Napredne arhitekture informacionih sistema

Vektorske baze podataka

Izvođači nastave:
dr Marko Vještica
Elena Akik
Sanja Radić

DAIS
Data Science and Information Systems Group

Sadržaj

Softverska podrška

- Za potrebe rada sa vektorskom bazom podataka, biće korišćen ***Milvus***, verzija v2.6.13
- ***Milvus*** je sistem otvorenog koda za upravljanje vektorskom bazom podataka, razvijen i održavan od strane kompanije *Zilliz*
- Projektovan je za skladištenje, indeksiranje i pretragu vektorskih ugradnji u realnom vremenu nad skupovima podataka koji broje milijarde vektora
- Prihvaćen kao projekat od strane *Linux Foundation AI & Data* (https://lfaidata.foundation/)
- Zvanična dokumentacija:
  - https://milvus.io/docs
- Referenca:
  - „Milvus: A Purpose-Built Vector Data Management System“ Wang et al., SIGMOD 2021 (https://dl.acm.org/doi/10.1145/3448016.3457550)

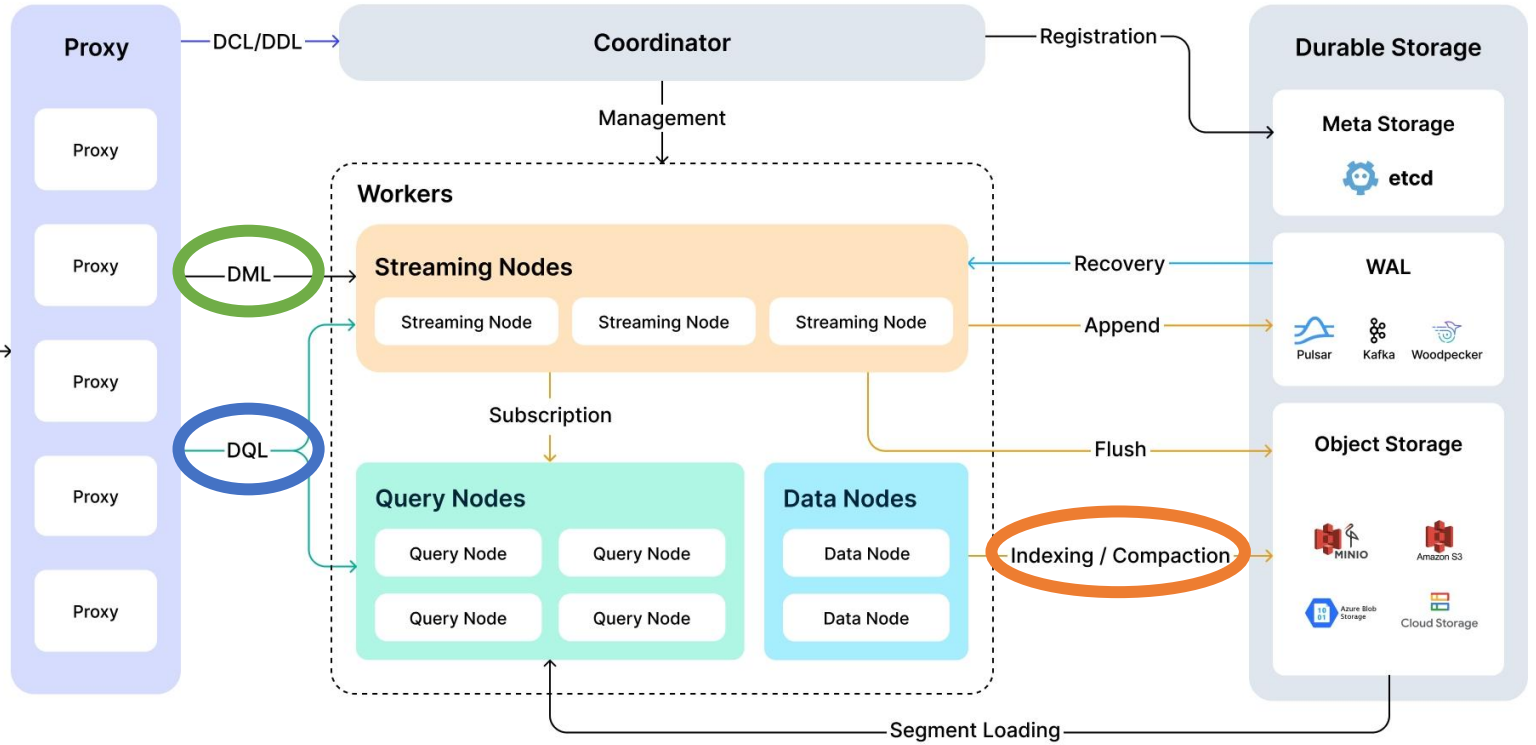
Softverska podrška

- Za potrebe rada sa vektorskom bazom podataka, biće korišćena vektorska baza podataka ***Milvus***, verzija v2.6.13
- *Milvus* je odabran zbog:
  - Specijalizacije za veštačku inteligenciju: dizajniran za upravljanje i pretragu milijardi vektorskih ugradnji
  - Visokih performansi: omogućava pretragu sličnosti u milisekundama, čak i nad ogromnim skupovima podataka
  - *Cloud-native* arhitekture: potpuno razdvajanje resursa za skladištenje i izračunavanje, što omogućava maksimalnu fleksibilnost i skalabilnost
  - Hibridne pretrage: podržava kombinovanje pretrage sličnosti sa tradicionalnim filtriranjem meta-podataka
  - Široke podrške indeksiranju: implementira napredne algoritme poput HNSW, IVD I DiskANN za optimalan balans između brzine i preciznosti
  - Visoke dostupnosti: obezbeđuje integritet podataka kroz *Write-Ahead Logging* (WAL) i distribuiranu obradu

Arhitektura vektorske baze podataka *Milvus*

Podaci ulaze preko proksija, idu u *Log Broker*, a zatim ih *Data Node* pakuje i šalje u *Object Storage*

Upit stiže do proksija, on ga šalje Query čvorovima, koji pretražuju podatke u memoriji i vraćaju rezultat



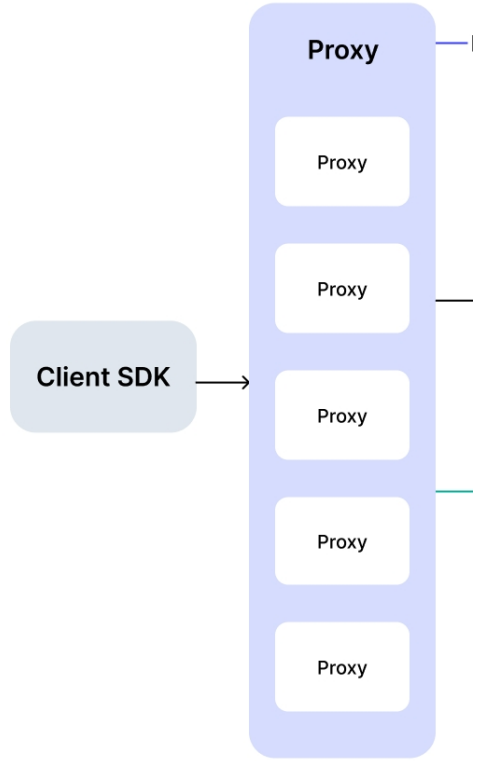
Sistem u pozadini stalno preuzima sirove podatke iz objektnog skladišta, gradi indekse i vraća ih nazad, kako bi pretraga bila brža

Figura 1. Prikaz arhitekture vektorske baze podataka *Milvus*
Izvor: Zvanična Internet strana dokumentacije vektorske baze podataka *Milvus*, link: https://milvus.io/docs/architecture_overview.md

Arhitektura vektorske baze podataka *Milvus*

Pristupni sloj (engl. *Access Layer*)

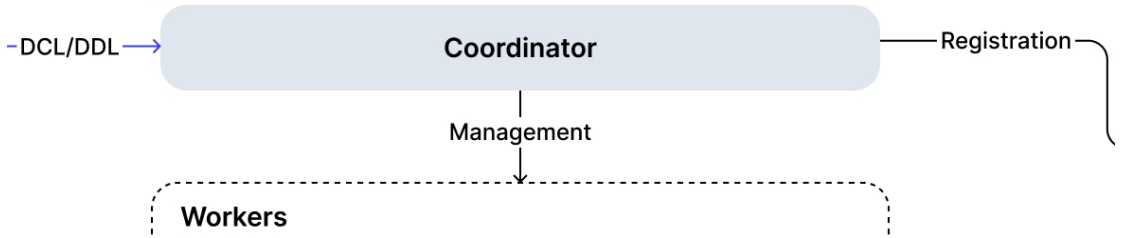
- **Proxy**
  - služi kao jedinstvena ulazna tačka sistema
  - Njegova uloga je validacija klijentskih zahteva, provera autorizacije i prosleđivanje upita odgovarajućim radnim čvorovima
  - Takođe vrši i agregaciju delimičnih rezultata pre slanja nazad klijentu



Arhitektura vektorske baze podataka *Milvus*

Koordinatorski servis (engl. *Coordinator Service*)

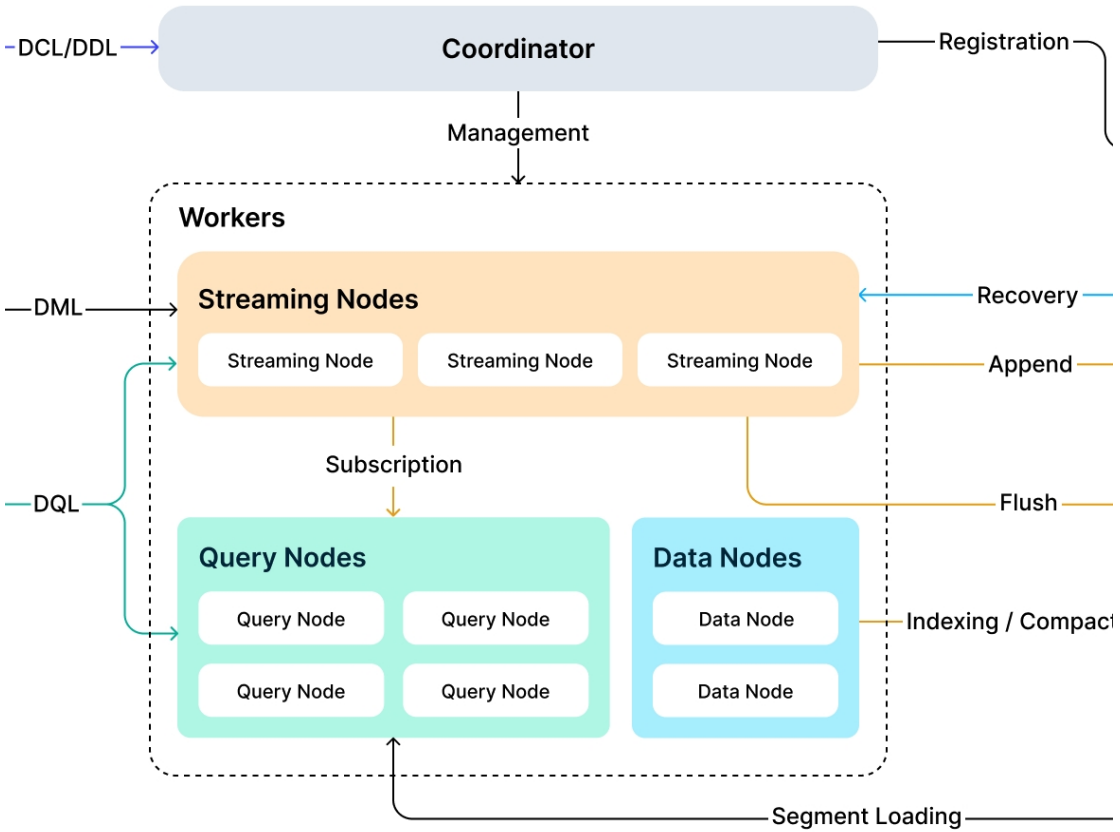
- **Glavni koordinator** (engl. *Root Coordinator*)
  - upravlja meta-podacima i dodeljuje vremenske oznake kako bi se osigurao redosled operacija
- **Koordinator upita** (engl. *Query Coordinator*)
  - nadgleda distribuciju podataka među upitnim čvorovima i brine o balansiranju opterećenja tokom pretrage
- **Koordinator podataka** (engl. *Data Coordinator*)
  - upravlja topologijom čvorova i odlučuje kada je potrebno izvršiti redukciju podataka radi uštede prostora



Arhitektura vektorske baze podataka *Milvus*

Radni čvorovi (engl. *Worker Nodes*)

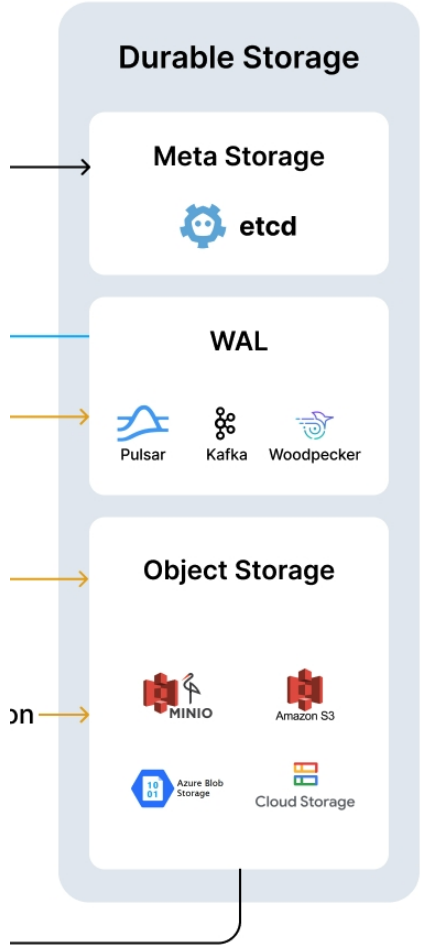
- **Čvorovi za upite** (engl. *Query Nodes*)
  - primaju zahteve za pretragu od proksija
  - drže podatke u RAM memoriji kako bi omogućili pretragu u realnom vremenu sa ekstremno niskim kašnjenjem
- **Čvorovi za podatke** (engl. *Data Nodes*)
  - preuzimaju podatke iz logova, organizuju ih u logičke segmente i šalju u trajno objektno skladište
- **Čvorovi za indeksiranje** (engl. *Index Nodes*)
  - specijalizovani za izgradnju indeksa
    - Izgradnja indeksa je procesorski intenzivna operacija, pa su ovi čvorovi odvojeni, kako ne bi usporavali samu pretragu



Arhitektura vektorske baze podataka *Milvus*

Trajno skladište (engl. *Durable Storage*)

- **Skladište meta-podataka** (engl. *Meta Storage – etcd*)
  - Čuva strukturu baze podataka, status čvorova i lokacije segmenata
  - Bez ove komponente, sistem ne bi znao gde se nalaze podaci
- **Posrednik za poruke** (engl. *Log Broker – Pulsar, Kafka, RocksDB*)
  - Mehanizam za osiguranje podataka u transakcijama
  - Svaka operacija pisanja se prvo beleži ovde (engl. *Write Ahead Log*), pre nego što se potvrdi, čime se garantuje da podaci neće biti izgubljeni u slučaju pada sistema
- **Objektno skladište** (engl. *Object Storage – miniO, S3*)
  - Glavno skladište za velike fajlove (indekse i sirove vektore)
  - Omogućava skladištenje neograničene količine podataka na ekonomičan način



Arhitektura vektorske baze podataka *Milvus*

- Postoje dva osnovna načina za pokretanje vektorske baze podataka *Milvus*
  - **Samostalni režim** (engl. *Standalone*)
    - jedna instanca koja pokreće sve komponente unutar jednog procesa
    - idealno za manje skupove podataka i lakša radna opterećenja (razvoj i testiranje)
    - eliminiše se potreba za složenim spoljnim servisima (poput *Kafka*-e ili *Pulsar*-a)
    - ovaj način će biti korišćen na vežbama
  - **Distribuirani režim** (engl. *Cluster*)
    - napredni način rada gde svaka komponenta radi nezavisno, na zasebnim serverima
    - pogodno za velike skupove podataka (milijarde vektora) i scenarije rada sa visokim opterećenjem
    - svaka komponenta može se zasebno skalirati

Arhitektura *Milvus standalone* sistema

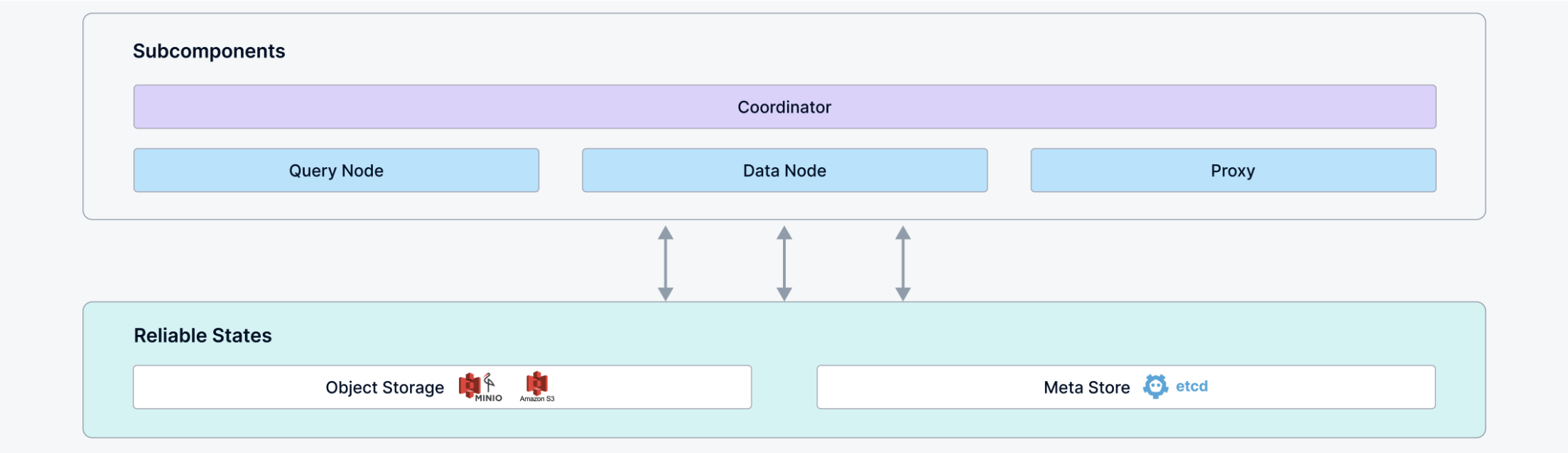


Figura 2. Prikaz arhitekture *Milvus standalone* sistema
Izvor: Zvanična Internet strana dokumentacije vektorske baze podataka *Milvus*,
link: https://milvus.io/docs/main_components.md

Arhitektura *Milvus standalone* sistema

- Komunikacija sa klijentom ide putem *gRPC* na portu 19530
- Pokretanje:

```
1  docker-compose up -d
```

- Konfiguracija kontejnera u okviru *docker-compose.yml* fajla je sledeća:
  - **milvusdb/milvus:v2.6.13**
    - Osnovna baza podataka, obuhvata pristupni sloj, koordinaciju i radne čvorove
  - **etcd**
    - Skladištenje meta-podataka i koordinacionog stanja
  - **minio**
    - Trajno skladištenje segmenata i indeksnih datoteka

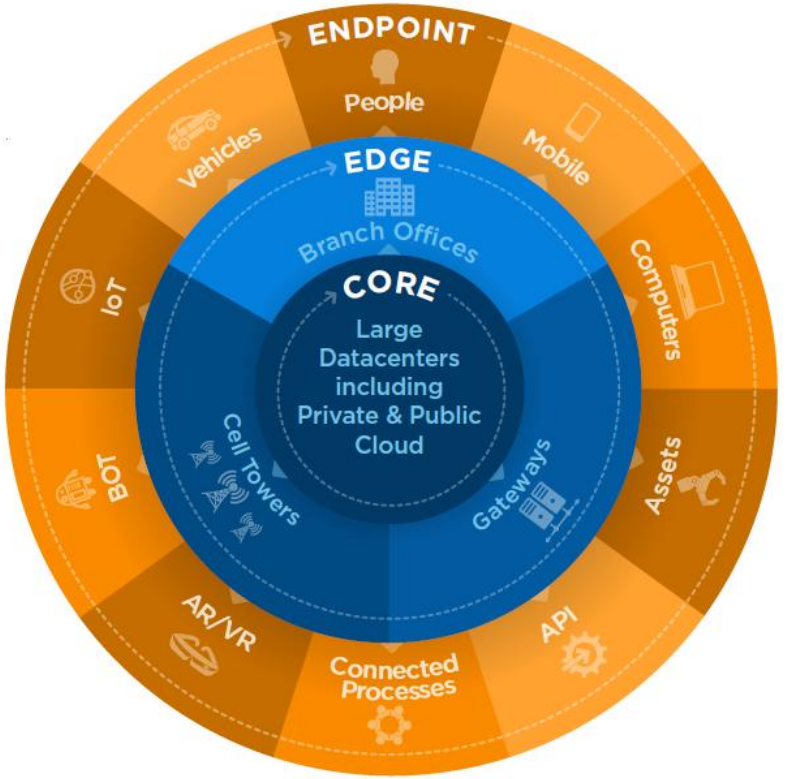
Sadržaj

Ograničenja tradicionalnih sistema za pretragu

- Tradicionalni sistemi za pretragu teksta zasnivaju se na **leksičkom podudaranju**, odnosno, **podudaranju nizova karaktera**
- **Problem**: semantički ekvivalentni pojmovi koji se leksički razlikuju ne bivaju pronađeni
  - Primer:
    - *Upit koji sadrži reč „majica“ ne pronalazi entitet koji sadrži frazu „polo majica“, niti „pamučna majica“*
    - *Upit koji sadrži uslov „LIKE ’%majica%’“ ne vraća nijedan rezultat ako reč majica ne postoji doslovno u atributu entiteta*
- Klasični sistemi ne mogu da odgovore na zahteve poput:
  - *„Pronađi slike koje liče na ovaj opis“*
  - *„Pronađi naučne radove koji govore o konceptu prikazanom na slici“*
- **Rešenje**:
  - reprezentacija semantičkog sadržaja u obliku numeričkog vektora – **vektorske ugradnje** (engl. *embedding vector*)

Uvod u vektorske baze podataka

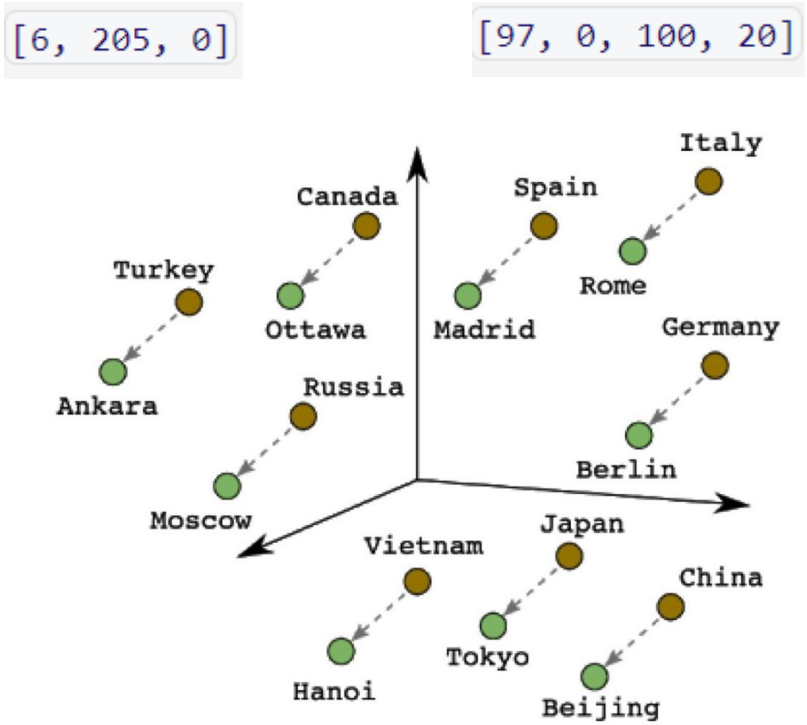
- **Vektorske baze podataka** služe za skladištenje i indeksiranje podataka različitih formata u formi vektora, odnosno, vektorske ugradnje (engl. *vector embedding*)
- Novi zahtevi koji se javljaju sa sve većom pojavom nestrukturiranih podataka:
  - *Potreba za većim memorijskim prostorom za skladištenje podataka*
  - *Duži vremenski period za izvođenje transformacije podataka, indeksiranje podataka, postavljanje upita nad podacima*



Uvod u vektorske baze podataka

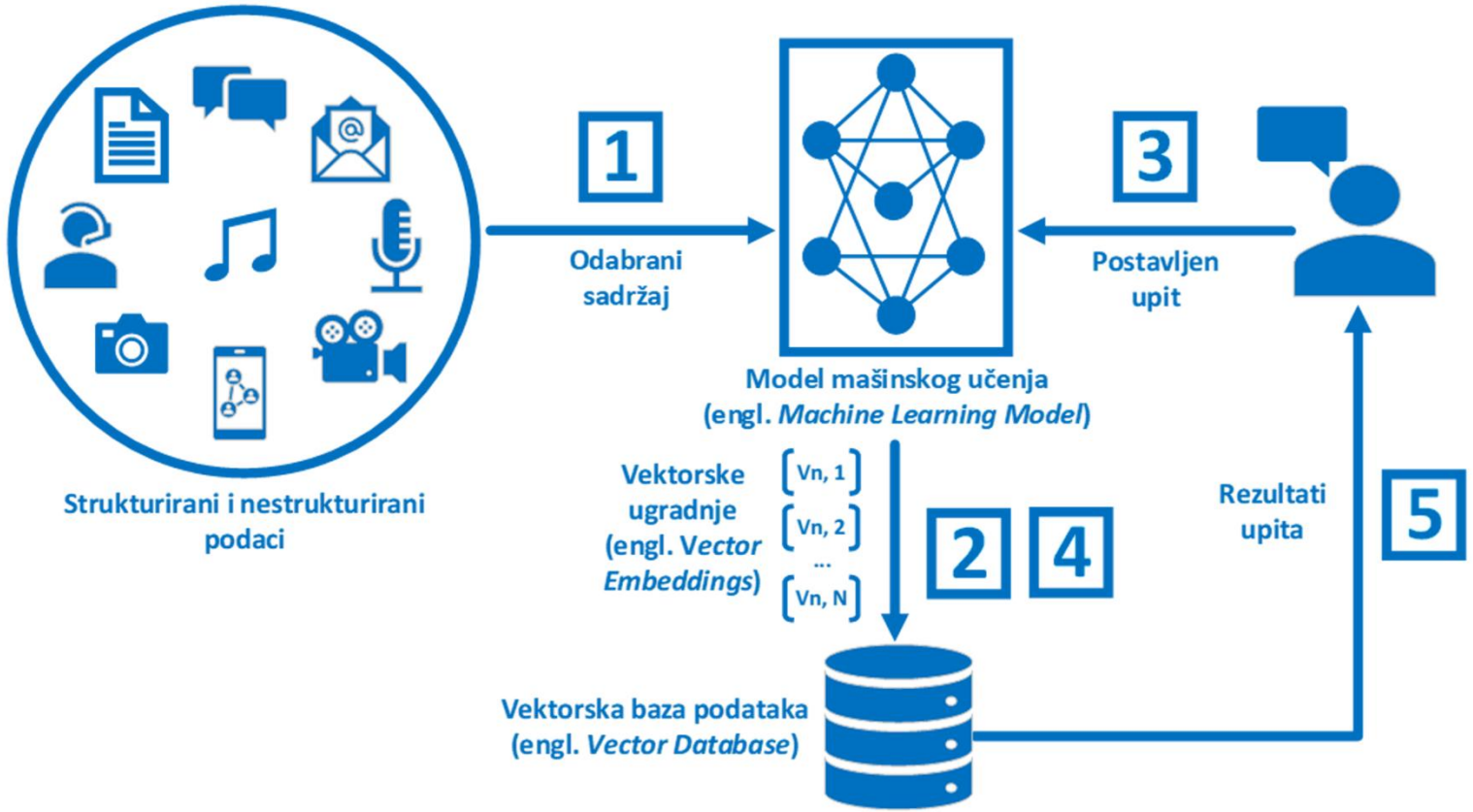
Uvod u vektorske baze podataka

- Osnovni pojmovi:
  - **Vektor** – N – dimenzioni niz vrednosti
  - **Vektorska ugradnja** – vektor koji reprezentuje podatke u vektorskom prostoru baze podataka, pri čemu semantički slični podaci imaju slične vektorske reprezentacije
  - **Pretraga vektora** – na osnovu vrednosti vektorskih ugradnji, moguće je izvršiti pretragu vektorskog prostora, odnosno, pretragu sličnih vektora



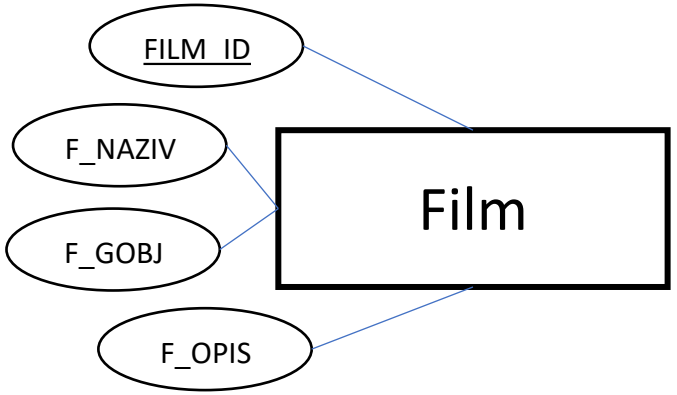
Uvod u vektorske baze podataka

- Prikaz rada sa vektorskom bazom podataka



Sadržaj

Osnovni koncepti vektorske baze



FILM_ID	F_NAZIV	F_GOBJ	F_OPIS
1	'Film 1'	1978	'Opis 1'
2	'Film 2'	1979	'Opis 2'
3	'Film 3'	1980	'Opis 3'
4	'Film 4'	1981	'Opis 4'

Klaster-Filmovi

Kolekcija-filmovi1

FILM_ID	F_NAZIV	F_GOBJ	F_NAZIV_VECTOR
1	'Film 1'	1978	[0.764, -0.980, 0.999, ...]
2	'Film 2'	1979	[-0.284, -0.640, 0.184, ...]
3	'Film 3'	1980	[-0.764, 0.187, -0.999, ...]
4	'Film 4'	1981	[0.764, -0.980, 0.999, ...]

Kolekcija-filmovi2

FILM_ID	F_OPIS	F_GOBJ	F_OPIS_VECTOR
1	'Opis 1'	1978	[-0.122, 0.187, -0.899, ...]
2	'Opis 2'	1979	[0.284, 0.880, -0.184, ...]
3	'Opis 3'	1980	[0.764, -0.337, 0.107, ...]
4	'Opis 4'	1981	[0.765, 0.280, -0.999, ...]

Osnovni koncepti vektorske baze

- **Klaster** – vođena *Milvus* instanca koja je povezana sa određenim računarskim resursima
- **Šema kolekcije** – struktura podataka koja pruža uvid u organizaciju podataka i veze između njih pre unosa u bazu
- **Kolekcija** – objekat u okviru klastera koji skladišti podatke u formi tabele, sa fiksnim brojem kolona i promenljviim brojem redova. U tabeli, svaki red predstavlja entitet, a svaka kolona određeni atribut tog entiteta
  - Ekvivalent šemi relacije u relacionim bazama podataka
- **Dinamička šema kolekcije** – prilikom unosa novih podataka nije neophodno poštovati format prvobitno formirane šeme kolekcije
  - *Primer*: Nakon formirane šeme sa 5 polja, ukoliko je potrebno dodati još dva polja, a podaci su već uneti u vektorsku bazu podataka, to je moguće uraditi ukoliko je prilikom kreiranja šeme kolekcije odobrena osobina dinamičnosti šeme kolekcije
- **Pojava tipa entiteta** – tip entiteta u kolekciji
  - Ekvivalent pojavi tipa entiteta u relacionim bazama podataka

Osnovni koncepti vektorske baze

- **Šema polja** – struktura koja definiše meta-podatke o polju entiteta
  - Mogući meta-podaci: naziv, tip podatka, maksimalna dužina, broj dimenzija, naznaka da je dato polje označeno kao jedinstveni identifikator
- **Polje** – instanca šeme polja
  - Polje može biti primarno ili neprimarno
  - Dozvoljeni tipovi podataka: *INT 8/32/64, DOUBLE, FLOAT, VARCHAR, BOOLEAN, JSON, FLOAT_VECTOR*
- **Ograničenje** – pravilo uspostavljeno na nivou određenog objekta baze podataka
  - Primer: nad poljem *F_NAZIV* moguće je definisati ograničenje maksimalnog broja karaktera, dok je nad vektorizovanim poljem *F_NAZIV_VECTOR* moguće definisati ograničenje dimenzija vektora
- **Indeks** – objekat baze podataka koji obezbeđuje direktan i brz pristup entitetima u tabeli
  - Formira se na osnovu određenog polja kolekcije
  - U *Zilliz Cloud* okruženju podržan je samo indeks tipa *AUTOINDEX*
  - Indeks se formira nad vektorizovanim poljem, u cilju što efikasnije pretrage
  - Moguće je kreirati indekse i za nevektorizovana polja

Sadržaj

Opšti postupak generisanja vektorskih ugradnji

- Bez obzira na modalitet ulaznih podataka, proces generisanja vektorske ugradnje prati isti konceptualni tok u četiri faze:
  - Pretprocesiranje – ulazni podatak se prevodi u format prihvatljiv za model
    - Tekst se tokenizuje, slika se deli na segmente, audio signal se transformiše u spektogram
    - Cilj je standardizacija ulaza u fiksnu strukturu
  - Enkodovanje – pretprocesirani ulaz prolazi kroz parametrizovanu neuronsku mrežu
  - Mreža uči reprezentacije u toku treniranja na velikim skupovima podataka uz odgovarajuću ciljnu funkciju
  - Agregacija – sekvencijalni ili prostorni izlaz enkodera (matrica reprezentacije) redukuje se u jedan vektor fiksne veličine
  - [opciono] Normalizacija – projekcija vektora na jediničnu sferu

Postupak generisanja vektorskih ugradnji za tekst

- Korak 1 – **Tokenizacija**
  - Ulazni tekst se razlaže na tokene prema rečniku modela korišćenjem algoritma podreči
  - Najčešći algoritmi: *WordPiece, Byte Pair Encoding* (*GPT* familija), *SentencePiece* (*mT5*, *LaBSE*)
    - Primer (WordPiece):
      - "running" → ["run", "##ning"]
      - "unbelievable" → ["un", "##believ", "##able"]
    - Prednost podreči: rečnik konačne veličine (~30 000 tokena) uz sposobnost reprezentacije reči koje nisu viđene u treniranju
  - Uz svaki token dodaju se specijalni tokeni: [CLS] na početak sekvence, [SEP] kao separator
    - [CLS] token se konvencionalno koristi za reprezentaciju cele sekvence
  - Ograničenje kontekstualnog prozora: modeli imaju maksimalnu dužinu ulazne sekvence u tokenima
    - Tekst duži od praga se skraćuje ili se primenjuje strategija segmentacije sa preklapanjem (engl. *chunking with overlap*)

Postupak generisanja vektorskih ugradnji za tekst

- Korak 2 – **Enkodovanje**
  - Reprezentacije na nivou tokena se inicijalizuju sintetizovanjem vektorskih ugradnji tokena, pozicionih vektorskih ugradnji i vektorskih ugradnji segmenata
  - Poziciona vektorska ugradnja – kodira poziciju tokena u sekvenci
    - *BERT* koristi apsolutne naučene pozicione vektorske ugradnje
    - Noviji modeli, poput *RoPE* ili *ALiBi*, koriste relativne pozicione kodove, koji se bolje generalizuju na duže sekvence
  - Prolazak kroz L slojeva transformer enkodera
    - Svaki sloj sadrži mehanizam višeslojne pažnje (engl. *Multi-head self attention*), normalizaciju sloja i neuronsku mrežu sa direktnim prostiranjem
    - Mehanizam pažnje – svaki token vidi sve ostale tokene u sekvenci, što omogućava kontekstualizaciju značenja
    - $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$
    - Referenca: "Attention Is All You Need„ Vaswani et al, 2017 (https://arxiv.org/abs/1706.03762)

Postupak generisanja vektorskih ugradnji za tekst

- Korak 3 – **Agregacija**
- Tri standardne strategije:
  - **Sažimanje pomoću klasifikacionog tokena** (engl. *Classification token pooling*) – koristi se izlazna reprezentacija [CLS] tokena
    - Korišćeno je u originalnom *BERT*-u za klasifikacione zadatke
  - **Sažimanje usrednjavanjem** (engl. *Mean pooling*) - aritmetička sredina izlaznih reprezentacija svih tokena (osim specijalnih)
    - Superiorniji za semantičku sličnost prema "Sentence-BERT" Reimers & Gurevych, 2019 (https://arxiv.org/abs/1908.10084)
    - Korišćen u all-MiniLM-L6-v2
  - **Maksimalno sažimanje** (engl. *Max pooling*) - uzima maksimalnu vrednost po svakoj dimenziji
    - ređe u upotrebi za semantičku pretragu
- Korak 4 – **Normalizacija**
  - *L2* normalizacija: $\hat{v} = v / \| v \|_2$
  - Kosinusna sličnost normalizovanih vektora: $\cos(u, v) = u \cdot v$
  - Milvus parametar: `normalize_embeddings=True` u SentenceTransformers biblioteci

Postupak generisanja vektorskih ugradnji za vizuelni sadržaj

- Korak 1 – **Pretprocesiranje slike**
  - Promena rezolucije: slika se skalira na fiksnu dimenziju prihvatljivu za model
    - Standardne vrednosti: 224x224 piksela (*ResNet, ViT-B*), 336x336 piksela (*ViT-L*/14@336)
  - Normalizacija piksela: vrednosti piksela [0, 255] → [0.0, 1.0]
    - Potom, srednja vrednost se oduzima i deli standardnom devijacijom kanala po statistikama skupa
  - Augmentacija: obuhvata nasumično isecanje i horizontalno rotiranje
    - Vrši se samo pri treniranju
- Korak 2 – **Enkodovanje**
- Dva dominantna pristupa:
  - **Konvolucione neuronske mreže**
    - Hijerarhijsko izdvajanje obeležja: niži slojevi detekuju ivice i teksture, dok dublji slojevi prepoznaju semantičke koncepte (lica, objekte, scene)
      - Primer: ResNet-50 – arhitektura od 50 slojeva sa rezidualnim vezama, gde je izlaz poslednjeg konvolutivnog sloja tenzor dimenzija
      - Agregacijom ovog tenzora dobijamo vektor od 2048 dimenzija
    - Referenca: "Deep Residual Learning for Image Recognition„ He et al, 2015 (https://arxiv.org/abs/1512.03385)
  - Vision Transformer

Postupak generisanja vektorskih ugradnji za vizuelni sadržaj

- Korak 2 – **Enkodovanje**
- Dva dominantna pristupa:
  - Vision Transformer
    - Slika se deli na nepreklapajuće isečke fiksne veličine
    - Svaki isečak se linearno projektuje u vektorsku reprezentaciju
    - Vrši se prolazak kroz standardni trensformer enkoder
      - Referenca: "An Image is Worth 16x16 Words" Dosovitskiy et al, 2020 (https://arxiv.org/abs/2010.11929)

Postupak generisanja vektorskih ugradnji za vizuelni sadržaj

- Korak 3 – **Agregacija**
  - Globalno sažimanje prosekom (engl. *Global average pooling*): izračunava se prostorna srednja vrednost po svim pozicijama tenzora obeležja, čime se dobija vektor fiksne veličine
  - Klasifikacioni token: specijalni token dodaje se na početak niza isečaka i njegova izlazna reprezentacija koristi se kao sažeta reprezentacija cele slike
- Korak 4 – **Projekcija** [opciono]
  - U multi-modalnim modelima (poput CLIP-a)
    - Izlazni vektor vizuelnog enkodera prolazi kroz dodatni linearni projekcioni sloj
    - Ovaj sloj preslikava vektor u zajednički multi-modalni prostor
    - Isti postupak primenjuje se na izlaz tekstualnog enkodera, čime se osigurava uporedivost vektora različitih modaliteta

Postupak generisanja vektorskih ugradnji za audio i video sadržaj

- Korak 1 – **Pretprocesiranje audio signala**
  - Sirovi audio signal: jednodimenzionalni vremenski niz uzorkovanih amplituda
    - tipična frekvencija uzorkovanja: 16 000 Hz ili 22 050 Hz
  - Kratkotrajna Fourierova transformacija (STFT) (engl. *Short-Time Fourier Transform*): signal se deli na kratke vremenski preklaplajuće prozore
    - za svaki prozor računaju se frekvencijske komponente
    - dobija se dvodimenzionalna reprezentacija: osa X = vreme, osa Y = frekvencija
  - Mel spektrogram: frekventna osa *STFT*-a preslikava se na Mel skalu (perceptualno linearnu ljestvicu blisku ljudskom sluhu)
    - rezultat je Mel spektrogram oblika
    - tretira se kao „slika" i ulazi u vizuelni enkoder ili specijalizovani audio enkoder
  - Log-Mel spektrogram: primeniti logaritam na energiju
    - odgovara logaritmičkoj prirodi percepcije glasnoće (*Weber–Fechnerov* zakon)

Postupak generisanja vektorskih ugradnji za audio i video sadržaj

- Korak 2 – **Enkodovanje**
- Dva pristupa:
  - **CNN + RNN / Transformer** nad Mel spektrogramom: mel spektrogram tretiran kao 2D slika, konvolucijama se ekstrahhuju lokalne odlike, potom sekvencijalni model uči temporalne zavisnosti
  - **Wav2Vec 2.0 / HuBERT** (Meta AI): direktan rad nad sirovim signalom *(waveform)*
    - konvolutivni modul ekstrahhuje lokalne odlike, Transformer enkoder modeluje dugoročne zavisnosti
    - trenirano samonadziranim učenjem *(self-supervised learning)* na nelabeliranom audio sadržaju
  - **Whisper enkoder** (OpenAI): Log-Mel spektrogram → Transformer encoder
    - primarno razvijen za prepoznavanje govora, ali enkoder se koristi i za audio ugradnje
- Korak 3 – Agregacija
  - **Mean pooling** po vremenskoj osi: aritmetička sredina svih vremenskih okvira pretvara se u vektor fiksne dimenzije bez obzira na trajanje ulaznog audio isečka

Postupak generisanja vektorskih ugradnji za audio i video sadržaj

- Video zapisi su po svojoj prirodi multi-modalni – sastoje se od vizuelnog toka, audio toka i opciono tekstualnog toka
- Pristup A – **Nezavisno enkodovanje modaliteta i fuzija**
  - Vizuelni tok: vrši se uzorkovanje ključnih kadrova, pri čemu se svaki kadar enkoduje vizuelnim enkoderom, te se dobije niz vizuelnih vektora
  - Audio tok: pretprocesiranje i enkodovanje (prema prethodno opisanom postupku)
  - Fuzija: primenjuje se konkatenizacija vektora modaliteta ili mehanizam pažnje za fuziju, gde se generiše jedinstveni video vektor
- Pristup B – **Video Transformer**
  - Prostorno-vremenska tokenizacija isečaka: video se tretira kao 3D tenzor dimenzija
  - Tenzor se deli na 3D isečke koji simultano kodiraju i prostorne i vremenske informacije
    - Primer: **VideoMAE** (Tong et al, 2022) – maskirano auto-enkodovanje video isečaka
      - treniranje putem samonadziranog učenja
    - Primer: **CLIP4Clip** – proširenje CLIP modela na video pretragu
      - niz kadrova se enkoduje CLIP vizuelnim enkoderom, nakon čega sledi agregacija po vremenskoj osi

Postupak generisanja vektorskih ugradnji

- Kategorije modela za generisanje vektorskih ugradnji
  - **Opštenamenski tekstualni modeli** (engl. *General-purpose sentence encoders*)
    - Trenirani na velikim, raznorodnim tekstualnim korpusima
    - Pogodni za semantičku pretragu i rangiranje u opštem domenu
    - all-MiniLM-L6-v2, all-mpnet-base-v2, paraphrase-MiniLM-L3-v2
  - **Retrieval-optimizovani modeli** (engl. *Asymmetric retrieval models*)
    - Trenirani na asimetričnim parovima upit-dokument
    - Nadmašuju opšte modele na *benchmark* skupovima za pretragu informacija
    - bge-large-en-v1.5, e5-large-v2, bge-m3
  - **Višejezični modeli** (engl. *Multilingual models*)
    - Zajednički vektorski prostor za 50-100 jezika
    - paraphrase-multilingual-mpnet-base-v2, multilingual-e5-large, LaBSE

Postupak generisanja vektorskih ugradnji

- Kategorije modela za generisanje vektorskih ugradnji
  - **Multi-modalni modeli** (engl. *Cross-modal models*)
    - Enkoduju više modaliteta (tekst, slika, audio) u zajednički vektorski prostor
    - CLIP ViT-B/32, CLIP ViT-L/14, ImageBind
  - **Domenski specijalizovani modeli** (engl. *Domain-adapted models*)
  - **Retki modeli** (engl. *Sparse neural retrieval models*)
  - **Modeli zasnovani na velikim jezičkim modelima** (engl. *LLM-based embedding models*)
    - Koriste enkoder ili dekoder velikih jezičkih modela umesto BERT bazirane arhitekture
    - Trenutno vodeći na MTEB listi po kvalitetu
    - text-embedding-3-large, Gemini text-embedding-004, NV-Embed-v2

Postupak odabira modela za generisanje vektorskih ugradnji

- Kriterijumi za odabir modela:
  - **Dimenzionalnost izlaznog vektora** – veća dimenzija nosi više informacija, ali povećava memoriju
    - Za N vektora dimenzije d memorija raste kao *O(N*d*4)* bajtova
  - **Domenska pokrivenost** – opšti modeli rade dobro za generalne modele, dok se za specijalizovane domene (biomedicina, pravo) koriste *fine-tuned* modeli
  - **Jezička pokrivenost** – monojezični i višejezični modeli
  - **Veličina modela i latencija inferencije** – kompaktni modeli i veliki modeli (veličina se ogleda kroz broj parametara)
    - Relevantno za sisteme koji vrše obradu podataka u realnom vremenu
- Gde pronaći modele?
  - **HuggingFace Hub** *(https://huggingface.co/models?pipeline_tag=sentence-similarity)*
    - centralni repozitorijum modela
  - **Massive Text Embedding Benchmark (MTEB) Leaderboard** *(https://huggingface.co/spaces/mteb/leaderboard)*
    - rangiranje modela po zadacima: Retrieval, STS, Clustering, Classification
    - metrike: NDCG@10, MRR@10, Recall@k

Evaluacija modela za generisanje vektorskih ugradnji

- **Unutrašnja evaluacija**
  - *MTEB Retrieval* zadatak – model se testira nad standardnim skupovima
  - Metrika *Normalized Discounted Cumulative Gain* (NDCG@10) – normalizovani diskontovani kumulativni dobitak za prvih 10 rezultata
    - Proste metrike (preciznost ili odziv) posmatraju da li je nešto dobro pronađeno, a NDCG meri koliko je o dobro urađeno
    - Meri sledeće komponente:
      - Relevantnost – svaki entitet dobija ocenu korisnosti (nebitno, delimično, veoma relevantno)
      - Poziciju – što je entitet niže na listi, njegova vrednost se „kažnjava“ (diskontuje)
      - Kumulativnost – sabira se vrednost svih prvih N rezultata kako bi se dobila jedna ocena za taj upit
      - Normalizaciju – deli se stvarni rezultata sa „idealnim“ rezultatom (koji bi bio postignut da su entiteti poređani u „savršenom“ porektu od najbitnijeg do najmanje bitnog). Ovde se dobija broj između 0 i 1.
    - Viši **NDCG@10** na opštim testovima ne garantuje bolje performanse u specifičnom domenu
    - Uvek treba uraditi validaciju nad sopstvenim podacima

Evaluacija modela za generisanje vektorskih ugradnji

- **Spoljašnja evaluacija**
  - Podrazumeva proveru učinka modela u realnom poslovnom procesu ili finalnoj verziji aplikacije
  - Unutrašnja evaluacija meri koliko je model „pametan“ na testu, a spoljašnja koliko je model „koristan“ u radu
  - Obuhvata sledeće:
    - Evaluaciju u okviru RAG sistema – vernost, relevantnost, stopa halucinacije
    - Poslovne i korisničke metrike (KPIs) – stopa konverzije, korisničko zadovoljstvo
    - Operativne performanse – latencija, trošak skladištenja
    - Validaciju na specifičnom domenu – testiranje nad „rečnikom“ korisnika, A\B testiranje
    - Robustnost i bezbednost / osetljivost na šum, prompt injection

Sadržaj

Osnovne operacije nad vektorskom bazom

- Kreiranje kolekcije
- Priprema podataka
- Unos podataka
- Postavljanje upita
- Brisanje entiteta
- Uklanjanje kolekcije

Osnovne operacije nad vektorskom bazom

- **Kreiranje kolekcije** – nakon kreiranja klastera, moguće je formirati novu kolekciju koja će sadržati podatke.
- Prilikom formiranja kolekcije, potrebno je definisati korisničke kredencijale.

```
from pymilvus import MilvusClient

CLUSTER_ENDPOINT = <vrednost_cluster_endpoint_parametra>
TOKEN = <vrednost_tokena>
COLLECTION_NAME = <naziv_kolekcije>
```

```
client = MilvusClient (uri = <CLUSTER_ENDPOINT>, token = <TOKEN>
```

- Potom, poziva se instanca *Milvus* klijenta, u cilju kreiranja kolekcije.

```
client.create_collection(
   collection_name = <COLLECTION_NAME>, # predefinisani naziv kolekcije
   dimension = <dimenzija_vektora>)  # dimenzija vektorizovanog polja
)
```

Osnovne operacije nad vektorskom bazom

- **Priprema podataka** predstavlja kompleksan korak pre rada sa vektorskom bazom. U toku tog procesa, neophodno je izvršiti sledeće pobrojane korake:
  - Prikupljanje podataka
    - Korisnički podaci, fizički unos, generisanje podataka
    - Predefinisani skup podataka
      - Kako postoji vektorizovano polje, pre unosa podataka u bazu neophodno je formirati dato polje na adekvatan način, shodno polju za koje se kreira vektorizovano polje i broju dimenzija vektora.
  - Definisanje šeme polja
    - Definisanje meta-podataka za polje atributa: naziv, opis, indikator da li je jednistveni identifikator, ograničenje maksimalne dužine, broj dimenzija...
  - Definisanje šeme kolekcije
    - Nakon formiranja šema za svako polje kolekcije, formira se šema kolekcije
  - Kreiranje indeksa nad poljem kolekcije
    - U cilju efikasnije pretrage podataka, nad vektorizovanim poljem kreira se indeks

Osnovne operacije nad vektorskom bazom

- **Priprema podataka**
  - Prikupljanje podataka

```
DATASET_PATH = <putanja_do_lokacije_skupa_podataka>

# način za čitanje podataka iz skupa podataka, u formi red po red
with open(DATASET_PATH) as f:
   data = json.load(f)
   rows = data['rows']
```

  - Definisanje šeme polja

```
field = FieldSchema(
   name = <naziv_polja>,
   dtype = <tip_podatka>,
   is_primary = <indikator_PK>,
   max_length = <brojčana_vrednost>,
   dim = <dimenzija_vektora>)
```

Osnovne operacije nad vektorskom bazom

- **Priprema podataka**
  - Definisanje šeme kolekcije

```
schema = CollectionSchema (
   fields = [<lista_šema_polja>],
   description = <opis_kolekcije>,
   enable_dynamic_fields = <indikator_odobrenja_dinamičkog_polja>
   )
```

  - Kreiranje kolekcije sa predefinisanim parametrima

```
collection = Collection(
   name = <naziv_kolekcije>,
   description = <opis_kolekcije>,
   schema = <naziv_šeme>
   )
```

Osnovne operacije nad vektorskom bazom

- **Priprema podataka**
  - Kreiranje indeksa nad poljem kolekcije

```
index_params = {
   "index_type": <tip_indeksa>,
   "metric_type": <naziv_metrike>,
   "params": <dodatni_parametri>
   }

collection.create_index(
   field_name = <naziv_polja>,
   index_params = <parametri_indeksa>,
   index_name = <naziv_indeksa>
   )
```

Osnovne operacije nad vektorskom bazom

```
CREATE TABLE Film (
           FILM_ID INTEGER NOT NULL,
           F_NAZIV VARCHAR2(30) NOT NULL,
           F_GOBJ INTEGER NOT NULL,
           F_OPIS VARCHAR2(250)
);
```

```
id_field = FieldSchema(name = ’film_id’, dtype = DataType.INT64,
                       is_primary = True)

fnaziv_field = FieldSchema(name = ’f_naziv’, dtype = DataType.VARCHAR,
                           max_length = 30 )

fgobj_field = FieldSchema(name = ’f_gobj’, dtype = DataType.INT64,
                          max_length = 30 )

fnaziv_vec_field = FieldSchema(name = ’f_naziv_vector’,
                               dtype = DataType.FLOAT_VECTOR,
                               max_length = 30 )
```

```
schema = CollectionSchema(
           fields=[id_field, fnaziv_field,
                   fgobj_field, fnaziv_vec_field],
           description='Title collection')
```

```
index_params = {
    'metric_type':'L2',
    'index_type':"AUTOINDEX",
    'params':{}     }
collection.create_index(field_name=“fnaziv_vec_field",index_params=index_params)
```

Osnovne operacije nad vektorskom bazom

- **Unos podataka**

```
# Pražnjenje kolekcije pre unosa podataka
client.delete(collection_name = <naziv_kolekcije>, pks = <brojčana_vrednost>)

# Unos pretprocesuiranih podataka u bazu
client.insert(
   collection_name = <naziv_kolekcije>,
   data = <formirani_skup_podataka> # proslediti pretprocesuiran skup podataka
   )
```

Osnovne operacije nad vektorskom bazom

- **Postavljanje upita**
  - **Osnovna pretraga**
    - Pretraga na osnovu vrednosti jednog vektora

```
search_vector1 = [<vektorska_ugradnja>]

result = client.search(
   collection_name = <naziv_kolekcije>,
   data = [<vektor_upita>],
   output_fields = [<lista_polja_u_rezultatu>],
   limit = <broj_izlaznih_rezultata>
                              )
# Ispis rezulatata u konzoli
print(json.dumps(result, indent = 4))
```

Osnovne operacije nad vektorskom bazom

- Lista operatora:

Operator	Opis operatora
add (&&)	Tačno, ako su oba operanda tačna
or (||)	Tačno, ako je bar jedan operand tačan
+, -, *, /	Sabiranje, oduzimanje, množenje, deljenje
**	Eksponent
%	Modus (džoker znak uz operator *like*)
<, >	Manje od, veće od
==, !=	Jednako, različito
<=, >=	Manje ili jednako, veće ili jednako
not	Negira dobijeno stanje uslova
like	Poredi vrednosti koristeći džoker operatore
In	Testira da li izraz odgovara bilo kojoj vrednosti u listi vrednosti

Osnovne operacije nad vektorskom bazom

- **Postavljanje upita**
  - **Osnovna pretraga**
    - Pretraga sa filterima
      - **Filter** – logički izraz koji se koristi za specifikaciju uslova *ANN* pretrage. Moguće je koristiti aritmetičke, logičke ili operatore poređenja, u cilju formiranja filtera.

```
search_vector1 = [<vektorska_ugradnja>]

result = client.search(
   collection_name = <naziv_kolekcije>,
# opciono polje, u slučaju da se ne radi pretraga sličnosti, nego samo pretraga nevektorizovanih
polja
   [data = <vektor_upita>],
   filter = <(napredni)uslov_filtriranja>
   output_fields = <[lista_polja_u_rezultatu]>,
   limit = <broj_izlaznih_rezultata>
   )
# Ispis rezultata u konzoli, u json formatu
print(json.dumps(result, indent = 4)) # indent – broj razmaka u sledećem redu u json formatu
```

Osnovne operacije nad vektorskom bazom

- **Brisanje entiteta**
  - Brisanje jednog entiteta, na osnovu identifikatora

```
client.delete(
   collection_name = <naziv_kolekcije>,
   pks = <vrednost_identifikatora>
   )
```

  - Brisanje više entiteta istovremeno, na osnovu identifikatora

```
client.delete(
   collection_name = <naziv_kolekcije>,
   pks = <[lista_identifikatora]>
   )
```

- **Uklanjanje kolekcije**

```
client.drop_collection(
   collection_name = <naziv_kolekcije>
   )
```

Pretraga približnih najbližih suseda (engl. *Approximate Nearest Neighbor search*)

- Pretraga približnih najbližih suseda sprovodi pretragu po sličnosti uz opciono skalarno filtriranje
- Pretražuje najsličnije vektore u odnosu na upitne vektore
- Vraća entitete koji su semantički slični upitu
- Podržava uslove filtriranja radi sužavanja rezultata
- Glavni tok:
  - Prihvata upitni vektor (vektore) kao ulazne podatke
  - Primenjuje opcioni izraz za skalarno filtriranje (ukoliko je zadat)
  - Vrši pretragu koristeći specifičan tip metrike
  - Vraća top K najsličnijih entiteta, zajedno sa njihovim distancama

Pretraga približnih najbližih suseda (engl. *Approximate Nearest Neighbor search*)

- Sintaksa:

```
res = client.search(
    collection_name="<naziv_kolekcije>",
    data=[[<vektor_upita>]],
    limit=<top_k>,
    filter='<uslov_filtriranja>',
    output_fields=["<naziv_polja>"],
    search_params={"metric_type": "<naziv_tipa_metrike>", "params": {}}
)
```

Hibridna pretraga (engl. *Hybrid Search*)

- Hibridna pretraga izvršava pretragu nad više vektora
- Kombinuje rezultate iz više različitih vektorskih polja
- Vraća rezultate pretrage nakon procesa ponovnog rangiranja
- Podržava različite parametre pretrage za svako pojedinačno vektorsko polje
- Glavni tok:
  - Prihvata više zahteva za pretragu, gde se svaki odnosi na različito vektorsko polje
  - Svaki zahtev ima sopstveno vektorsko polje i specifične parametre pretrage
  - Preuzima najbolje rezultate za svaki pojedinačni zahtev
  - Kombinuje i ponovo rangira rezultate koristeći specifičan algoritam za rangiranje
    - *WeightedRanker* – težinski
    - *RRFRanker - Reciprocal Rank Fusion*
  - Vraća konačne, spojene i ponovo rangirane rezultate

Hibridna pretraga (engl. *Hybrid Search*)

- Sintaksa:

```
res = collection.hybrid_search(
    reqs=[
        AnnSearchRequest(
            data=[[<vektor_upita_1>]],
            anns_field="<vektorizovano_polje_1>",
            param={"metric_type": "<naziv_tipa_metrike>", "params": {"nprobe": <vrednost_nprobe>}},
            limit=<vrednost_limita>
        ),
        AnnSearchRequest(
            data=[[<vektor_upita_2>]],
            anns_field="<vektorizovano_polje_2>",
            param={"metric_type": "<naziv_tipa_metrike>", "params": {"nprobe": <vrednost_nprobe>}},
            limit=<vrednost_limit>
        )
    ],
    rerank=WeightedRanker(<težina_1>, <težina_2>),
    limit=<finalna_vrednost_limita>
)
```

Pretraga u opsegu (engl. *Range Search*)

- Pretraga u opsegu pronalazi vektore unutar određenog radijusa u odnosu na upitni vektor
- Sužava opseg pretrage samo na vektore koji se nalaze unutar specifičnog opsega sličnosti
- Koristi parametre radijus i range_filter za definisanje granica pretrage
- Glavni tok:
  - Prihvata upitni vektor i parametre za radijus i filter opsega
  - Vrši pretragu nad vektorima unutar zadatog opsega
  - Vraća sve entitete koji se nalaze unutar definisane granice sličnosti

Pretraga u opsegu (engl. *Range Search*)

- Sintaksa:

```
search_params = {
    "metric_type": "<naziv_tipa_metrike>",
    "params": {
        "radius": <vrednost_radijusa>,
        "range_filter": <vrednost_range_filtera>
    }
}

res = client.search(
    collection_name="<naziv_kolekcije>",
    data=[[<vektor_upita>]],
    limit=<vrednost_limita>,
    search_params=search_params
)
```

Iterator pretrage (engl. *Search Iterator*)

- Iterator pretrage vraća iterator za prolazak kroz rezultate pretrage
- Koristan je kada rezultati pretrage sadrže veliku količinu podataka
- Omogućava grupnu obradu rezultate pretrage
- Glavni tok:
  - Inicijalizuje iterator pretrage sa zadatim parametrima pretrage
  - Prolazi kroz rezultate u grupama
  - Obrađuje svaku grupu rezultata pojedinačno
  - Zatvara iterator nakon završetka obrade

Iterator pretrage (engl. *Search Iterator*)

- Sintaksa:

```
1  R<SearchIterator> response = milvusClient.searchIterator(SearchIteratorParam.newBuilder()
2      .withCollectionName("<naziv_kolekcije>")
3      .withBatchSize(<veličina_batch_grupe>)
4      .withVectorFieldName("<naziv_vektorizovanog_polja>")
5      .withFloatVectors(<vektor>)
6      .withParams(<vrednost_parametara>)
7      .withMetricType(MetricType.<naziv_tipa_metrike>)
8      .build());
9
10 SearchIterator searchIterator = response.getData();
11 while (true) {
12     List<QueryResultsWrapper.RowRecord> batchResults = searchIterator.next();
13     if (batchResults.isEmpty()) {
14         searchIterator.close();
15         break;
16     }
17     // Process batch
18 }
```

Pretraga punog teksta (engl. *Full Text search – BM25*)

- Pretraga punog teksta izvršava tekstualnu pretragu zasnovanu na ključnim rečima koristeći BM25 algoritam
- Pretvara tekst u retke vektore (engl. *sparse vectors*) koji predstavljaju BM25 rezultate
- Omogućava precizno podudaranje ključnih reči uporedo sa semantičkom pretragom
- Podržava višejezičnu analizu teksta
- Glavni tok:
  - Unosi se sirovi tekst
  - *Milvus* automatski generiše i skladišti retke vektore
  - BM25 algoritam rangira rezultate na osnovu učestalosti termina i relevantnosti
  - Može se kombinovati sa hibridnom pretragom

Pretraga punog teksta (engl. *Full Text search – BM25*)

- Sintaksa:

```
1   from langchain_milvus import Milvus, BM25BuiltInFunction
2
3   vectorstore = Milvus.from_documents(
4       documents=<korpus_dokumenata>,
5       embedding=<naziv_modela_vektorskih_ugradnji>,
6       builtin_function=BM25BuiltInFunction(
7           input_field_names="<naziv_ulaznog_polja>",
8           output_field_names="<naziv_izlaznog_polja>"
9       ),
10      vector_field=["<naziv_dense_polja>", "<naziv_sparse_polja>"],
11      connection_args={"uri": "<uri>"}
12  )
13
14  results = vectorstore.similarity_search("<tekst_upita>", k=<k>)
```

Grupisanje pretrage (engl. *Grouping Search*)

- Rezultati su grupisani prema specifičnom polju kako bi se osigurala raznolikost
- Izbegava vraćanje više rezultata iz iste grupe
- Vraća ciljani broj entiteta unutar svake grupe
- Glavni tok:
  - Izvršava *ANN* pretragu sa definisanim poljem za grupisanje
  - Grupiše rezultate prema tom specifičnom skalarnom polju
  - Vraća određeni broj entiteta po grupi
  - Opciono, nameće strogo ograničenje veličine grupe

Grupisanje pretrage (engl. *Grouping Search*)

- Sintaksa:

```
res = client.search(
    collection_name="<naziv_kolekcije>",
    data=[[<vektor_upita>]],
    limit=<vrednost_limita>,
    group_by_field="<naziv_polja>",
    group_size=<veličina_grupe>,
    strict_group_size=<True/False>,
    search_params={"metric_type": "<naziv_tipa_metrike>", "params": {}}
)
```

Filtrirana pretraga (engl. *Filtered Search*)

- Filtrirana pretraga sprovodi filtriranje meta-podataka pre izvršavanja *ANN* pretrage
- Smanjuje opseg pretrage sa cele kolekcije na entitete koji ispunjavaju uslove filtriranja
- Podržava složene logičke izraze za filtriranje
- Glavni tok:
  - Prvo primenjuje uslov skalarnog filtriranja
  - Sprovodi *ANN* pretragu samo nad filtriranim entitetima
  - Vraća rezultate koji zadovoljavaju i filter i vektorsku sličnost

Filtrirana pretraga (engl. *Filtered Search*)

- Sintaksa:

```
res = client.search(
    collection_name="<naziv_kolekcije>",
    data=[[<vektor_upita>]],
    filter='<logički_izraz>',
    limit=<vrednost_limita>,
    output_fields=["<naziv_polja>"]
)
```

Pretraga po ključnim rečima (engl. *Text Match – Keyword Match*)

- Omogućava precizno pronalaženje eniteta na osnovu specifičnih termina
- Primarno se koristi za filtriranu pretragu radi ispunjavanja konkretnih uslova
- Može uključiti skalarno filtriranje
- Glavni tok:
  - Prihvatanje tekstualnog upita sa specifičnim ključnim rečima ili frazama
  - Pronalaženje identičnih termina u tekstualnim poljima
  - Mogućnost kombinovanja sa vektorskom pretragom
  - Vraćanje entiteta koji se podudaraju sa zadatim kriterijumima ključnih reči
- Sintaksa:

```
res = client.search(
    collection_name="<naziv_kolekcije>",
    data=[[<vektor_upita>]],
    filter='TEXT_MATCH(<naziv_polja>, "<ključna_reč>")',
    limit=<vrednost_limita>
)
```

Podudaranje fraza (engl. *Phrase Match*)

- Vraća rezultate isključivo kada se tačan redosled reči iz upita pojavljuje uzastopno i u ispravnom poretku unutar entiteta
  - Opciono, postoji podesivi parametar koji dozvoljava određeni broj reči između traženih termina, čime se omogućava približno podudaranje
- Glavni tok:
  - Aktivacija tekstualne analize na polju tipa `VARCHAR` postavljanjem `enable_analyzer=True` i `enable_match=True`
  - Opciono konfigurisanje analizatora za tokenizaciju teksta
  - Korišćenje izraza `TEXT_MATCH` za definisanje polja i fraze koja se pretražuje

```
1  results = client.search(
2      collection_name="<naziv_kolekcije>",
3      anns_field="<vektorizovano_polje>",
4      data=[<vektor_upita>],
5      filter=filter,
6      limit=<top_k>,
       output_fields=["<naziv_polja_1>", "<naziv_polja_2>"]
   )
```

Sadržaj

Zadaci za vežbu – pokretanje projekta

- Pozicionirati se u folder projekta.

```
cd Desktop/NAIS-projekat-2026
```

- Pokrenuti sve potrebne servise.

```
docker compose up -d etcd minio standalone attu vector-database-service ollama streamlit
```

- Pratiti logove tokom upisa podataka (potrebno je sačekati da se svi podaci upišu, pre testiranja upita).

```
docker logs -f nais-projekat-2026-vector-database-service-1
```

- Proveriti da li su kolekcije kreirane i napunjene podacima.

```
curl http://localhost:8000/lab/v1/fashion/stats
curl http://localhost:8000/api/v1/sciq/stats
# očekivani odgovor: {"collection": "...", "row_count": 1000}
#                    {"collection": "...", "row_count": ~500}
```

Zadaci za vežbu – pokretanje projekta

- Otvoriti interfejse u pretraživaču:
  - Testiranje zadataka: http://localhost:8000/docs
  - Attu: http://localhost:8090
  - Streamlit: http://localhost:8501

Zadaci za vežbu – Podaci

- **Fashion Product Images Small**
  - Skup podataka sa slikama i meta-podacima modnih proizvoda
  - Podaci sadrže 44 441 entitet i 10 obeležja za svaki proizvod (identifikator, pol, kategorija, podkategorija, tip artikla, boja, sezona, godina, namena i naziv proizvoda)
  - Slike su rezolucije 60x80 piksela, u RGB formatu
  - Izvor skupa podataka
    - Matična Internet strana: https://huggingface.co/datasets/ashraq/fashion-product-images-small
      - Podaci su dostupni u više formata
        - Može se koristiti kroz *Hugging Face datasets* biblioteku ili kao *CSV* tabela sa folderom slika
    - Ova verzija je optimizovana za brže treniranje modela i zauzima manje prostora
    - Vlasnik i autor originalnog skupa je Param Aggarwal
    - Licenca i mogućnosti korišćenja skupa podataka
      - CC0: Public Domain
        - podaci su javno dostupni za istraživački rad i komercijalnu upotrebu

Zadaci za vežbu – Podaci

- **SciQ – Skup podataka za naučna pitanja**
  - Skup podataka sa pitanjima iz oblasti prirodnih nauka namenjen za mašinsko učenje i QA sisteme
  - Podaci sadrže 13 679 pitanja sa višestrukim izborom odgovora, pri čemu su pitanja dobijena putem *crowdsourcing*-a
  - Struktura svakog pitanja obuhvata: pitanje, tri distraktora (netačna, ali smislena odgovora) i podržavajući teskt (dokaz ili kontekst iz kog je izvučen odgovor)
  - Pitanja su generisali ljudi, a zatim su verifikovana
  - Izvor skupa podataka
    - Matična Internet strana: https://huggingface.co/datasets/allenai/sciq
      - Podaci su podeljeni u tri skupa: trening, validacioni i test skup
      - Podaci su dostupni u više formata
        - Može se koristiti kroz *Hugging Face datasets* biblioteku ili kao *CSV* tabela sa folderom slika
    - Vlasnik i autor originalnog skupa je Allen Institute for AI (AI2)
    - Licenca i mogućnosti korišćenja skupa podataka
      - CC BY-NC 4.0 (Attribution-NonCommercial 4.0 International)
        - Podaci su javno dostupni za istraživački rad, ali uz ograničenja za direktnu komercijalnu upotrebu i obavezno navođenje izvora.

Sadržaj

Korisni linkovi

- Milvus Documentation: https://milvus.io/docs
- PyMilvus SDK (Python Client): https://milvus.io/api-reference/pymilvus/v2.6.x/About.md
- Ollama API Guide: https://docs.ollama.com/api/introduction
- Sentence-Transformers (SBERT): https://sbert.net/docs/sentence_transformer/usage/usage.html
- FastAPI Framework: https://fastapi.tiangolo.com/
- PyTorch Library: https://docs.pytorch.org/docs/stable/index.html
- Docker Compose: https://docs.docker.com/compose/
- Python Eureka Client: https://pypi.org/project/py-eureka-client/