



UNIVERZITET U NIŠU  
ELEKTRONSKI FAKULTET  
Katedra za računarstvo



## **RETRIEVAL-AUGMENTED GENERATION (RAG) TEHNIKA ZA UNAPREĐENJE CHATBOT APLIKACIJE**

Završni rad

Studijski program: Elektrotehnika i računarstvo

Modul: Računarstvo i informatika

Kandidat:

Božidar Mitić, br. indeksa: 18282

Mentor:

Prof. dr Aleksandar Stanimirović

Niš, Septembar 2024. godina

Univerzitet u Nišu  
Elektronski Fakultet

**RETRIEVAL-AUGMENTED GENERATION (RAG) TEHNIKA ZA UNAPREĐENJE CHATBOT  
APLIKACIJE**

**ENHANCING CHATBOT APPLICATIONS USING RETRIEVAL-AUGMENTED GENERATION  
(RAG) TECHNIQUE**

Završni rad

Studijski program: Elektrotehnika i računarstvo

Modul: Računastvo i informatika

Student: Božidar Mitić, br. ind. 18282

Mentor: Prof. dr Aleksandar Stanimirović

*Zadatak: Upoznati se sa konceptom i osnovnim pojmovima vezanim za retrieval-augmented generation (RAG) tehniku. Detaljno proučiti mogućnosti primene RAG tehnike za unapređenje rada chatbot aplikacija baziranih na velikim jezičkim modelima. U praktičnom delu rada, korišćenjem velikih jezičkih modela i RAG tehnike, implementirati prototip chatbot aplikacije koja pruža informacija o pravilima za igranje različitih društvenih igara.*

Datum prijave rada: \_\_\_\_\_

Datum predaje rada: \_\_\_\_\_

Datum odbrane rada: \_\_\_\_\_

Komisija za ocenu i odbranu:

1. \_\_\_\_\_

2. \_\_\_\_\_

3. \_\_\_\_\_

## Sadržaj

1	Uvod.....	5
2	Generativna veštačka inteligencija i veliki jezički modeli.....	6
2.1	Generativna veštačka inteligencija.....	6
2.2	Veliki jezički modeli .....	6
2.3	Primena velikih jezičkih modela.....	8
3	Retrieval-augmented generation (RAG) .....	9
3.1	Komponente .....	10
3.1.1	Unapred obučeni model (Pre-trained model) .....	10
3.1.2	Skladištenje podataka (Data Storage) .....	10
3.1.3	Mehanizam za pronalaženje (Retrieval mechanism) .....	10
3.1.4	Popularne primene RAG-a u oblasti veštačke inteligencije.....	12
3.1.5	Prednosti i nedostaci .....	13
3.2	Osnovni RAG sistem .....	14
3.2.1	Indeksiranje i embedding (Indexing and Embedding) .....	14
3.2.2	Pretraga (Retrieval).....	15
3.2.3	Generisanje (Generation) .....	16
3.3	Napredni RAG Sistem .....	17
3.3.1	Prevođenje upita (Query Translation).....	17
3.3.2	Rutiranje (Routing) .....	22
3.3.3	Kreiranje upita (Query Construction) .....	23
3.3.4	Indeksiranje (Indexing) .....	24
3.3.5	CRAG .....	27
3.3.6	RAG vs Fine-tuning (Fino podešavanje) .....	29
3.3.7	Budućnost RAG-a .....	30
4	Implementacija RAG sistema sa Q&A Chatbot-om za pravila društvenih igara.....	31
4.1	Izazovi projekta.....	31
4.1.1	Potreba za tačnim informacijama.....	32
4.2	Arhitektura chatbot aplikacije za pravila društvenih igara.....	32
4.2.1	Flask.....	33
4.2.2	LangChain.....	33
4.2.3	Pinecone .....	34
4.3	Implementacija ključnih funkcionalnosti.....	34

4.3.1	Učitavanje dokumenata.....	34
4.3.2	Inicijalizacija baze.....	36
4.3.3	Kreiranje i čuvanje vektora .....	37
4.3.4	Pretraga i generisanje (Retrieval and generation) .....	39
4.4	Primer rada aplikacije .....	43
5	Zaključak.....	46
6	Literatura.....	47

# 1 Uvod

Sa brzim razvojem tehnologije, količina informacija kojima su korisnici svakodnevno izloženi postaje sve veća i kompleksnija. Ovaj eksponencijalni rast informacija predstavlja značajan izazov za obradu i ekstrakciju korisnog sadržaja iz velikih skupova podataka. Tradicionalne metode pretrage i obrade često nisu dovoljne da pruže relevantne i precizne informacije u realnom vremenu, što otvara prostor za primenu naprednijih tehnika kao što je Retrieval-Augmented Generation (RAG).

RAG predstavlja kombinaciju tehnika tradicionalnih sistema za pretragu informacija (baze podataka) sa mogućnostima generativnih velikih jezičkih modela (LLMs). Zbog svoje fleksibilnosti i moći da obrađuje raznovrsne informacije, RAG se primenjuje u mnogim oblastima, uključujući, ali ne ograničavajući se na, chatbot aplikacije, automatizovanu analizu dokumenata, generisanje odgovora na složena pitanja, pa čak i personalizovane preporuke.

Cilj ovog rada je da istraži i prikaže potencijal RAG tehnike, s posebnim osvrtom na njenu primenu u unapređenju funkcionalnosti chatbot aplikacija. Posebna pažnja biće posvećena kako teoretskom, tako i praktičnom aspektu integracije RAG sistema, uključujući analizu njegovog uticaja na efikasnost, tačnost, i skalabilnost chatbot aplikacija. Pored toga, istražiće se i mogućnosti daljeg razvoja i prilagođavanja ove tehnike za specifične industrijske primene.

Drugo poglavlje obuhvata upoznavanje sa generativnom veštačkom inteligencijom i velikim jezičkim modelima koji spadaju u najvažnije komponente svakog RAG sistema, kao njihovim primenama.

U trećem poglavlju rada, akcenat će biti stavljen na komponente RAG sistema, uključujući unapred obučene modele, skladištenje podataka, i mehanizme za pronalaženje relevantnih informacija. Posebno će biti obrađene popularne primene RAG tehnike u veštačkoj inteligenciji, s naglaskom na prednosti koje ona donosi u poređenju sa tradicionalnim pristupima.

Četvrto poglavlje biće posvećeno praktičnoj implementaciji, gde će biti detaljno objašnjena integracija RAG sistema u chatbot aplikaciju. Prikazaće se koraci u indeksiranju podataka, njihovoj pretrazi i generisanju odgovora. Cilj ovog dela je da prikaže kako se RAG može koristiti za poboljšanje funkcionalnosti chatbot sistema kroz konkretne primere.

## 2 Generativna veštačka inteligencija i veliki jezički modeli

Za razumevanje načina funkcionisanja RAG sistema, neophodno je razumeti pojmove i osnove funkcionisanja generativne veštačke inteligencije (Generative Artificial Intelligence - GenAI) i velikih jezičkih modela (Large Language Model - LLM)

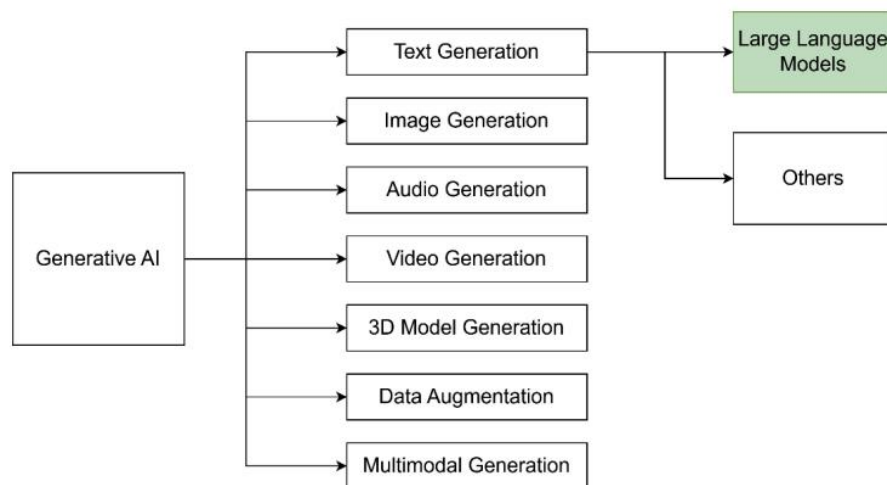
### 2.1 Generativna veštačka inteligencija

GenAI predstavlja sistem koji ima mogućnost generisanja novog sadržaja kao što su tekst, slike, audio i video. Za razliku od specijalizovanih AI sistema koji su dizajnirani za specifične zadatke poput klasifikacije slika ili prepoznavanja govora, GenAI modeli mogu kreirati potpuno nove sadržaje koje je često vrlo teško, a možda i nemoguće razlikovati od sadržaja koji je stvorio čovek.

Ovi sistemi koriste tehnike mašinskog učenja kao što su neuronske mreže koje se obučavaju na ogromnim količinama podataka. Generativni modeli mogu modelovati verovatnoću distribucije podataka i semplovati tu distribuciju za generisanje novih primera. To rade tako što uče obrasce i strukturu u okviru podataka za obuku [35].

### 2.2 Veliki jezički modeli

Jedna od najbrže rastućih grana u oblasti GenAI-a je generisanje prirodnih jezika pomoću velikih jezičkih modela. Ovde spadaju tehnike kao što su Natural Language Processing (NLP) i Natural Language Understanding (NLU)

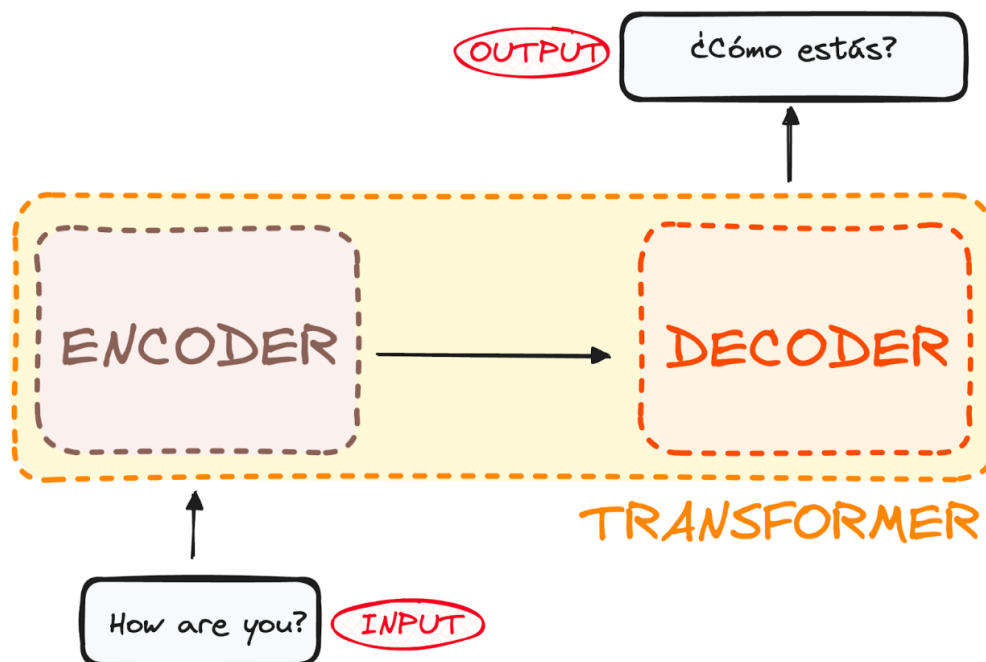


Slika 1. Podela GenAI-a [35]

Veliki jezički modeli predstavljaju deo generativne veštačke inteligencije, koji su specifično dizajnirani za razumevanje i generisanje ljudskog jezika [5]. To se može videti na slici 1. Oni funkcionišu kao neuronske mreže obučene na ogromnim količinama tekstualnih podataka (članci, knjige, transkripti video zapisa, itd.), što je razlog za reč „veliki“ u njihovom imenu. Za razliku od sistema za generisanje teksta zasnovanih na pravilima, ono što ove modele čini posebnim je njihova sposobnost da stvore potpuno nov, originalan tekst koji deluje prirodno [35].

Obučavanjem velikih jezičkih modela na velikim količinama podataka nekog jezika, ovi modeli stižu duboko razumevanje strukture i karakteristika određenog jezika, kao što su sintaksa i kompleksna gramatika, razumevanje idioma, semantika, kulturne reference i slično. Nakon toga, oni su u stanju da vrše rezimiranje tekstova, a čak i generisanje potpuno novih tekstova pretpostavljajući sledeću reč u rečenici na osnovu sadržaja sa kojim su upoznati [35].

U jednom trenutku uvodi se pojam **pažnje (attention)**, prvi put viđen u [3]. Ideja je inspirisana ljudskom sposobnošću selektivnog fokusiranja na određene delove informacija, dok se ostale zanemaruju. Cilj je da se na neki način ova jako specifična ljudska osobina izimitira i integriše u domenu velikih jezičkih modela, omogućavajući im mnogo viši nivo razumevanja i generisanja jezika. Na ovaj način su nastali **transformeri**, koji se sastoje od kodera i dekodera (Slika 2) i koriste koncept pažnje i na taj način uče zavisnostni na velikim udaljenostima u tekstu, a ne samo zavisnosti reči koje su blizu jedna do druge. Zbog toga modeli transformera imaju sposobnost razumevanja, ne samo pojedinačnih reči, već i celokupnog konteksta u kojem se te reči nalaze [35]. Transformer model obrađuje podatke tako što prvo tokenizuje ulazne podatke, a zatim istovremeno izvodi matematičke operacije kako bi otkrio odnose između tokena [4], [6]. Ovaj pristup omogućava modelu da prepozna obrasce u podacima na način sličan onome kako bi to učinio čovek



Slika 2. Arhitektura transformera [4]

## 2.3 Primena velikih jezičkih modela

Veštačka inteligencija postaje sve prisutnija u ljudskim životima, donoseći sa sobom promene u različitim aspektima društva i industrije. Kako se njene mogućnosti šire, generativna veštačka inteligencija i veliki jezički modeli pronalaze primenu u mnogim oblastima. Ovi modeli omogućavaju automatizaciju složenih zadataka, stvaranje novog sadržaja, kao i bolje razumevanje i interpretaciju ljudskog jezika. Štaviše, oni doprinose pristupačnosti pomažući osobama sa invaliditetom, putem aplikacija za pretvaranje teksta u govor i generisanje sadržaja u pristupačnim formatima. Od zdravstva do finansija, veliki jezički modeli transformišu industrije ubrzavajući procese, poboljšavajući korisničko iskustvo i omogućavajući efikasnije i na podacima zasnovano donošenje odluka.

Primene velikih jezičkih modela [1]:

- **Generisanje teksta:** Sposobnost generisanja jezika, poput pisanja e-mejllova, blogova ili drugog sadržaja kao odgovor na zadate upite, koji se zatim mogu dodatno usavršavati. Odličan primer za usavršavanje je retrieval-augmented generation (RAG), o čemu će biti reči kasnije.
- **Sažimanje sadržaja:** Sažimanje dugih članaka, priča u novinama, istraživačkih izveštaja, korporativne dokumentacije, u sažete tekstove prilagođene dužini izlaznog formata.
- **AI asistenti:** Chatbot-ovi koji odgovaraju na korisnička pitanja, obavljaju zadatke u pozadini i pružaju detaljne informacije na prirodnom jeziku kao deo integrisanog rešenja za korisničku podršku.
- **Generisanje koda:** Pomaže programerima u izradi aplikacija, pronalaženju grešaka u kodu i otkrivanju bezbednosnih problema u više programskih jezika, pa čak i „prevođenju“ između njih.
- **Analiza osećanja:** Analiza teksta kako bi se utvrdio ton korisnika u cilju razumevanja povratnih informacija.
- **Prevođenje jezika:** Pruža širu pokrivenost organizacijama širom sveta sa tečnim prevodima i višejezičnim mogućnostima.

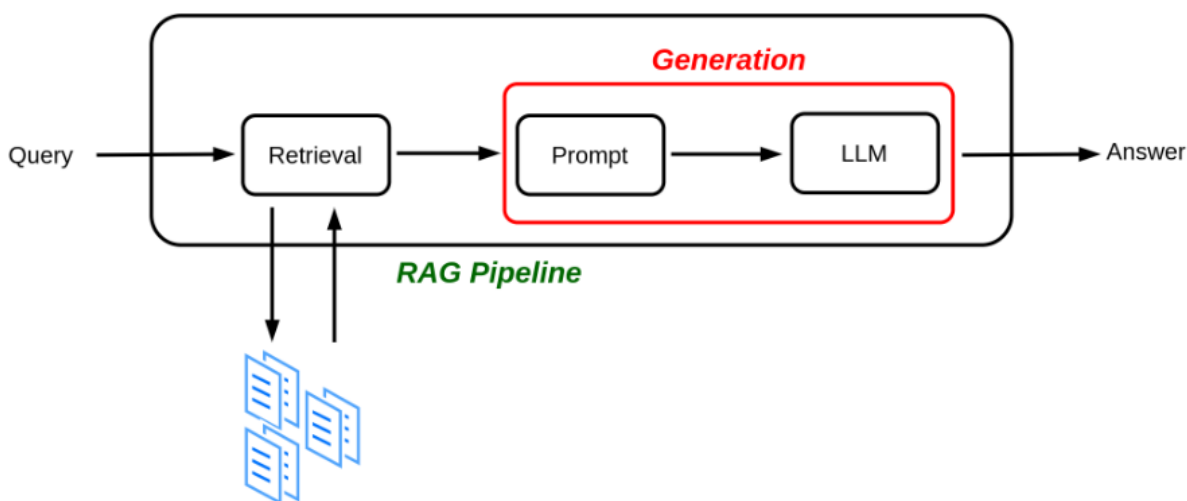
Uskoro će veliki jezički modeli značajno uticati na razne industrije, od finansija i osiguranja, preko ljudskih resursa, pa sve do zdravstva i šire, automatizujući korisničku samouslugu, ubrzavajući vreme odgovora na sve veći broj zadataka, kao i pružajući veću preciznost, poboljšano usmeravanje i inteligentno prikupljanje konteksta.



### 3 Retrieval-augmented generation (RAG)

Kada se govori o problemima velikih jezičkih modela, važno je istaći njihove nedostatke, kao što su nemogućnost dinamičkog ažuriranja ili pristupanja spoljnim informacijama, rizik od generisanja netačnih informacija ili „halucinacija” i nedostatak transparentnosti u načinu na koji dolaze do određenih zaključaka. Ovi modeli, iako moćni, inherentno su statični, oslanjaju se isključivo na podatke na kojima su obučeni, a oni mogu postati zastareli ili nedovoljno relevantni za određene zadatke. Obučavanje velikih jezičkih modela je veoma zahtevno, kako u pogledu vremena, tako i resursa, što onemogućava njihovo konstantno ažuriranje. U međuvremenu, kako tehnologija napreduje, količina informacija na internetu eksponencijalno raste, što dodatno otežava zadatak održavanja modela aktuelnim. Ovi izazovi su podstakli potragu za inovativnim rešenjem koje može prevazići ova ograničenja.

Tu na scenu stupa Retrieval-Augmented Generation (RAG). RAG pokušava da reši ove probleme kombinovanjem generativnih sposobnosti velikih jezičkih modela sa spoljnim sistemom za pretragu koji može pristupiti i uključiti najrelevantnije, najnovije informacije iz velike baze podataka ili izvora znanja, što je prikazano na slici 3. Na taj način, RAG poboljšava tačnost, relevantnost i činjeničnost generisanog teksta, što ga čini posebno korisnim za zadatke koji zahtevaju specifične, aktuelne ili detaljne informacije. Ovaj hibridni pristup ne samo da poboljšava kvalitet generisanog sadržaja, već omogućava i transparentnije i prilagodljivije AI sisteme.



Slika 3. Osnovni način korišćenja RAG-a [2]

RAG se prvi put javlja u [36]. Ovaj rad predstavlja prekretnicu u razvoju tehnika za kombinovanje pretrage i generisanja teksta. U njemu autori uvode RAG kao pristup koji kombinuje moć velikih jezičkih modela sa informacijama iz spoljnjih izvora, poput Vikipedije, kako bi se poboljšala tačnost i relevantnost generisanih odgovora u zadacima koji zahtevaju obimno znanje.

## 3.1 Komponente

RAG sadrži nekoliko glavnih komponenti: unapred obučeni model (pre-trained model), mehanizam za pronalaženje (retrieval mechanism) i skladištenje podataka (data storage). U pozadini, arhitektura je nešto kompleksnija, ali je neophodno prvo razumeti funkciju glavnih komponenti pre nego što se pređe na detaljnije procese i dodatne elemente.

### 3.1.1 Unapred obučeni model (Pre-trained model)

Unapred obučeni model je ključni deo RAG arhitekture. Ovaj model, koji obrađuje sekvence teksta, obučeni su na velikim količinama podataka. Njegova glavna uloga je da generiše tekst na osnovu unosa i informacija koje dobija putem mehanizma za pretragu. Korišćenjem naučenih parametara, model kreira smislen i kontekstualno relevantan odgovor. U kontekstu RAG-a, unapred obučeni model kombinuje informacije iz više izvora, generišući odgovore koji su precizniji i bogatiji informacijama nego kada bi se oslanjao samo na svoje interne parametre.

### 3.1.2 Skladištenje podataka (Data Storage)

Skladištenje podataka predstavlja ključnu komponentu koja omogućava čuvanje i brz pristup velikim količinama informacija koje RAG model koristi za generisanje odgovora. Baze podataka za dokumenta (Document Databases) služe kao veliki i pristupačni sistemi za skladištenje eksternih skupova podataka ili dokumenata iz kojih RAG model preuzima informacije. Ove baze podataka moraju biti dovoljno velike i dobro organizovane da bi omogućile efikasan pristup potrebnim informacijama.

Skalabilna rešenja za skladištenje su takođe od suštinskog značaja, posebno kada se radi o ogromnim skupovima podataka. Cloud storage i distribuirani fajl sistemi često se koriste za ovu svrhu, omogućavajući sistemu lako skaliranje u skladu sa rastućim zahtevima za skladištenjem i obradom podataka. Pored toga, ovakva rešenja omogućavaju visok nivo dostupnosti i otpornosti na kvarove, čime se osigurava kontinuitet u radu modela [30].

### 3.1.3 Mehanizam za pronalaženje (Retrieval mechanism)

Mehanizam za pronalaženje je komponenta koja se koristi za pretragu i dohvatanje informacija iz različitih vrsta baza podataka, bilo da je reč o vektorskim, SQL, graf ili dokumentnim bazama. Kada se postavi upit, ovaj mehanizam ga obrađuje i koristi odgovarajuću tehniku pretrage, bilo da je to pretraga po ključnim rečima, relacijama među podacima, ili vektorska pretraga u slučaju numeričkih reprezentacija. Odabrani rezultati predstavljaju delove podataka ili dokumente koji su najrelevantniji za postavljeni upit. Mehanizam za pronalaženje omogućava pronalazak najaktuelnijih i najpreciznijih informacija, koje se zatim kombinuju sa znanjem unapred obučenog modela, čime se poboljšava celokupni izlaz generisan od strane RAG-a [29].

### 3.1.3.1 Klasične tehnike pronalaženja informacija

Neke od klasičnih tehnika za pronalaženje informacija [29]:

- **Frekvencija termina-inverzna frekvencija dokumenta (Term Frequency-Inverse Document Frequency (TF-IDF)):** Ova tehnika se široko koristi za procenu relevantnosti dokumenta u odnosu na postavljeni upit. TF-IDF izračunava značaj svakog termina u dokumentu na osnovu njegove učestanosti u samom dokumentu i retkosti kroz sve dokumente. Termin sa najvišim TF-IDF skorom smatra se najrelevantnijim.
- **Torba reči (Bag of Words – BoW):** Ova tehnika predstavlja dokument kao torbu, odnosno skup pojedinačnih reči. Za svaku reč se pamti broj ponavljanja i nije bitan njihov redosled. BoW je korisna za obradu upita sa greškama u pisanju i smanjenje uticaja reči poput „i“, „ali“, „the“, itd.
- **Dodeljivanje težina terminima (Term weighting):** Ova tehnika dodeljuje težine svakom terminu u dokumentu na osnovu njegove relevantnosti za upit. Težine se koriste za izračunavanje ukupnog skora dokumenta, koji se zatim poredi sa pragom relevantnosti kako bi se odredila njegova važnost.

### 3.1.3.2 Savremene tehnike pronalaženja informacija

Sa razvojem tehnologije, došlo je i do razvoja tehnika pronalaženja informacija. Neke od savremenijih tehnika su [29]:

- **BM25:** Ova tehnika je proširenje TF-IDF metode koja uzima u obzir dužinu dokumenta i broj termina koje on sadrži. BM25 dodeljuje više skorove dokumentima sa većom dužinom i više termina, jer se smatra da su oni informativniji.
- **Pristupi zasnovani na neuronskim mrežama:** Ove tehnike koriste neuronske mreže za učenje složenih obrazaca u matrici upit-dokument. Neuronske mreže mogu naučiti da izvlače relevantne karakteristike iz vektora dokumenata i upita, poput dužine rečenica, oznaka vrsta reči i imenovanih entiteta.
- **Modeli zasnovani na dubokom učenju:** Ovi modeli koriste tehnike dubokog učenja, poput konvolucionih neuronskih mreža (CNN) i rekurentnih neuronskih mreža (RNN), za učenje složenih obrazaca u matrici upit-dokument. Modeli dubokog učenja mogu naučiti da izvlače relevantne karakteristike iz vektora dokumenata i upita, kao što su vektorski prikaz reči (word embeddings), vektorski prikaz rečenica (sentence embeddings) i latentne semantičke strukture.

### 3.1.4 Popularne primene RAG-a u oblasti veštačke inteligencije

Pre nego što se detaljno objasni kako funkcioniše RAG sistem, biće reči o njegovim popularnim primenama u oblasti veštačke inteligencije. Sa sve širim razvojem veštačke inteligencije, naročito velikih jezičkih modela, u različitim industrijama i oblastima, RAG se pojavio kao ključna inovacija koja dodatno unapređuje mnoge od tih primena. U nastavku će biti predstavljene neke od tih najpopularnijih oblasti u kojima je RAG doneo značajne koristi [29].

1. **Kreativno pisanje:** RAG može biti korišćen za generisanje kreativnog sadržaja poput priča, pesama i članaka. Kroz analizu postojećih tekstova, jezički model uči obrasce i strukture jezika, omogućavajući mu da kreira originalan i zanimljiv sadržaj, često neprimetno različit od onog koji pišu ljudi.
2. **Dizajn proizvoda:** RAG se može primeniti u dizajnu inovativnih proizvoda generisanjem novih ideja na osnovu postojećih dizajna proizvoda ili stvaranjem potpuno novih konceptata. Na primer, jezički model može analizirati uspešne dizajne i generisati nove sa unapređenom funkcionalnošću ili estetikom.
3. **Medicinska dijagnoza:** RAG može pomagati lekarima u postavljanju medicinskih dijagnoza generisanjem mogućih dijagnoza na osnovu simptoma i medicinske istorije pacijenata. Jezički model analizira velike količine medicinskih podataka i identifikuje obrasce i odnose koje ljudski lekar može prevideti, čime doprinosi tačnijim dijagnozama i tretmanima.
4. **Prevođenje jezika:** RAG može biti korišćen za generisanje prevoda tekstova, govora ili čak celih knjiga. Kroz analizu postojećih prevoda, jezički model uči obrasce i nijanse jezika, omogućavajući mu da proizvodi tačne i prirodne prevode.
5. **Virtuelni asistenti:** RAG se može koristiti za kreiranje virtuelnih asistenata koji mogu komunicirati s korisnicima, odgovarati na pitanja ili čak izvršavati zadatke. Jezički model analizira velike količine podataka i uči kako da odgovara korisnicima na način sličan ljudima, omogućavajući mu da pruži personalizovana iskustva i poboljša interakcije sa korisnicima.
6. **Razvoj igara:** RAG se može primeniti na generisanje nivoa u igrama, likova ili čak celih igara. Jezički model analizira postojeće igre i uči obrasce i strukture dizajna igara, omogućavajući mu da kreira nove i inovativne igre koje često ne zaostaju za onima koje kreiraju ljudi.
7. **Arhitektonski dizajn:** RAG se može koristiti za generisanje arhitektonskih dizajna na osnovu postojećih zgrada ili čitavih gradova. Jezički model analizira velike količine arhitektonskih podataka i uči obrasce i odnose između različitih dizajnerskih elemenata, omogućavajući mu da kreira inovativne i funkcionalne dizajne.
8. **Modni dizajn:** RAG se može primeniti na generisanje modnih dizajna na osnovu postojećih stilova ili čak čitavih kolekcija. Jezički model analizira velike količine modnih podataka i uči obrasce i odnose između različitih dizajnerskih elemenata, omogućavajući mu da kreira inovativne i stilizovane dizajne.
9. **Edukativni sadržaj:** RAG može biti korišćen za generisanje edukativnog sadržaja kao što su lekcije, kvizovi ili čak celi kursevi. Jezički model analizira postojeće obrazovne materijale i uči obrasce i strukture učenja, omogućavajući mu da kreira kvalitetan edukativni sadržaj prilagođen potrebama pojedinih učenika.

### 3.1.5 Prednosti i nedostaci

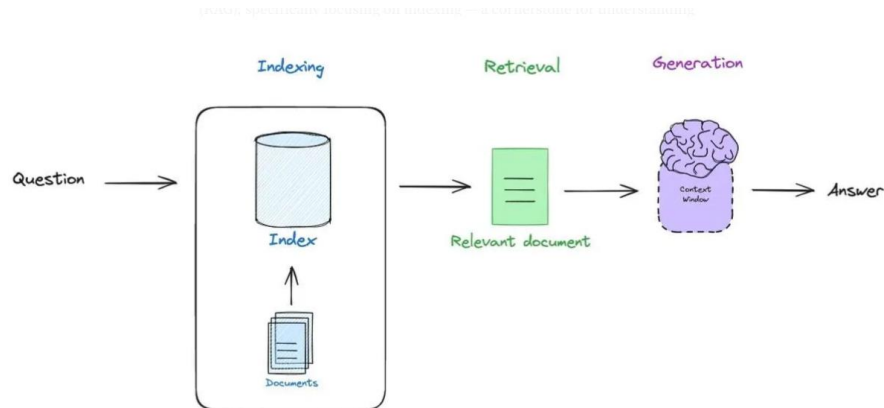
RAG prevazilazi ograničenja tradicionalnih jezičkih modela koji se oslanjaju isključivo na podatke na kojima su obučeni i time omogućava korišćenje aktuelnih informacija i šireg spektra izvora podataka, što donosi niz prednosti [30]:

1. **Osiguravanje relevantnosti uz ažurirane informacije:** RAG omogućava velikim jezičkim modelima pristup najnovijim dostupnim informacijama, što je od ključne važnosti za teme koje zavise od vremenski osetljivih podataka. Ovo osigurava da generisani sadržaj ostane relevantan i odražava najnovije događaje i podatke.
2. **Povećanje preciznosti i tačnosti odgovora:** Integrisanjem spoljnih izvora informacija, RAG značajno poboljšava tačnost i nivo detalja u odgovorima generisanim od strane velikih jezičkih modela. Ovo poboljšanje je naročito uočljivo kod odgovora na činjenična pitanja, gde su preciznost i tačnost od najveće važnosti.
3. **Smanjenje pristrasnosti kroz eksterno preuzimanje podataka:** Jedna od inovativnih primena RAG-a je njegova sposobnost da se suoči sa pristrasnostima koje mogu postojati u podacima nad kojim su obučeni jezički modeli. Preuzimanjem i integrisanjem informacija iz različitih eksternih izvora, RAG uvodi širu perspektivu koja pomaže u smanjenju pristrasnosti u rezultatima modela.
4. **Poboljšano kontekstualno razumevanje:** RAG-ova komponenta za pretragu ističe se u prepoznavanju i pristupanju informacijama specifičnim za korisnički upit. Ova sposobnost omogućava generisanje odgovora koji su ne samo tačni, već i kontekstualno prilagođeni potrebama korisnika, pružajući personalizovanije i relevantnije iskustvo.
5. **Prevazilaženje ograničenja unapred treniranih modela:** Tradicionalni jezički modeli često se suočavaju sa izazovom ograničenog znanja zasnovanog samo na podacima na kojima su obučeni. RAG rešava ovaj problem pristupanjem i integrisanjem dodatnih eksternih informacija, proširujući bazu znanja modela i njihovu primenljivost.

RAG donosi brojne prednosti, ali ima i svoje nedostatke. Jedan od glavnih problema je povećana latencija, jer sistem mora da pretraži spoljne izvore informacija pre nego što generiše odgovor, što može usporiti proces. Pored toga, RAG zavisi od kvaliteta i ažurnosti tih eksternih izvora, pa ako su podaci zastareli ili neadekvatni, to može negativno uticati na tačnost rezultata. Takođe, implementacija RAG sistema zahteva dodatne resurse, jer je potrebno održavati bazu podataka i razviti efikasne strategije pretrage, što sve može povećati kompleksnost u odnosu na tradicionalne modele koji ne koriste pretragu.

## 3.2 Osnovni RAG sistem

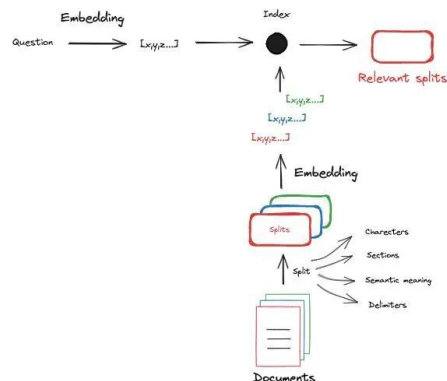
Na slici 5 prikazan je osnovni proces RAG sistema, gde se postavljeno pitanje pretražuje kroz indeksirane dokumente, relevantan dokument se pronalazi, a potom se generiše odgovor koristeći pribavljen kontekst.



Slika 5. Osnovni RAG sistem [37]

### 3.2.1 Indeksiranje i embedding (Indexing and Embedding)

Zamišlja se osoba u ogromnoj biblioteci sa milionima knjiga raspoređenih po nebrojenim policama. Njen zadatak je da pronade informacije o vrlo specifičnoj temi. Kako bi to uradila? Mogla bi lutati kroz hodnike, knjigu po knjigu, ali to bi bilo neefikasno i oduzelo bi mnogo vremena. Ovde na scenu nastupa indeksiranje. Indeksiranje u veštačkoj inteligenciji omogućava organizaciju i brzo preuzimanje informacija iz velikih skupova digitalnih podataka i koristi se kada se radi sa vektorskim bazama podataka [9].



Slika 6. Učitavanje, indeksiranje i embedding [37]

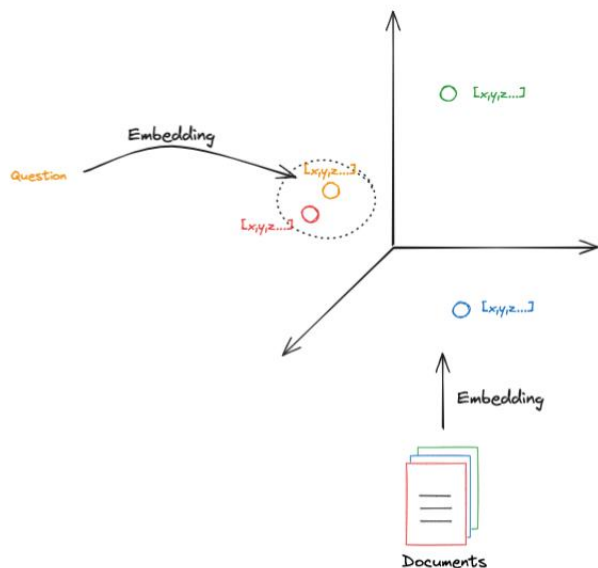
Proces započinje učitavanjem podataka iz različitih izvora, što omogućava kreiranje pretražive baze podataka. Embedding predstavlja prevođenje ovih dokumenata u format koji omogućava računaru brzu pretragu. Ovo najčešće obavljaju veoma kompleksni embedding modeli. Dokumenti se transformišu u numeričke reprezentacije, odnosno vektore. Proces učitavanja, indeksiranja i embedding-a može se videti na slici 6. Rane metode, poput Bag of Words, koristile su retke vektore zasnovane na čuvanju broja ponavljanja reči u dokumentu, dok moderni modeli mašinskog učenja stvaraju guste vektore koji hvataju suštinu teksta. Indeksiranje uključuje deljenje dokumenata na manje delove, njihovu konverziju u vektore (embedding), a zatim organizaciju tih dokumenata na način koji omogućava brz pristup [9].

### 3.2.2 Pretraga (Retrieval)

Kada se govori o pretrazi, misli se na proces upotrebe mehanizma za pretragu u cilju pronalaska odgovarajućih dokumenata ili delova dokumenata koji odgovaraju korisničkom upitu. Kao što je već rečeno, pretragu je moguće obaviti nad različitim vrstama baza podataka, ali ovde će fokus biti na najčešći slučaj kada je u pitanju RAG, a to su vektorske baze podataka.

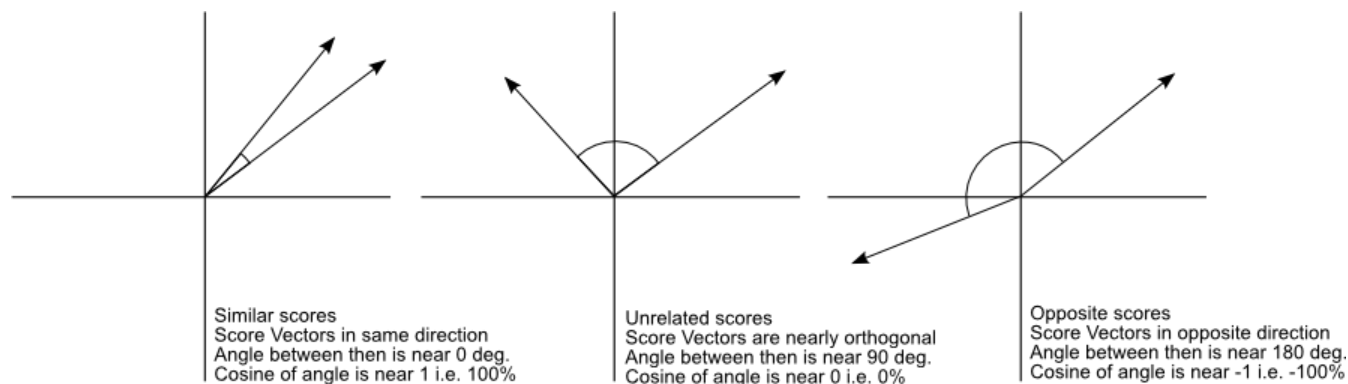
Kod vektorskih baza podataka, kao što se dokumenti prevode u vektore, isto tako je neophodno i korisnički **upit(query)** prevesti u vektor da bi pretraga bila moguća. Vektori se upoređuju pomoću procesa koji se zove **provera sličnosti (similarity check)**. Provera sličnosti omogućava pribavljanje sličnih dokumenata na osnovu semantičkog značenja [23].

Ukoliko je svaki dokument predstavljen preko trodimenzionalnog vektora, onda oni zauzimaju neku poziciju u trodimenzionalnom prostoru (koordinatnom sistemu). Dokumenti koji su slični na osnovu semantičkog značenja nalaze se međusobno blizu u ovom prostoru. Vrš se pribavljanje jednog ili više dokumenata čiji su vektori bliski sa vektorom upita.



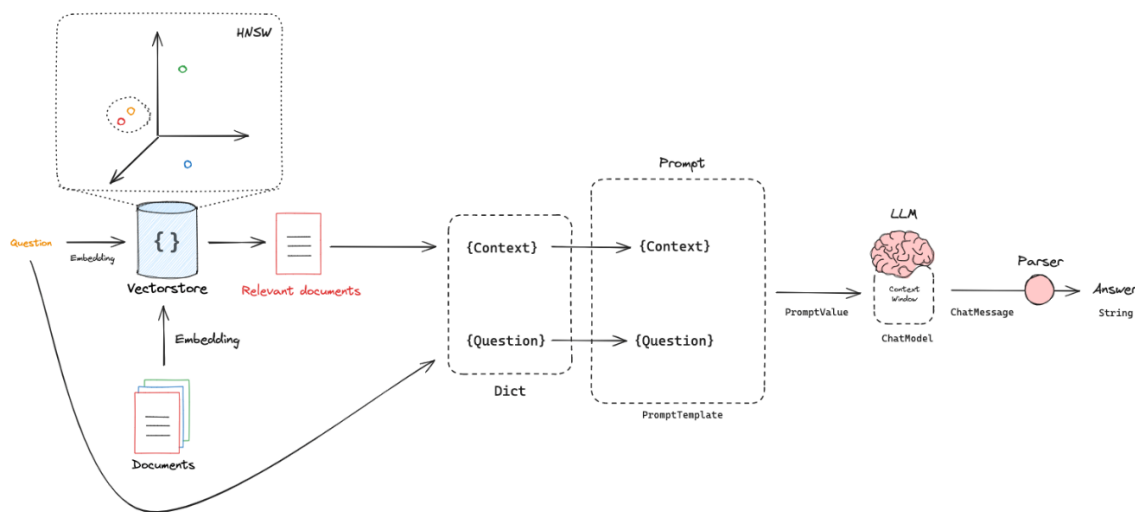
Slika 7. Pretraga (Retrieval) pomoću provere sličnosti [37]

Jedna od najčešće korišćenih metoda za poređenje vektora je **kosinusna sličnost (cosine similarity)** [13]. U najprostijem obliku, meri se kosinus ugla između dva vektora, pri čemu vrednost može da se kreće od -1 do 1. Vrednost 1 označava da su vektori potpuno slični i da imaju identične smerove, dok vrednost 0 označava da su vektori ortogonalni, što znači da između njih ne postoji sličnost. Vrednost -1 bi značila da su vektori potpuno suprotni u smislu smerova. Sve ovo prikazano je na slici 8, gde su za primer uzeti dvodimenzionalni vektori.



Slika 8. Kosinusna sličnost za dvodimenzionalne vektore [13]

### 3.2.3 Generisanje (Generation)



Slika 9. Slanje prompt-a velikom jezičkom modelu [38]

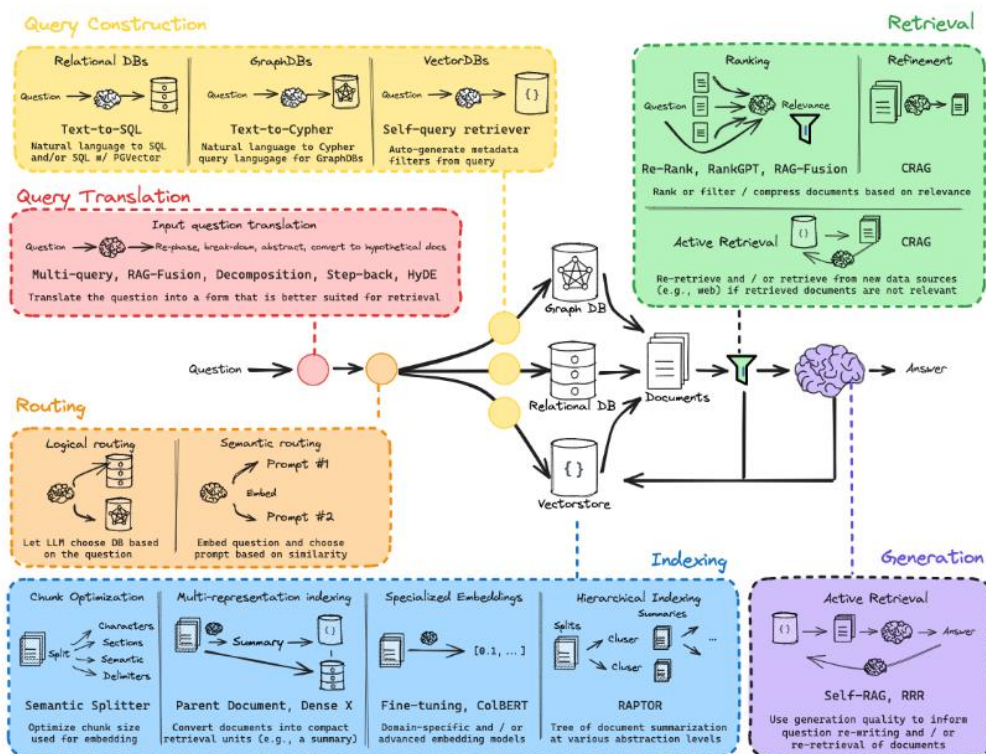
**Generisanje** je poslednji korak osnovnog RAG sistema. U ovom koraku se formira **prompt**, koji uglavnom sadrži dva ključna elementa: **kontekst** i **pitanje**. Kao kontekst prosleđuju se relevantne informacije koje su preuzete iz baze podataka u prethodnom koraku, dok se kao pitanje prosleđuje originalni korisnički upit. Prompt se pažljivo oblikuje da bi model bolje razumeo šta korisnik traži i u kom formatu bi odgovor trebalo da bude. Na osnovu ovog prompt-a, veliki



jezički model generiše konačan odgovor (Slika 9). Model koristi kako informacije iz prompta, tako i svoje prethodno naučeno znanje, kako bi kreirao najrelevantniji i tačan odgovor. Ovaj proces može da proizvede različite formate odgovora, od kratkih, direktnih rešenja, do opširnih sažetaka, u zavisnosti od korisničkog upita.

### 3.3 Napredni RAG Sistem

Za rešavanje kompleksnijih problema, potrebno je unaprediti osnovne RAG sisteme. Ta unapređenja predstavljaju napredne RAG sisteme (Slika 10), koji omogućavaju efikasniju obradu složenijih zahteva.



Slika 10. Napredni RAG sistem

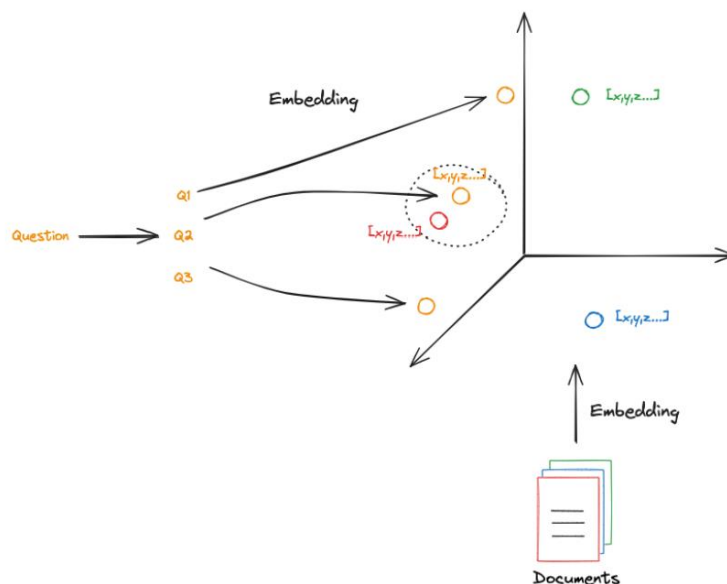
#### 3.3.1 Prevođenje upita (Query Translation)

**Prevođenje upita** se odnosi na proces transformisanja inicijalnog upita u različite forme ili pod-upite u cilju poboljšanja procesa pronalaženja informacija [8]. Ova transformacija može uključivati razbijanje kompleksnog upita na jednostavnije delove, preformulisane upite radi boljeg razumevanja konteksta, ili prilagođavanje upita tako da bolje odgovara podacima u bazi. Cilj prevođenja upita je da se maksimizira relevantnost informacija koje će sistem preuzeti. U nekim slučajevima, upit može biti preformulisan kako bi se bolje podudarao sa sličnim terminima

i konceptima unutar dokumenata. Ovaj korak je posebno važan kada se koristi više izvora podataka ili višejezični sistemi.

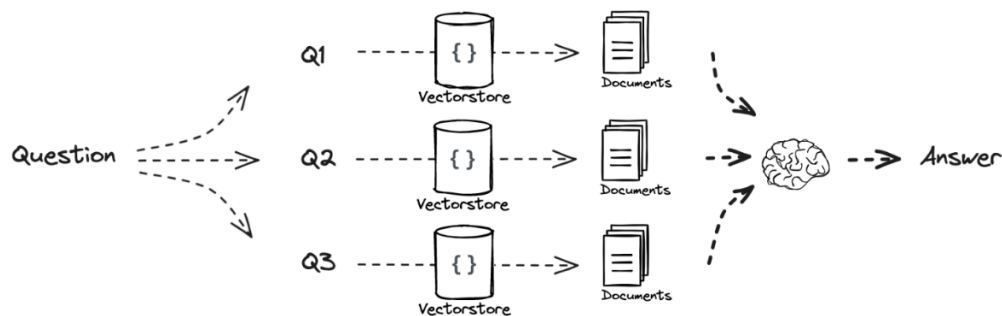
### 3.3.1.1 Multi-query

**Multi-query** podrazumeva korišćenje velikih jezičkih modela za generisanje više upita, na osnovu prvobitnog korisničkog upita [10], [15]. Za svaki upit se vrši embedding i pretraga baze (Slika 11). Ovi upiti se mogu izvršavati paralelno, a dobijeni rezultati se zatim zajedno prosleđuju modelu. Ova tehnika je naročito korisna kada jedno pitanje zavisi od više pod-pitanja, što omogućava potpunije prikupljanje informacija.



Slika 11. Embedding i provera sličnosti kod multi-query strategije [39]

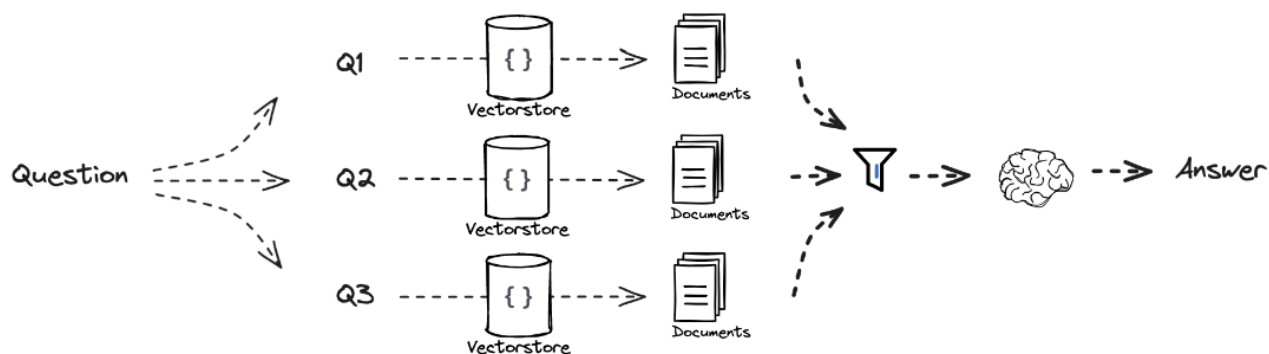
Pretraga baza na osnovu upita se mogu izvršavati paralelno (prikazano na slici 12), što znači da nije neophodno pristupiti istoj bazi podataka za svaki od upita, već se relevantni dokumenti mogu pribaviti iz većeg broja različitih baza podataka koji će se na kraju koristiti za formiranje jednog odgovora.



Slika 12. Paralelna pretraga baza kod multi-query strategije [39]

### 3.3.1.2 RAG-Fusion

**RAG-Fusion** ima nekih sličnosti u poređenju sa **multi-query** strategijom. Ovde se takođe koristi veliki jezički model za generisanje većeg broja upita na osnovu prvobitnog upita. Kod RAG-Fusion strategije, postoje više različitih retriever-a (pretraživača) koji mogu koristiti različite načine pretrage, gde svaki od njih radi pretragu za jedan od generisanih upita. Na primer, jedan retriever može raditi vektorsku pretragu, dok drugi može koristiti ključne reči. Ono što izdvaja RAG-Fusion je proces **rangiranja** dokumenata. Kada retriever-i preuzmu relevantne informacije, RAG-Fusion vrši **evaluaciju** i **rangiranje** tih dokumenata na osnovu njihove relevantnosti i na taj način omogućava da se najvažnije informacije organizuju tako da budu u prvom planu prilikom generisanja odgovora, osiguravajući da konačan rezultat bude što precizniji i relevantniji [17], [14].



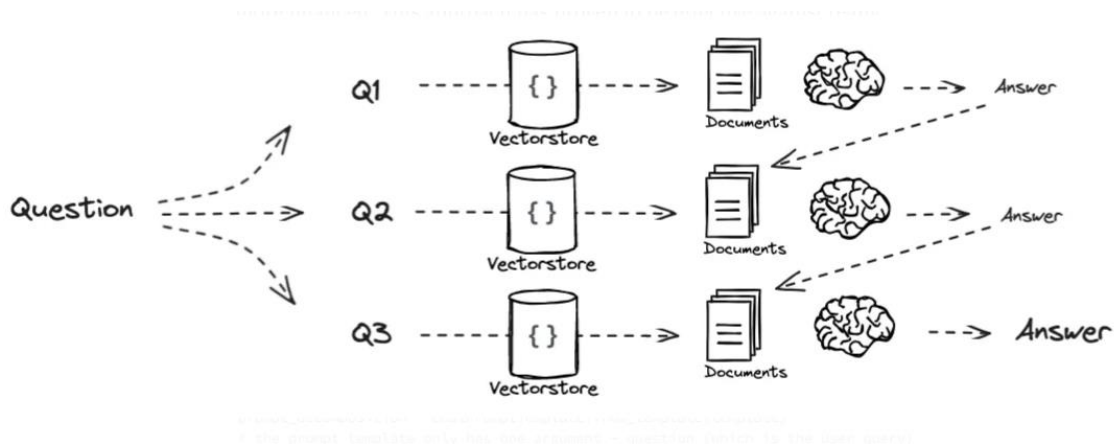
Slika 13. RAG-Fusion, pribavljanje i rangiranje dokumenata [40]

### 3.3.1.3 Decomposition

**Decomposition** je strategija koja podrazumeva razlaganje složenog pitanja ili problema na manje, lakše savladive pod-probleme, koji se zatim rešavaju nezavisno, a njihova rešenja se na kraju spajaju kako bi se dobio sveobuhvatan odgovor [14], [16].

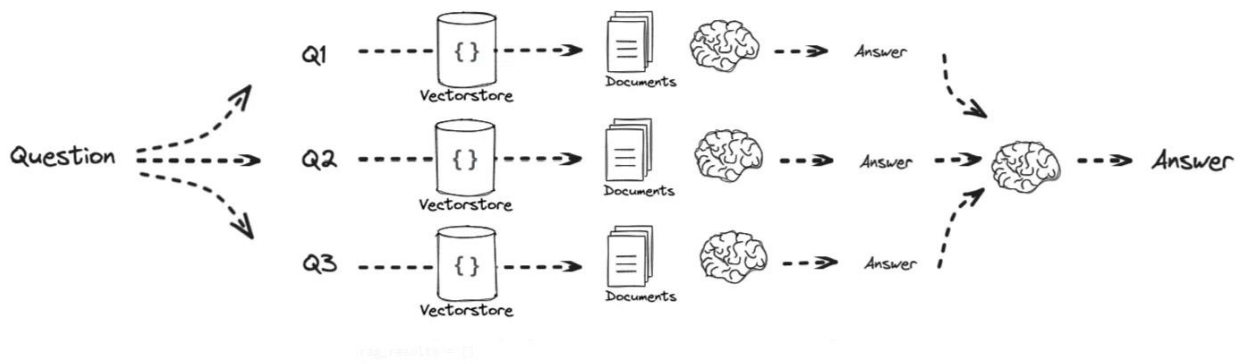
Postoje dva pristupa [18]:

1. **pristup rekurzivnog odgovaranja (Recursive Answering Approach)**: Ovde se pitanja postavljaju jedno po jedno, a za njihovo odgovaranje koriste se zajedno odgovori na prethodna pitanja i kontekst odnosno dokumenti koji su pribavljeni za trenutno pitanje, kao što je prikazano na slici 14. Na taj način zadržava se stara perspektiva i sinhronizuje rešenje sa novom perspektivom, dajući detaljnije rešenje. Ovaj pristup se pokazao efikasnim kod veoma složenih upita.



Slika 14. Pristup rekurzivnog odgovaranja [41]

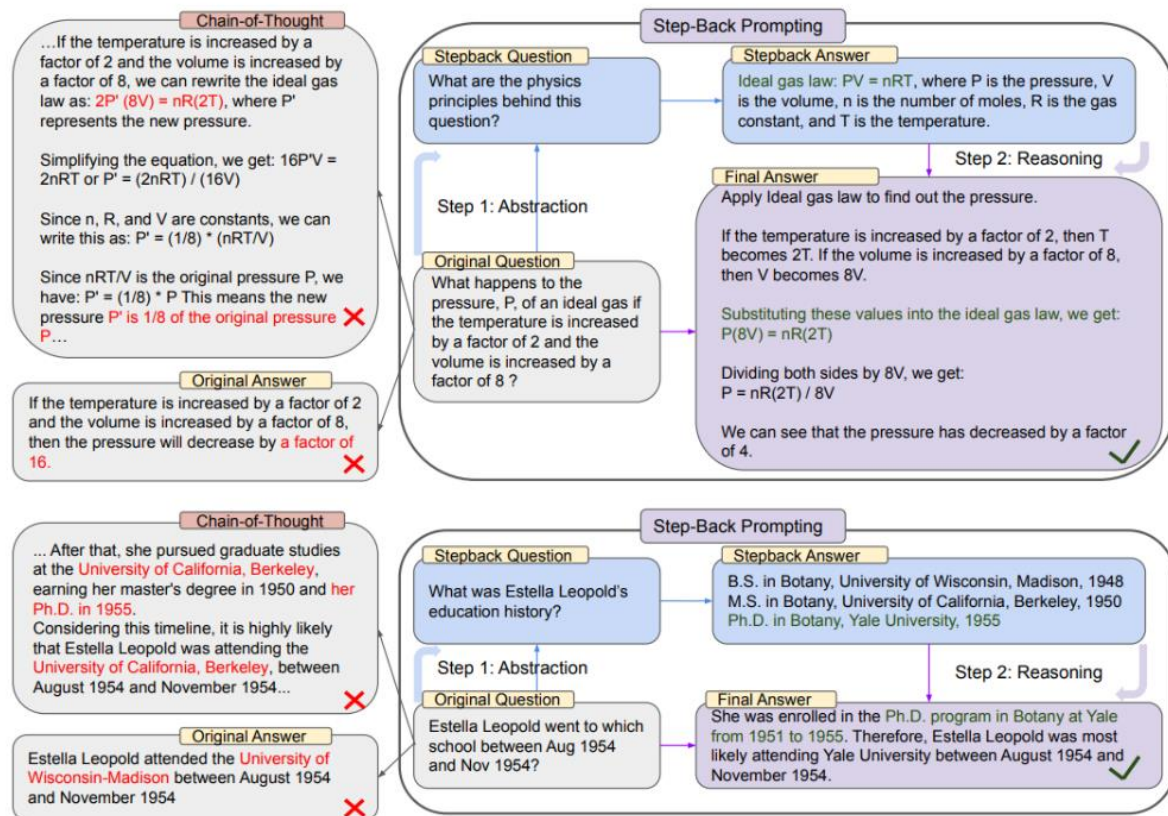
2. **Pristup paralelnog odgovaranja (Parallel Answering Approach):** U ovom pristupu (Slika 15), korisnički upit se razlaže na više delova, kao i ranije. Razlika je u tome što se ti delovi rešavaju paralelno, gde se svako pitanje obrađuje individualno, a zatim se svi odgovori kombinuju kako bi se dobio mnogo detaljniji kontekst, koji se zatim koristi za odgovaranje na korisnički upit. Ovaj pristup predstavlja efikasno rešenje za većinu slučajeva.



Slika 15. Pristup paralelnog odgovaranja [12]

#### 3.3.1.4 Step-back prompting

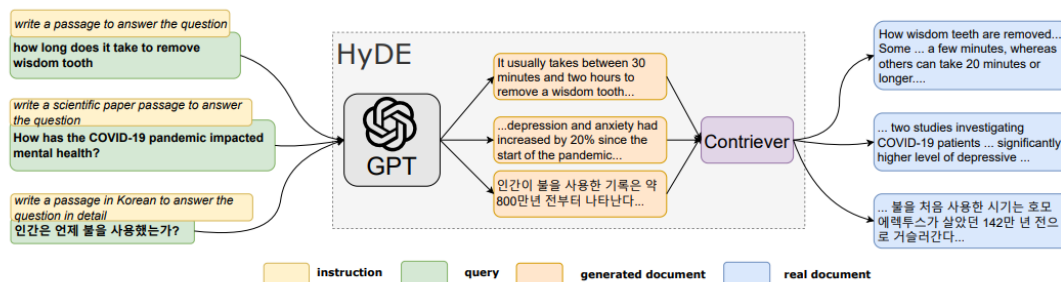
**Step-back prompting**, je tehnika prvi put viđena u [42], koja omogućava modelu da se fokusira na dublje razmišljanje pre nego što reši zadatak. Ona podrazumeva kreiranje apstraktnog pitanja na osnovu korisničkog upita kako bi se dobila šira perspektiva zadatka i prikupio dodatni kontekst. Nakon toga, direktni kontekst se generiše na osnovu korisničkog upita, a oba konteksta se kombinuju za dobijanje preciznijeg odgovora. Ova tehnika je posebno efikasna za pitanja koja zahtevaju razmišljanje, jer omogućava modelu da istraži širu sliku korisničkog upita i uporedi je sa specifičnim kontekstom.



Slika 16. Primer korišćenja step-back prompting-a [42]

### 3.3.1.5 HyDE (Hypothetical Document Embeddings)

HyDE se prvi put pominje u [7]. Ideja iza ovog pristupa je da su dokumenti veliki delovi podataka koji sadrže informacije iz dobro formulisanih, gustih tekstova, dok korisnički upit nije tako dobro konstruisan. Iz tog razloga, konstruišemo hipotetički dokument ili odgovor na osnovu korisničkog upita, koji kada se pretvori u vektor, u teoriji je bliži delovima dokumenta nego sam korisnički upit u visoko-dimenzionalnom vektorskom prostoru [19], [20]. Ilustracija HyDE modela prikazana je na slici 17.



Slika 17. Ilustracija HyDE modela [7]

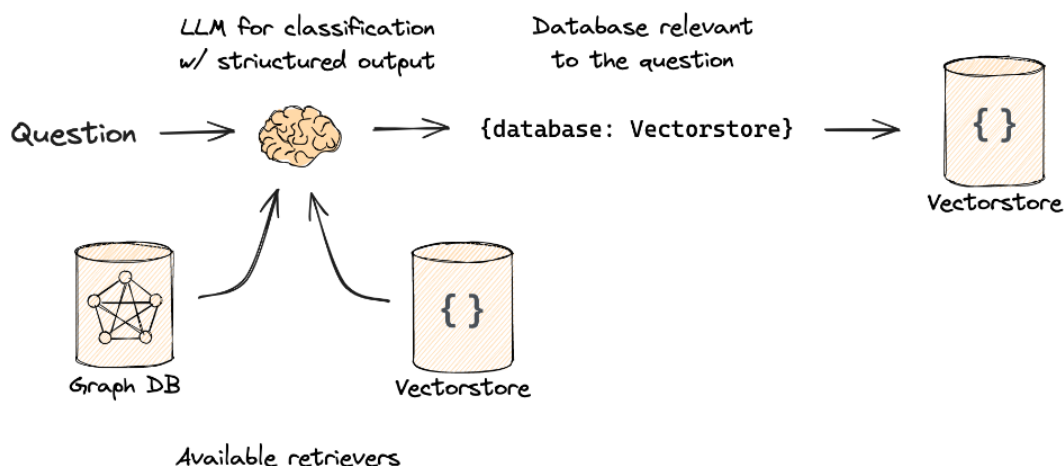


### 3.3.2 Rutiranje (Routing)

Ponekad kod kompleksnijih RAG aplikacija postoje više različitih baza podataka za podatke iz različitih oblasti. **Rutiranje** se najčešće odnosi na proces odlučivanja koja baza podataka ili mehanizam za pretragu će biti korišćeni za pribavljanje relevantnih dokumenata na osnovu korisničkog upita, ali se ne odnosi nužno na izbor baze podataka, već je moguće vršiti i odlučivanje koji prompt koristiti. U suštini, cilj **rutiranja** je da se na osnovu korisničkog upita odredi gde ga treba proslediti dalje. Dva najčešće korišćenih tipova **rutiranja** su:

#### 1. Logičko rutiranje (Logical routing):

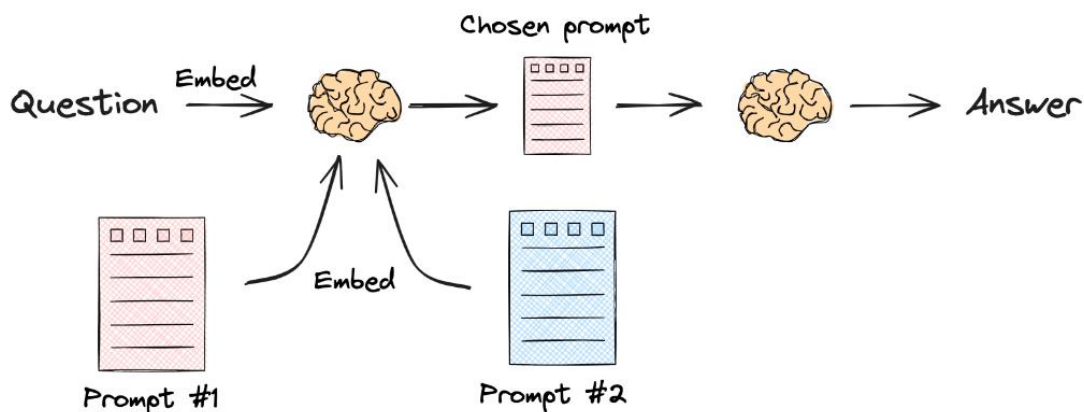
Kod ovog tipa rutiranja, velikom jezičkom modelu se prosleđuje znanje o tome koje sve vrste baza postoje i šta svaka od njih čuva. Ime je nastalo na osnovu činjenice da veliki jezički model koristi **logiku** da bi odlučio kojoj bazi podataka pristupiti ili koji retriever koristiti [8]. Na slici 18 možemo videti proces logičkog rutiranja.



Slika 18. Logičko rutiranje [21]

#### 2. Semantičko rutiranje (Semantic routing):

**Semantičko rutiranje** se odvija u dva koraka. U prvom koraku se vrši embedding nad korisničkim upitom i na primer, promptovima. Nakon toga se njihove numeričke reprezentacije porede u vektorskom prostoru i bira se prompt koji najviše odgovara datom upitu. Odatle i ime „semantičko”, jer vrši poređenje na osnovu semantičkog značenja [8]. Na slici 18 možemo videti proces semantičkog rutiranja.



Slika 19. Semantičko rutiranje [21]

### 3.3.3 Kreiranje upita (Query Construction)

Postoji veliko interesovanje o povezivanju ljudskog jezika sa različitim tipovima podataka, bilo da su strukturirani, nestruktuirani ili čak polustrukturirani podaci [11]. Ovaj korak naprednog RAG sistema radi baš to, prevodi korisnički upit sa ljudskog jezika u odgovarajući format koji mogu izvršiti i razumeti ciljani tipovi baza podataka[8].

Od koristi je pomenuti sledeće vrste [11]:

#### 1. Text-to-metadata-filter:

Koristi se za prevođenje upita u format koji razumeju vektorske baze podataka sa filterima za metapodatke, omogućavajući strukturane upite nad embedded-ovanim nestruktuiranim dokumentima. Ova vrsta je uglavnom od najvećeg interesa kada su u pitanju moderni RAG sistemi.

#### 2. Text-to-SQL:

Kada su u pitanju strukturirani podaci, veliki izvor predstavljaju relacione baze podataka, pa odatle potreba za prevođenje ljudskog jezika u SQL naredbe. Važno je naglasiti da je u cilju generisanja što tačnijih naredbi potrebno proslediti sve CREATE TABLE naredbe u bazi, nakon čega sledi nekoliko primera SELECT naredbi.

#### 3. Text-to-SQL+semantic:

U savremenom radu sa podacima, kombinovano skladištenje strukturiranih i nestruktuiranih podataka postaje sve češće. Dodavanje podrške za vektore u relacionim bazama podataka igra ključnu ulogu u omogućavanju hibridne pretrage. Moguće je vršiti proveru sličnosti nad vektorima unutar relacione baze podataka i na taj način poboljšava text-to-SQL pretragu uz pomoć semantičkih operatora.

#### 4. Text-to-Cypher:

Cypher predstavlja upitni jezik definisan za rad sa graf bazama podataka. Graf baze se koriste zato što rešavaju neke probleme i nedostatke koji se javljaju kod drugih tipova baza podataka. Vektorske baze podataka ne razumeju međusobne odnose između vektora, dok relacione baze, iako dobro modeluju vezu među entitetima, promene šeme mogu da budu mnogo skupe. Graf baze podataka su idealne za odnose tipa više-prema-više i za hijerarhije koje je teško predstaviti u tabelarnoj formi.

### 3.3.4 Indeksiranje (Indexing)

Osnove indeksiranja i embedding-a su obrađene u poglavlju posvećenom osnovnama RAG sistema. U ovom poglavlju će fokus biti na različitim metodama indeksiranja.

#### 3.3.4.1 Chunk Optimization

Chunking, kao što sam naziv sugerishe, podrazumeva razbijanje velikog dokumenta na manje, kontekstualne delove (chunk-ove). Time što se dokument deli na lakše upravljive delove, omogućava se modelu da ga efikasnije obradi [24].

Najpre treba razmisliti o tome kakav je tekst nad kojim se vrši embedding. Vršenje embedding-a nad prevelikim ili premalim segmentima može dovesti do manje optimalnih rezultata. Iz tog razloga je vrlo važno odabrati optimalnu veličinu segmenata kako bi se osigurala tačnost dobijenih rezultata.

Izbor odgovarajuće strategije segmentacije zahteva pažljivo razmatranje faktora kao što su priroda sadržaja, embedding model, optimalna veličina bloka, očekivana dužina i složenost korisničkih upita, kao i način na koji će rezultati biti korišćeni u konkretnoj aplikaciji [22].

Mogu se izdvojiti 5 novoa [22], [24]:

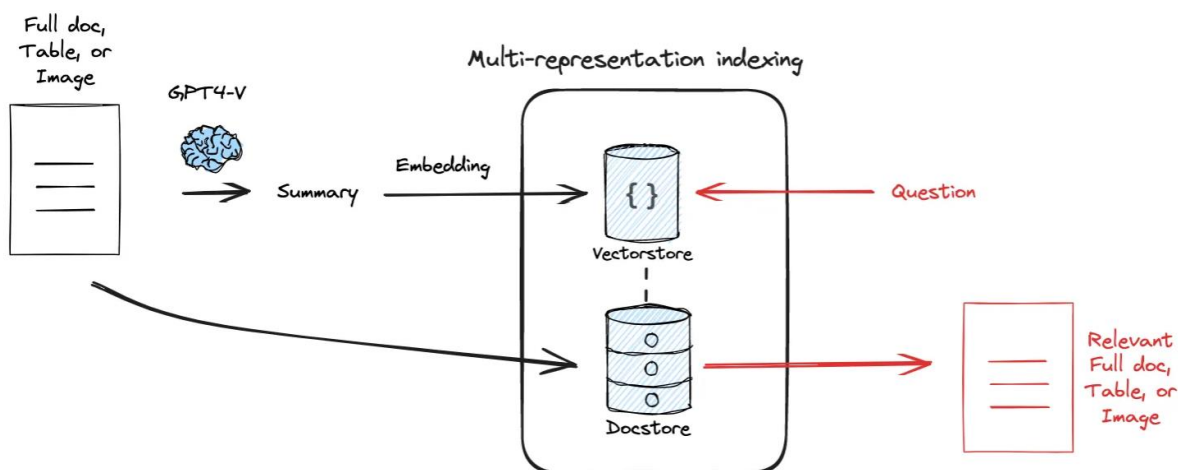
- **Fixed Size Chunking:** Tekst se deli na segmente fiksne dužine, bez obzira na sadržaj ili strukturu.
- **Recursive Chunking:** Tekst se deli na manje segmente na hijerarhijski način koristeći skup separatora, i rekurzivno se deli dok se ne postigne željena veličina segmenta.
- **Document-Based Chunking:** Dokument se deli prema njegovoj strukturi, uzimajući u obzir tok i format sadržaja.
- **Semantic Chunking:** Tekst se deli na osnovu semantičkog značenja, grupišući delove teksta koji su semantički slični.
- **Agentic Chunking:** Koristi velike jezičke modele da odluči koliko i koji deo teksta treba uključiti u segment na osnovu konteksta.



### 3.3.4.2 Multi-representation indexing/ Multi vector representation

Inspiracija i ideja potiču iz rada [25] koji uvodi metodu korišćenja velikih jezičkih modela za kreiranje propozicija, odnosno sažetaka, koji su optimizovani za pretragu. Zaključak je da se mogu dobiti mnogo bolji rezultati ukoliko se koriste propozicije umesto čistih dokumenata.

Multi-representation indexing, čiji proces se nalazi na slici 20, funkcioniše tako što se najpre kreiraju propozicije (sažeci), koji se potom embedduju i na osnovu njih se vrši provera sličnosti sa korisničkim upitom. Celi dokumenti se čuvaju zasebno u odgovarajućem skladištu dokumenata. Kada se pronađu najsličniji sažeci, vraćaju se celi dokumenti kojima oni odgovaraju [21].



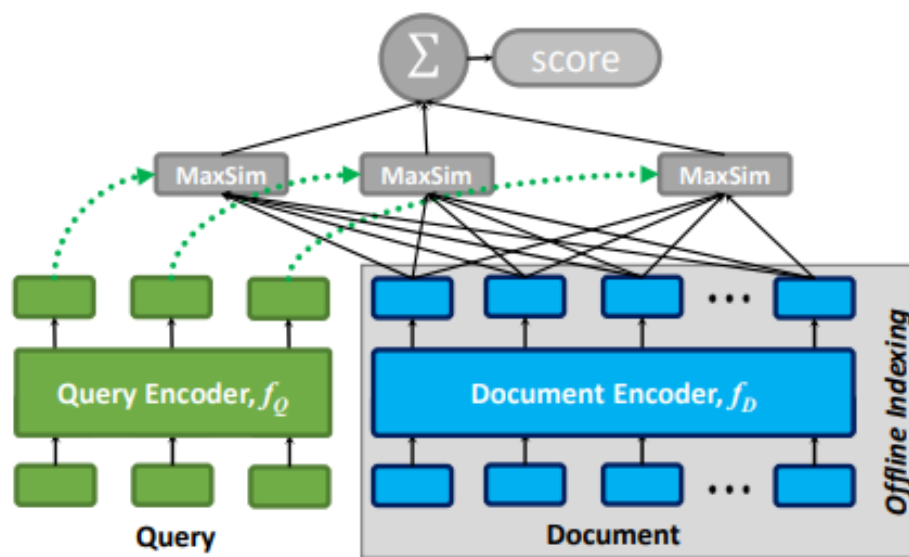
Slika 20. Multi-representation indexing [12]

### 3.3.4.3 Specialized embeddings (ColBERT)

**ColBERT**(Contextualized late interaction over **BERT**) se prvi put javlja u radu [26] kao ideja za efikasnije pretraživanje i rangiranje dokumenata korišćenjem dubokih jezičkih modela, konkretno BERT-a. Ovaj model uvodi inovativnu arhitekturu kasne interakcije, koja omogućava nezavisno kodiranje upita i dokumenata, a zatim koristi brzu, ali preciznu interakciju za procenu sličnosti. Time se smanjuju računarski troškovi, omogućava pretraga velikih kolekcija dokumenata, i značajno ubrzava procesiranje upita u poređenju sa tradicionalnim BERT modelima.

Umesto da koristi klasičnu metodu gde se svaki upit-dokument par mora obraditi zajedno, ColBERT uvodi kasnu interakciju (late interaction), gde se upiti i dokumenti nezavisno enkodiraju koristeći BERT. Upit se najpre tokenizuje i priprema, dodaje se specijalni token [Q] kako bi se označilo da je to upit. BERT model potom obrađuje upit i generiše vreću embedding-a (bag of embeddings), što su vektorske reprezentacije reči iz upita. Dokumenti se sličnim procesom enkodiraju. Pre dokumenta se dodaje specijalni token [D] kako bi se označilo da je to dokument,

a nakon BERT obrade generiše se vreća embedding-a za dokument. Ponekad se uklanjaju tokeni interpunkcija kako bi se smanjio broj embedding-a. Kada se upit i dokument enkodiraju u vreće embedding-a, ColBERT izračunava sličnost između embedding-a iz upita i dokumenta koristeći kosinusnu sličnost koja je ranije detaljnije objašnjena. Zatim računa relevantnost dokumenta prema upitu na osnovu maksimuma sličnosti između embedding-a [26]. Arhitektura ovog načina indeksiranja može se videti na slici 21.



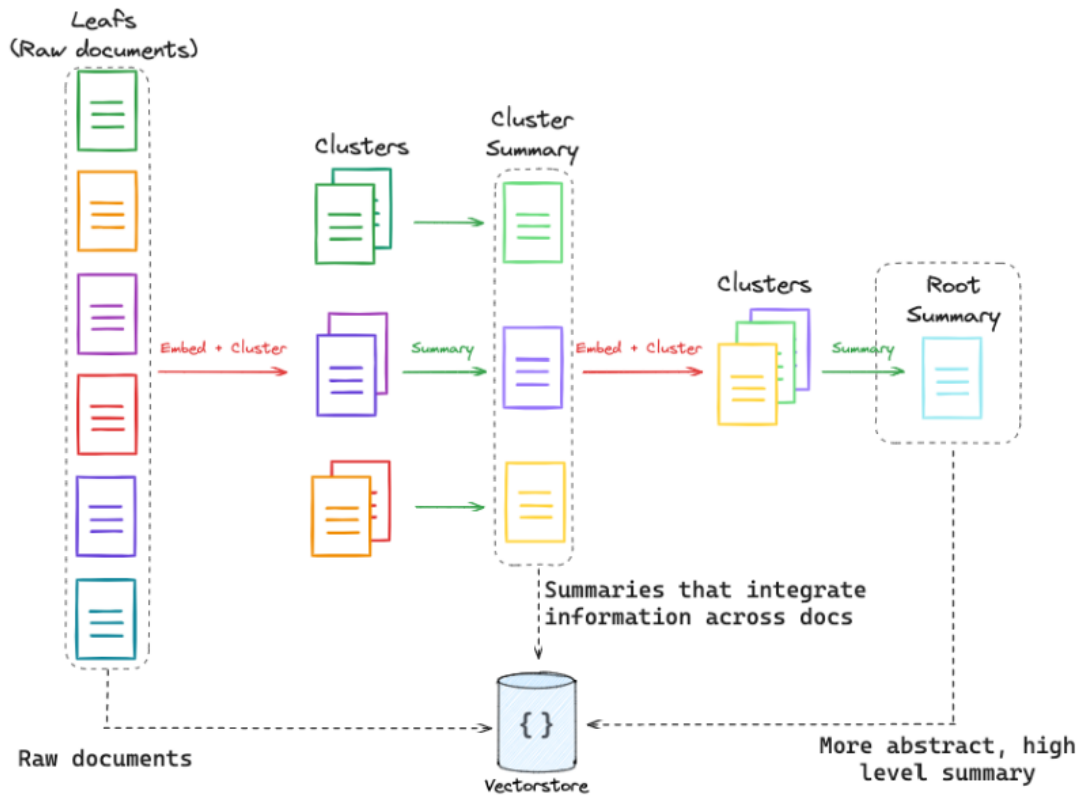
Slika 21. Opšta arhitektura ColBERT-a [26]

#### 3.3.4.4 Hierarchical indexing (RAPTOR)

**RAPTOR (Recursive Approach for Parsing Texts for Optimal Retrieval)** je inovativna metoda, prvi put viđena u [27], dizajnirana za unapređenje performansi RAG sistema, posebno kada se radi sa dugim i složenim tekstovima koji često imaju hijerarhijsku strukturu i više podtema. Dok tradicionalni RAG sistemi obično preuzimaju kratke, neprekidne delove teksta iz skupa dokumenata, što ograničava njihovu sposobnost da obuhvate širi kontekst dokumenta. RAPTOR rešava ovaj problem kreiranjem rekurzivnog stabla.

RAPTOR segmentira dokumente u kratke delove (oko 100 tokena). Ovi delovi se enkodiraju koristeći BERT-bazirani enkoder (SBERT) i formiraju listove stabla. Konačno stablo se dobija rekurzivnim enkodiranjem, klasterovanjem i sažimanjem delova teksta, stvarajući više slojeva apstrakcije. RAPTOR koristi algoritam za klasterovanje kako bi grupisao slične delove teksta. Nakon klasterovanja, vrši se sažimanje sadržaja grupisanih tekstova pomoću jezičkog modela. Sažeti tekst se ponovo enkodira i dalje klasteriše, pri čemu se proces nastavlja dok dalja klasterizacija ne postane neizvodljiva. Rezultat je višeslojno stablo gde svaki nivo predstavlja različit stepen sažimanja, od detaljnih informacija na dnu do apstraktnih sažetaka na vrhu. Koristi se meko klasterovanje, omogućavajući da delovi teksta pripadaju više klastera. Ova fleksibilnost

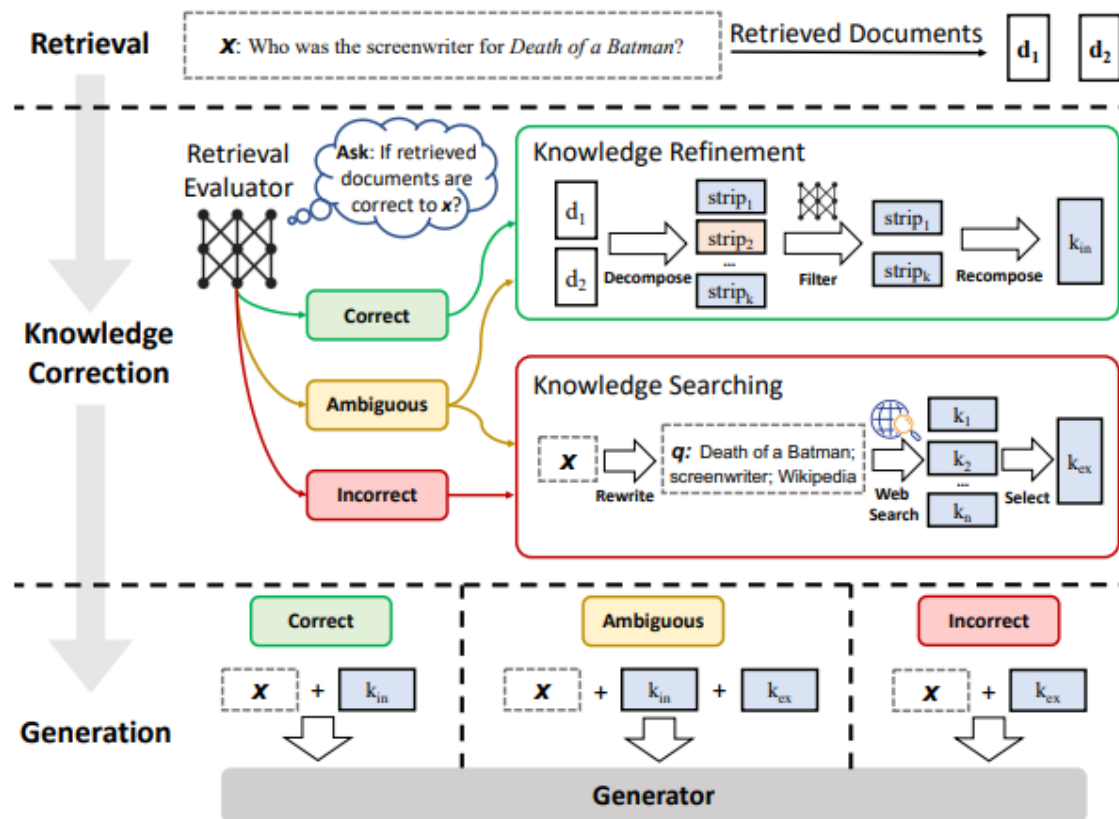
je važna jer pojedinačni segmenti teksta često sadrže informacije relevantne za više tema, što omogućava modelu da bolje obuhvati složenost stvarnih dokumenata [27].



Slika 22. RAPTOR stablo [12]

### 3.3.5 CRAG

**CRAG (Corrective Retrieval-Augmented Generation)** se prvi put javlja u [28], kao rešenje za problem netačnih ili nerelevantnih dokumenata prikupljenih tokom pretrage kod RAG-a. Ključni problem koji CRAG rešava je kako model postupa kada pretraga ne uspe da pronađe tačne ili relevantne informacije. CRAG je dizajniran tako da bude lagan i skalabilan, što znači da se može lako integrisati u različite RAG sisteme bez velikog povećanja resursa. U eksperimentima se pokazalo da CRAG značajno poboljšava performanse na zadacima koji zahtevaju generisanje teksta, kako u kratkoj, tako i u dugoj formi.



Slika 23. Pretraga i generisanje kod Crag-a [28]

Crag uvodi sloj evaluacije (može se videti na slici 28) preuzetih dokumenata kako bi osigurao njihovu relevantnost za postavljeno pitanje. Kada model preuzme dokumente, Crag koristi poseban evaluacioni model da proceni koliko su ti dokumenti relevantni za upit korisnika. Ova procena se izražava kao nivo poverenja (confidence degree), koji zatim aktivira jednu od tri moguće akcije [28]:

1. **Correct (Tačno)** – Ako su dokumenti ocenjeni kao relevantni, oni prolaze kroz dodatni korak obrade. Crag koristi tehniku koja rastavlja informacije iz dokumenata, filtrira nebitne delove, a zatim ponovo sklapa preostale informacije u precizniji oblik. Ova faza se naziva **knowledge refinement** (pročišćavanje znanja).
2. **Incorrect (Netačno)** – Ako dokumenti nisu relevantni, odbacuju se i preusmerava se pretraga na šire izvore, kao što je pretraga putem interneta, u cilju pronalaska tačnijih informacija. Ova metoda proširuje tradicionalne RAG pristupe tako što koristi veći opseg informacija kada su lokalni izvori neadekvatni.
3. **Ambiguous (Neodređeno)** – Ukoliko se ne može jasno utvrditi da li su dokumenti tačni ili netačni, koristi se „neodređena“ akcija, koja kombinuje oba izvora (lokalnu pretragu i pretragu interneta) kako bi postigao uravnotežen pristup.

### 3.3.6 RAG vs Fine-tuning (Fino podešavanje)

RAG nije konvencionalna metoda za fino podešavanje u mašinskom učenju. Umesto toga, RAG je arhitektura koja kombinuje jezičke modele sa sistemom za pretragu kako bi poboljšala generaciju odgovora korišćenjem eksternih podataka. Dok tradicionalno fino podešavanje prilagođava unapred treniran model određenom zadatku kroz dodatnu obuku, RAG dinamički uključuje relevantne informacije iz eksternih izvora tokom generacije, čime poboljšava kvalitet i relevantnost odgovora. Međutim, komponente RAG sistema, poput jezičkog modela i sistema za pretragu, mogu se dodatno fino podešavati kako bi se unapredila njihova preciznost i performanse. U tabeli 1 prikazano je poređenje između RAG-a i finog podešavanja [30]:

Tabela 1. Poređenja RAG-a i finog podešavanja

	RAG	Fino podešavanje
Ažuriranje Znanja	Direktno ažurira bazu podataka za pretragu, što je pogodno za dinamična okruženja podataka i zahteva ređe ponovno obučavanje	Oslanja se na statičke podatke, što zahteva ponovnu obuku za ažuriranja i manje je pogodno za informacije koje se često menjaju
Spoljno Znanje	Efikasno koristi spoljne resurse, posebno sa strukturiranim i nestrukturiranim bazama podataka	Usaglašava spoljno znanje sa obukom velikih jezičkih modela, ali je manje efikasno za dinamične izvore podataka
Obrada Podataka	Zahteva minimalnu obradu podataka	Ograničenja u datasetovima mogu uticati na performanse
Prilagođavanje Modela	Fokusira se na integraciju spoljnog znanja, ali nudi manje prilagodljivosti u ponašanju ili stilu modela	Omogućava veću kontrolu nad ponašanjem modela, stilom i domenom znanja, prilagodljiv je specifičnim potrebama.
Interpretabilnost	Pruža veću interpretabilnost zbog mogućnosti praćenja odgovora do izvora podataka	Često funkcioniše kao crna kutija, što otežava razumevanje odluka modela, rezultirajući manjom interpretabilnošću
Zahtevi za Latenciju	Uključuje pretragu podataka, što može povećati latenciju	Obično nudi manju latenciju jer funkcioniše bez potrebe za pretragom podataka nakon obuke
Smanjenje Halucinacija	Manje je sklono halucinacijama zbog oslanjanja na pretražene dokaze	Može smanjiti halucinacije sa obukom specifičnom za domen, ali je i dalje podložno njima kod nepoznatih ulaza
Etika i Privatnost	Pitanja se javljaju zbog skladištenja i pretrage tekstova iz spoljnih baza podataka	Etička pitanja i pitanja privatnosti mogu nastati zbog osetljivog sadržaja u podacima za obuku.
Računarski Resursi	Zahteva resurse za strategije pretrage i održavanje baze podataka	Zahteva resurse za pripremu datasetova, definisanje ciljeva i računarsku snagu za obučavanje

### 3.3.7 Budućnost RAG-a

Budućnost RAG modela u oblasti veštačke inteligencije i obrade prirodnog jezika donosi niz uzbudljivih unapređenja koja će poboljšati njihovu efikasnost i proširiti primenu u različitim domenima [30]. Jedno od ključnih unapređenja je **poboljšana integracija pretrage i generisanja** informacija. Ovo će omogućiti RAG modelima da se fokusiraju na bolje povezivanje ovih komponenti kako bi omogućili nesmetano funkcionisanje, odnosno obezbeđivanje da preuzete informacije budu visoko relevantne i tačne za zadatak koji je u toku.

RAG modeli će se takođe širiti u specifične domene poput **zdravstva, prava i finansija**, što će omogućiti da modeli efikasno koriste znanja iz određenih industrija, pružajući precizne i stručne odgovore na složene upite specifične za te oblasti.

Još jedna bitna oblast unapređenja je **povećanje efikasnosti i skalabilnosti** RAG sistema. Optimizacija računarskih procesa omogućavaće modelima da obrađuju veće skupove podataka i sve složenije upite, čime će biti otvoren put za širu komercijalnu i industrijsku primenu. Takođe, unaprediće se **razumevanje konteksta**, što će omogućiti modelima da bolje interpretiraju nijanse, dvosmislenosti i specifične namere korisnika.

U skladu sa brzim promenama u svetu, očekuje se razvoj naprednijih mogućnosti za **preuzimanje informacija u realnom vremenu**. To će podrazumevati preuzimanje aktuelnih podataka iz dinamičnih izvora, poput interneta ili ažurnih baza podataka, što će ove sisteme učiniti posebno korisnim za obradu novosti, trendova i aktuelnih događaja.

Budućnost donosi i **multimodalne RAG sisteme** koji će ići dalje od teksta, uključujući slike, video zapise i audio sadržaje. Ova unapređenja će stvoriti sveobuhvatnije sisteme sposobne da razumeju i generišu odgovore u različitim tipovima medija.

**Interaktivno učenje i integracija povratnih informacija** dodatno će poboljšati RAG sisteme, omogućavajući im da se unapređuju kroz vreme zahvaljujući učenju iz korisničkih interakcija i kontinuiranom poboljšanju performansi na osnovu povratnih informacija u realnom vremenu. Etička pitanja takođe će postati sve važnija, sa naporima da se obezbedi **odgovorna upotreba** RAG modela, rešavajući probleme kao što su pristrasnost, dezinformacije i transparentnost u preuzimanju i obradi podataka.

Očekuje se i unapređenje u **višelingvističkim sposobnostima** RAG modela, omogućavajući efikasniju pretragu i generisanje sadržaja na više jezika. Pored toga, potencijal za veću **saradnju sa ljudskom inteligencijom** omogućiće da RAG sistemi rade zajedno sa ljudskim ekspertima, kombinujući efikasnost i skalabilnost veštačke inteligencije sa kreativnošću i prosuđivanjem ljudi.

Kako se tehnologija RAG-a nastavlja razvijati, ovi modeli postaće sve sofisticiraniji, prilagodljiviji i ključni za razne primene, pomerajući granice sposobnosti veštačke inteligencije u razumevanju i generisanju ljudskog jezika.

## 4 Implementacija RAG sistema sa Q&A Chatbot-om za pravila društvenih igara

U ovom poglavlju biće opisana implementacija Q&A chatbot-a baziranog na RAG tehnici za pravila društvenih igara. Sistem je razvijen korišćenjem Flask-a za API, LangChain-a za integraciju sa velikim jezičkim modelima, i Pinecone-a za upravljanje vektorskom bazom podataka. Cilj ovog projekta je da se korisnicima omogući da postavljaju pitanja o pravilima različitih društvenih igara i dobijaju tačne i relevantne odgovore. Kompletan kod implementiranog sistema nalazi se na github repozitorijumu: <https://github.com/BoziCCCa/RAG-Thesis>

Korisnici mogu uneti svoja pitanja putem korisničkog interfejsa, koji komunicira sa backend-om preko Flask API-ja. Backend koristi LangChain za generisanje odgovora na osnovu pretraživanja vektorske baze podataka koja sadrži pravila igara. Sistem implementira mehanizme za napredno pretraživanje i generisanje odgovora korišćenjem RAG tehnike, što omogućava pružanje preciznih informacija i uvida u pravila društvenih igara.

Pored objašnjenja osnovnih funkcionalnosti sistema, u ovom poglavlju će biti prikazani izazovi projekta, kao i delovi koda koji ilustruju kako se integracija između različitih komponenti ostvaruje i kako sistem obrađuje korisničke upite.

### 4.1 Izazovi projekta

Implementacija Q&A chatbot-a za pravila društvenih igara nosi sa sobom mnoge izazove koji čine tradicionalne metode pretrage i generisanja odgovora neadekvatnim. Pravilnici društvenih igara su često napisani u različitim formatima i stilovima, a informacije su često rasute po velikim i kompleksnim dokumentima. Tradicionalni sistemi za pretragu se uglavnom oslanjaju na jednostavno pretraživanje ključnih reči, što može rezultirati netačnim ili nerelevantnim odgovorima kada je potrebno precizno odgovoriti na specifična pitanja.

Jedan od ključnih izazova je varijabilnost pravila u različitim društvenim igrama. Svaki pravilnik može biti organizovan na jedinstven način, što otežava jednostavno pretraživanje ključnih informacija. Neki pravilnici su sažeti i strukturisani, dok su drugi opširni i obuhvataju specifične situacije i izuzetke.

U ranijim oblastima je detaljno objašnjeno kako RAG sistem funkcioniše kao i njegove prednosti i primene. U kontekstu ovog projekta, RAG sistem se izdvaja kao posebno koristan zbog svoje sposobnosti da efikasno pretražuje i generiše odgovore iz kompleksnih i nestrukturisanih izvora kao što su pravilnici društvenih igara.

Pravilnici društvenih igara često sadrže raznolike i specifične informacije, a jednostavna pretraga ključnih reči često nije dovoljna da obuhvati sve nijanse i kontekste pravila. RAG nudi rešenje ovih problema omogućavajući dubinsku pretragu kontekstualnih informacija i generisanje tačnih odgovora prilagođenih korisničkom upitu. Na ovaj način, RAG sistem može prepoznati

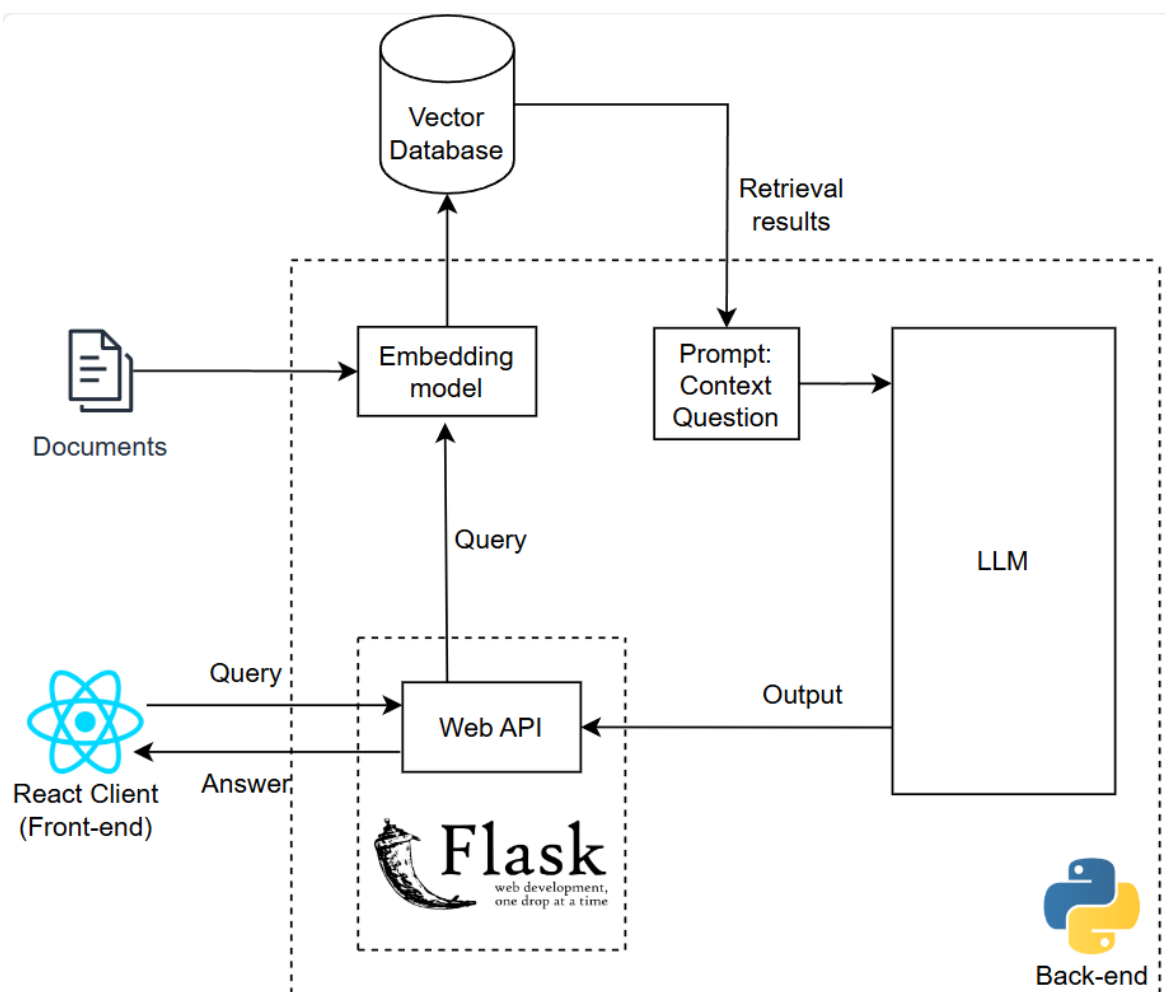


specifične delove pravilnika relevantne za pitanje korisnika, što značajno unapređuje tačnost i korisnost odgovora.

#### 4.1.1 Potreba za tačnim informacijama

Za chatbot aplikaciju koja se bavi pravilima društvenih igara, ključno je da informacije budu tačne i relevantne, posebno kada su pravila igre bitna za donošenje odluka tokom igre gde pogrešne odluke mogu dovesti do frustracije igrača. RAG pomaže u rešavanju ovog problema koristeći pretragu vektorskih reprezentacija podataka, što osigurava da su povučeni podaci zaista relevantni i u skladu sa postavljenim pitanjem.

#### 4.2 Arhitektura chatbot aplikacije za pravila društvenih igara



Slika 24. Arhitektura chatbot aplikacije za pravila društvenih igara



Kompletna arhitektura aplikacije prikazana je na slici 24. Za implementaciju ovog projekta, za logiku i komunikaciju sa velikim jezičkim modelom na serverskoj strani korišćen je Python, dok je za korisnički interfejs na klijentskoj strani korišćen React. U nastavku će biti malo više reči o korišćenim tehnologijama.

#### 4.2.1 Flask

**Flask** je popularan mikro-framework za Python, razvijen sa ciljem da omogućiti jednostavno i brzo kreiranje web aplikacija i API-ja. Flask predstavlja skup biblioteka i modula koji omogućavaju programerima da razvijaju web aplikacije bez potrebe da se bave detaljima nižeg nivoa poput protokola, upravljanja nitima i sličnim tehničkim aspektima [31]. U ovom projektu, Flask je korišćen za kreiranje API-ja koji omogućava komunikaciju između front-enda i back-enda. Zbog svoje jednostavne konfiguracije, Flask je bio idealan izbor za implementaciju (Slika 25), jer omogućava brzu i efikasnu razmenu podataka između korisničkog interfejsa i jezičkog modela koji se koristi za generisanje odgovora.

```
app = Flask(__name__)
CORS(app)
initialize()

@app.route('/ask-question', methods=['POST'])
def ask_question():
    data = request.get_json()
    query = data.get("question")

    if not query:
        return jsonify({"error": "No question provided"}), 400

    answer = rag_fusion_pipeline(query)
    return jsonify({"answer": answer})

if __name__ == '__main__':
    app.run(debug=True)
```

Slika 25. Flask API implementacija

#### 4.2.2 LangChain

**LangChain** predstavlja framework koji služi za razvoj aplikacija koje koriste velike jezičke modele. Nudi alate za integraciju velikih jezičkih modela sa spoljnim izvorima podataka, API-jima i korisničkim interakcijama, omogućavajući složene procese poput odgovaranja na pitanja, sažimanja i pretraživanja dokumenata [32].

Framework se sastoji od dve open-source biblioteke, čije konkretne primene će biti opisane kasnije :

1. langchain-core – Osnovne apstrakcije i LangChain jezik izraza.
2. langchain-community – 3rd-party integracije

### 4.2.3 Pinecone

Kao što je već pomenuto, vektorske baze podataka igraju ključnu ulogu u procesu pretrage informacija iz baza podataka. U tu svrhu koristi se **Pinecone**, upravljana vektorska baza podataka sa jednostavnim API-jem, koja omogućava dugoročno skladištenje vektora za AI aplikacije [33].

Svaka Pinecone baza podataka se sastoji od jednog ili većeg broja indeksa. Indeks predstavlja strukturu podataka koja omogućava čuvanje, pretragu i pristup vektorima koji se nalaze u njemu. Svaki indeks ima i svoju dimenziju koja označava dimenzionalnost vektora koji se mogu čuvati u njemu. Dimenzionalnost vektora zavisi od embedding modela koji se koristi (u projektu je ta vrednost 1536).

## 4.3 Implementacija ključnih funkcionalnosti

Nakon detaljnog proučavanja ključnih alata i tehnologija, sledeći korak je razumevanje celokupne funkcionalnosti projekta. Ovaj deo će prikazati kako različite komponente sistema međusobno funkcionišu i doprinose ostvarivanju ciljeva projekta. Data je analiza kako se informacije obrađuju, skladište i koriste za generisanje odgovora, što je ključno za efikasno rešavanje problema vezanih za pravila društvenih igara.

### 4.3.1 Učitavanje dokumenata

Dokumenti se, u zavisnosti od obima projekta i potreba za skalabilnošću, mogu čuvati u posebnim bazama podataka. Međutim, s obzirom na to da se ovaj projekat ne bavi velikim količinama dokumenata, odlučeno je da se koristi lokalni fajl sistem za čuvanje dokumenata. Ovaj pristup omogućava jednostavniju implementaciju i manje resursno zahtevan sistem, jer lokalni fajl sistem pruža dovoljno prostora i fleksibilnosti za trenutne potrebe projekta. Ukoliko se obim dokumentacije poveća, postoji mogućnost prelaska na drugi vid čuvanja dokumenata.

LangChain nudi širok spektar document loader-a (TextLoader, PyPDFLoader, YouTubeLoader, HTMLLoader), koji predstavljaju specijalizovane alate za učitavanje podataka iz različitih izvora kao što su tekstualni fajlovi, PDF-ovi, web stranice, baze podataka ili čak transkripti video zapisa. Njihova glavna funkcija je da automatizuju proces prikupljanja podataka iz tih izvora, formatiraju ih i pripreme za dalje obrade.

Document loader-i u LangChain-u pružaju metodu **load**, koja je ključna za učitavanje podataka iz podešenog izvora. Kao izlaz ove metode dobijaju se Document objekti, koji sadrže:

- **Tekstualni sadržaj** – Glavni deo dokumenta, npr. Sadržaj stranica iz PDF-a ili transkript videa.
- **Metapodaci** – Dodatne informacije o dokumentu, poput izvora, stranice, autora, URL-a, itd.

Neki document loader-i podržavaju i opcionu metodu **lazy\_load**, koja omogućava „lazy loading“ podataka. To znači da se podaci učitavaju postepeno, samo kada su potrebni, što smanjuje potrošnju memorije kod većih setova podataka.

U ovom projektu koristi se PyPDFLoader jer su pravila društvenih igara obično dostupna u obliku zasebnih PDF fajlova koji se jednostavno mogu preuzeti. Ova metoda omogućava efikasno i precizno učitavanje pravila direktno iz tih fajlova, čime se olakšava njihov dalji rad u sistemu.

```
import os
import re
from concurrent.futures import ThreadPoolExecutor
from langchain_community.document_loaders import PyPDFLoader

DATA_PATH = "data"

def clean_document(text):
    cleaned_text = re.sub(r'\s+', ' ', text)
    cleaned_text = cleaned_text.strip()
    return cleaned_text

def load_pdf(file_path):
    loader = PyPDFLoader(file_path)
    documents = loader.load_and_split()
    return documents

def load_documents():
    pdf_files = [os.path.join(DATA_PATH, file_name) for file_name in os.listdir(DATA_PATH) if file_name.endswith(".pdf")]

    with ThreadPoolExecutor() as executor:
        results = list(executor.map(load_pdf, pdf_files))

    flat_documents = [doc for result in results for doc in result]

    cleaned_documents = []
    for doc in flat_documents:
        doc.page_content = clean_document(doc.page_content)
        cleaned_documents.append(doc)

    return cleaned_documents
```

Slika 26. Učitavanje dokumenata

U nastavku su dati ključni elementi koda sa slike 26:

- **DATA\_PATH** – putanja do direktorijuma u kome se nalaze PDF fajlovi
- **clean\_document** – Ova funkcija prima tekst kao ulaz i čisti ga tako što uklanja višak belih prostora (tabulator, novi red) i trimuje krajeve. Ovaj korak je važan za pripremu teksta pre obrade
- **load\_pdf** – Ova funkcija koristi **PyPDFLoader** za učitavanje PDF fajla sa zadate putanje (**file\_path**) koja se prosleđuje kao parametar. Zatim koristi metodu PyPDFLoader-a **load\_and\_split** da podeli PDF sadržaj na stranice. Rezultat je lista **Document** objekata koji sadrže tekst i metapodatke.

- **load\_documents** – Ova funkcija najpre pretražuje direktorijum **DATA\_PATH** i generiše listu (**pdf\_files**) putanja do svih PDF fajlova koji se nalaze u njemu. Zatim koristi **ThreadPoolExecutor** za paralelno učitavanje PDF fajlova koristeći funkciju **load\_pdf**. Ovo omogućava efikasnije procesiranje više fajlova istovremeno. Učitani dokumenti se spajaju u jednu listu (**flat\_documents**), zatim se tekst u svakom **Document** objektu čisti koristeći funkciju **clean\_document**. Čisti dokumenti se zatim dodaju u listu **cleaned\_documents**

#### 4.3.2 Inicijalizacija baze

Sada kada su dokumenti učitani, spremni su za njihovu dalju obradu i čuvanje u vektorsku bazu podataka koju najpre treba inicijalizovati.

```
import os
from dotenv import load_dotenv
from pinecone import Pinecone, ServerlessSpec

def create_new_index(pc, index_name):
    pc.create_index(
        name=index_name,
        dimension=1536,
        metric="cosine",
        spec=ServerlessSpec(
            cloud="aws",
            region="us-east-1"
        )
    )

def initialize_pinecone():
    index_name="quickstart-index"
    load_dotenv()
    pc = Pinecone(api_key=os.getenv("PINECONE_API_KEY"))
    existing_indexes = [index['name'] for index in pc.list_indexes()]

    if index_name not in existing_indexes:
        create_new_index(pc, index_name)

    index = pc.Index(index_name)
    return index
```

Slika 27. Pinecone inicijalizacija i kreiranje indeksa

U nastavku su dati ključni elementi koda sa slike 27:

- **create\_new\_index** – funkcija koja u Pinecone bazi kreira novi indeks sa zadatim imenom, dimenzijom 1536 (mora da se poklapa sa dimenzijom vektora koje vraća embedding model) i metrikom „**cosine**”, što označava da će se u ovom indeksu pretraga vršiti na osnovu kosinusne sličnosti koja je već detaljnije objašnjena. Funkcija takođe podešava i specifikacije indeksa
- **initialize\_pinecone** – ova funkcija učitava API ključeve pomoću **load\_dotenv**, proverava da li indeks već postoji i, ako nije prisutan, kreira ga korišćenjem funkcije **create\_new\_index**, a zatim vraća instancu tog indeksa za dalju upotrebu u Pinecone operacijama.

Ukratko, ovaj deo koda omogućava efikasno upravljanje i pristupanje Pinecone indeksima kako bi se vektori dokumenata mogli sačuvati za brzo pretraživanje.

#### 4.3.3 Kreiranje i čuvanje vektora

Sledeći korak u obradi dokumenata odnosi se na odluku o načinu njihove podele i indeksiranja. U ovom projektu, odluka je bila da se koriste LangChain-ovi Text Splitteri. Kada se radi sa dugim tekstovima, neophodno je podeliti ih na manje delove ili chunk-ove, kako bi odgovarali kontekstualnom prozoru modela. LangChain nudi nekoliko različitih tipova Text Splittera koji olakšavaju ovu obradu [34], neki od njih su:

- Recursive
- HTML
- Code
- Token
- Character
- Semantic

U ovom projektu, odluka je bila da se koristi **RecursiveCharacterTextSplitter** (Slika 28) zbog njegove sposobnosti da zadrži semantički povezane delove teksta zajedno. Pravila društvenih igara često sadrže informacije koje su tematski povezane unutar različitih sekcija, a ovaj tip chunking-a omogućava da se tekst podeli na manje delove na način koji čuva tu povezanost. **RecursiveCharacterTextSplitter** pokušava da inteligentno raspodeli tekst kako bi se obezbedila relevantnost unutar svakog dela, što olakšava preciznije pretrage i poboljšava kvalitet odgovora na upite korisnika o pravilima igre.

```
def split_documents(documents, chunk_size=300, chunk_overlap=100):
    text_splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
        chunk_size=chunk_size,
        chunk_overlap=chunk_overlap,
        is_separator_regex=False,
    )
    return text_splitter.split_documents(documents)
```

Slika 28. Chunking funkcija

```
def embed_and_index_documents(documents):
    docs = split_documents(documents)

    embeddings = []
    vectors = []

    chunk_ids = calculate_chunk_ids(docs)
    for i, doc in enumerate(docs):
        embedding = get_embeddings(doc.page_content)
        embeddings.append(embedding)
        doc.metadata["page_content"] = doc.page_content
        vectors.append((chunk_ids[i], embedding, doc.metadata))

    existing_ids = get_existing_vector_ids()
    new_vectors = [(id, embedding, metadata) for id, embedding, metadata in vectors if id not in existing_ids]
    if new_vectors:
        print(f"Adding {len(new_vectors)} new vectors to Pinecone.")

        add_vectors_to_database(new_vectors)
        print(f"Indexed {len(new_vectors)} vectors into Pinecone.")
    else:
        print("No new vectors to add.")
```

Slika 29. Glavna funkcija za indeksiranje i embedding

Funkcija **embed\_and\_index\_documents**, čiji je kod prikazan na slici 29, obavlja čitav proces embedding-a i indeksiranja ranije učitanih dokumenata. Prvo se dokumenti dele na manje delove (chunk-ove) pomoću funkcije `split_documents`, a zatim se za svaki chunk izračunava jedinstveni identifikator (Slika 31). Ovi identifikatori se koriste za proveru postojanja duplikata, sa ciljem da se u indeks dodaju samo novi vektori. Na taj način se izbegava dodavanje već postojećih podataka, čime se štede vreme i resursi. Tokom izračunavanja identifikatora, poziva se funkcija `get_embeddings`, koja kreira embedding za svaki chunk pomoću određenog embedding modela (u projektu se koristi **OpenAI model text-embedding-3-small**, što pokazuje slika 30), čime se dobija njihova numerička reprezentacija. Na samom kraju, novi vektori se dodaju u bazu podataka pozivom funkcije **add\_vectors\_to\_database**.

```
from langchain_openai import OpenAIEmbeddings
embeddings_model = OpenAIEmbeddings(
    model="text-embedding-3-small",
    api_key=os.getenv("OPENAI_API_KEY")
)

def get_embeddings(content):
    return embeddings_model.embed_query(content)
```

Slika 30. Implementacija OpenAI modela za kreiranje embedding-a

```
def calculate_chunk_ids(docs):
    last_page_id = None
    current_chunk_index = 0
    chunk_ids = []

    for doc in docs:
        source = doc.metadata.get("source")
        page = doc.metadata.get("page")
        current_page_id = f"{source}:{page}"

        if current_page_id == last_page_id:
            current_chunk_index += 1
        else:
            current_chunk_index = 0

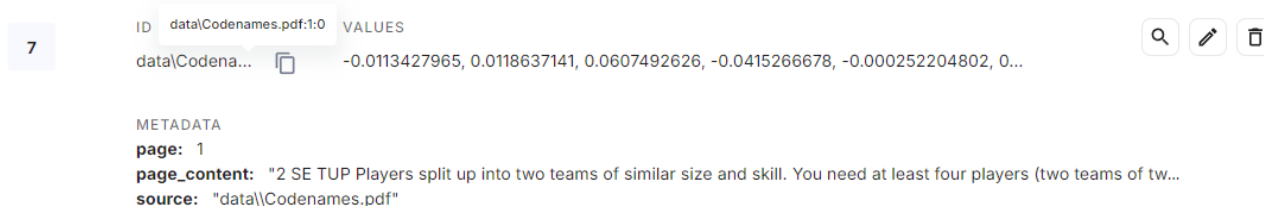
        chunk_id = f"{current_page_id}:{current_chunk_index}"
        last_page_id = current_page_id

        chunk_ids.append(chunk_id)

    return chunk_ids
```

Slika 31. Računanje identifikatora za svaki chunk

Primer dodatog vektora u bazu može se videti na slici 32. gde se vidi njegov jedinstveni identifikator kao i dodatni metapodaci.



The screenshot shows a database table with two columns: 'ID' and 'VALUES'. The 'ID' column contains the value 'data\Codenames.pdf:1:0'. The 'VALUES' column contains a long list of floating-point numbers representing a vector. Below the table, there is a 'METADATA' section with three fields: 'page: 1', 'page\_content: "2 SE TUP Players split up into two teams of similar size and skill. You need at least four players (two teams of tw...", and 'source: "data\\Codenames.pdf"'. On the right side of the table, there are three icons: a magnifying glass, a pencil, and a trash can.

ID	VALUES
data\Codenames.pdf:1:0	-0.0113427965, 0.0118637141, 0.0607492626, -0.0415266678, -0.000252204802, 0...

METADATA

page: 1

page\_content: "2 SE TUP Players split up into two teams of similar size and skill. You need at least four players (two teams of tw..."

source: "data\\Codenames.pdf"

Slika 32. Vektor u bazi sa svojim metapodacima

#### 4.3.4 Pretraga i generisanje (Retrieval and generation)

Cela funkcionalnost LangChain-a zasniva se na konceptu **lanaca (chains)**, koji predstavljaju osnovni način organizacije i upravljanja radnim tokovima. Svaka operacija u LangChain-u, bilo da se radi o interakciji sa jezičkim modelima, pretrazi u bazama podataka, ili primeni različitih alata, strukturisana je u sekvence koraka unutar lanca. Ovaj pristup omogućava modularnost i fleksibilnost, jer se svaki korak može nezavisno kontrolisati i prilagoditi specifičnim potrebama aplikacije. Na taj način, lanci obezbeđuju jednostavno povezivanje različitih komponenti sistema i optimizuju ceo proces rada [32].

Postoje dva tipa lanaca koje LangChain podržava:

1. **LCEL (LangChain Execution Language) chains**: Ovi lanci nude fleksibilnost i omogućavaju korisnicima da lako modifikuju svaki deo lanca.
2. **Legacy Chains** – kreirani pomoću starijih metoda nasleđivanja klasa.

Prednost LCEL lanaca leži u mogućnosti jednostavnog modifikovanja i praćenja svakog koraka lanaca. Pored toga, ovi lanci podržavaju strimovanje (streaming), obavljanje asinhronih operacija, kao i obradu više upita odjednom (batch processing). Iz tih razloga, u ovom projektu koriste se LCEL lanci, kod kojih se za nadovezivanje internih koraka koristi **operator** | (Slika 33). Ovaj operator povezuje dva koraka tako da izlaz prvog automatski postaje ulaz drugog, čime se pojednostavljuje radni tok i omogućava lakše kreiranje složenih procesa.

```
generation_chain = generation_prompt | llm | StrOutputParser()
```

Slika 33. Primer LCEL lanca

Prvi korak u procesu pretrage, koji omogućava davanje tačnih i relevantnih odgovora, jeste prevođenje korisničkog upita. U ovom projektu, za tu svrhu je korišćen **RAG-Fusion** pristup. RAG-Fusion je odličan izbor kada je u pitanju RAG sistem za pravila o društvenim igrama iz nekoliko razloga:

- **Višestruki upiti:** Društvene igre često imaju složena i specifična pravila koja mogu biti opisana na različite načine u dokumentaciji. RAG-Fusion omogućava generisanje više upita na osnovu jednog korisničkog pitanja, čime se pokrivaju različiti načini na koje pravila mogu biti formulisana.
- **Kombinovanje relevantnih informacija:** Pravila društvenih igara često imaju povezane koncepte koji su raspoređeni u različitim delovima dokumenta. RAG-Fusion omogućava sistemu da prikupi informacije iz više izvora, što pomaže da se svi važni delovi pravila objedine i predstave korisniku na jasan i konzistentan način.
- **Preciznost pretrage:** Korišćenjem tehnike Reciprocal Rank Fusion, ovaj pristup olakšava rangiranje najbitnijih i najrelevantnijih dokumenata, što je posebno korisno kada je potrebno pronaći specifičan deo pravila u dokumentima koji sadrže puno informacija.
- **Kontekstualna relevantnost:** RAG-Fusion omogućava da se ključne informacije efikasno sažmu u finalni odgovor, kako bi korisnik dobio tačnu i potpunu informaciju koja je direktno vezana za njegovo pitanje o pravilima igre.

```
from langchain_openai import ChatOpenAI
load_dotenv()
llm = ChatOpenAI(model_name="gpt-4o-mini-2024-07-18", temperature=0, api_key=os.getenv("OPENAI_API_KEY"))
```

Slika 34. Inicijalizacija velikog jezičkog modela

```
rag_fusion_prompt = ChatPromptTemplate.from_template("""
You are a helpful assistant that generates multiple search queries based on a single input query about board game rules.
Generate multiple search queries related to: {question}
Output (4 queries):
""")
generate_queries_chain = rag_fusion_prompt | llm | StrOutputParser()
```

Slika 35. RAG Fusion prompt i chain

U ovom projektu koristi se veliki jezički model GPT 4o mini, obučen 18. jula 2024. godine. Nakon inicijalizacije modela (Slika 34), potrebno je kreirati prompt (Slika 35) koji se šalje modelu i na osnovu inicijalnog korisničkog upita generišu se četiri nova upita. Ovaj pristup doprinosi većoj pouzdanosti i relevantnosti dobijenih informacija. Zatim se svaki upit zasebno koristi za pretragu vektorske baze podataka. Glavna funkcija zadužena za to je **retrieve\_documents** (Slika 35).



```
def retrieve_documents(queries):
    query_splits=split_queries(queries)
    query_embeddings = embeddings_model.embed_documents(query_splits)

    results = []
    index=initialize_pinecone()
    for query_embedding in query_embeddings:
        response = index.query(
            vector=query_embedding,
            top_k=5,
            include_metadata=True
        )
        results.append(response["matches"])
    return results
```

Slika 36. Funkcija retrieve\_documents

Prvi korak funkcije **retrieve\_documents** (Slika 36) podrazumeva podelu upita na manje delove (chunk-ove), slično načinu na koji su dokumenti prethodno deljeni. Međutim, zbog manjeg obima upita u poređenju sa celokupnim dokumentima, korišćena je manja veličina chunk-ova (chunk\_size=50, chunk\_overlap=20) kako bi se optimizovala obrada. Nakon podele, vrši se embedding svakog chunk-a, kako bi se dobila njihova numerička reprezentacija, odnosno vektor. Svaki vektor se zatim koristi za pretragu baze po kosinusnoj sličnosti i uzimaju se pet najbližih vektora koji se dodaju u listu **results**. Sledeći korak predstavlja rangiranje upravo pribavljenih dokumenata pomoću funkcije **reciprocal\_rank\_fusion** (Slika 37).

```
from langchain.load import dumps, loads

def reciprocal_rank_fusion(results: list[list[dict]], k=60):
    fused_scores = {}

    for docs in results:
        for rank, doc in enumerate(docs):
            doc_id = doc['id']
            score = doc['score']
            page_content=doc['metadata']['page_content']
            doc_str = dumps({'id': doc_id, 'score': score, 'page_content':page_content})

            if doc_str not in fused_scores:
                fused_scores[doc_str] = 0

            fused_scores[doc_str] += score / (rank + k)

    reranked_results = [
        (loads(doc), score)
        for doc, score in sorted(fused_scores.items(), key=lambda x: x[1], reverse=True)
    ]

    return reranked_results
```

Slika 37. Funkcija za rangiranje pribavljenih dokumenata

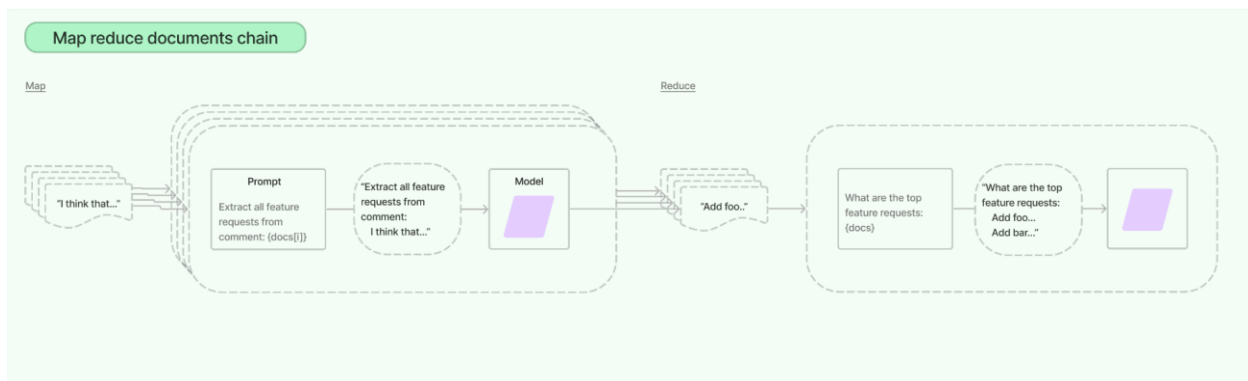
Prvi korak funkcije **reciprocal\_rank\_fusion** (Slika 37) je kreiranje rečnika **fused\_scores**, koji čuva zbirne ocene dokumenata na osnovu njihovih **pozicija (rank)** u listi rezultata. Svaki dokument identifikuje se jedinstvenim ID-jem, **ocenom (score)** i sadržajem stranice, koji se pretvaraju u JSON string pomoću funkcije **dumps**. Zatim se izračunava nova ocena za svaki dokument tako što se zbirna ocena povećava za vrednost dobijenu deljenjem originalne ocene sa pozicijom dokumenta u listi rezultata. Nakon obrade svih dokumenata, funkcija vraća listu dokumenata sortiranu po konačnoj oceni, pri čemu se rezultati vraćaju u strukturu Document klase pomoću funkcije **loads**.

Nakon završenog rangiranja, uzimaju se pet najboljih dokumenata, kao što se može videti na slici 38.

```
ranked_docs = reciprocal_rank_fusion(retrieved_docs)
ranked_docs_content = [doc[0]['page_content'] for doc in ranked_docs]
top_ranked_docs_content = ranked_docs_content[0:5]
```

Slika 38. Uzimanje najboljih dokumenata

Za dalju obradu dokumenta koristi se **MapReduce**. MapReduce je tehnika koja se koristi za efikasnu obradu velikih skupova podataka. U prvom koraku (**Map** faza), svaki dokument se zasebno obrađuje kako bi se dobio sažetak ključnih informacija. U drugom koraku (**Reduce** faza) ti pojedinačni sažeci se kombinuju u jedan krajnji, sjedinjeni sažetak, osiguravajući tačnost i relevantnost informacija. Ovaj princip rada prikazan je na slici 39.



Slika 39. Map Reduce princip rada [32]

Prvo je potrebno kreirati posebne prompt-ove za **map** i **reduce** korake. Prompt za **map** korak može se videti na slici 40, dok je prompt za **reduce** korak prikazan na slici 41. Nakon toga, treba formirati chain-ove koji će prosleđivati ove promptove velikom jezičkom modelu radi obrade.

```
map_prompt = ChatPromptTemplate.from_template("""
Write a concise summary for the following document: {doc}.
The summary should contain crucial information regarding this question about board game rules: {question}.
""")
map_chain = map_prompt | llm
```

Slika 40. Map prompt i chain

```
reduce_prompt = ChatPromptTemplate.from_template("""
The following is a set of summaries:
{docs}
Take these and distill them into a final, consolidated summary
of the main themes. You are writing summaries about board game rules, so you need to be precise and correct.
The summary should have all the crucial information regarding this question:
{question}
""")
reduce_chain = reduce_prompt | llm | StrOutputParser()
```

Slika 41. Reduce prompt i chain

Kao odgovor, od velikog jezičkog modela dobija se konačan sažetak koji će se koristiti kao kontekst za odgovor na korisničko pitanje. Kao i do sada, treba kreirati prompt i chain pomoću koga će se preneti kontekst i pitanje velikom jezičkom modelu (Slika 42), kako bi se na kraju dobio konačan odgovor.

```
generation_prompt = ChatPromptTemplate.from_template("""
Answer the question based only on the following context:
{context}

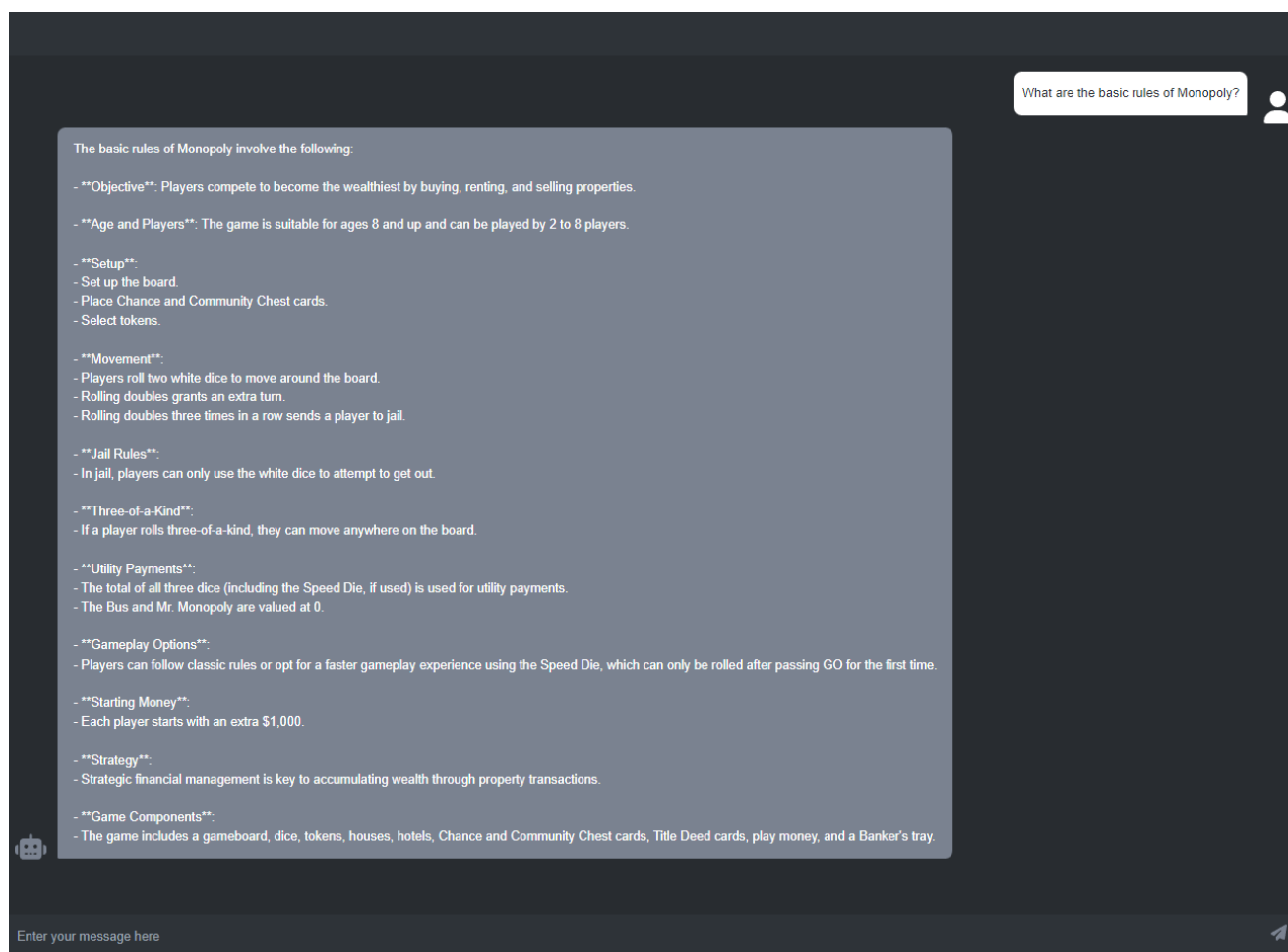
Question: {question}
""")
generation_chain = generation_prompt | llm | StrOutputParser()
```

Slika 42. Prompt i chain za generisanje konačnog odgovora

#### 4.4 Primer rada aplikacije

U ovom poglavlju biće obrađen rad aplikacije i prikaz rezultata koji su dobijeni prilikom korišćenja aplikacije. Za primer rada, kao izvor podataka korišćena su pravila društvenih igara: **Codenames**, **Monopoly (Monopol)**, **Risk (Riziko)** i **Ticket to ride**. Igre su izabrane zbog svoje popularnosti.

Postavljanje pitanja vrši se unosom teksta u polje na dnu ekrana i pritiskom na dugme pored polja, ili jednostavno pritiskom na taster Enter. Korisnik će dobiti odgovor na pitanje samo ako relevantan kontekst postoji u dokumentima.



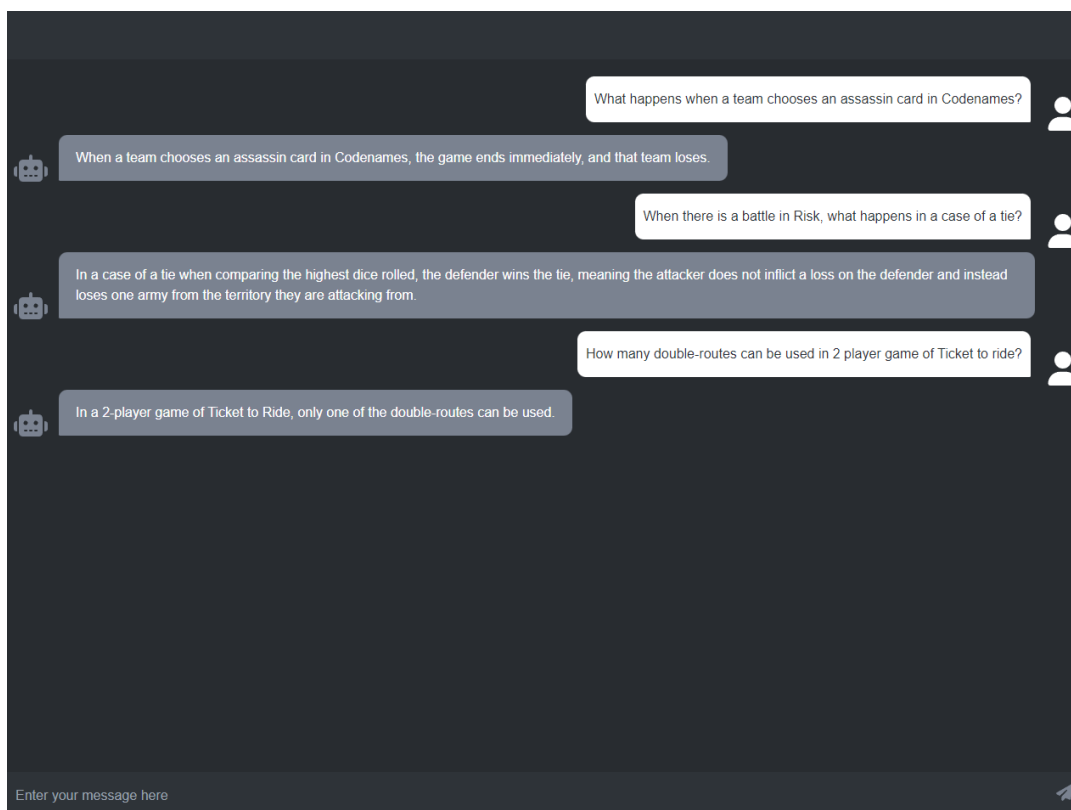
Slika 43. Primer odgovora na opšte pitanje o pravilima igre Monopol

Na slici 43 može se videti primer očekivanog odgovora za postavljeno opšte pitanje o igri Monopol. Može se uočiti da iako se dokumenti dele na manje chunk-ove, chatbot nema nikakvih problema pri odgovaranju na uopštena pitanja, poput „What are the basic rules of Monopoly?” (Koja su osnovna pravila Monopola?).

Slika 44 pokazuje odgovore na konkretna i specifična pitanja o pravilima ostalih društvenih igara. Može se uočiti kako chatbot precizno odgovara na postavljena pitanja. Na primer:

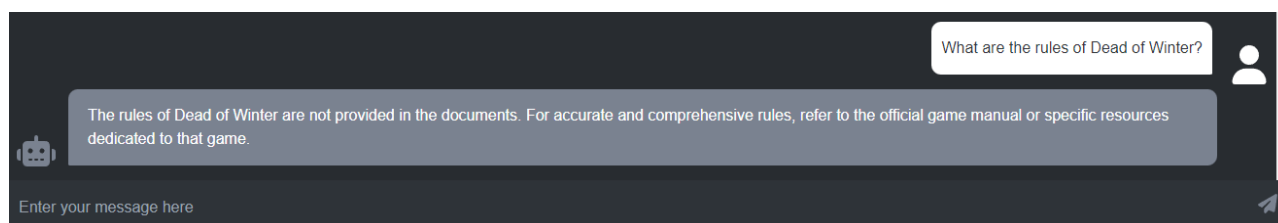
- „What happens when a team chooses an assassin card in Codenames?” (Šta se dešava kada tim izabere kartu ubice u igri Codenames?) – Igra se odmah završava, a tim gubi.
- „When there is a battle in Risk, what happens in a case of a tie?” (Šta se dešava u igri Riziko ako dođe do izjednačenja tokom bitke?) – U slučaju izjednačenja, branioc pobeđuje, a napadač gubi jednu armiju.
- „How many double-routes can be used in a 2-player game of Ticket to Ride?” (Koliko duplih ruta može biti korišćeno u igri Ticket to Ride za dva igrača?) – Samo jedna dupla ruta može biti korišćena u igri za dva igrača.

Može se zapaziti da deljenje dokumenata na chunk-ove i rangiranje pribavljenih dokumenata omogućava lakše i efikasnije pronalaženje relevantnih informacija, čak i za pitanja o veoma specifičnim pravilima.



Slika 44. Primer odgovora na konkretna pitanja o pravilima ostalih društvenih igara

Ovaj projekat je napravljen tako da radi isključivo sa informacijama koje se nalaze u dokumentima. Iz tog razloga, ukoliko korisnik postavi pitanje o društvenoj igri za koju ne postoji dokument sa pravilima, chatbot će ga obavestiti o tome, što se može videti na slici 45.



Slika 45. Primer odgovora na pitanje o pravilima igre koja se ne nalazi u dokumentima

## 5 Zaključak

U ovom radu obrađena je primena RAG-a u cilju unapređenja chatbot aplikacija. U teorijskom delu objašnjeni su koncepti, kao što su veliki jezički modeli i generativna veštačka inteligencija, kao i ograničenja koja se javljaju u njihovom korišćenju. Zatim je prikazano kako RAG tehnika, integracijom velikih jezičkih modela i mehanizama pretrage, pokušava da prevaziđe ta ograničenja, omogućavajući chatbot-ovima da pružaju relevantne i precizne odgovore na osnovu eksternih izvora podataka.

Praktična implementacija sistema, koja uključuje korišćenje Pinecone-a kao vektorsku bazu podataka i LangChain framework-a u Python-u za obradu upita, pokazala je kako RAG tehnika može uspešno da obradi korisničke upite i odgovori na njih na osnovu eksternih dokumenata. Na primerima društvenih igara Monopol, Riziko, Codenames i Ticket to ride, demonstrirano je kako chatbot može generisati tačne i relevantne odgovore na osnovu dostupnih pravila igara.

Korišćenje tehnologija kao što su Pinecone i LangChain ima brojne prednosti. Pinecone omogućava efikasno i brzo pretraživanje vektora, što čini ovaj sistem skalabilnim, čak i kada se obrađuje veliki broj dokumenata. LangChain, s druge strane, nudi fleksibilnost u kreiranju lanaca za pretragu i generisanje odgovora, što olakšava planiranje i praćenje svakog koraka posebno. Međutim, postoje i izazovi. Brzina obrade može zavisi od kvaliteta internet konekcije, s obzirom na to da se Pinecone servis koristi kao cloud rešenje. Što se tiče alternativa, postoje i druga rešenja poput FAISS-a za lokalno indeksiranje vektora, koje može biti prikladnije za manje sisteme ili sisteme koji rade offline. Međutim, Pinecone je odabran zbog svoje skalabilnosti i jednostavnosti korišćenja. Takođe, LangGraph, kao napredniji alat za rad sa grafovima u kontekstu RAG arhitekture, omogućava organizaciju celokupnog procesa i njegovo predstavljanje u obliku grafa, što ga čini efikasnijim rešenjem za složenije sisteme. Ograničenja sistema uključuju zavisnost od kvaliteta eksternih dokumenata, kao i manje precizne odgovore na apstraktne upite, ali pružene tehnologije predstavljaju solidnu osnovu za dalji razvoj RAG sistema.

Rezultati potvrđuju da RAG tehnika značajno poboljšava tačnost odgovora, kako na opšte, tako i na specifične upite u chatbot aplikacijama, što ukazuje na potencijalnu primenu ove tehnologije u industrijama poput korisničke podrške i obrazovanja. Buduća istraživanja mogu se fokusirati na proširenje RAG sistema za različite domene i industrije, kao i unapređenje tačnosti pretrage u kompleksnijim scenarijima.

## 6 Literatura

- [1] Ibm, „What are large language models (llms)?,” IBM, <https://www.ibm.com/topics/large-language-models> (pristupljeno 1. septembra 2024.)
- [2] A. Y. Mehdi Allahyari, A practical approach to retrieval augmented generation systems, <https://mallahyari.github.io/rag-ebook> (pristupljeno 4. septembra 2024.)
- [3] A. Vaswani et al., „Attention is all you need,” arXiv.org, <https://arxiv.org/abs/1706.03762> (pristupljeno 2. septembra 2024.)
- [4] J. Ferrer, „How transformers work: A detailed exploration of Transformer architecture,” DataCamp, <https://www.datacamp.com/tutorial/how-transformers-work> (pristupljeno 2. septembra 2024.)
- [5] „What are large language models?: A Comprehensive Llms Guide,” Elastic, <https://www.elastic.co/what-is/large-language-models> (pristupljeno 1. septembra 2024.)
- [6] „What is a transformer model?,” IBM, <https://www.ibm.com/topics/transformer-model> (pristupljeno 2. septembra 2024.)
- [7] L. Gao, X. Ma, J. Lin, and J. Callan, „Precise zero-shot dense retrieval without relevance labels,” arXiv.org, <https://arxiv.org/abs/2212.10496> (pristupljeno 9. septembra 2024.)
- [8] S. Singh, „Mastering rag: Advanced methods to enhance retrieval-augmented generation,” Medium, <https://medium.com/@samarrana407/mastering-rag-advanced-methods-to-enhance-retrieval-augmented-generation-4b611f6ca99a> (pristupljeno 10. septembra 2024.)
- [9] G. Palanisamy, „Navigating the basics of rag indexing-part 1 for beginners,” Medium, <https://medium.com/@gokulpalanisamy/navigating-the-basics-of-rag-indexing-part-1-for-beginners-a07fda1aa8c3> (pristupljeno 6. septembra 2024.)
- [10] LangChain, „Query transformations,” LangChain Blog, <https://blog.langchain.dev/query-transformations> (pristupljeno 7. septembra 2024.)
- [11] LangChain, „Query construction,” LangChain Blog, <https://blog.langchain.dev/query-construction> (pristupljeno 10. septembra 2024.)
- [12] Langchain-Ai, „Langchain-ai/RAG-from-scratch,” GitHub, <https://github.com/langchain-ai/rag-from-scratch/tree/main> (pristupljeno 11. septembra 2024.)
- [13] C. S. Perone, „Machine Learning :: Cosine Similarity for Vector Space Models (Part III),” Terra Incognita, <https://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii> (pristupljeno 7. septembra 2024.)

- [14] Y. and A. News, „Learn generative AI rag from scratch with Lance Martin’s 14 steps.,” Building AI Retrieval-Augmentation (RAG) from Scratch, <https://www.linkedin.com/pulse/learn-generative-ai-rag-from-scratch-lance-martins-14-steps-klxve> (pristupljeno 8. septembra 2024.)
- [15] S. Majid, „Rag techniques: Multi query,” DEV Community, <https://dev.to/shawonmajid/rag-techniques-multi-query-2p5h> (pristupljeno 7. septembra 2024.)
- [16] K. Dhungana, „Advanced rag: Decomposition technique in Langchain,” Medium, <https://medium.com/@kbdhunga/advanced-rag-decomposition-technique-in-langchain-c0959541cfec> (pristupljeno 8. septembra 2024.)
- [17] D. Shah, „Reciprocal Rank Fusion (RRF) explained in 4 mins.,” Medium, <https://medium.com/@devalshah1619/mathematical-intuition-behind-reciprocal-rank-fusion-rrf-explained-in-2-mins-002df0cc5e2a> (pristupljeno 8. septembra 2024.)
- [18] Raghunaathan, „Query translation for Rag (retrieval augmented generation)applications,” Medium, <https://raghunaathan.medium.com/query-translation-for-rag-retrieval-augmented-generation-applications-46d74bff8f07> (pristupljeno 8. septembra 2024.)
- [19] P. Nayak, „Advanced Rag-improving retrieval using hypothetical document embeddings(hyde),” Medium, <https://medium.aiplanet.com/advanced-rag-improving-retrieval-using-hypothetical-document-embeddings-hyde-1421a8ec075a> (pristupljeno 9. septembra 2024.)
- [20] D. Jaju, „Hyde - hypothetical document embeddings,” Medium, <https://medium.com/@dineshkumarjaju/hyde-hypothetical-document-embeddings-9b3fbfca8d2d> (pristupljeno 9. septembra 2024.)
- [21] Raghunaathan, „Indexing and routing strategies in retrieval-augmented generation (RAG) Chatbots,” Medium, <https://blog.gopenai.com/indexing-and-routing-strategies-in-retrieval-augmented-generation-rag-chatbots-06271908e9f6> (pristupljeno 10. septembra 2024.)
- [22] Thallyscostal, „Chunking strategies optimization for retrieval augmented generation (RAG) in the context of...,” Medium, <https://medium.com/@thallyscostal/chunking-strategies-optimization-for-retrieval-augmented-generation-rag-in-the-context-of-e47cc949931d> (pristupljeno 10. septembra 2024.)
- [23] freeCodeCamp.org, „Learn RAG From Scratch – Python AI Tutorial from a LangChain Engineer,” YouTube, <https://www.youtube.com/watch?v=sVcwVQRHic8&t=5558s> (pristupljeno 6. septembra 2024.)
- [24] A. Mishra, „Five levels of chunking strategies in Rag: Notes from greg’s video,” Medium, [https://medium.com/@anuragmishra\\_27746/five-levels-of-chunking-strategies-in-rag-notes-from-gregs-video-7b735895694d](https://medium.com/@anuragmishra_27746/five-levels-of-chunking-strategies-in-rag-notes-from-gregs-video-7b735895694d) (pristupljeno 10. septembra 2024.)



- [25] T. Chen et al., „Dense X retrieval: What retrieval granularity should we use?,” arXiv.org, <https://arxiv.org/abs/2312.06648> (pristupljeno 10. septembra 2024.)
- [26] O. Khattab and M. Zaharia, „Colbert: Efficient and effective passage search via contextualized late interaction over Bert,” arXiv.org, <https://arxiv.org/abs/2004.12832> (pristupljeno 11. septembra 2024.)
- [27] P. Sarthi et al., „Raptor: Recursive abstractive processing for tree-organized retrieval,” arXiv.org, <https://arxiv.org/abs/2401.18059> (pristupljeno 11. septembra 2024.)
- [28] S.-Q. Yan, J.-C. Gu, Y. Zhu, and Z.-H. Ling, „Corrective retrieval augmented generation,” arXiv.org, <https://arxiv.org/abs/2401.15884> (pristupljeno 11. septembra 2024.)
- [29] Et Tu Code, *LLM, Transformer, RAG AI: Mastering Large Language Models, Transformer Models, and Retrieval-Augmented Generation (RAG) Technology*. 2024
- [30] Dr. Ray Islam, *Retrieval-Augmented Generation (RAG) Empowering Large Language Models (LLMs)*. Dr. Ray Islam (Mohammad Rubyet Islam). 2023
- [31] „What is Flask Python,” What is Flask Python - Python Tutorial, <https://pythonbasics.org/what-is-flask-python/> (pristupljeno 14. septembra 2024.)
- [32] „LangChain Documentation,” LangChain, <https://python.langchain.com/v0.1> (pristupljeno 14. septembra 2024.)
- [33] „The vector database to build knowledgeable AI,” Pinecone, <https://docs.pinecone.io> (pristupljeno 14. septembra 2024.)
- [34] „Welcome to flask,” Welcome to Flask - Flask Documentation (3.0.x), <https://flask.palletsprojects.com> (pristupljeno 14. septembra 2024.)
- [35] A. Gheorghiu, *Building Data-Driven Applications with LLAMAINDEX a Practical Guide on Retrieval-Augmented Generation (RAG) to Enhance LLM Applications*. Birmingham, UK: Packt Publishing Ltd, 2024.
- [36] P. Lewis et al., „Retrieval-augmented generation for knowledge-intensive NLP tasks,” arXiv.org, <https://arxiv.org/abs/2005.11401> (pristupljeno 4. septembra 2024)
- [37] L. Martin, „Rag from scratch: Retrieval,” Google Slides, [https://docs.google.com/presentation/d/124I8jlBRCbb0LAUhdmDwbn4nREqxSxZU1RFeTGXUGc/edit#slide=id.g267060cc54f\\_0\\_0](https://docs.google.com/presentation/d/124I8jlBRCbb0LAUhdmDwbn4nREqxSxZU1RFeTGXUGc/edit#slide=id.g267060cc54f_0_0) (pristupljeno 6. septembra 2024.)
- [38] L. Martin, „Rag from scratch: Generation,” Google Slides, [https://docs.google.com/presentation/d/1eRJwzbdSv71e9Ou9yeqziZrz1UagwX8B1kL4TbL5\\_Gc/edit#slide=id.g2b46f2cb556\\_0\\_0](https://docs.google.com/presentation/d/1eRJwzbdSv71e9Ou9yeqziZrz1UagwX8B1kL4TbL5_Gc/edit#slide=id.g2b46f2cb556_0_0) (pristupljeno 7. septembra 2024.)

- [39] L. Martin, „RAG from scratch: Query Translation (Multi-Query),” Google Slides, [https://docs.google.com/presentation/d/15pWydIszbQG3Ipur9COftTduutTZm6ULdkkyX-MNrY8I/edit#slide=id.g268cd4ba153\\_0\\_0](https://docs.google.com/presentation/d/15pWydIszbQG3Ipur9COftTduutTZm6ULdkkyX-MNrY8I/edit#slide=id.g268cd4ba153_0_0) (pristupljeno 7. septembra 2024.)
- [40] L. Martin, „RAG from scratch: Query Translation (RAG Fusion),” Google Slides, [https://docs.google.com/presentation/d/1EwykmdVSQqlh6XpGt8APOMYp4q1CZqgeclAx61pUcjI/edit#slide=id.g268cfa48f45\\_0\\_0](https://docs.google.com/presentation/d/1EwykmdVSQqlh6XpGt8APOMYp4q1CZqgeclAx61pUcjI/edit#slide=id.g268cfa48f45_0_0) (pristupljeno 8. septembra 2024.)
- [41] L. Martin, „Rag from scratch: Query translation (decomposition),” Google Slides, [https://docs.google.com/presentation/d/1O97KYrsmYEmhpQ6nkvOVAqQYMJvIaZulGFgmz4cuuVE/edit#slide=id.g268fdc1fda2\\_0\\_0](https://docs.google.com/presentation/d/1O97KYrsmYEmhpQ6nkvOVAqQYMJvIaZulGFgmz4cuuVE/edit#slide=id.g268fdc1fda2_0_0) (pristupljeno 8. septembra 2024.)
- [42] H. S. Zheng et al., „Take a step back: Evoking reasoning via abstraction in large language models,” arXiv.org, <https://arxiv.org/abs/2310.06117> (pristupljeno 9. septembra 2024.)