



UNIVERZITET U NIŠU  
ELEKTRONSKI FAKULTET



## **Primena vektorskih baza podataka kod pretrage slika po sličnosti i semantici**

Diplomski rad  
Studijski program: Elektrotehnika i računarstvo  
Modul: Računarstvo i informatika

Student:

Slavko Petrović, br. ind. 17345

Mentor:

Prof. dr Leonid Stoimenov

Niš, maj 2024. godina

Univerzitet u Nišu  
Elektronski Fakultet

**Primena vektorskih baza podataka kod pretrage slika po sličnosti i semantici**

**Application of vector databases in image search by similarity and semantics**

Diplomski rad  
Studijski program: Elektrotehnika i računarstvo  
Modul: Računarstvo i informatika

Student: Slavko Petrović, br. ind. 17345

Mentor: Prof. dr Leonid Stoimenov

*Zadatak: Proučiti osnovne koncepte, način rada i primenu vektorskih baza podataka. Uporediti ih sa postojećim relacionim i nerelacionim bazama podataka. U praktičnom delu rada realizovati aplikaciju koja demonstrira pretragu slika po sličnosti i semantici korišćenjem Weaviate vektorske baze podataka i CLIP modela za vektorizaciju podataka.*

Datum prijave rada: 28.02.2024

Datum predaje rada:

Datum odbrane rada:

Komisija za ocenu i odbranu:

---

1. Predsednik Komisije

---

2. Član

---

3. Član

# Sadržaj

<b>1. UVOD .....</b>	<b>4</b>
<b>2. PODACI .....</b>	<b>5</b>
2.1. Struktuirani podaci .....	5
2.2. Nestruktuirani podaci .....	6
2.3. Polustruktuirani podaci .....	7
<b>3. VEKTORSKE BAZE PODATAKA.....</b>	<b>8</b>
3.1. Način rada .....	8
3.2. Vektorske ugradnje .....	9
3.3. Indeksiranje .....	12
3.4. Metrike sličnosti .....	15
3.5. Filtriranje .....	17
3.5.1. Filtriranje posle upita .....	17
3.5.2. Filtriranje zajedno sa upitom .....	18
3.5.3. Filtriranje pre upita .....	18
<b>4. PRIMENA I OSOBINE VEKTORSKIH BAZA PODATAKA .....</b>	<b>20</b>
4.1. Najpoznatije vektorske baze podataka .....	22
4.1.1. Pinecone baza podataka .....	23
4.1.2. Milvus baza podataka .....	23
4.1.3. Chroma baza podataka .....	24
4.1.4. Weaviate baza podataka.....	24
4.2. Poređenje sa relacionim bazama podataka .....	25
4.3. Poređenje sa nerelacionim bazama podataka .....	25
<b>5. PRETRAŽIVANJE SLIKA PO SLIČNOSTI I SEMANTICI .....</b>	<b>27</b>
5.1. Instalacija Weaviate baze podataka, Python i Node paketa .....	27
5.2. Popunjavanje Weaviate baze podacima.....	28
5.3. Pretraživanje vektora.....	31
<b>6. ZAKLJUČAK.....</b>	<b>35</b>
<b>7. LITERATURA .....</b>	<b>36</b>

# 1. UVOD

Kako nas digitalno doba pokreće u eru u kojoj dominiraju veštačka inteligencija i mašinsko učenje, vektorske baze podataka su se pojavile kao nezamenljiv alat za skladištenje, pretraživanje i analizu vektora podataka. Razumevanje ovog tipa baze podataka je od velikog značaja za sve koje rade u toj oblasti.

Uzbuđenje oko vektorskih baza podataka usko je povezano izdavanjem ChatGPT-a. Od kraja 2022. godine, javnost je počela da razume mogućnost najsavremenijih modela velikih jezika (engl. Large Language Models), dok su programeri shvatili da vektorske baze podataka mogu dodatno unaprediti ove modele.

Sve više i više kompanija se odlučuju da u svoje postojeće sisteme uvedu vektorske baze podataka ili da dodaju mogućnost vektorskog pretraživanja u svojim SQL ili NoSQL bazama. Razlog toga je što se većina podataka na internetu nalaze u obliku nestruktuiranih podataka poput teksta, slike i video snimaka. To omogućava kompanijama da dobiju dragocene informacije iz velikog obima podataka i donose bolje poslovne odluke. Ova transformacija u načinu skladištenja i obrade podataka predstavlja odgovor na zahtev tržišta za agilnijim i efikasnijim rešenjima u doba digitalne ekspanzije.

Takođe, efikasna obrada podataka postaje od ključnog značaja za aplikacije koje koriste velike jezičke modele, generativnu veštačku inteligenciju i semantičku pretragu. Ove aplikacije oslanjaju se na vektorske reprezentacije podataka, koje sadrže semantičke informacije neophodne za razumevanje i održavanje dugoročne memorije, ključne za izvršavanje složenih zadataka veštačke inteligencije. Stoga je neophodno imati specijalizovanu bazu podataka koja efikasno rukuje ovim tipom podataka. Mašinsko učenje kroz razne modele omogućuje pretvaranje kompleksnih nestruktuiranih i polustruktuiranih podataka u numeričke reprezentacije zvane vektorske ugradnje.

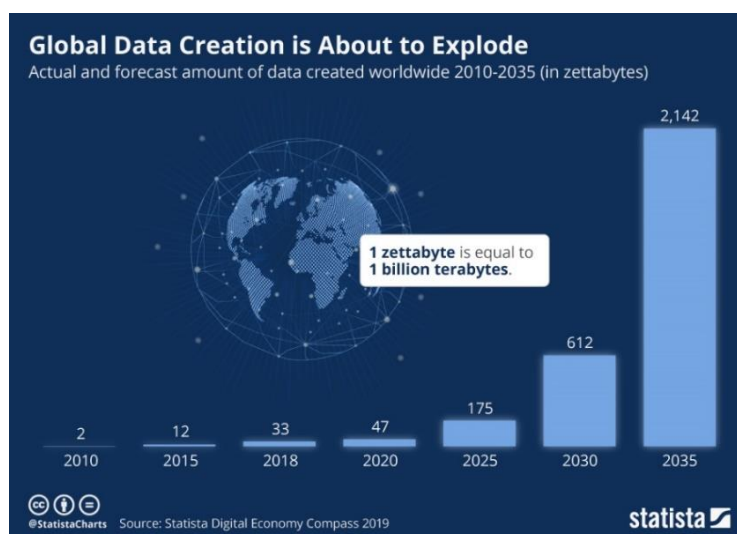
Rad sa vektorskim podacima predstavlja izazov jer tradicionalne baze podataka zasnovane na skalarima nisu adekvatne za praćenje kompleksnosti i obima takvih podataka. Ovo otežava izvlačenje relevantnih uvida i izvođenje analiza u realnom vremenu. Vektorske baze podataka su posebno dizajnirane kako bi se nosile sa ovom vrstom podataka, pružajući bolje performanse, skalabilnost i fleksibilnost potrebne za maksimalno iskorišćavanje podataka. Glavna funkcionalnost koju pružaju vektorske baze podataka su jednostavno pretraživanje po sličnosti i semantici.

U teorijskom delu ovog diplomskog rada objašnjeni su sledeći koncepti: vrste digitalnih podataka, način rada vektorskih baza podataka, transformacija podataka u vektore, indeksiranje, filtriranje, metrika sličnosti, poređenje sa već postojećim bazama podataka kao i samu primenu vektorskih baza. Kako bi se na praktičnom primeru bolje objasnio sam proces vektorskih baza podataka napravljena je aplikacija koja demonstrira pretragu slika po sličnosti i semantici. Aplikacija je implementirana pomoću Weaviate vektorske baze podataka i CLIP modela za vektorizaciju podataka. Na kraju rada dat je zaključak kao i spisak korišćene literature.

## 2. PODACI

Podaci su danas značajniji nego ikada pre. Nepravilno skladištenje poslovnih podataka može dovesti do kritičnih neuspeha u svemu, od donošenja odluka preko gubitka klijenata do alokacije resursa. Iako se često koriste kao sinonimi, podatke i informacije treba jasno razgraničiti. Podaci predstavljaju neobrađene činjenice ili statističke podatke, koji sami po sebi nemaju inherentnu vrednost. Mogu se pojaviti u različitim oblicima, kao što su tekst, slike, video zapisi, grafikoni ili simboli. Tek kroz njihovu interpretaciju i analizu podaci se pretvaraju u informacije, znanje koje ima praktičnu primenu.

Prema IBM-ovoj studiji, predviđa se da će globalni obim podataka dostići 175 zetabajta u 2025. godini a da će nakon toga ono početi eksponencijalno rasti. Da bi bolje razumeli koja je to veličina možemo zamisliti ovo: 35ZB predstavlja veličinu od jedan trilion sati video filma, a kad bi pogledali sve to bilo bi nam potrebno 115 miliona godina. Ovaj masivni i kontinuirano rastući volumen podataka jasno ilustrira potrebu za efikasnim upravljanjem i strategijama za njihovo korišćenje [4]. Na slici 1. dat je grafički prikaz ekspanzije podataka.



Slika 1. Trenutna i predviđena količina podataka koja je kreirana širom sveta između 2010-2035 izražena u zetabajtima [1]

Digitalni podaci se mogu klasifikovati u tri oblika:

- Struktuirani podaci
- Nestruktuirani podaci
- Polustruktuirani podaci

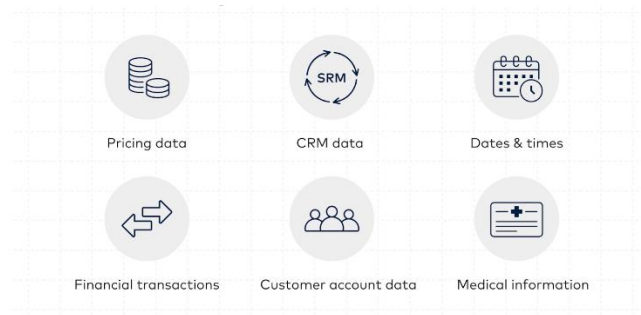
U narednim odeljcima detaljnije ćemo se osvrnuti na svaku od ovih vrsta podataka. Cilj je da analiziramo njihove karakteristike, skladištenje i potencijalne primene.

### 2.1.Struktuirani podaci

Strukturni podaci se obično sastoje iz alfanumeričkih znakova prevedenih iz izvornog formata u unapred definisani format pre nego što se unesu u određeni model podataka. Ovaj model podataka skladišti struktuirane podatke u ćelijama, kolonama i redovima. Jezik struktuiranih upita (SQL) je programski jezik koji se koristi za upravljanjem struktuiranim podacima. Podaci koji su skladišteni u relacijskim bazama podataka je savršeni primer

struktuiranih podataka.

Za struktuirane podatke često se kaže da su to kvantitativni podaci, što znači da imaju prebrojiv broj elemenata. Uobičajeno, lakše je organizovati, pretraživati, i analizirati struktuirane podatke. Na slici 2. prikazani su najčešći tipovi struktuiranih podataka.



Slika 2. Primeri struktuiranih podataka [3]

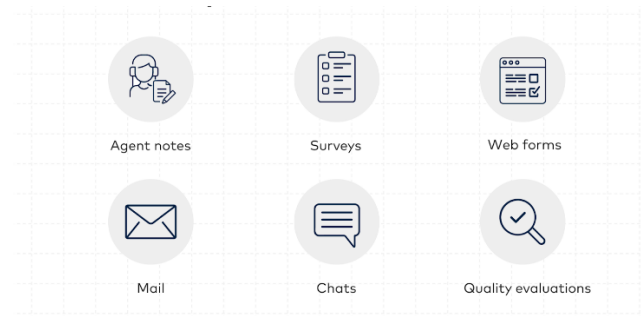
## 2.2. Nestruktuirani podaci

Nestruktuirani podaci kao što samo ime govori su podaci koji nemaju poznatu formu. To su podaci koji nemaju prepoznatljivu strukturu ili nisu u obliku koji može lako da koristi kompjuterski program. Oko 80-90% podataka kompanija je u ovom formatu; na primer, beleške, sobe za časkanje, PowerPoint prezentacije, slike, video zapisi, telo e-pošte, itd.

Jedna od ključnih prednosti nestruktuiranih podataka je ta što oni pomažu u pružanju kvalitativnih informacija korisnih kompanijama za razumevanje trendova i promena. Primarni nedostatak rada sa ovim tipom podataka je dodatna složenost koje iziskuje specijalizovane veštine, alate i razumevanje za analizu i korišćenje informacija.

Za skladištenje nestruktuiranih podataka koristimo nerelacione baze kao što su: MongoDB, Cassandra, Redis, Neo4J. Ako postoji potreba da se veliki podaci čuvaju u sirovim, izvornim formatima kao skladište se koristi jezero podataka(engl. Data lake). [4]

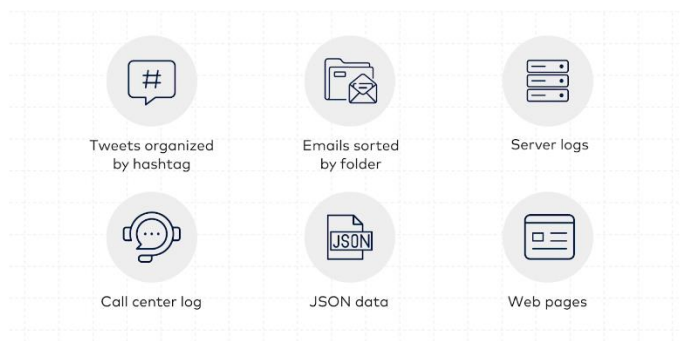
Ponekad, numerički i tekstualni podaci mogu biti nestruktuirani jer je modeliranje u obliku tabele neefikasno. Na primer podaci koje očitava senzor predstavljaju konstantan tok numeričkih vrednosti ali bi kreiranje tabele sa dve kolone – vremenskom oznakom i očitanom vrednošću bilo i neefikasno i nepraktično. Na slici 3. prikazani su tipovi podataka koji se klasifikuju kao nestruktuirani.



Slika 3. Primeri nestruktuiranih podataka [3]

## 2.3. Polustrukturirani podaci

Polustrukturirani podaci imaju osobine i nestruktuiranih i strukturiranih podataka. Nemaju predefinisani model podataka i kompleksniji su od strukturiranih podataka, ali su lakši za skladištenje od nestruktuiranih podataka. Polustrukturirani podaci koriste ‘metapodatke’ da identifikuju specifične karakteristike i skaliraju podatke u zapise. Metapodaci omogućavaju da se polustrukturirani podaci bolje klasifikuju i pretražuju u odnosu na nestruktuirane podatke. Najpoznatiji formati ovog tipa podataka su JSON, CSV i XML. Na slici 4. možemo videti tipove polustrukturiranih podataka.



Slika 4. Primeri polustrukturiranih podataka [3]

### 3. VEKTORSKE BAZE PODATAKA

U oblasti veštačke inteligencije, ogromne količine podataka zahtevaju efikasno rukovanje i procesuiranje. Kako ulazimo u naprednije aplikacije veštačke inteligencije, kao što su prepoznavanje slika, glasovna pretraga ili mašina za preporuku, priroda podataka postaje sve složenija. Tu vektorske baze podataka dolaze do izražaja. Za razliku od tradicionalnih baza podataka koje čuvaju skalarne vrednosti, vektorske baze podataka su jedinstveno dizajnirane da rukuju višedimenzionalnim tačkama podataka, koje se često nazivaju vektori.

One predstavljaju tip baze podataka koja indeksira i skladišti podatke kao visokodimenzione vektore, koje predstavljaju matematičku reprezentaciju karakteristika ili atributa, takođe poznati kao vektori ugradnje (engl. Vector embeddings). Svaki vektor ima određeni broj dimenzija, koje mogu da se kreću od desetina do hiljada, u zavisnosti od složenosti i granularnosti podataka.

Vektorske baze podataka nude niz funkcija, uključujući i CRUD operacije, filtriranje metapodataka i horizontalno skaliranje. Ove funkcionalnosti omogućavaju efikasno upravljanje podacima i pruža lakše rukovanje velikim količinama vektorskih podataka. [10]

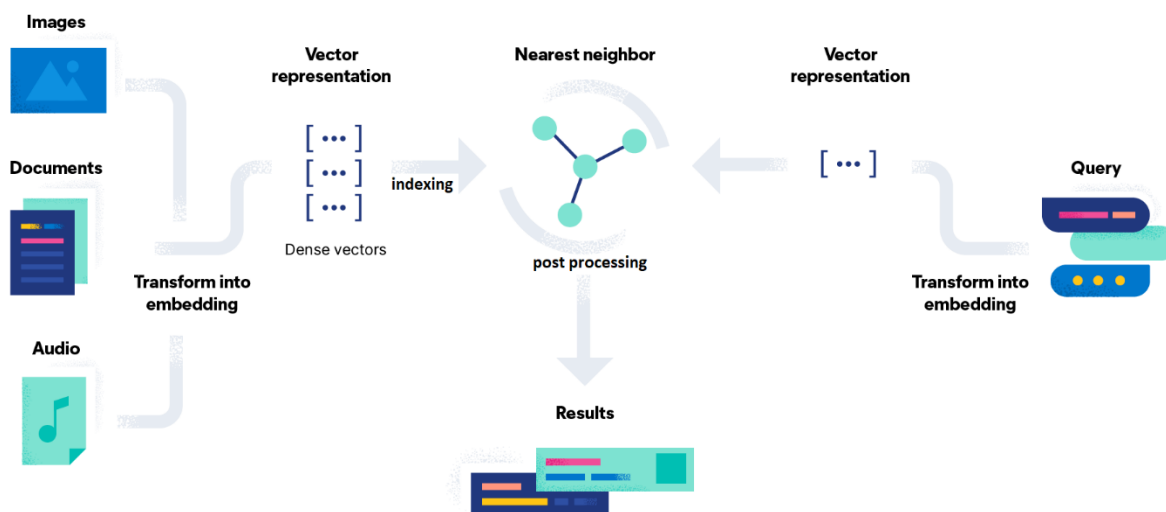
Njihova primarna prednost je sposobnost da brzo i precizno locira i preuzima podatke prema njihovoj vektorskoj blizini ili sličnosti. Ovo omogućava pretrage ukorenjene u semantičkoj ili kontekstualnoj relevantnosti umesto da se oslanjaju samo na tačna podudaranja kriterijuma kao kod konvencionalnih baza podataka. Vektorske baze podataka ne samo da čuvaju vektore ugradnje već i omogućavaju skladištenje i filtriranje metapodataka. To znači da pored upita zasnovanog na sličnosti vektora, takođe možemo da filtriramo i tražimo vektore na osnovu povezanih metapodataka, čineći proces pretraživanja dinamičnijim i fleksibilnijim. Još jedna prednost vektorskih baza podataka je njihova skalabilnost. Kako obim podatak vremenom sve više i više raste, rastu i zahtevi korisnika, a vektorske baze omogućavaju horizontalno skaliranje kako bi zadovoljila te potrebe.

Vektorske baze podataka su takođe dizajnirane da podržavaju ažuriranja u realnom vremenu, što omogućava dinamičku promenu i ažuriranje podataka bez ugrožavanja performansi. Ovo je posebno važno u scenarijama u kojim se podaci stalno menjaju, kao što su analitika u realnom vremenu, IoT aplikacije ili senzorsko učitavanje podataka.

#### 3.1. Način rada

U prethodnom delu rada, istaknuta je ključna funkcija i prednost vektorskih baza podataka, a to je njihova sposobnost izvođenja pretraga zasnovanih na sličnosti. To joj omogućuje reprezentacija podataka u obliku vektora koji se dobija transformacijom nestruktuiranih podataka pomoću neuronskih mreža. Da bi se pretraga izvršavala brzo vektorske baze podataka koriste kombinaciju različitih algoritama koji svi učestvuju u pretraživanju vektorskog prostora. Algoritam koji daje najtačnije rezultate naziva se kNN (engl. k-Nearest Neighbor), odnosno k-najbližih suseda, međutim on nije najkorišćeniji zato što poredi vektor upita sa svim ostalim vektorima i vraća stvarnih k najbližih suseda. U realnim sistemima to nije efikasno, baza mora imati brzu pretragu i davati što tačnije rezultate. To se postiže optimizacijom kNN algoritma korišćenjem indeksiranja kao i ANN algoritma (engl. Approximate Nearest Neighbor), odnosno algoritma približno najbližih suseda.





Slika 5. Tok rada vektorskih baza podataka[11]

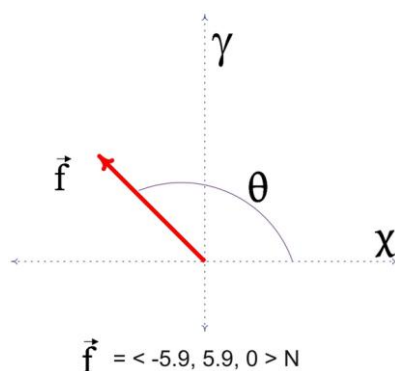
Na slici 5. prikazan je uobičajeni tok rada vektorskih baza podataka, koji možemo izdvojiti u pet celina:

1. Transformacija – pretvaranje nestruktuiranih tipova podataka u njihovu vektorsku reprezentaciju uz pomoć neuronskih mreža ili alata koji je koriste (Word2Vec, RasNet50, CNN...)
2. Indeksiranje – korišćenjem heširanja (LSH), kvantizacije (IVFPQ) ili tehnika zasnovanih na grafu (HNSW), vektorska baza podataka indeksira vektore tako što ih mapira u datu strukturu podataka. Ovo omogućava bržu pretragu.
3. Transformacija upita u vektorske ugradnje, korišćenje istih modela kao kod prve stavke.
4. Pretraživanje – uz pomoć indeksa i ANN algoritma, metrika sličnosti i vektora upita, vrši se pretraživanje vektorskog prostora.
5. Obrada rezultata – poslednji korak pri generisanju rezultata. Korišćenjem drugačijih metrika sličnosti dobijeni rezultati se re-sortiraju. Uz pomoć filtriranja i metapodataka odbacuju se nepoželjna rešenja.

U narednim odlomcima ovog diplomskog rada detaljno će biti objašnjena svaka od ovih pet celina.

### 3.2. Vektorske ugradnje

Da bi razumeli vektorske baze podataka prvo moramo razumeti sta je vektor. U matematici i fizici, vektor je veličina koja ima intezitet, pravac i smer. On se može podeliti na komponente, na primer, u dvodimenzijalnom prostoru, vektor ima X (horizontalnu) i Y (vertikalnu) komponentu. Kada se govori o mašinskom učenju i nauci o podacima (engl. Data science) vektor je uređena lista ili niz brojeva koji mogu predstavljati bilo koju vrstu podataka, uključujući i nestruktuirane i polustruktuirane podatke. Na slici 6. prikazan je vektor  $\vec{f}$  koji se nalazi u Dekartovom koordinatnom sistemu i zaklapa ugao  $\theta$  sa X osom.[16]



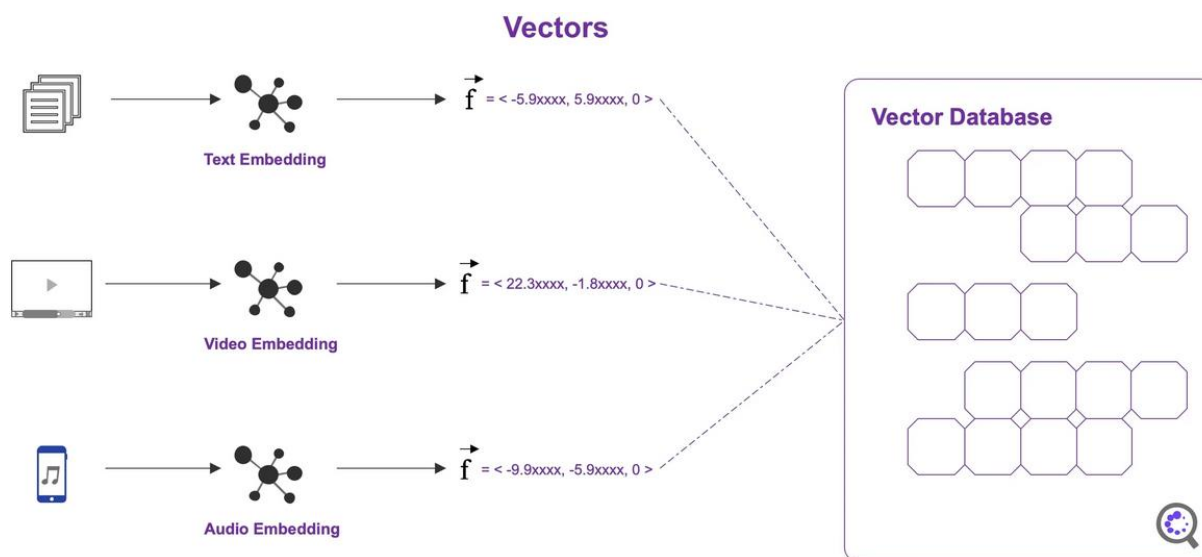
Slika 6. Vektorska reprezentacija[16]

Google-ova definicija vektorskih ugradnji kaže:

*„Vektorske ugradnje su način predstavljanja podataka kao tačke u n-dimenzionalnom prostoru tako da se slični podaci grupišu zajedno”.*[30]

One predstavljaju numerički prikaz podataka koji obuhvata semantičke odnose i sličnosti, što omogućava izvođenje matematičkih operacija i poređenje podataka.

Svaki vektor u vektorskoj bazi podataka odgovara objektu ili stavci, bilo da je reč o slici, videu zapisu, dokumentu ili bilo kom drugom podatku. Ovi vektori su uglavnom dugački i složeni, izražavajući lokaciju svakog objekta duž desetina ili čak hiljada dimenzija. Na primer, vektorska baza podataka filmova može da locira filmove po dimenzijama kao što su vreme trajanja, žanr, godina izdanja, broj zajedničkih glumaca itd. Ako su ovi vektori kreirani ispravno, onda će slični filmovi verovatno završiti zajedno i blizu jedni drugih u vektorskoj bazi podataka.

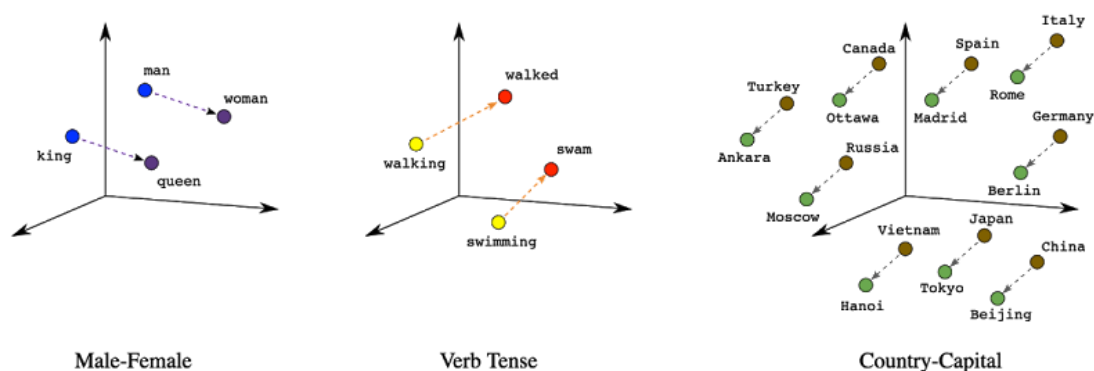


Slika 7. Transformacija nestruktuiranih podataka u vektorske ugradnje pomoću odgovarajućeg modela[16]

Postoje više tipova vektorskih ugradnji koje se koriste u različitim aplikacijama, i najkorišćeniji su (Slika 7.):

- Ugradnje reči (engl. Word Embeddings) kao što sam naziv implicira predstavlja pretvaranje reči u vektore. Tehnike poput Word2Vec, GloVe uče ugrađivanje reči hvatanjem semantičkih odnosa i kontekstualnih informacija iz velikih korpusa teksta.
- Ugradnje slika (engl. Image Embeddings) predstavlja slike kao vektore uzimajući u obzir razne vizualne karakteristike. Tehnike kao što su konvolucione neuronske mreže (engl. Convolutional Neural Networks) i unapred obučeni modeli kao što su ResNet50 i VGG generišu vektore za zadatke kao što su klasifikacija slika, detekcija objekta i sličnost slika.
- Ugradnje rečenica (engl. Sentence embeddings) predstavlja cele rečenice kao vektore. Modeli poput USE (engl. Universal Sentence Encoder) i SkipThought generišu ugradnje koje obuhvataju i kontekst i značenje rečenica.
- Ugradnje dokumenata (engl. Document embeddings) sa svojim modelima kao što su Doc2Vec i ParagraphVectors omogućavaju da transformišu dokumenta u vektore gde će se grupisati slični.

Word2Vec je tehnika koju je kompanija Google implementirala 2013. godine za ugradnju reči. Kao ulaz uzima reč a kao izlaz dobija se n-dimenzionalni vektor tako da kada se unese u vektorski prostor sinonimi se grupišu. Na slici 8. data je pojednostavljena vizuelna reprezentacija ugradnje reči.



Slika 8. Vizuelna reprezentacija reči kao vektora gde se sinonimi grupišu[6]

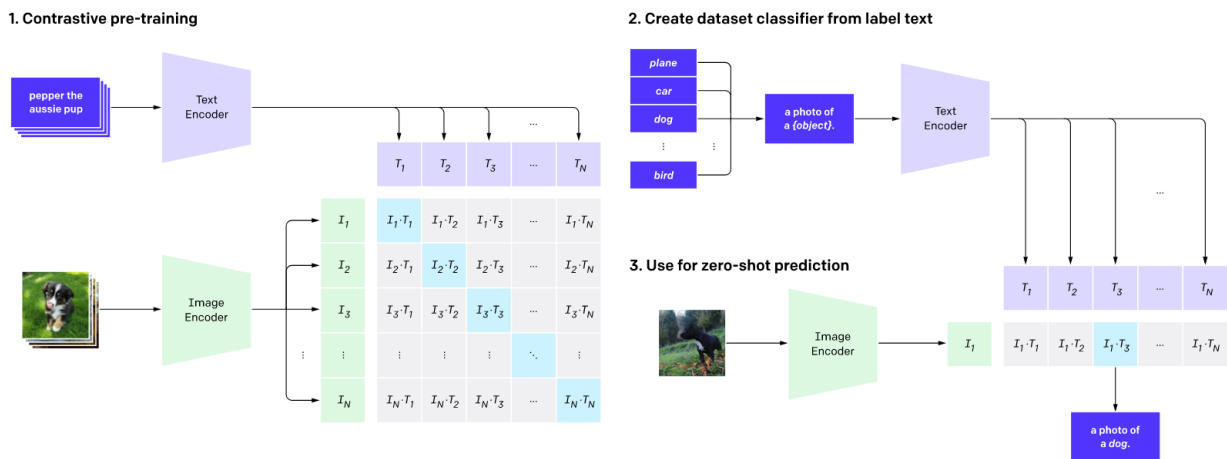
Sa ove slike možemo videti da se slične reči grupišu u blizini, da se vektori kralj i kraljica nalaze u neposrednoj blizini. Isto tako, svi sinonimi bi se grupisali, poput hodao, šetao, koračao. Takođe, možemo primetiti da kada bi primenili vektorske operacije na ključne reči imali bismo predstavu šta možemo dobiti. Najpoznatiji primer je relacija između reči kralj i kraljica koja je data formulom  $kralj - muškarac + žena = kraljica$ . Da ovo izrazimo rečima, zbir vektora kralja i žene trebalo bi da bude otprilike isti kao za vektor kraljicu koja je naravno na neki način „ženski kralj”. Ovo je prilično zgodna karakteristika vektora reči, iako se ova osobina ne prevodi uvek na koristan način za ugrađivanje složenijih tipova podataka, poput slika i dužih delova teksta. [6]

CLIP (engl. Contrastive Language-Image Pre-training) je model dubokog učenja kojeg je razvila kompanija OpenAI 2021. godine. On omogućava da vektorske ugradnje slika i teksta dele isti prostor, dozvoljavajući direktna poređenja između ova dva modaliteta. To se postiže obučavanjem modela da približi povezane slike i tekstove, dok neposredne odvaja. Tako da, na primer, fotografija ptice i rečenica „slika ptice“ će imati veoma slične vektorske ugradnje i bile

bi blizu jedna drugoj u vektorskom prostoru.

Ovaj model postiže izuzetne „zero-shot“ performanse na različitim testiranjima kao što je ImageNet koji predstavlja skup podataka za klasifikaciju slika. „Zero-shot“ učenje se odnosi na činjenicu da model nije eksplicitno obučen ni na jednom od 1.28 miliona primera obuke u ImageNet skupu podataka. Ipak, CLIP model je parirao tačnosti originalnog ResNet-50 modela koji je obučen nad tim podacima.

CLIP prvo obučava enkoder slike i enkoder teksta da predvide koji tekstovi odgovaraju kojim slikama u skupu podataka. Zatim koristi tu sposobnost da transformiše CLIP u nulti klasifikator. Sve klase u skupu podataka se zatim pretvaraju u oznake poput „fotografija {objekta}“. Ovim postupkom dobijamo hiljade različitih vektorskih reprezentacija koje odgovaraju svim mogućim klasama. Na kraju, izračunava se skalarni proizvod između vektorskih ugradnji slika i vektorskih ugradnji teksta. Budući da je CLIP obučen na način da slike i tekst budu u istom vektorskom prostoru, a skalarni proizvod procenjuje sličnosti između ugradnji, verovatno je da će najveći skalarni proizvod biti sa „fotografijom {objekta}“. Ovaj proces je grafički prikazan na slici 9.[30]



Slika 9. Princip rada CLIP modela [30]

### 3.3. Indeksiranje

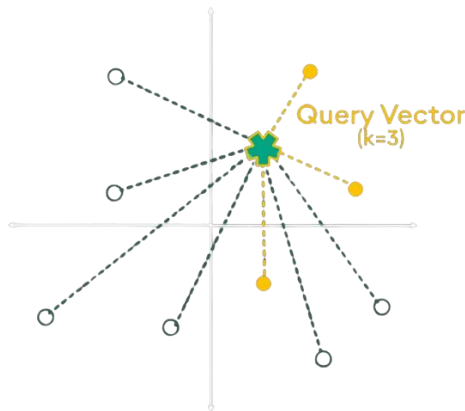
Čuvanje podataka u obliku vektora nije optimalno. Da bi se ubrzao proces pretraživanja, neophodno je indeksiranje vektora. Vektorsko indeksiranje predstavlja ključni koncept u vektorskim bazama podataka, jer omogućava značajno povećanje brzine procesa pretraživanja po sličnosti, uz minimalan gubitak preciznosti. Indeksiranje mapira vektore u nove strukture podataka, što omogućava efikasniju pretragu po sličnosti ili udaljenosti, kao što je pretraga najbližih suseda između vektora.

Najkorišćeniji tipovi indeksa su:

- Flat
- LSH (engl. Locality Sensitive Hashing)
- IVF (engl. InVerted File)
- HNSW (engl. Hierarchical Navigable Small Worlds)

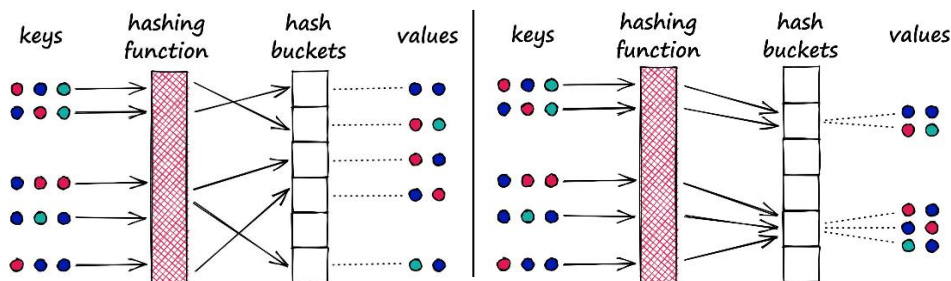
Flat indeksi, poznatiji kao indeksi grube sile (engl. Brute Force) daju najbolju tačnost od svih metoda indeksiranja, međutim oni su dosta spori. To je zato što oni predstavljaju direktan prikaz vektorskih ugradnji. Sa Flat indeksima pretraga je iscrpna: vrši se izračunavanje sličnosti između vektora upita i svakog drugog vektora ugradnji, zatim se k broj vektorskih ugradnji vraća korišćenjem kNN(k-nearest neighbors) algoritma[17]. Na slici 10. žutom bojom je označeno koje „vektorske tačke“ se vraćaju kao rezultat, a crnom isprekidanom linijom dat

je simboličan prikaz da Flat indeks uzima sve „vektorske tačke“ u obzir.



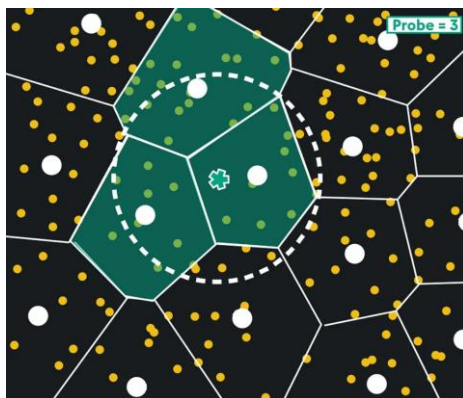
Slika 10. Grafički prikaz rada Flat indeksa i kNN algoritma[17]

LSH je strategija indeksiranja koja optimizuje brzinu i pronalaženje najbližeg suseda, umesto da se vrši iscrpna pretraga kao kod Flat indeksiranja. Indeks je izgrađen pomoću funkcije heširanja. Vektorske ugradnje koje se nalaze u blizini, tj. koji su slični, heširaju se u istu „kantu“ (Slika 11). Kada se prosledi upit, njegovi najbliži susedi se mogu pronaći heširanjem upita, a zatim izračunavanjem metrike sličnosti za sve ostale vektore koji se nalaze u istoj „kanti“.[16]



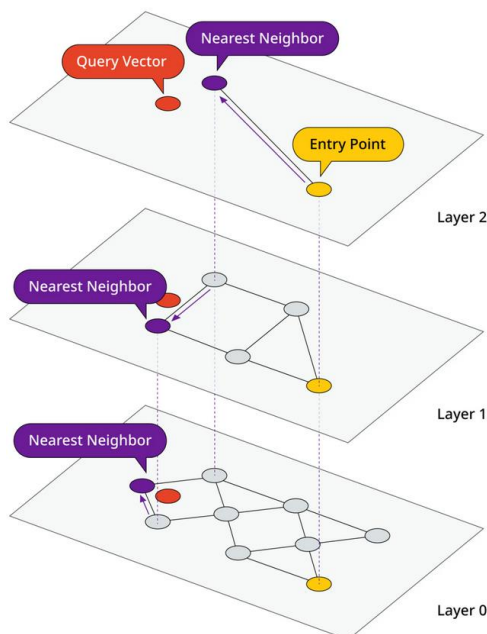
Slika 11. Klasično heširanje (levo) i LSH heširanje (desno)[18]

IVF indeksi su slični LSH indeksima po tome što je cilj da se prvo mapira vektor upita na manji podskup vektorskog prostora a zatim se samo pretražuje taj manji prostor za približno najbliže susede. U IVF-u, vektorski prostor se prvo deli ili grupiše, a zatim se pronalaze težišta svakog klastera. Onda za vektor upita nalazimo najbliže težište. Zatim za taj centar tražimo sve vektore u povezanom klasteru. Mana ovog indeksiranja je mogućnost da se vektor upita nalazi blizu ivice klastera. U ovom slučaju najbliži vektori mogu biti u susednom klasteru, te generalno moramo da pretražimo više klastera[16]. Na slici ispod, prikazani su Voronoi ćelije, žutom bojom su prikazani vektori ugradnje, klasteri razdvojeni belom linijom a centar klastera označen belom tačkom (Slika 12).



Slika 12. Grafički prikaz IVF indeksiranja[17]

HSNW spada u podgrupu grafovskih indeksa. Oni koriste čvorove (engl. Nodes) i ivice (engl. Edges) da bi konstruisali strukturu nalik mreži. Čvorovi predstavljaju vektorske ugradnje, dok ivice predstavljaju odnose između tih vektora. Dva čvora su povezana na osnovu njihove blizine, često definisane Euklidskom distancom (engl. Euclidean Distance). Svaki čvor u grafu se povezuje sa drugim čvorom koji se nazivaju “prijatelji” i svaki čvor čuva listu prijatelja. Počevši od unapred definisane početne tačke (engl. Entry point), algoritam prolazi kroz povezane čvorove prijatelja dok ne pronađe najbližeg suseda vektoru upita. Svaka pretraga se nastavlja preko čvorova najbližih suseda sve dok se ne pronađe bliže teme – dostiže se lokalni minimum za udaljenost. Ulazna tačka je uobičajeno na čvorovima visokog stepena (čvorovima sa mnogo veza) da bi se smanjila mogućnost prevremenog zaustavljanja započinjanjem na vrhovima niskog stepena[17]. Umesto da se ovo radi na jednom sloju, što bi zahtevalo dosta vremena i memorije, indeksni prostor je podeljen na hijerarhijske slojeve. Da bi pretražili indeks, prvo tražimo najviši sloj grafa. Najbliže podudaranje sa ovog grafa se zatim prenosi na sledeći sloj dole gde ponovo nalazimo najbliže podudaranje sa vektorom upita. Nastavljamo ovaj proces sve dok ne dođemo do najnižeg sloja na grafu (Slika 13). [16]



Slika 13. Primer pretrage na HNSW grafu[16]

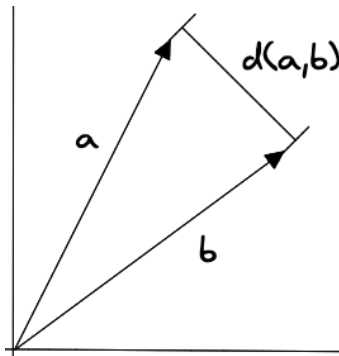
### 3.4. Metrike sličnosti

Osnovna svrha vektorskih baza podataka jeste olakšavanje pretraživanja po sličnosti. Znamo da se slični objekti i podaci nalaze u istom vektorskom prostoru, a da bi odredili koji je vektor najbliži drugome moramo izračunati njihovu međusobnu udaljenost. Matematička interpretacija podataka kod vektorskih baza nam dozvoljava da vršimo matematičke operacije nad njima, kao što je računanje distance između dva vektora da bi odredili njihovu sličnost. U nastavku ovog poglavlja će biti detaljno razmatrane neke od najčešće korišćenih metrika sličnosti.

Euklidska distanca ili L2 distanca predstavlja pravolinijsku udaljenost između dva vektora. Ukoliko imamo dva vektora, vektor  $A = [a_1, a_2, \dots, a_n]$  i vektor  $B = [b_1, b_2, \dots, b_n]$  distanca  $d$  između ova dva vektora se može izračunati formulom:

$$d(A, B) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2} \quad (1)$$

Ukoliko je dobijena vrednost jednaka 0 reč je o istim vektorima a sve veće vrednosti od 0 predstavljaju različitost tih vektora, gde što je veća vrednost distance veća je i različitost. Neke baze podataka ne računaju koren pošto je odnos isti a korenovanje kao operacija skupa. Grafički prikaz ove udaljenosti prikazan je na slici 14.

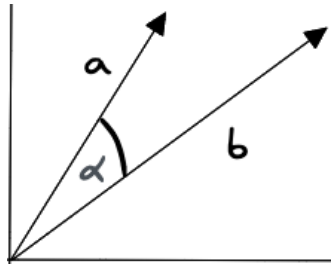


Slika 14. Grafički prikaz Euklidkse (L2) distance između vektora A i B[19]

Kosinusna sličnost ili kosinusna udaljenost meri kosinus ugla između vektora. Ukoliko uvažimo prethodnu interpretaciju vektora A i B, vrednost  $\cos \alpha$  možemo izračunati formulom,

$$\cos \alpha = \frac{A * B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}} \quad (2)$$

Gde  $A * B$  predstavlja skalarni proizvod vektora, dok  $\|A\|$  i  $\|B\|$  predstavljaju intezitet vektora. Vrednost dobijena formulom 2 može da se kreće od -1 do 1, gde 1 predstavlja identične vektore, 0 ortogonalne vektore a -1 vektore koji su suprotni. Slika 15 predstavlja grafički prikaz ove metrike.

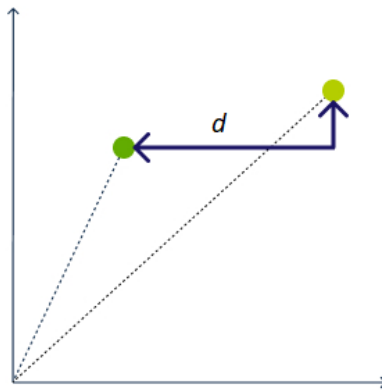


Slika 15. Grafički prikaz metrike kosinusne sličnosti[19]

Manhattan ili L1 udaljenost, meri rastojanje između dva vektora duž osa pod pravim uglom. Može se predstaviti formulom,

$$d(A, B) = \sum_{i=1}^n |a_i - b_i| \quad (3)$$

Gde  $A$  i  $B$  predstavljaju vektorske ugradnje koje se porede. Slično kao kod L2 metrike, vrednost ukoliko je vrednost  $d$  jednaka 0 reč je o identičnim vektorima, a svaka veća vrednost predstavlja veću različitost vektora. Grafički prikaz dat je na slici 16.



Slika 16. Grafički prikaz Manhattan (L1) metrike[12]

Vredi pomenuti da vektorske ugradnje ne moraju biti skup decimalnih brojeva, već mogu biti i skup binarnih brojeva. Tada navedene metrike i nemaju bas mnogo smisla, te se koriste druge metrike. Najkorišćenije među njima jesu Hamming rastojanje i Jaccard rastojanje.

Hammingovo rastojanje se računa kao broj pozicija na kojima se vektori razlikuju i logično je da se radi o identičnim vektorima ako je rastojanje 0 i da se različitost vektora proporcionalno povećava povećanjem Hammingovove distance.

Jaccardovo rastojanje meri različitosti između skupova podataka i dobija se oduzimanjem Jaccardovog koeficijenta od broja 1. Jaccardov koeficijent se može odrediti kao kardinalnost preseka skupova podeljena sa kardinalnošću njihove unije i matematički se može odrediti formulom:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad (4)$$

Ovaj koeficijent može ići u opsegu 0 do 1, gde 1 predstavlja totalno poklapanje vektora a 0 da su to vektori bez ijedne sličnosti.[20]



### 3.5. Filtriranje

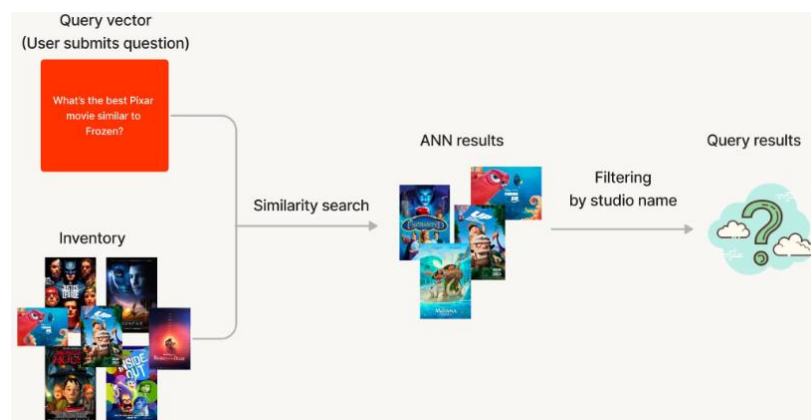
Kao što je već rečeno u radu, svaki vektor koji se skladišti u bazi podataka može imati svoje metapodatke koje se takođe čuvaju. Pored pretraživanja vektora po sličnosti, vektorske baze podataka takođe mogu da filtriraju rezultate na osnovu vektora upita, odnosno upita za metapodatke. Zbog toga, vektorska baza podataka uobičajeno sadrži dva indeksa, vektorski indeks i indeks metapodataka, dok neke baze podataka kombinuju ove indekse u jedan i time izbegavaju proces filtriranja.

Iako se to razlikuje od baze do baze, postoje tri osnovna tipa filtriranja podataka, i to su:

1. Posle upita (engl. Post-query)
2. Zajedno sa upitom (engl. In-query)
3. Pre upita (engl. Pre-query)

#### 3.5.1. Filtriranje posle upita

U ovom pristupu, filtriranje metapodataka se vrši nakon pretrage vektora upita. Ovo će pomoći da se osigura da se svi relevantni podaci uzmu u obzir, ali takođe može dovesti do dodatnih troškova i usporiti proces upita jer se nerelevantni rezultati moraju filtrirati nakon što se pretraga završi.[22]



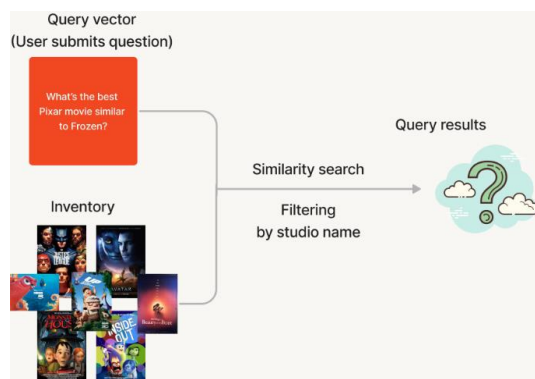
Slika 17. Grafički primer filtriranja posle upita[21]

Na slici 17 možemo videti kako se to generalno izvršava. Zamislimo da imamo vektorsku bazu podataka gde se nalaze crtani filmovi, a mi želimo da nađemo najbolji Pixar-ov film koji je sličan crtanom filmu Frozen. Prvo se izvršava pretraga po sličnosti gde će međurezultat biti N najbližijih filmova filmu Frozen, a nakon toga se izvršava filtriranje koji od tih filmova je izdat od Pixar-a.

Odavde možemo zaključiti dva ključna problema ovog filtriranja. Prvi je da mi ne možemo predvideti koliki set rezultata ćemo imati, a drugi da li će rezultat uopšte postojati. Može se desiti da filter izbaci sve rezultate koji su vraćeni ANN algoritmom a da pritom sigurno u bazi postoji neki objekat koji odgovara našem upitu. Oba problema se može donekle rešiti vraćanjem većeg broja rezultata pre filtriranja, ali onda ovo dodatno usporava ceo proces pretrage i filtriranja.

### 3.5.2. Filtriranje zajedno sa upitom

Filtriranje zajedno sa upitom je strategija u kojoj se ANN pretraga i filtriranje izvršavaju istovremeno (Slika 18.). Tokom pretrage, i vektor sličnosti i metapodaci moraju biti izračunati i obrađeni, što dovodi do velike potražnje za sistemom.

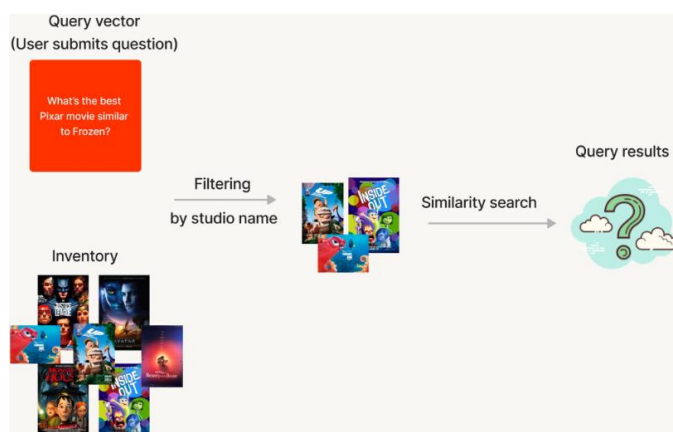


Slika 18. Grafički primer filtriranja zajedno sa upitom[21]

Za efikasno izvršenje filtriranja zajedno sa upitom, vektorska baza podataka mora da uči u memoriju kako vektorske, tako i skalarne podatke relevantne za filtriranje. Međutim, u situacijama kada postoji obimno filtriranje koje uključuje pretragu po sličnosti i skalarno filtriranje atributa, postoji realna opasnost od preopterećenja memorije (engl. OOM-OutOfMemory). Strategije filtriranja koje kombinuju pretragu po sličnosti i skalarno filtriranje su optimalne samo za vektorske baze podataka koje su organizovane po principu redova, prilagođavajući se tako potrebama i zahtevima za efikasno upravljanje podacima.

### 3.5.3. Filtriranje pre upita

Filtriranje pre upita je tehnika koja uključuje početni skrining potencijalnih podudaranja pre sprovođenja glavnog upita za pretragu vektora. Iako ovo može pomoći da se smanji prostor za pretragu, takođe može dovesti do toga da sistem previdi relevantne rezultate koji ne odgovaraju kriterijumima filtera metapodataka. Pored toga, opsežno filtriranje metapodataka može usporiti proces upita zbog dodatnih računarskih troškova[9].



Slika 19. Grafički primer filtriranja pre upita[21]

Sa slike 19. možemo videti da prvo vrši filtriranje filmova na osnovu koja je izdavačka kuća u pitanju i da se vraća lista vektora filmova koji zadovoljavaju uslove filtera. Nakon toga, ANN

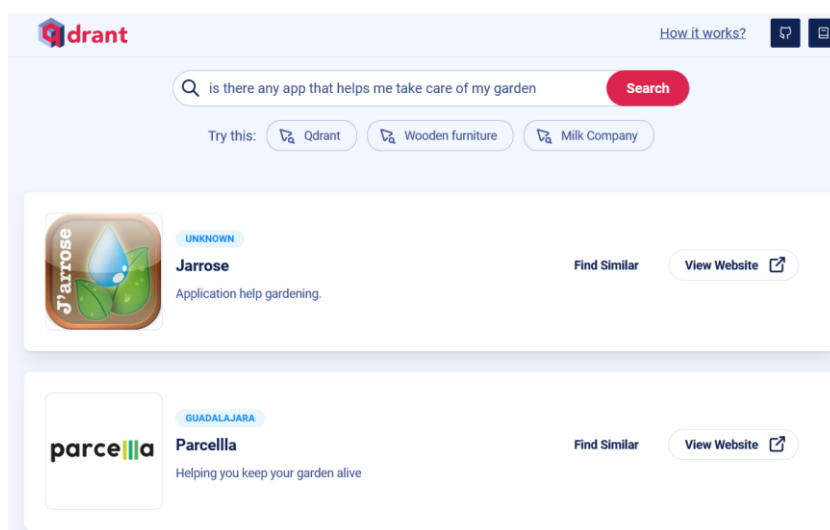
algoritam će se izvršiti na samo porciju vektora koji su dobijeni kao međurezultat, odnosno prihvatljivih vektora. Ovaj metod eliminiše nepotrebne rezultate u ranoj fazi, poboljšavajući efikasnost i tačnost naknadne pretrage vektora.

Iako ovo zvuči kao dobro rešenje, ipak nije. Pošto se svi podaci filtriraju pre nego što se pređe na ANN algoritam pretrage, ova implicira da je to pretraga grube sile koja dosta usporava sistem.

## 4. PRIMENA I OSOBINE VEKTORSKIH BAZA PODATAKA

Zahvaljujući svom jedinstvenom dizajnu i mogućnostima brze pretrage po sličnosti vektorskih ugradnji, vektorske baze podataka nude značajne mogućnosti za različite aplikacije, podstičući transformaciju u načinu na koji kompanije upravljaju, analiziraju i izvlače uvide iz nestruktuiranih podataka. U nastavku rada biće obrađeni najučestalije primene i osobine ovih baza podataka.

Semantičko pretraživanje – U pronalaženju informacija i obradi prirodnog jezika (engl. Natural Language Processing) vektorske baze podataka mogu poboljšati tačnost i efikasnost semantičkih pretraga. Korišćenjem dubokog učenja, posebno neuronskih mreža sa višestrukim slojevima, tehnika poput Word Embedding-a i Transformer modela, kompanije mogu da koriste vektorske baze podataka za traženje sličnih reči, fraza ili dokumenata[23]. Na slici 20. prikazana je Qdrant-ova demo aplikacija koja omogućava semantičku pretragu.



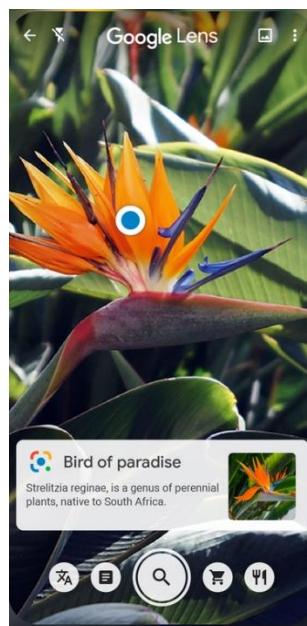
Slika 20. Qdrant-ova demo aplikacija za semantičku pretragu[24]

Pretraživanje po sličnosti i sistemi za preporuku – Najveća prednost vektorskih baza podataka je njena sposobnost za pretraživanje vektora po sličnosti, i sposobnost da vrati najbližije podatke vektoru upita. Sposobnost transformacije nestruktuiranih tipova podataka, kao što su muzika, video i slika dozvoljavaju kompanijama da to iskoriste u cilju poboljšanja sistema za preporuku. Iako nije nigde specificirano koju arhitekturu kompanije koriste na osnovu jedinstvene funkcionalnosti koje one pružaju može se gotovo sa sigurnošću tvrditi da one koriste vektorsku bazu podataka ili vektorsko pretraživanje. Neke od poznatijih kompanija koje koriste pretragu po sličnosti su Amazon, kao jedna od najvećih platforma za elektronsku trgovinu koristi sisteme preporuka baziranu na vektorskim bazama podataka, Netflix koristi ove baze podataka za svoj sistem preporuku serija i filmova, kao i Spotify i Google Lens koje ću detaljnije objasniti.

Spotify kao najpopularnija striming platforma za muziku koristi vektorske podatke. Muzički audio zapisi se mogu predstaviti kao vektori na osnovu audio karakteristika, a korisnici mogu biti predstavljeni na osnovu njihovih navika slušanja. Ovo omogućava Spotify-u da preporuči sličnu muziku korisnicima pronalaženjem pesama sa vektorima bliskim onima u kojima korisnik uživa. Inženjeri koji rade u Spotify-u su takođe napravili dve biblioteke za

vektorsko pretraživanje, 2013. godine biblioteku zvanu Annoy i 2022. godine poboljšanu verziju hnsplib biblioteke zvanu Voyager.[25]

Google Lens donosi revolucionarnu pretragu slika po sličnosti. Pretvaranjem slike u vektore i poređenje sa već zapamćenim vektorima Google Lens ima mogućnost da vrati precizne rezultate. Ako slika sadrži neki određeni proizvod, kao što su farmerke ili patike, Google Lens je u stanju da prikaže i dodatne informacije o tom proizvodu kao i da pronađe web sajt koji prodaje baš taj ili sličan proizvod (Slika 21.).[26]



Slika 21. Primer rada Google Lens aplikacije[27]

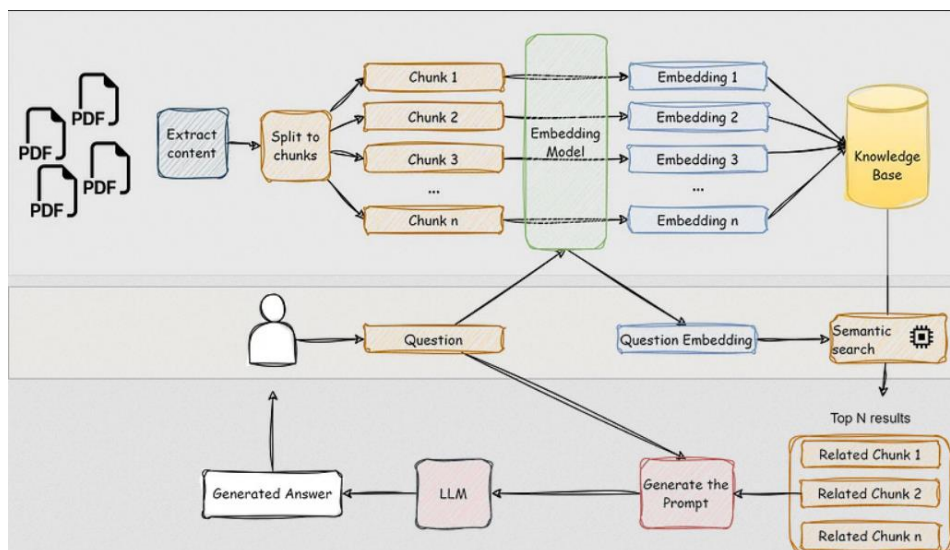
Otkrivanje anomalija – Uočavanje odstupanja je podjednako bitno kao i prepoznavanje sličnosti. Vektorske baze podataka se mogu koristiti i u finansijama, zdravstvu, bezbednosti i otkrivanju prevara, gde je cilj identifikovanje anomalnog ponašanja. Predstavljajući normalno i anormalno ponašanje kao vektora, korisnici mogu da koriste pretragu sličnosti i brzo identifikuju potencijalne pretnje ili lažne aktivnosti.[23]

U bioinformatičari vektorske baze podataka se mogu koristiti za skladištenje i ispitivanje genetskih sekvenci, proteinskih struktura i drugih bioloških podataka koji se mogu predstaviti kao vektori, što uključuje DNK, RNK i proteina koji se često predstavljaju kao nizovi nukleotida ili aminokiselina. Pronalaženjem sličnih vektora istraživači mogu identifikovati slične genetske sekvence ili strukture proteina pomažući im da se unapredi naše razumevanje bioloških sistema i bolesti i tako pomogne u otkrivanju novih genetičkih varijanti i razvoju novih lekova.[23]

AI čet botovi – Vektorske baze podataka omogućavaju čet botovima sa veštačkom inteligencijom da imaju “pamćenje” ukoliko zapamtimo i pronalazimo poruke iz istorije prepiske koje su relevantne za trenutnu konverzaciju.

RAG (engl. Retrieval Augmented Generation), odnosno pronalaženje proširene generacije predstavlja je pristup koji se koristi za poboljšanje konteksta koji se pruža LLM-u. Vektorska baza podataka se koristi da poboljša *prompt* prosleđen LLM-u dodavanjem dodatnog konteksta uz upit.

Na slici 22. možemo videti tok rada RAG aplikacije. Gornja polovina slike predstavlja kako se podaci čuvaju u bazi. Korišćenjem prave metodologije fajlovi se prvo dele na manje celine koje se posle transformišu pomoću nekog embedding modela i skladište se u bazi podataka. Kada korisnik unese pitanje ono se takođe transformiše u vektorske ugradnje i onda se izvršava semantička pretraga koja vraća N rezultata. Pitanje, zajedno sa rezultatima semantičke pretrage se prosleđuje LLM-u koji efikasnije obrađuje informacije što na kraju dovodi do tačnijih i smislenijih rezultata.



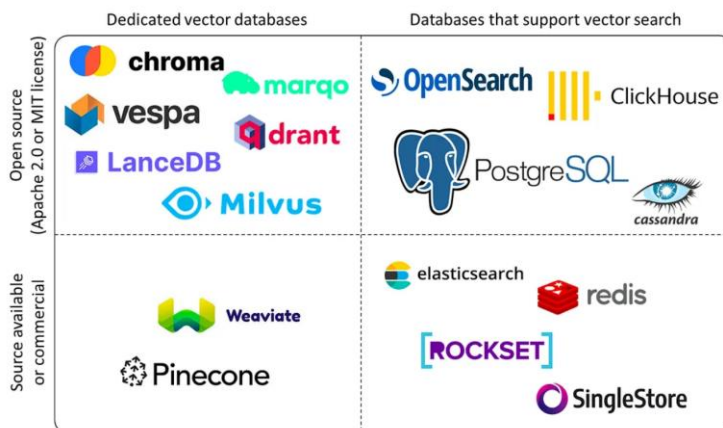
Slika 22. Tok rada RAG aplikacije[28]

## 4.1. Najpoznatije vektorske baze podataka

Najpoznatije vektorske baze podataka (Slika 23.) su:

- Pinecone
- Milvus
- Chroma
- Weaviate

U narednim odlomcima ovog diplomskog rada biće dat kratak opis, prednosti i mane ovih baza podataka.

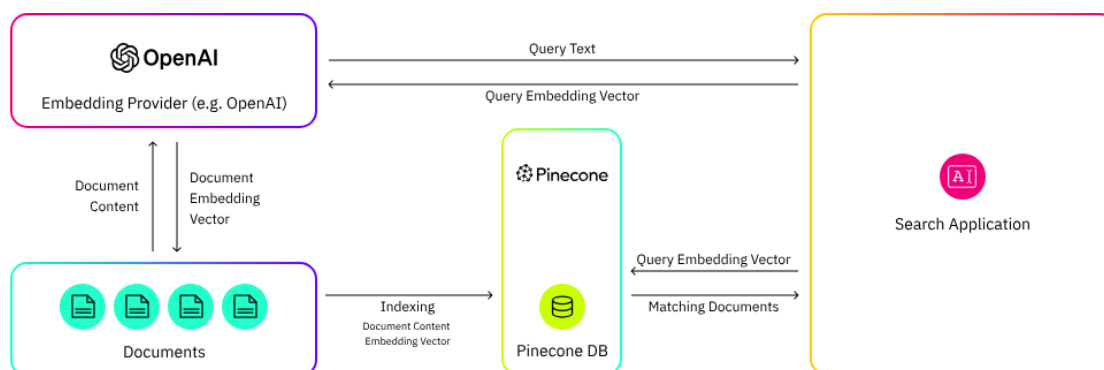


Slika 23. Primeri vektorskih baza podataka i baza koje podržavaju vektorsku pretragu [8]

### 4.1.1. Pinecone baza podataka

Pinecone je vektorska baza podataka zasnovana na oblaku koja nudi besprekorni API i infrastrukturu. Ona eliminiše potrebu da korisnici upravljaju infrastrukturom, omogućavajući im da se usresrede na razvoj i proširenje svojih AI rešenja. Pinecone se ističe u brznoj obradi podataka, podržava filtere metapodataka i retko gust indeks za tačne rezultate.

Prednosti su mu pretraga u realnom vremenu, skalabilnost, automatsko indeksiranje i podrška za python programski jezik dok su mane visoka cenovna struktura, posebno za velike implementacije sa značajnim količinama podataka [13]. Na slici 24. data je tipična arhitektura aplikacije koja koristi Pinecone kao bazu podataka.

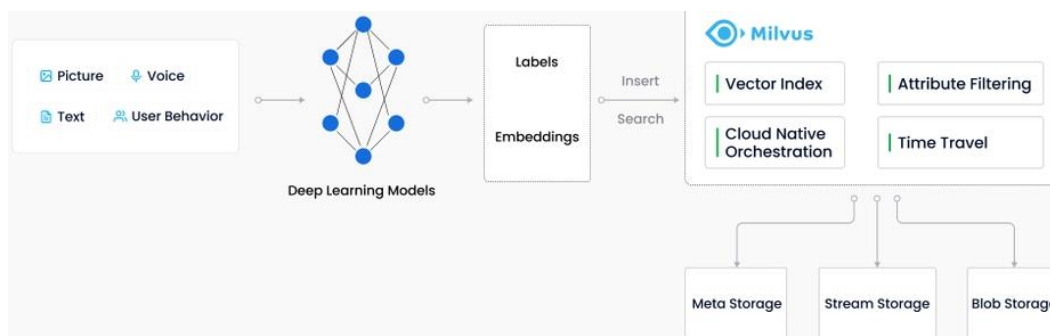


Slika 24. Arhitektura aplikacije koja koristi Pinecone kao bazu [8]

### 4.1.2. Milvus baza podataka

Milvus je vektorska baza podataka otvorenog koda, dizajnirana da olakša ugrađivanje vektora, efikasno pretraživanje sličnosti i AI aplikacije. Ova baza pojednostavljuje, nestruktuirano pretraživanje i pruža jedinstveno korisničko iskustvo nezavisno od okruženja primene. Objavljen pod Apache License 2.0, Milvus nudi pretragu trilionskih vektorskih skupova podataka u milisekundama. Razlikuje se od drugih baza po tome što kombinuje skalarno filtriranje sa vektorskom sličnošću, što kulminira u hibridnom rešenju za pretragu. Slučajevi korišćenja za Milvus bazu uključuju pretragu slika, čet botove i pretragu hemijske strukture.[15]

Glavne vrline ove baze su munjevito brza obrada i podrška za GPU, jednostavan API dizajn, skalabilan dizajn sa visokim performansama kao i to što je otvorenog koda sa jakom zajednicom, a kao mane možemo izdvojiti to što zauzima značajan resurs kada se pokreće lokalno i traženje podataka nije intuitivno. Na slici 25. prikazan je tok rada Milvus baze podataka.

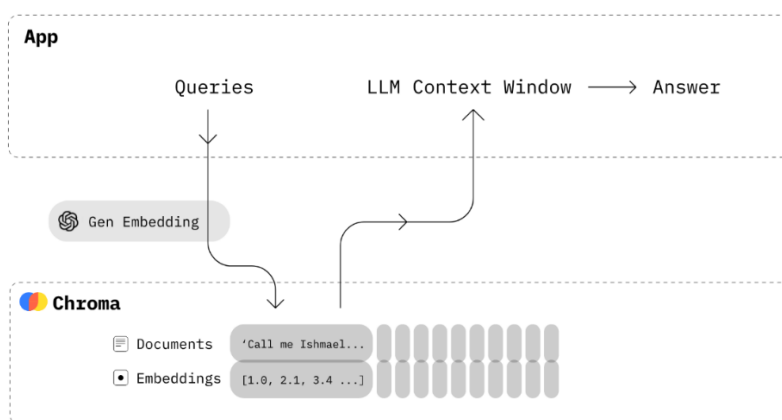


Slika 25. Milvus – tok rada[14]



### 4.1.3. Chroma baza podataka

Chroma je baza podataka otvorenog koda koja ima za cilj da pojednostavi proces kreiranja LLM (engl. Large Language Model) aplikacija koje pokreće obrada prirodnog jezika, tako što će znanje, činjenice i veštine učiniti dostupnim za modele mašinskog učenja na nivou LLM-a (Slika 26.). Mnogi inženjeri su izrazili želju za “ChatGPT, ali za podatke” a Chroma baza nudi ovu vezu putem preuzimanja dokumenta zasnovanog na ugrađivanju. Prednosti su joj to što je otvorenog koda sa odličnom zajednicom, mogućnost proširenog upita. Mane ove baze podataka je kompleksnost razvijanja aplikacije, zahteva više truda i stručnosti u poređenju sa drugim bazama podataka kao i nedostatak performansi u određenim scenarijima visoke propusnosti u realnom vremenu.[13]

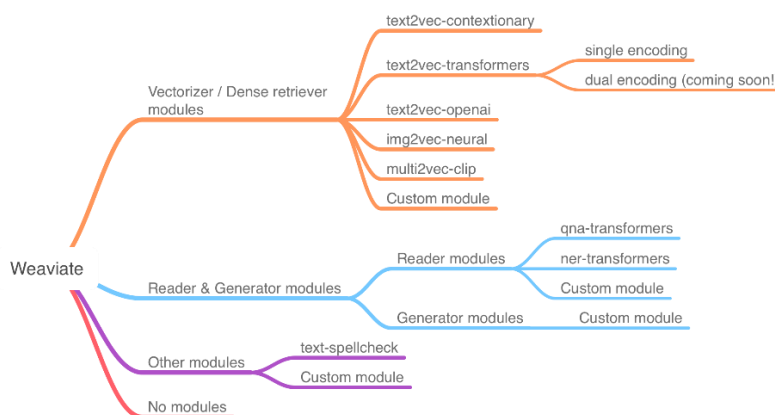


Slika 26. Izgradnja LLM aplikacija korišćenjem Chroma baze podataka [8]

### 4.1.4. Weaviate baza podataka

Weaviate je vektorska baza podataka otvorenog koda koja čuva objekte i njihove vektore radi omogućavanja kombinovane pretrage vektora. Razlikuje se od drugih vektorskih baza podataka po tome što pruža različite module koji se mogu instalirati radi automatskog upravljanja podacima, uključujući vektorizaciju. Takođe, Weaviate može koristiti više od jednog modula u zavisnosti od raznolikosti podataka (Slika 27.).[12]

Glavna prednost Weaviate baze podataka je brzina pretrage. Može brzo pronaći 10 najbližih suseda iz miliona objekata za samo nekoliko milisekundi. Takođe podržava GraphQL i smatra se veoma lakom za korišćenje. Što se tiče performansi, Weaviate može imati nešto slabije performanse u poređenju s drugim vektorskim bazama podataka poput Milivusa.[14]



Slika 27. Dostupni tipovi modula Weaviate baze podataka[12]



## 4.2. Poređenje sa relacionim bazama podataka

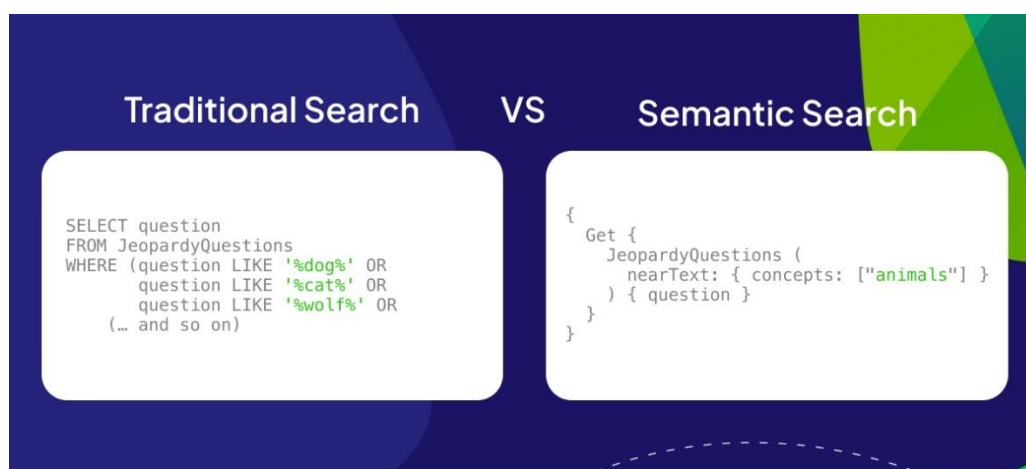
Vektorske baze podataka pružaju ključne prednosti u brzom i preciznom pretraživanju podataka putem sličnosti. Ova tehnologija omogućava pronalaženje relevantnih informacija na osnovu semantičkog ili kontekstualnog značenja i ta karakteristika čini ih otpornim na greške u kucanju te im omogućava da interpretiraju semantički kontekst pretrage, što rezultira preciznijim rezultatima. Kod tradicionalnih baza podataka pretraživanje se oslanja na tačna podudaranja ili unapred definisane kriterijume.

Glavna razlika između vektorskih i relacionih baza podataka leži u optimizaciji za različite tipove podataka. Dok relacione baze podataka efikasno rukuju strukturiranim podacima organizovanim u tabelama, vektorske baze podataka su prilagođene za skladištenje nestruktuiranih podataka i njihovih vektorskih reprezentacija.

Vektorske baze podataka su posebno korisne za aplikacije koje zahtevaju brzo i precizno uparivanje podataka na osnovu sličnosti, što ih čini idealnim za primene u oblastima kao što su analitika podataka, mašinsko učenje i obrada prirodnog jezika. Njihova sposobnost da efikasno tumače semantički kontekst čini ih ključnim alatom za savremene sisteme pretrage i analitike.

Na primer, zamislimo da imamo bazu podataka koja čuva pitanja za TV kviz i želimo da preuzmемо sva pitanja koja sadrže životinju. Pošto se pretraga u tradicionalnim bazama oslanja na podudaranja ključnih reči, mi bismo imali veliki upit koji ispituje sve vrste životinja. Sa pretragom možemo jednostavno tražiti koncept “životinje”. Pošto većina vektorskih baza ne samo da skladišti vektorsku reprezentaciju podataka već ih čuva zajedno sa izvornim podacima, one ne samo da su sposobne za vektorsku pretragu već i omogućavaju tradicionalnu pretragu po ključnim rečima (Slika 28.).[12]

Moram napomenuti, da vektorske baze nikako ne može zameniti relacione baze podataka, već svaki tip baza ima svoju prednost i na osnovu našeg tipa problema mi trebamo izabrati odgovarajući tip baze podataka.



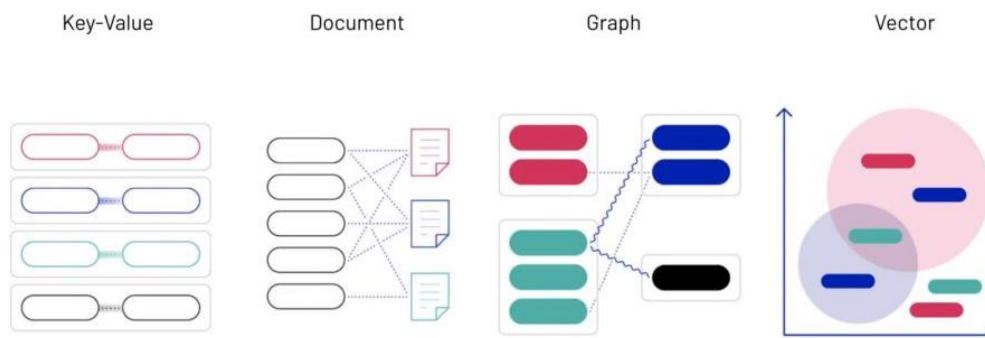
Slika 28. Primer tradicionalnog i semantičkog pretraživanja[12]

## 4.3. Poređenje sa nerelacionim bazama podataka

Nerelacione baze podataka, poznate i kao NoSQL baze podataka, fleksibilni su tip baze koja može da rukuje različitim tipovima podataka, uključujući struktuirane, polustruktuirane i nestruktuirane podatke. Glavna prednost nerelacionih baza podataka je u tome što mogu veoma dobro da rukuju velikim podacima i lako se mogu vertikalno skalirati. Međutim, nerelacione baze podataka mogu da se muče sa složenim upitima i visokodimenzionalnim podacima, dok su vektorske baze podataka dosta efikasnije u tom pogledu. Vektorske baze podataka mogu

pretraživati u višedimenzionalnom prostoru, omogućavajući im da brzo pronađu vektore najbližijim datom vektoru. Pored toga, vektorske baze podataka mogu da obrađuju i podatke poput slika i teksta, što nerelacionim bazama podataka može biti teško.

Danas su mnoge postojeće baze podataka već omogućile vektorsku podršku i pretragu vektora. Međutim, one obično ne indeksiraju ugrađivanje vektora, što usporava pretragu vektora. Na slici 29. dati su grafički prikazi NoSQL baze podataka.



Slika 29. NoSQL baza podataka[8]

## 5. PRETRAŽIVANJE SLIKA PO SLIČNOSTI I SEMANTICI

Za potrebe praktičnog dela ovog diplomskog rada, realizovana je manja aplikacija koja uključuje dve ključne funkcionalnosti vektorskih baza podataka - semantičku pretragu i pretragu po sličnosti. Za vektorsku bazu podataka izabrana je Weaviate baza kao i njen model multi2vec-clip koji omogućava transformaciju slika i teksta u istom vektorskom prostoru. Ovaj model detaljnije je opisan u odeljku 3.2. Vektorske ugradnje.

U nastavku rada biće prikazan postupak instalacije i pokretanje Weaviate baze podataka pomoću Docker compose opcije, instaliranje Python i Node paketa koji omogućavaju komunikaciju sa našom bazom kao i pothranjivanje podacima i prikazivanje osnovnih funkcionalnosti same aplikacije.

Kod kompletnog projekta, kao i samog diplomskog rada dostupan je na Github repozitorijumu:

<https://github.com/SlavkoPetrovic/Diplomski-rad>

### 5.1. Instalacija Weaviate baze podataka, Python i Node paketa

Zbog jednostavnosti preuzimanja i instaliranja kao i samog rada sa Weaviate vektorskom bazom podataka izabrana je instalacija preko docker kontejnera, odnosno pomoću docker compose opcije i yml fajla. Docker je alat dizajniran tako da olakšava razvoj, implementaciju i pokretanje aplikacija koristeći kontejnere, koji su paketi pojedinačnih aplikacija koji sadrže sve neophodno za izvršavanje i pokretanje.

Weaviate se takođe može pokrenuti preko njihovog cloud servisa, kubernetes-a kao i AWS ili Google Marketplace-a.[29]

U okviru docker-compose.yml fajla dodat je deo koji omogućava korišćenje multi2vec-clip modela. Kod koji omogućava pokretanje pomoću docker-a dostupan je na sledećem linku:

<https://weaviate.io/developers/weaviate/installation/docker-compose#starter-docker-compose-file>

Pod uslovom da je na našoj mašini instaliran docker, onda pomoću Windows terminala i Command Prompt-a potrebno je pozicionirati se u istom folderu gde se nalazi docker-compose.yml fajl i izvršiti sledeću komandu.

```
docker-compose up --build
```

Ova komanda se koristi za pokretanje Docker Compose servisa sa dodatkom *--build* opcije. Ova opcija navodi da Docker prvo izvrši izgradnju (engl. build) svih Docker slika definisanih u docker-compose.yml fajlu pre nego što pokrene kontejnere. Time osiguravamo da se svaka promena u tom fajlu primeni pre pokretanja servisa.

Korišćenjem docker desktopa ili sledeće komande možemo proveriti da li se uspešno izvršilo podizanje baze, trebalo bi da se pokrenu dva kontejnera, jedan za weaviate bazu, a drugi za multi2-vec model.

```
docker ps
```

Pomoću Weaviate klienta omogućena je implementacija u programskim jezicima Python, Go, Java i Javascript/Node. Funkcionalnosti za kreiranje šeme i pothranjivanje baze podacima izgrađena je pomoću Python programskog jezika, a za pretraživanje i prikazivanje rezultata napisan je kod u Javascript-u, odnosno Node-u. Pre svega treba proveriti da li imamo instaliran Python i pip, kao i Node.js i npm. Pod pretpostavkom da je to uspešno instalirano, možemo preći na instaliranje potrebnih paketa i modula.

Da bi aplikacija mogla da se izvršava potrebno je instalirati python paket koji omogućava konekciju kao i sam API za komunikaciju sa bazom pomoću komande,

```
pip install weaviate-client==4.4b4
```

ili sledećom komandom ukoliko se pokreće iz mog repozitorijuma:

```
pip install -r requirements.txt
```

Što se tiče Node modula potrebno je instalirati komandom sledeće:

```
npm install express multer weaviate-client
```

- Express modul je framework za Node.js koji se koristi za pravljenje web aplikacija i API-ja
- Multer predstavlja middleware za rukovanje upload-ovanja slika
- Weaviate-client je modul koji omogućava komunikaciju i API za rad sa Weaviate bazom podataka

Što se tiče podataka, preuzeto je ukupno 140 slika koje se mogu klasifikovati u 28 klasa, za svaku klasu po 5 slika. U realnim sistemima ovo predstavlja mali broj podataka, prvenstveno to što ima po 5 slika za svaku klasu ali to je urađeno sa razlogom zato što prikazujem 6 najpribližnijih rezultata, gde će 6. slika uglavnom biti ona koja je približna klasi prve slike. Slike nisu licencirane, preuzete su sa sajtova koji dozvoljavaju korišćenje u komercijalne svrhe bez navođenja izvora. Preuzete slike nalaze se u folderu data/images.

## 5.2. Popunjavanje Weaviate baze podacima

Ukoliko je sve uspešno izvršeno, možemo napraviti add\_data.py skriptu koja će ostvariti konekciju sa Weaviate bazom, napraviti potrebne kolekcije i šeme kao i izvršiti konverziju slika u vektorske ugradnje i smestiti ih u vektorskom prostoru.

U nastavku ovog odeljka biće prikazane i objašnjene najbitnije funkcije koje su napisane za ove potrebe.

```
9  COLLECTION_NAME = "DiplomskiRadCollection"
10 imgdir = Path("data/images")
11
12
13 def connect() -> WeaviateClient:
14     return weaviate.connect_to_local()
15
16
17 def delete_existing(client: WeaviateClient) -> bool:
18     client.collections.delete(COLLECTION_NAME)
19     return True
```

Slika 30. Konekcija i brisanje postojeće šeme

Na slici 30. prikazane su funkcije za kreiranje konekcije kao i brisanje postojeće kolekcije.

Varijabla **COLLECTION\_NAME** je postavljena na simboličan naziv „*DiplomskiRadCollection*“ i služi za identifikaciju kolekcije u vektorskoj bazi podataka, a u varijabli **imgdir** čuvamo relativnu putanju do naših slika.

Funkcija **connect()** koja se nalazi u 13. redu služi za kreiranje konekcije na Weaviate bazu pomoću njihovog klijenta. Funkcijom **Connect\_to\_local()** omogućeno je povezivanje sa lokalnom Weaviate instancom koja je deploy-ovana pomoću docker compose-a sa standardnim konfiguracijama porta.

**Delete\_existing(..)** funkcijom omogućeno je brisanje zadate kolekcije. Ovo je posebno važno ukoliko želimo da promenimo indeks koji koristi naša baza, kao i metriku distance.

Na slici 31. možemo videti funkciju koja obavlja kreiranje kolekcije sa zadatom metrikom i indeksom.

```
22 def define_collection(client: WeaviateClient) -> bool:
23     client.collections.create(
24         name=COLLECTION_NAME,
25         vector_index_config=wvc.config.Configure.VectorIndex.hnsw(
26             distance_metric=wvc.config.VectorDistance.MANHATTAN
27         ),
28         vectorizer_config=wvc.config.Configure.Vectorizer.multi2vec_clip(
29             image_fields=["image"],
30             vectorize_collection_name=False,
31         ),
32         generative_config=wvc.config.Configure.Generative.openai(),
33         properties=[
34             wvc.Property(
35                 name="image",
36                 data_type=wvc.config.DataType.BLOB,
37             ),
38             wvc.Property(
39                 name="filename",
40                 data_type=wvc.config.DataType.TEXT,
41                 skip_vectorization=True,
42             ),
43         ],
44     )
45     return True
```

Slika 31. Kreiranje kolekcije

Parametrom **name** označili smo naziv naše kolekcije, a parametrom **vector\_index\_config** odredili smo koji indeks će biti korišćen kao i koja metrika.

Weaviate baza podržava dva tipa indeksa, HSNW i Flat indekse kao i kosinusnu, dot, L2, L1 i hammingovu udaljenost. Ukoliko nije navedena, podrazumevani indeks je HSNW a podrazumevana metrika udaljenosti je kosinusna udaljenost. U ovom kodu prikazano je podešavanje **HSNW** indeksa i **MANHATTAN (L1)** metrike udaljenosti. Radi poređenja rezultata koji se vraćaju pozivanjem upita biće prikazana i kosinusna metrika. Jedina promena u kodu jeste u 26. liniji koda gde umesto **MANHATTAN** unosimo **COSINE**.

Parametrom **vectorizer\_config** podešeno je korišćenje **multi2vec\_clip** modula koji se u ovom slučaju koristi za transformaciju slika u vektorske ugradnje.

Parametrom **generative\_config** omogućeno je korišćenje openai modela, konkretno

GPT 3.5 koji služi za pronalaženje proširene generacije (RAG) na osnovu podataka koji su smešteni u Weaviate instanci.

I poslednjim parametrom *properties* označena je lista svojstava koja će se čuvati u bazi. Prvo svojstvo predstavlja sliku koja je tipa BLOB a drugi je naziv samog fajla koji nećemo vektorizovati zato što sam imena slika zadavao smislenim imenima koje bi mogle uticati na tačnost pretrage.

```
48 def import_data(client: WeaviateClient) -> BatchObjectReturn:
49     mm_coll = client.collections.get(COLLECTION_NAME)
50
51     data_objs = list()
52     for f in imgdir.glob("*.jpg"):
53         b64img = base64.b64encode(f.read_bytes()).decode()
54         data_props = {"image": b64img, "filename": f.name}
55         data_obj = wvc.data.DataObject(
56             | properties=data_props, uuid=generate_uuid5(f.name)
57         )
58         data_objs.append(data_obj)
59
60     insert_response = mm_coll.data.insert_many(data_objs)
61
62     print(f"{len(insert_response.all_responses)} insertions complete.")
63     print(f"{len(insert_response.errors)} errors within.")
64     if insert_response.has_errors:
65         for e in insert_response.errors:
66             | print(e)
67
68     return insert_response
```

Slika 32. Pothranjivanje podataka u bazu

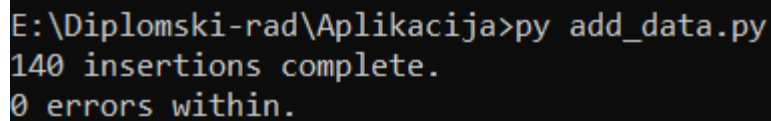
Na slici 32. funkcija *import\_data* omogućava popunjavanje baze podataka slikama, tj. vektorskim ugradnjama. Kroz sve fajlove koje se nalaze u folderu *imgdir* nalazi one koje imaju ekstenziju *.jpg* i popunjava se lista *data\_objs* sa svojstvima koje su navedene i objašnjene u prethodnom pasusu. Funkcija na 60. liniji izvršava popunjavanje podataka u naznačenoj kolekciji. Njenu povratnu vrednost možemo iskoristiti da proverimo da li su sve slike uspešno ubačene ili se desila neka greška. Na slici 33. prikazan je redosled pozivanja objašnjenih funkcija koje će se izvršavati pri startovanju skripte.

```
70 def main():
71     | client = connect()
72     | delete_existing(client)
73     | define_collection(client)
74     | import_data(client)
75
76 if __name__ == "__main__":
77     | main()
78
```

Slika 33. Pozivanje funkcija u main

Pozicioniranjem u folderu gde se nalazi skripta, i pozivanjem sledeće komande izvršavaju se navedene funkcije. Na slici 34. možemo videti da su se svih 140 slika uspešno učitale.

`py add_data.py`



```
E:\Diplomski-rad\Applikacija>py add_data.py
140 insertions complete.
0 errors within.
```

Slika 34. Izvršavanje skripte add\_data.py

### 5.3. Pretraživanje vektora

U skripti app.js, napisanoj pomoću JavaScript-a/Node.js-a, implementirana je funkcionalnost pretraživanja vektorske baze podataka, kao i prikazivanje rezultata pretrage. Ova funkcionalnost omogućava korisnicima da unesu sliku po kojoj bi pretraživali vektorski prostor i dobili slične slike ili da unesu tekst koji bi semantičkom analizom uz pomoć CLIP modela uspešno vratili najtačnije rezultate. Obzirom da deo prikazivanje rezultata nije od relevantnog značaja za ovu temu diplomskog rada, u nastavku će samo biti objašnjene funkcionalnosti pretrage.

```
117 app.post('/search', upload.single('image'), async (req, res) => {
118   const searchText = req.body.searchText;
119   const image = req.file;
120
121   if (image) {
122     const imgData = fs.readFileSync(image.path).toString('base64');
123     const response = await client.graphql.get()
124       .withClassName(COLLECTION_NAME)
125       .withNearImage({ image: imgData })
126       .withFields(['filename _additional { distance }'])
127       .withLimit(6)
128       .do();
129     const distances = response.data.Get.DiplomskiRadCollection.map(item => item._additional.distance);
130     const fullImagesPath = response.data.Get.DiplomskiRadCollection.map(item => item.filename);
131     res.send(generateImageSearchResults(distances, fullImagesPath));
132   } else {
133     const response = await client.graphql.get()
134       .withClassName(COLLECTION_NAME)
135       .withNearText({ concepts: [searchText] })
136       .withFields(['filename _additional { distance }'])
137       .withLimit(6)
138       .do();
139     const distances = response.data.Get.DiplomskiRadCollection.map(item => item._additional.distance);
140     const fullImagesPath = response.data.Get.DiplomskiRadCollection.map(item => item.filename);
141     res.send(generateImageSearchResults(distances, fullImagesPath));
142   }
143 });
```

Slika 35. Pretraživanje baze na osnovu slika ili teksta

Na slici 35. prikazana je *post* metoda koja omogućava pretraživanje po sličnosti slika ili semantici teksta. Ukoliko je zadat i tekst i slika, funkcija će uzeti u obzir samo sliku dok se ona ne poništi.

Koristeći Weaviate client koji smo prethodno istancirali i odradili konekciju na našu bazu, mi možemo pretraživati vektorski prostor. Weaviate client omogućava korišćenje GraphQL-a, gRPC-a ili RESTful API-ja za komunikaciju i slanje upita. U ovom kodu prikazano je korišćenje GraphQL-a gde možemo tačno specificirati koja polja želimo da dobijemo kao povratnu vrednost.



Prateći 123. liniju koda, možemo videti kako se konstruiše vektor upita koristeći weaviate klijent i GraphQL. Pozivajući metodu *get()* govorimo da želimo da dobijemo podatke, iz naše kolekcije pomoću *withClassName(..)* funkcije. Nakon toga u argumentu funkcije *withNearImage(..)* kao parametar dostavljamo upload-ovanu sliku upita kodiranu u base64 string. Metodom *withFields(..)* određujemo koje povratne vrednosti želimo da dobijemo. Radi jednostavnosti same aplikacije izabrano je da se vrati naziv slike kao i metrika udaljenosti koja je rečena u prethodnom podglavlju. Sa funkcijom *withLimit(6)* određujemo da želimo da dobijemo top 6 rezultata najbližih vektoru upita, i završavamo funkcijom *.do()* koja govori da je to kraj upita i da treba da nam vrati rezultat. Nakon upita mapiramo dobijene udaljenosti i naziv slika i pomoću funkcije napisane na 131. liniji.

Jedina stvar koja se razlikuje u pretrazi slika i teksta jeste što umesto funkcije *withNearImage(..)* koristimo funkciju *withNearText(..)* gde kao parametar dostavljamo uneseni tekst pretrage.

Samu aplikaciju i server možemo startovati pozicioniranjem u dati folder i kucanjem komande,

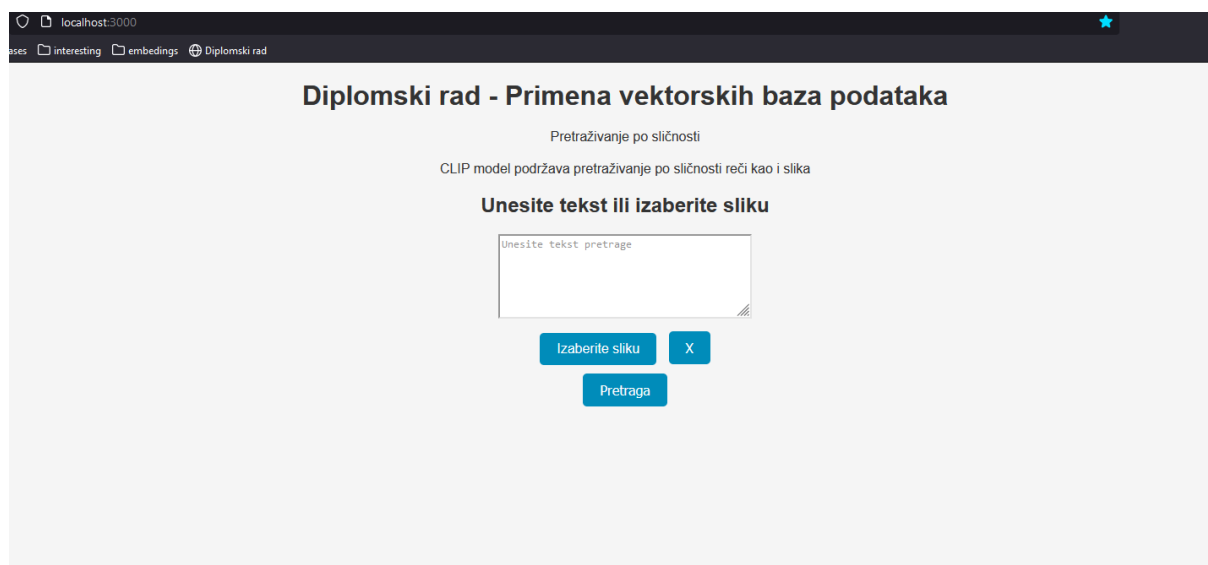
```
npm app.js
```

```
E:\Diplomski-rad\Aplikacija>node app.js  
Server is running on port 3000
```

Slika 36. Prikaz konzole nakon startovanja

Na slici 36. možemo videti da je aplikacija startovana na portu 3000. Interfejsu same aplikacije možemo pristupiti na sledećem linku:

<http://localhost:3000/>

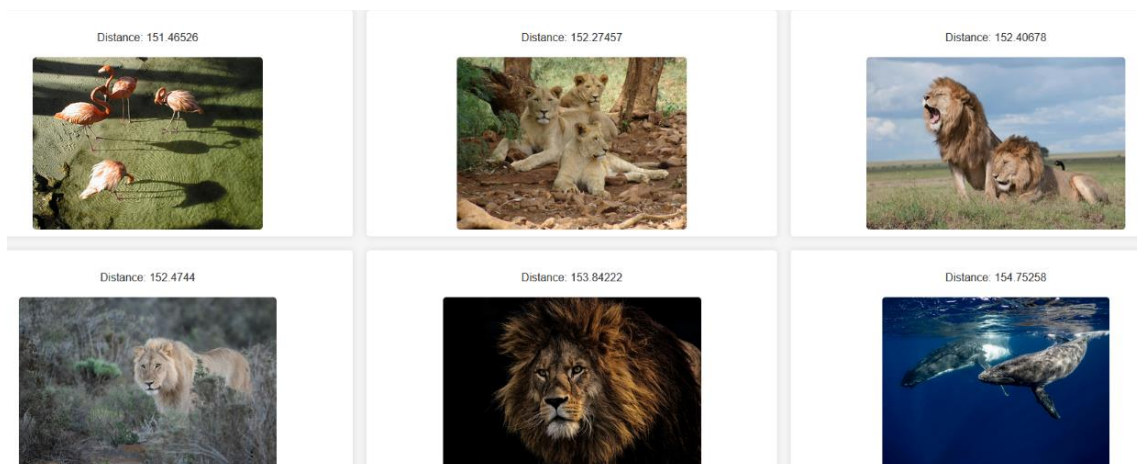


Slika 37. Početna strana aplikacije

Na slici 37. prikazana je početna strana aplikacije. Postoji *text-box* gde možemo napisati šta želimo da pretražimo. Selektovanje slike omogućeno je klikom na dugme *Izaberite sliku* gde se otvara *file explorer* koji nam olakšava izbor slike. Kao što je iznad rečeno, ukoliko je unesen i tekst i otvorena slika, klikom na dugme *Pretraga* biće uzeta u obzir samo slika. Ukoliko ipak želimo da pretražimo po tekstu moguće je poništiti izbor slike na dugme *X*.



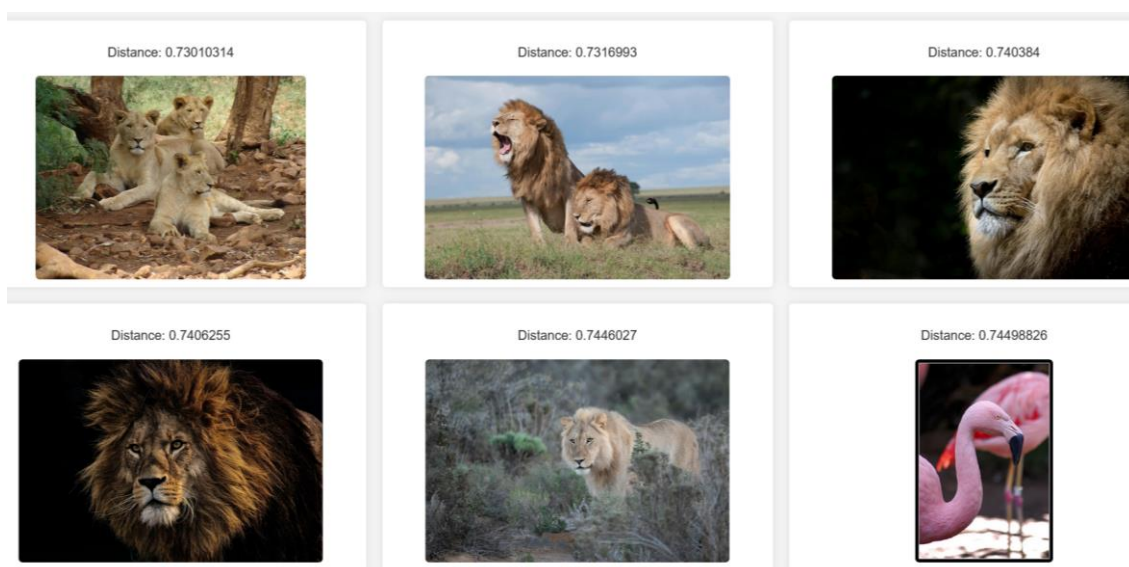
Na slici 38. prikazan je rezultat pretrage po tekstu. U *text box-u* unet je pojam „дивље животиње“, na ćirilici. Ovo prikazuje da CLIP model može vršiti pretragu na većini jezika. Prikazane su 6 najpribližnijih slika kao i njihova L1 distanca. Kod Weaviate baze podataka distanca ide u vrednostima od 0 do broj dimenzija gde 0 predstavlja identične vektore.



Slika 38. Prikaz rezultata pretraživanja po tekstu, L1 metrika sličnosti

Na slici 39. takođe je prikazan rezultat pretrage po tekstu pojma „дивље животиње“, ali kao metrika sličnosti izabrana je kosinusna sličnost. Kod Weaviate baze podataka, ova vrednost ide u vrednostima od 0 do 2, gde 0 pokazuje da se radi o identičnim vektorima, dok vrednost 2 pokazuje da se radi o suprotnim vektorima.

Možemo primetiti da rezultati pretrage nisu isti iako su i slike i upit isti. To nam govori da moramo pažljivo birati kako metriku sličnosti tako i indekse u zavisnosti od količine podataka koje se pretražuje kao i potrebne preciznosti. Tokom testiranja aplikacije ustanovljeno je da najpreciznije rezultate daje kosinusna sličnost a da kao najnepreciznije rezultate daje L2-Euklidska distanca.



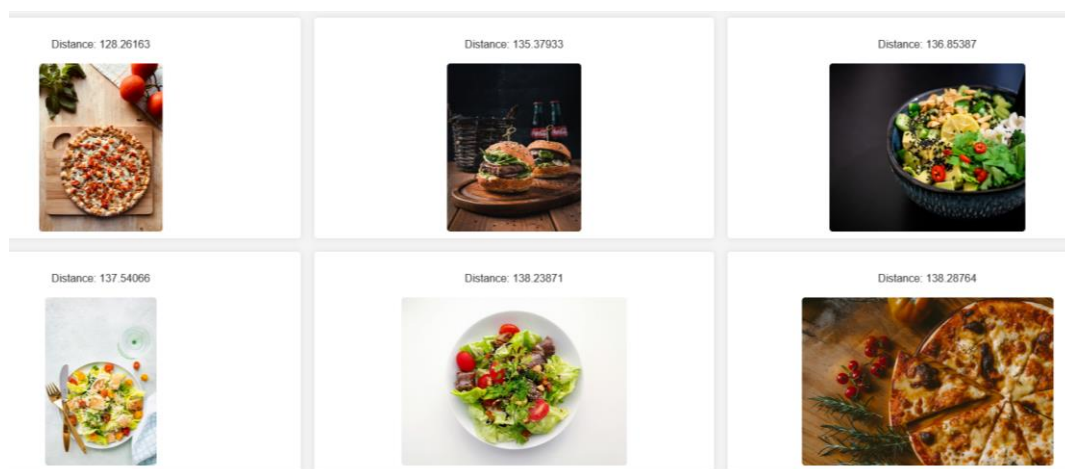
Slika 39. Prikaz rezultata pretraživanja po tekstu, kosinusna metrika sličnosti

U krajnjem delu ovog odeljka prikazana je i mogućnost pretrage sličnosti slike. Sliku po kojoj želimo da pretražimo našu vektorsku bazu podataka prikazana je na slici 40.



Slika 40. Burrito, meksička hrana

Na slici 41. prikazani su rezultati pretrage sličnosti i kao metrika sličnosti korišćena je Manhattan metrika.



Slika 41. Prikaz rezultata pretraživanja po slici, L1 metrika sličnosti

## 6. ZAKLJUČAK

U današnjem dobu, svedoci smo brze evolucije veštačke inteligencije koja preoblikuje svaki sektor industrije koju dotakne. Iako donosi obećavajuće mogućnosti automatizacije i inovacija, nije lišena izazova. U tom kontekstu, značaj vektorskih baza podataka u oblasti veštačke inteligencije i mašinskog učenja sve više dobija na značaju. Ove baze su od ključnog značaja za napredne pretraživače i sisteme preporuka proizvoda, doprinoseći povećanju preciznosti i unapređenju korisničkog iskustva putem algoritama mašinskog učenja koji razumeju korisničke potrebe.

Ovaj diplomski rad istakao je važnost optimalnog rukovanja podacima, kao i važnost implementacije vektorskih baza podataka i vektorskog pretraživanja u modernim informacionim sistemima.

U okviru drugog podglavlja objašnjeni su pojmovi podataka i informacije. Predstavljene su kategorije digitalnih podataka kao i primeri za svaku od njih.

U trećem podglavlju objašnjeni su pojmovi vektora i vektorskih podataka. Dat je tok rada vektorskih baza podataka. Detaljnije je obrađen proces vektorizacije nestruktuiranih podataka kao i indeksiranje, metrike sličnosti i filtriranje rezultata. Opisani su i algoritmi koji se koriste tokom pretraživanja.

U četvrtom podglavlju navedeni su primeri primene vektorskih baza podataka kao i detaljniji opis kompanija i način na koje ih implementiraju. Navedene su najkorišćenije vektorske baze podataka sa svojim prednostima i manama, dato je poređenje vektorskih baza podataka sa tradicionalnim relacionim bazama kao i sa ostalim NoSQL bazama.

U petom odeljku ovog diplomskog rada prikazana je implementacija aplikacije koja koristi Weaviate vektorsku bazu podataka kao i njen model multi2-clip koji se koristi za vektorizaciju. Prikazana je instalacija i podizanje baze koristeći docker compose funkcionalnosti, instalacija potrebnih paketa i modula. Na kraju odeljka prikazani su primeri rada aplikacije koristeći pretragu slika po tekstu kao i po sličnosti slika.

Sama aplikacija je manjeg obima, ali neosporno prikazuje primenu vektorskih baza podataka.

Kao proširenje aplikacije moguće je odraditi sledeće:

- Dodavanje više slika i klasa kako bi pretraga vraćala tačnije rezultate
- Ugradnja same aplikacije u realne sisteme kao sisteme preporuka
- Menjanjem metrike sličnosti može se doći do preciznijih rezultata
- Dodavanje metapodataka i uključivanje filtera u proces pretraživanja

## 7. LITERATURA

- [1] “Structured vs unstructured data: An overview,” MongoDB, <https://www.mongodb.com/unstructured-data/structured-vs-unstructured> (Pristupljeno Feb. 29, 2024).
- [2] C. Education, “Structured vs. unstructured data: What’s the difference?,” IBM Blog, <https://www.ibm.com/blog/structured-vs-unstructured-data/> (Pristupljeno Feb. 29, 2024).
- [3] Fivetran, “Structured vs. Unstructured Data: Business Benefits,” Fivetran, <https://www.fivetran.com/learn/structured-vs-unstructured-data> (Pristupljeno Feb. 29, 2024).
- [4] AltexSoft, “Structured vs unstructured data: What is the difference?,” AltexSoft, <https://www.altexsoft.com/blog/structured-unstructured-data/> (Pristupljeno Feb. 29, 2024).
- [5] What is a vector database?, <https://www.cloudflare.com/learning/ai/what-is-vector-database> (Pristupljeno Mar. 3, 2024).
- [6] “Meet Ai’s multitool: Vector embeddings | google cloud blog,” Google, <https://cloud.google.com/blog/topics/developers-practitioners/meet-ais-multitool-vector-embeddings> (Pristupljeno Mar. 3, 2024).
- [7] “Vector databases: Intro, use cases, top 5 vector DBS,” V7, <https://www.v7labs.com/blog/vector-databases> (Pristupljeno Mar. 3, 2024).
- [8] M. Ali, “The 5 best vector databases: A list with examples,” DataCamp, <https://www.datacamp.com/blog/the-top-5-vector-databases> (Pristupljeno Mar. 3, 2024).
- [9] R. Schwaber-Cohen, “What is a vector database & how does it work? use cases + examples,” Pinecone, <https://www.pinecone.io/learn/vector-database/> (Pristupljeno Mar. 3, 2024).
- [10] “What is a vector database?,” IBM, <https://www.ibm.com/topics/vector-database> (Pristupljeno Mar. 4, 2024).
- [11] “What is a vector database?: A comprehensive vector database guide,” Elastic, <https://www.elastic.co/what-is/vector-database> (Pristupljeno Mar. 4, 2024).
- [12] L. M. Advocate, “A gentle introduction to vector databases: Weaviate - Vector Database,” Weaviate Vector Database RSS, <https://weaviate.io/blog/what-is-a-vector-database> (Pristupljeno Mar. 4, 2024).
- [13] Woyera, “Pinecone vs. chroma: The pros and cons,” Medium, <https://medium.com/@woyera/pinecone-vs-chroma-the-pros-and-cons-2b0b7628f48f> (Pristupljeno Mar. 12, 2024).
- [14] A. Tyagi, “Vector database: How does it work & top 15 vector databases 2024,” Analytics Vidhya, <https://www.analyticsvidhya.com/blog/2023/12/top-vector-databases/> (Pristupljeno Mar. 12, 2024).
- [15] “Top 16 best vector databases for 2024: Detailed list,” lakeFS, <https://lakefs.io/blog/12-vector-databases-2023/> (Pristupljeno Mar. 13, 2024).
- [16] A. Cantarero, “What is a vector index? an introduction to vector indexing,” DataStax, <https://www.datastax.com/guides/what-is-a-vector-index#what-is-a-vector-index> (Pristupljeno Mar. 16, 2024).
- [17] N. Kanungo, “Vector indexing: A roadmap for vector databases,” Medium, <https://medium.com/kx-systems/vector-indexing-a-roadmap-for-vector-databases-65866f07daf5> (Pristupljeno Mar. 16, 2024).

- [18] “Locality sensitive hashing (LSH): The Illustrated Guide,” Pinecone, <https://www.pinecone.io/learn/series/faiss/locality-sensitive-hashing/> (Pristupljeno Mar. 16, 2024).
- [19] R. Schwaber-Cohen, “Vector similarity explained,” Pinecone, <https://www.pinecone.io/learn/vector-similarity/> (Pristupljeno Mar. 16, 2024).
- [20] A. Shmyga, “Vectors similarity. Jaccard similarity.,” Medium, <https://medium.com/@shmyga.87/vectors-similarity-jaccard-similarity-b67925895fd2> (Pristupljeno Mar. 16, 2024).
- [21] Edx, [https://courses.edx.org/asset-v1:Databricks+LLM101x+2T2023+type@asset+block@Module\\_2\\_slides.pdf](https://courses.edx.org/asset-v1:Databricks+LLM101x+2T2023+type@asset+block@Module_2_slides.pdf) (Pristupljeno Mar. 22, 2024).
- [22] J. Briggs, “The missing where clause in vector search,” Pinecone, <https://www.pinecone.io/learn/vector-search-filtering/> (Pristupljeno Mar. 22, 2024).
- [23] J. Solanki, “What is vector database? concepts and examples: Decube,” RSS, <https://www.decube.io/post/vector-database-concept> (Pristupljeno Mar. 22, 2024).
- [24] “Semantic search demo,” Qdrant, <https://demo.qdrant.tech/> (Pristupljeno Mar. 27, 2024).
- [25] Spotify Engineering, “Introducing voyager: Spotify’s new nearest-neighbor Search Library,” Spotify Engineering, <https://engineering.atspotify.com/2023/10/introducing-voyager-spotifys-new-nearest-neighbor-search-library/> (Pristupljeno Mar. 27, 2024).
- [26] L. Wang, “8 ways Google Lens can help make your life easier,” Google, <https://blog.google/products/google-lens/google-lens-features/> (Pristupljeno Mar. 27, 2024).
- [27] L. Wang, “New google lens features to help you be more productive at home,” Google, <https://blog.google/products/google-lens/new-google-lens-features-help-you-be-more-productive-home/> (Pristupljeno Mar. 28, 2024).
- [28] M. Younes, “Bring your data to life: Creating a chatbot with LLM, LangChain, vector DB (locally on Docker)...,” Medium, <https://medium.com/@mutazyounes/bring-your-data-to-life-creating-a-chatbot-with-llm-langchain-vector-db-locally-on-docker-ed647e546f85> (Pristupljeno Mar. 28, 2024).
- [29] “How to install Weaviate: Weaviate - Vector Database,” Weaviate Vector Database RSS, <https://weaviate.io/developers/weaviate/installation> (pristupljeno Apr. 1, 2024).
- [30] “Clip: Connecting text and images,” CLIP: Connecting text and images, <https://openai.com/research/clip> (Pristupljeno Apr. 25, 2024).