✓ Bazat e të dhënave vektoriale (Vector Databases)
- 5. RAG (Gjenerim i përforcuar nga kërkimi)
- 6. Promptimi
- 7. Agjentët (Agents)
- 8. Përshtatja e modelit (Fine-Tuning)
- 9. Vlerësimi (Evaluation)
- 10. Siguria & Përafrimi (Safety & Alignment)
- 11. Vendosja e modelit (Deployment)
- 12. Integrimi i Projektit Final



Bibliografi

- J. Johnson, M. Douze, and H. Jégou, “Billion-Scale Similarity Search with GPUs,” *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2021.
- M. Malkov and D. Yashunin, “Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 824–836, 2020.
- H. Jégou, M. Douze, and C. Schmid, “Product Quantization for Nearest Neighbor Search,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.
- N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” in *Proc. EMNLP*, 2019.
- P. Lewis *et al.*, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” in *Proc. NeurIPS*, 2020.

