Izrada chatbota baziranog na OpenAI API-ju uz primjenu RAG

Adžić, Samuel

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

Izrada chatbota baziranog na OpenAI API-ju uz primjenu RAG

Završni rad

Samuel Adžić

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P: Obrazac za ocjenu završnog rada na sveučilišnom prijediplomskom studiju

Ocjena završnog rada na sveučilišnom prijediplomskom studiju

Ime i prezime pristupnika:	Samuel Adžić
Studij, smjer:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	R4599, 27.07.2021.
JMBAG:	0165089093
Mentor:	doc. dr. sc. Hrvoje Leventić
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Izrada chatbota baziranog na OpenAI API-ju uz primjenu RAG
Znanstvena grana završnog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada:	Istražiti i opisati funkcioniranje RAG koncepta (Retrieval Augmented Generation), objasniti vektorske prostore i vektor embedding. Implementirati chatbot koji će koristiti RAG za komunikaciju s korisnikom, gdje će sustav pretraživati dokumente koje će prethodno pripremiti aplikacija izrađena za tu primjenu. (Rezervirano: Samuel Adžić, 3. sveučilišni)

Datum prijedloga ocjene završnog rada od strane mentora:	18.09.2024.
Prijedlog ocjene završnog rada od strane mentora:	Izvrstan (5)
Datum potvrde ocjene završnog rada od strane Odbora:	25.09.2024.
Ocjena završnog rada nakon obrane:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio sveučilišni prijediplomski studij:	27.09.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 27.09.2024.

Ime i prezime Pristupnika:	Samuel Adžić
Studij:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R4599, 27.07.2021.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Izrada chatbota baziranog na OpenAI API-ju uz primjenu RAG**

izrađen pod vodstvom mentora doc. dr. sc. Hrvoje Leventić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD

Živimo u svijetu u kojem tehnologija napreduje iz sata u sat, nova otkrića i spoznaje mijenjaju način na koji živimo i način na koji razmišljamo. Jedno od novijih otkrića i izuma koji upravo to donose su *chatbotovi*. *Chatbot* je računalni program dizajniran za simulaciju razgovora s čovjekom tako da koristi prirodni jezik putem tekstualnih ili glasovnih interakcija. *Chatbotovi* su postali neizostavno pomagalo u različitim industrijama, od korisničke podrške i osobnih asistenata do obrazovanja i zdravstva. Njihova sposobnost da automatiziraju komunikaciju, pruže informacije i podršku korisnicima te poboljšaju korisničko iskustvo čini ih ključnim u razvoju i planiranju strategija. *Chatbotovi* su evoluirali od jednostavnih pravila i skripti do sofisticiranih modela temeljenih na umjetnoj inteligenciji, sposobnih razumjeti i generirati prirodni jezik s visokom razinom točnosti. Jedan od naprednih pristupa u razvoju *chatbota* je primjena modela za generaciju teksta u kombinaciji s metodama pretraživanja informacija, poznat pod nazivom Retrieval-Augmented Generation (RAG). Ovaj pristup omogućava *chatbotovima* ne samo generiranje odgovora na temelju naučenih obrazaca, već i pretraživanje relevantnih informacija iz vanjskih izvora podataka, čime se značajno poboljšava točnost i relevantnost odgovora. Cilj ovog rada je detaljno istražiti i implementirati *chatbot* baziran na OpenAI API-ju uz primjenu RAG metodologije. OpenAI pruža napredne generativne modele poput GPT-3, koji su sposobni generirati visokokvalitetan tekst, dok RAG metodologija omogućava pretraživanje i integraciju dodatnih informacija iz velikih skupova podataka. Kombinacija ovih tehnologija može stvoriti moćnog *chatbota* sposobnog odgovoriti na kompleksne upite s visokom razinom preciznosti. Uvođenjem RAG metodologije, *chatbotovi* postaju mnogo više od jednostavnih alatki za generiranje teksta – postaju sofisticirani sustavi za pretraživanje i pružanje informacija, sposobni zadovoljavati složene potrebe korisnika u različitim kontekstima.

1.1. Zadatak završnog rada

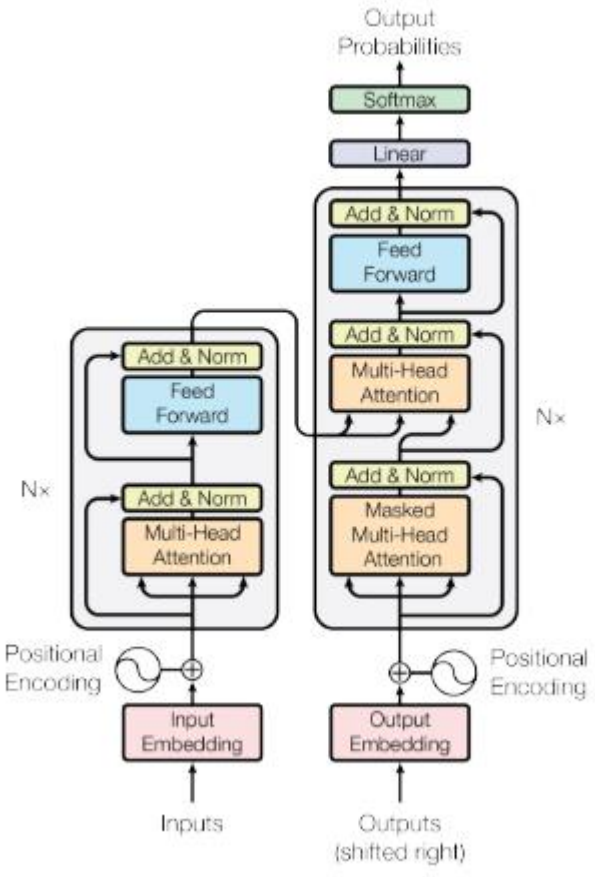
Istražiti i implementirati *chatbot* baziran na OpenAI API-ju uz primjenu Retrieval-Augmented Generation (RAG) metodologije. Cilj je demonstrirati kako kombinacija generativnih modela i metoda pretraživanja informacija može poboljšati točnost i relevantnost odgovora *chatbota*. Kroz ovaj rad će se razviti i evaluirati sustav koji koristi RAG pristup za pružanje sofisticiranih i informiranih odgovora na korisničke upite.

2. TEORIJSKE OSNOVE

2.1. Što su transformeri

Transformeri su napredni modeli dubokog učenja, prvi put predstavljeni u radu *Attention is All You Need* [1]. . Oni su postali temeljna tehnologija za mnoge zadatke obrade prirodnog jezika, uključujući strojno prevođenje, sažimanje teksta, generiranje teksta i odgovaranje na pitanja. Transformeri su revolucionirali obradu prirodnog jezika zbog svoje sposobnosti da obrade sekvencijalne podatke efikasno i s velikom preciznošću, zahvaljujući svom mehanizmu pažnje (engl. *attention*).

2.1.1. Arhitektura transformera



Sl. 2.1. Transformer - Arhitektura modela [1]

Arihitektura standardnog transformera je prikazana na slici 2.1. i sastoji se od dva glavna dijela: enkodera (lijeva strana) i dekodera (desna strana).

Prvi korak unutar enkodera jest pretvaranje ulaznih riječi ili tokena u vektorske reprezentacije, proces koji nazivamo *input embedding*. Budući da transformeri ne koriste sekvencijalne informacije na način kako to čine rekurentne neuronske mreže (RNN)(engl. *Recurrent neural network*), tim vektorskim reprezentacijama dodaje se pozicijsko kodiranje kako bi se modelu omogućilo učenje o redoslijedu riječi unutar rečenice. Enkoder se sastoji od NNN identičnih slojeva. Svaki sloj uključuje mehanizam višestruke pažnje (*Multi-Head Attention*), koji omogućuje modelu da istovremeno procjenjuje važnost svake riječi u odnosu na sve ostale riječi u sekvenci. Osim toga, skokovi (engl. *Residual Connections*) i slojevi normalizacije (engl. *Layer Normalization*) doprinose stabilnosti treniranja modela, dok potpuno povezana mreža (engl. *Feed Forward*) obrađuje svaku poziciju unutar sekvencije pojedinačno [1].

U dekoderu, izlazne riječi ili tokeni također se pretvaraju u vektorske reprezentacije. Kao i kod enkodera, pozicijsko kodiranje se dodaje tim vektorskim reprezentacijama. Dekoder se sastoji od NNN identičnih slojeva, gdje svaki sloj uključuje maskiranu višestruku pažnju (engl. *Masked Multi-Head Attention*) koja sprječava model da gleda buduće pozicije u sekvenci. Također, sloj višestruke pažnje koristi izlaz enkodera kako bi dekoder mogao fokusirati pažnju na relevantne dijelove ulazne sekvence [2]. Skokovi i slojevi normalizacije stabiliziraju treniranje, dok potpuno povezana mreža obrađuje svaku poziciju u sekvenci.

Izlazni sloj uključuje linearni sloj koji pretvara vektorske reprezentacije u *logite* (ne-normalizirane vjerojatnosti) za svaki token u vokabularu. Nakon toga, *softmax* funkcija pretvara *logite* u vjerojatnosti, gdje svaka vjerojatnost odgovara određenom tokenu u vokabularu [2].

2.1.2. Proces rada transformera

Rad transformera započinje pretvorbom ulazne sekvence riječi ili tokena u vektorske reprezentacije putem procesa koji se naziva *input embedding*. Ove vektorske reprezentacije pružaju modelu informacije o značenju riječi, no ne sadrže podatke o njihovom redoslijedu u rečenici. Kako bi se modelu omogućilo prepoznavanje sekvencijalnih odnosa među riječima, vektorske reprezentacije prolaze kroz fazu pozicijskog kodiranja(engl. *Positional Encoding*). Pozicijsko kodiranje dodaje informacije o poziciji svake riječi u rečenici, što omogućuje modelu razlikovanje redoslijeda i odnosa među riječima [1]. Dekoder, koji generira izlaznu sekvencu, koristi prethodno generirane tokene kao ulaz. Kako bi se spriječilo da dekoder "vidi" buduće tokene u sekvenci, koristi se maskirana višestruka pažnja (engl. *Masked Multi-Head Attention*), koja osigurava da dekoder generira jedan token u trenutku, uzimajući u obzir informacije iz prethodno generiranih tokena i relevantne dijelove ulazne sekvence [3]. Sloj pažnje dekodera

koristi izlaz enkodera kako bi fokusirao pažnju na ključne dijelove ulazne sekvence, čime se omogućava bolje razumijevanje konteksta i poboljšava kvaliteta generiranih odgovora [2]. Konačni slojevi dekodera uključuju linearni sloj koji pretvara vektorske reprezentacije u *logite*, što su ne-normalizirane vjerojatnosti za svaki token u vokabularu. Nakon toga, *softmax* funkcija pretvara te *logite* u vjerojatnosti, pri čemu svaki token iz vokabulara dobiva svoju vjerojatnost. Token s najvećom vjerojatnošću odabire se kao sljedeći u sekvenci [1][3].

2.2. Što su embeddingzi

Embeddingzi su ključni koncept u prirodnoj obradi jezika (engl. *Natural Language Processing*) i dubokom učenju. *Embeddingzi* su numeričke reprezentacije stvarnih objekata koje sustavi strojnog učenja (engl. *Machine learning*) i umjetne inteligencije (engl. *Artificial Intelligence,* AI) koriste kako bi razumjeli složena i kompleksna znanja i odnose poput ljudi [4]. Primjerice, računalni algoritam zna da je razlika između dva i tri jednaka jedan, što ukazuje na blisku povezanost ta dva broja u usporedbi sa razlikom brojeva dva i dvjesto. Međutim, podaci iz stvarnog, svakidašnjeg svijeta uključuju složenije i kompleksnije odnose. Ptičje gnijezdo i medvjeđi brlog su analogni parovi, dok su dan i noć suprotnosti. *Emeddingzi* pretvaraju te objekte i njihov odnos i vezu u numeričke reprezentacije razumljive računalu. Cijeli proces je automatiziran, pri čemu noviji sustavi umjetne inteligencije sebi sami stvaraju *embeddinge* tijekom procesa treninga i koriste ih po potrebi za obavljanje novih zadataka [4].

2.2.1. Princip rada embeddinga

Princip rada *embeddinga* uključuje nekoliko koraka koji omogućavaju transformaciju diskretnih podataka(riječi) u kontinuirane vektorske reprezentacije. Prvi korak je prikupljanje podataka koji će se koristiti u treningu modela. Količina podataka može varirati od nekoliko tisuća do nekoliko stotina milijuna riječi, o tome će i ovisiti sposobnost modela da razlučiva odnose među objektima. Drugi je korak priprema podataka na način da se razdvajaju na pojedine riječi ili fraze od kojih svaka ima jedinstveni identifikator. Treći je korak treniranje *embedding* modela. Postoji nekoliko pristupa poput Word2Vec, GloVe (Global Vectors for Word Representation), Transformerski modeli(BERT, GPT)… Neki od njih će biti detaljnije objašnjeni u nastavku ovog rada. Četvrti je korak generiranje vektorskih reprezentacija u kojem za svaku riječ stvara visoko-dimenzionalni vektor gdje svaka dimenzija kodira određeni aspekt značenja riječi. Peti je korak učenje odnosa među riječima, to jest riječi koje su semantički slične imaju slične vektorske reprezentacije. Nakon ovih koraka *embeddingzi* su spremni za korištenje u raznim zadacima obrade prirodnog jezika.

2.3. Metrike vektorske sličnosti: skalarni produkt i kosinusna sličnost

Vektorske sličnosti su ključne za razumijevanje odnosa između *embeddinga* u prostoru.

2.3.1. Skalarni produkt

Skalarni produkt(engl. *dot product*) je jako često korištena i popularna metrika sličnosti. Skalarni produkt uzima dva vektora istih dimenzija i vraća skalarni broj. Vrijednost skalarnog produkta mogu varirati od negativne beskonačnosti do pozitivne beskonačnosti pri čemu negativne vrijednosti ukazuju na suprotne smjerove, pozitivne vrijednosti na iste smjerove, a vrijednost nula kada su vektori okomiti jedan na drugog. Veća vrijednost skalarnog produkta ukazuje na veću sličnost [5].

Za dva vektora a i b u n-dimenzionalnom prostoru, skalarni se produkt definira kao:

$$a \cdot b = \sum_{i=1}^{n} a_i b_i \tag{2-1}$$

Gdje su $a_i \; i \; b_i$ komponente vektora a i b.

Geometrijski gledano, skalarni produkt se može izraziti kao:

$$a \cdot b = \|a\|\|b\| \cos\theta \tag{2-2}$$

Gdje su $\|a\|\|b\|$ norme vektora, θ kut između vektora a i b.

Svojstva skalarnog produkta

- Komutativnost

  $$a \cdot b = b \cdot a$$

- Distributivnost

  $$a \cdot (b+c) = a \cdot b + a \cdot c$$

- Bilinearnost

  $$(ka) \cdot b = k(a \cdot b)$$

- Ortogonalnost

  Ako su dva vektora ortogonalna:

  $$a \cdot b = 0$$

2.3.2. Kosinusna sličnost

Kosinusna sličnost(engl. *cosine similarity*) je mjera sličnosti između dva vektora u višedimenzionalnom prostoru, koja uzima u obzir smjer vektora, ali ne i njihovu magnitudu. Ova metrika je posebno korisna kada se želi uspoređivati sličnosti između vektora bez obzira na njihovu duljinu ili veličinu. Vrijednost kosinusne sličnosti (cos(θ)) se kreće od -1 (nije slično) do +1 (vrlo slično). Kod izračuna ove mjere sličnosti posebnu važnost ima kut između vektora, dok zanemaruje normu istih. Za izračun kosinusne sličnosti između dva vektora jednostavno se podijeli skalarni produkt oba vektora s umnoškom njihovih duljina[5].

Za dva vektora a i b u n-dimenzionalnom prostoru, kosinusna se sličnost definira kao:

$$cosine\ similarity(a, b) = \frac{a \cdot b}{\|a\|\|b\|} \tag{2-3}$$

Gdje je $a \cdot b$ skalarni produkt vekotra a i b, a $\|a\|\ i\ \|b\|$ su norme vektora a i b.

Geometrijski gledano kosinusna sličnost mjeri kosinus kuta θ između vektora a i b u višedimenzionalnom prostoru.

2.3.3. Sličnosti i različitosti

Glavna razlika između kosinusne sličnosti i skalarnog produkta je u tome što kosinusna sličnost usredotočuje se na usmjerenost vektora, dok skalarni produkt uzima u obzir i usmjerenost i veličinu vektora. Kosinusna sličnost je često preferirana kada je važno usporediti smjerove vektora bez obzira na njihovu duljinu, dok se skalarni produkt koristi kada je važna i orijentacija i "intenzitet" zajedničkih karakteristika vektora.

2.4. Osnovni RAG principi

RAG *pipeline* odnosi se na koncept korištenja Retrieval-Augmented Generation (RAG) u procesu prirodnog jezičnog procesiranja. RAG je tehnika koja poboljšava jezične modele uključivanjem dohvaćenih točnih i ažuriranih informacija iz autoritativne vanjske baze znanja. Ovo se kombinira s tehnikama oblikovanja upita, što omogućava generiranje visokokvalitetnih, aktualnih i točnih odgovora, čak i na pitanja s kojima se model nikad prije nije susreo, a također smanjuje pojavu grešaka generiranih modelom, poznatih kao 'halucinacije'.

2.4.1. Priprema i prikupljanje dokumenata

U ovoj fazi se vrši priprema dokumenata i podataka koji će se koristiti kao izvor informacija u fazi dohvaćanja. Dokumenti mogu biti različitih formata kao na primjer PDF, TXT datoteke, CSV… Dokumenti se učitavaju kako bi bili spremni za daljnju obradu, odnosno dijeljenje istih na manje dijelove, obično rečenice ili paragrafe koristeći heurističke metode za podjelu. Nakon toga dokument se dijeli na segmente prigodne za pohranu i pretraživanje, koji se popularno nazivaju *chunkovi,* a proces razdvajanja teksta na takve segmente naziva se *chunking*. Svaki se segment transformira u vektorsku reprezentaciju s pomoću *embedding* modela kako bi bio spreman za pohranu u bazu podataka.

2.4.2. Faza dohvaćanja informacija (Retrieval)

Korisnikov upit pokreće proces pretraživanja i generiranja odgovora. Upit se najprije pretvara u vektorsku reprezentaciju korištenjem *embedding* modela, kako bi mogao biti korišten za pretragu u bazi podataka. Baza podataka, koja pohranjuje vektorske reprezentacije dokumenata, po primitku vektorske reprezentacije korisničkog upita pretražuje najrelevantnije segmente (*chunkove*) na temelju jedne od mjera sličnosti, poput skalarnog produkta ili kosinusne sličnosti. Na temelju tih mjerenja vraćaju se najrelevantniji segmenti teksta. Metoda *Top-k chunks* označava odabir *k* najbližih ili najsličnijih dijelova u odnosu na korisnički upit, predstavljen njegovom vektorskom reprezentacijom. Odabrani dijelovi zatim se kombiniraju kako bi formirali kontekst koji će biti korišten u fazi generiranja odgovora.

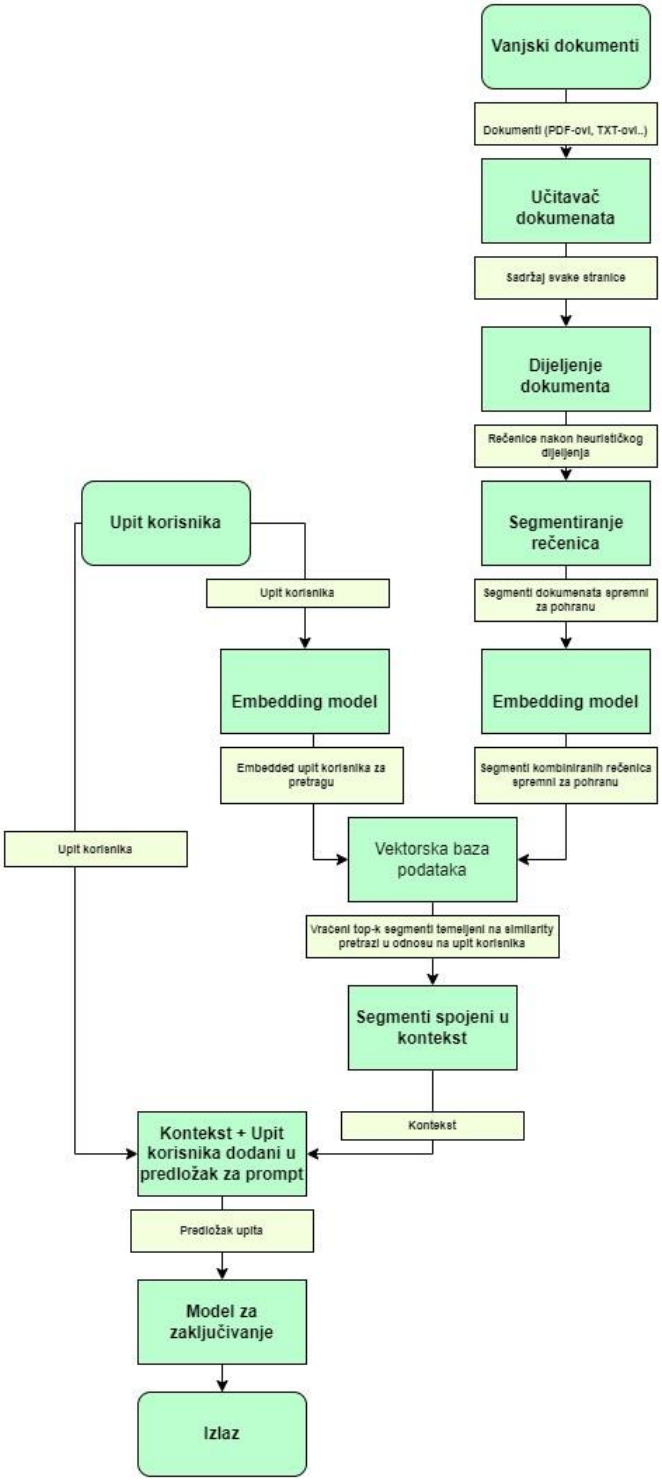
2.4.3. Faza obogaćivanja informacija (Augmentation)

Nakon što su najrelevantniji segmenti odabrani iz vektorske baze, oni se kombiniraju kako bi se kreirao kontekst. Originalni se upit korisnika dodaje kontekstu. To je vrlo bitan korak jer model mora razumjeti specifične informacije koje korisnik traži. Nakon toga se kontekst i korisnički upit se integriraju u jedinstveni predložak upita. Ovaj predložak sadržava sve potrebne informacije koje će generativni model koristiti za kreiranje odgovora.

2.4.4. Faza generiranja odgovora (Generation)

Predložak upita koji uključuje kontekst i upit korisnika se šalje generativnom modelu. Generativni model, koji može biti baziran na raznim arhitekturama, najčešće se radi o transformer arhitekturi kao što je GPT-3, koristi predložak upita za generiranje odgovora. Model koristi unaprijed utrenirane reprezentacije jezika kako bi interpretirao što bolji odgovor. Konačni se generirani odgovor vraća korisniku kroz korisničko sučelje. Oblik odgovora može varirati ovisno o aplikaciji

koja ga prikazuje, to jest može biti u tekstualnom obliku ili neki drugi interaktivni element poput slike, grafa, videa…



Slika 2.4.: Graf prikazuje princip rada RAG *pipeline-a*

3. PREGLED PODRUČJA

3.1. Metode embeddinga i njihovi rezultati

Metode *embedinga* riječi u obradi prirodnog jezika se mogu uglavnom svrstati u dvije kategorije: *Embedding* na temelju frekvencije i *Embedding* na temelju predviđanja. Postoje i druge tehnike *embeddinga* koje ne spadaju niti u jednu od ovih kategorija.

3.1.1. Metode embeddinga na temelju frekvencije

Embeddingzi na temelju frekvencije su reprezentacije riječi u gomili teksta zasnovane na njihovoj frekvenciji pojavljivanja i odnosu s drugim riječima. Dvije su uobičajene tehnike za generiranje *embeddinga* na temelju frekvencije: TF-IDF i matrica su-pojavljivanja(*the co-occurrence matrix*).

TF-IDF (Term Frequency-Inverse Document Frequency) mjeri koliko se neki termin često pojavljuje u dokumentu. Izračunava se kao broj pojavljivanja termina u dokumentu podijeljen s ukupnim brojem termina u dokumentu.

IDF (Inverse Document Frequency) mjeri koliko je termin jedinstven u zbirci dokumenata. Izračunava se kao logaritam ukupnog broja dokumenata podijeljen s brojem dokumenata koji sadrže taj termin.

BM25 je algoritam zasnovan na TF-IDF (Term Frequency-Inverse Document Frequency) modelu pretraživanja, koji je optimiziran za rangiranje dokumenata prema njihovoj relevantnosti za određeni upit. BM25 pripada obitelji funkcija poznatih kao *Okapi* BM25 i često se koristi u sustavima za pretraživanje informacija i tražilice jer poboljšava preciznost rangiranja dokumenata [6]. BM25 primjenjuje logaritamsku normalizaciju učestalosti pojmova kako bi spriječio da prečesto pojavljivanje riječi u dokumentu previše utječe na konačnu ocjenu dokumenta.

TF-IDF težina je produkt je TF i IDF vrijednosti nekog dokumenta. Izrazi s visokim TF-IDF težinama se smatraju važnijima u kontekstu dokumenta.

Co-occurrence matrix (matrica) je pristup u kojem oko svake riječi se definira kontekstni prozor, na primjer rečenica ili paragraf. Nakon toga se konstruira matrica gdje redovi i stupci predstavljaju riječi, a svaka ćelija sadrži koliko se puta par riječi pojavi zajedno unutar kontekstnog prozora. Tehnike poput singularnog vrijednosnog razlaganja (SVD) mogu se primijeniti za smanjenje dimenzionalnosti matrice i dohvaćanja latentnih semantičkih odnosa između riječi. Rezultirajući se *embeddingzi* mogu koristiti za mjerenje sličnosti između riječi na temelju njihovih *co-occurence* matrica.

Obje metode su vrijedne za hvatanje važnih odnosa između riječi u korpusu te se mogu koristiti za izgradnju reprezentacija riječi koje se mogu primijeniti u raznim zadacima u procesu obrade prirodnog jezika (engl. *Natural Language Processing*, NLP).

3.1.2. Metode embeddinga na temelju predviđanja

Metode *embeddinga* na temelju predviđanja se generiraju treniranjem modela za predviđanje riječi u danom kontekstu. Neke od popularnijih *embedding* metoda na temelju predviđanja uključuju Word2Vec (Skip-gram i CBOW), FastText i Global Vectors for Word Representation (GloVe).

Word2Vec je tehnika *embeddinga* koja radi na način da analizira veliku količinu tekstualnih podataka, kao što na primjer članci, knjige ili web stranice. Iz tih tekstova identificira kontekst svake riječi, odnosno riječi koje se pojavljuju prije i poslije nje u rečenici ili odlomku teksta [7]. Dva glavna modela Word2Vec-a su Skip-gram i CBOW.

Skip-gram je model koji predviđa okolne riječi s obzirom na ciljanu riječ. U cilju mu je trenirati model tako da, za svaku riječ u korpusu, može predvidjeti riječi koje se pojavljuju u njenom kontekstu. Model zapravo uči predstavljati riječi koje se često pojavljuju zajedno u sličnim vektorima [8]. Ova značajka omogućuje modelu da hvata semantičke odnose među riječima i zbog toga je dosta efikasan.

CBOW (Continuous Bag of Words) je model koji predviđa ciljanu riječ na temelju njenog konteksta. Umjesto predviđanja okolnih riječi, CBOW koristi okolne riječi da predvidi ciljanu [9]. Puno je brži za treniranje u usporedbi sa Skip-gramom jer koristi prosječnu vektorsku reprezentaciju okolnih riječi za predviđanje ciljane riječi. Koristan je u procesu generiranja *embeddinga* za riječi koje se rjeđe pojavljuju.

FastText je tehnika *embeddinga* razvijena od strane FAIR-a(Facebook AI research) koja poboljšava Word2Vec uključivanjem informacija o pod-riječima(sekvence znakova(engl. *character n-grams*)). FastText radi na način da razbija svaku riječ u sekvence znakova i uči vektorske reprezentacije za te sekvence znakove koji se nazivaju n-grami. Vektorska je reprezentacija riječi tada suma vektora njenih n-grama [10]. Ova je tehnika učinkovita pri obradi riječi koje nisu viđene tijekom treninga, te pomaže u prepoznavanju riječi sličnih u smislu pravopisa što je bitno za jezike s bogatom morfologijom.

GloVe (Global Vectors for Word Representation) je model koji su razvili istraživači sa Stanforda, koja koristi globalne statistike pojavljivanja(engl. *co-occurrence*) riječi za generiranje vektorskih reprezentacija riječi. GloVe koristi matricu pojavljivanja koja bilježi koliko često se dvije riječi

pojavljuju zajedno u prozoru konteksta u cijelom korpusu. Modelira omjer učestalosti, što modelu omogućuje da uči semantičke odnose između riječi. Primjenjuje tehnike faktorizacije matrice na matricu su-pojavljivanja kako bi smanjio njenu dimenzionalnost i stvorio kompaktne vektore riječi što omogućuje identifikaciju latentnih semantičkih odnosa među riječima [4]. Ovaj model kombinira neke od ideja iz drugih *embedding* pristupa kao što su Word2Vec iz kojeg koristi kontekstualne informacije za učenje vektora riječi, te TF-IDF iz kojeg koristi matricu su-pojavljivanja riječi.

3.1.3. Rijetki i gusti vektori

Podjela *embeddinga* na *sparse* (rijetke) i *dense* (guste) vektore odnosi se na način na koji su riječi, rečenice ili dokumenti prikazani u vektorskom prostoru za različite zadatke obrade prirodnog jezika.

Sparse embedding su vektori u kojima je većina vrijednosti nula. Ovaj pristup čest je u ranim metodama vektorizacije riječi, poput spomenutog tf-idf. Svaka riječ iz vokabulara predstavlja jednu dimenziju vektora, što znači da kako vokabular raste, raste i dimenzionalnost vektora. Zbog toga *sparse* vektori često imaju vrlo veliku dimenzionalnost, s većinom komponenti postavljenih na nulu [11]. Ovaj način predstavljanja teksta jednostavan je i intuitivan, ali ima nekoliko ključnih nedostataka. *Sparse* vektori ne uzimaju u obzir kontekst riječi i njihovu semantičku sličnost. Osim toga, zbog velike dimenzionalnosti, *sparse* vektori postaju vrlo neučinkoviti za rad s velikim vokabularima, a kako broj riječi raste, modeli koji se temelje na *sparse* vektorima postaju sporiji i zahtjevniji za memoriju.

Za razliku od *sparse* vektora, *dense* vektori omogućuju modelima da prepoznaju semantičku sličnost između riječi. Ovi vektori omogućuju modelu da nauči značenje riječi kroz njihove odnose s drugim riječima, čineći *dense* vektore moćnim alatom za zadatke poput strojnih prijevoda, generiranja teksta i analize sentimenta [11]. Jedna od ključnih prednosti *dense* vektora je ta što su sposobni hvatati kontekst riječi, a ne samo njihovu prisutnost. *Dense* vektori također su mnogo učinkovitiji u pogledu memorije i računalnih resursa. Svaka dimenzija *dense* vektora ne predstavlja određenu riječ, već apstraktnu značajku koja opisuje semantičke odnose između riječi. Ova značajka omogućuje modelima da bolje generaliziraju i uče o odnosima između riječi, što nije moguće sa *sparse* vektorima. *Dense* vektori, posebno u naprednim modelima poput BERT-a, mogu razumjeti i razlikovati različita značenja iste riječi u različitim kontekstima, što *sparse* vektori ne mogu [12].

3.1.4. Druge metode embeddinga

GPT (Generative Pre-trained Transformer) je model *embeddinga* temeljen na transformer arhitekturi, razvijen od strane OpenAI. GPT koristi arhitekturu transformer dekodera. Ključni koraci u implementaciji GTP modela jesu pred trening(engl. *pre-training*) i *fine* trening(engl. *fine-tuning*). U pred treningu GPT je treniran na ogromnom korpusu tekstualnih podataka prikupljenih s interneta. Model se zatim trenira pomoću zadatka predviđanja slijedeće riječi u tekstu. Model uči razumjeti jezik i kontekst bez eksplicitno označenih podataka. *Fine* trening je faza koja dolazi nakon u kojoj se GPT može trenirati za specifične zadatke koristeći nadzirano učenje. Ono uključuje skupljanje skupa podataka za obavljanje specifičnog zadatka i treniranje na specifičnim, manjim skupovima podataka kako bi prilagodio svoje generativne sposobnosti potrebama zadatka. Tijekom godina je izlazilo je nekoliko verzija GPT sa različitim značajkama. GPT-1 je verzija izašla dvije tisuće osamnaeste godine. Sadrži dvanaest slojeva transformer dekodera, te ima sto deset milijuna parametara, što znači da model ima sto deset milijuna pojedinačnih težina(numeričke vrijednosti koje povezuju neurone između slojeva u neuronskoj mreži) koje se treniraju kako bi se postigle najbolje performanse. Trening je obavljen na *BookCropusu,* to jest velikom skupu knjiga koji sadrži oko sedam tisuća naslova. Prvi u seriji modela koji je pokazao da trenirani jezični modeli mogu učinkovito generirati smislen tekst [13]. GPT-2 verzija je izašla dvije tisuće devetnaeste godine. Sadrži četrdeset osam slojeva transformer dekodera, ima jednu i pola milijardnu parametara. Treniran na raznolikom korpusu koji sadrži oko osam milijuna dokumenata. Značajno je nadmašio prvu verziju u generiranju teksta, razumijevanju konteksta i izvođenju različitih zadataka u obradi prirodnog jezika [13]. GPT-3 je verzija objavljena dvije tisuće dvadesete. Sadrži devedeset šest slojeva transformer dekodera. Ima 175 milijardi parametara. Treniran je na korpusu podataka koji obuhvaća tekstove s cijeloga interneta. Model koji je postavio nove standarde u generiranju i razumijevanju teksta. Može obavljati zadatke iz raznih područja s visokim postotkom točnosti i uspješnosti [14] . GPT-4 je noviji model objavljen dvije tisuće dvadeset treće godine. Specifičnosti modela nisu službeno objavljene, ali nagađa se kako model sadrži sto dvadeset slojeva novijih, naprednijih transformer dekodera. Pretpostavke su i nagađanja, također da gotovo dva bilijuna parametara mu daju sposobnost da uči još složenije obrasce i generira još sofisticiraniji tekst. Najnoviji modeli su GPT-4o i GPT-4o-mini. GPT-4o je model objavljen dvije tisuće dvadeset četvrte godine i predstavlja značajan napredak u obradi prirodnog jezika. Službenih podataka o arhitekturi nema, ali ostvareno je da prepozna i generira vrlo složene obrasce u tekstu omogućavajući mu da stvara izrazito sofisticirane odgovore i da se još bolje prilagodi specifičnim potrebama korisnika [15]. GPT-4o-mini je optimizirana verzija

GPT-4o modela, namijenjena korisnicima koji zahtijevaju visoku učinkovitost i točnost, ali s manje resursa.

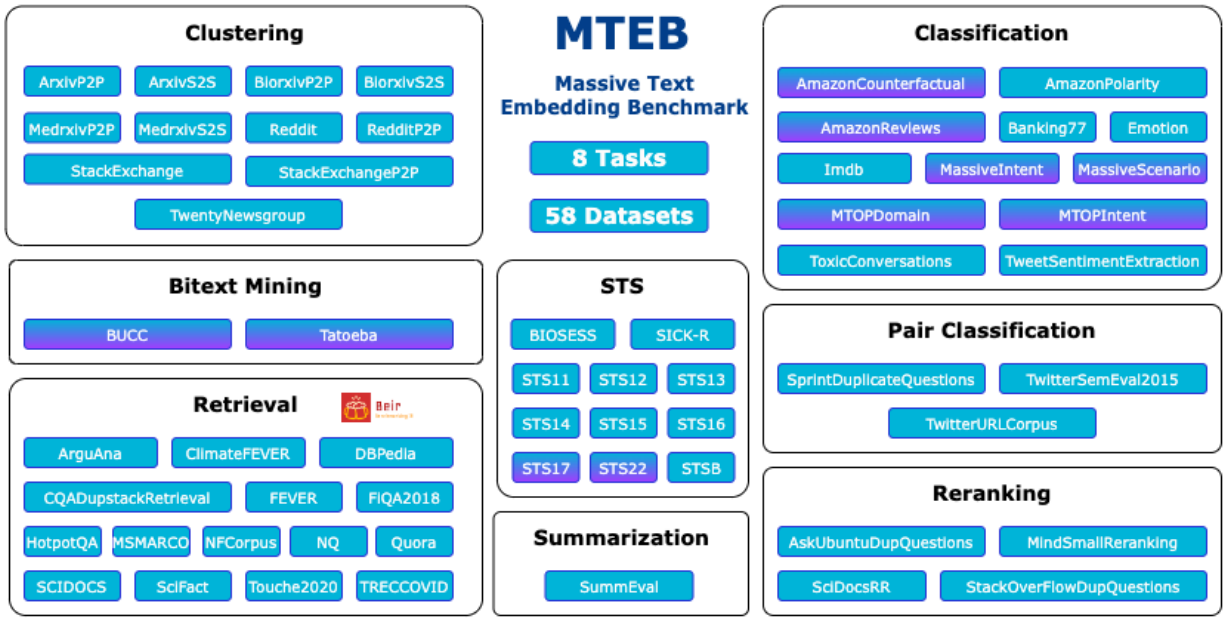
ELMO (Embeddings from Language Models) je tehnika za stvaranje *embeddinga* osmišljena od strane istraživača sa Sveučilišta Washington i Allen Institute for AI. ELMo koristi duboke bidirekcionalne LSTM mreže za generiranje vektorskih reprezentacija riječi. LSTM (engl. *Long Short-Term Memory*) su posebna verzija RNN mreže, dizajnirana za rješavanje problema dugoročnog pamćenja, uvodeći posebne čvorove nazvane ćelije koje imaju unutarnju strukturu koja omogućuje kontrolu toka informacija kroz mehanizme koji se nazivaju vrata (engl. *gates*). Za razliku od ostalih metoda, ELMo generira reprezentacije koje ovise o cijeloj rečenici. Pomoću spomenutih LSTM mreža, model uzima u obzir i prethodni i budući kontekst prilikom generiranja *embeddinga*. Princip rada je u početku isti kao i kod ostalih metoda, gdje se svaka riječ pretvara u reprezentacijski vektor. Međutim, nakon toga ti vektori prolaze kroz LSTM slojeve na način da jedan LSTM obrađuje tekst u smjeru od početka do kraja rečenice (*forward* LSTM), dok drugi obrađuje tekst u obrnutom smjeru (*backward* LSTM). Reprezentacije iz oba sloja kombiniraju se kako bi se generirao konačni *embedding* za svaku riječ. Upravo zbog ovog svojstva, model uzima u obzir složenije jezične obrasce i kontekst [16].

BERT (Bidirectional Encoder Representations from Transformers) razvijen je od strane Google *AI Language* tima. Slično kao kod ELMo-a, BERT koristi princip bidirekcionalnosti, odnosno koristi transformer arhitekturu koja omogućuje bidirekcionalno učenje, što znači da može razumjeti svaku riječ u rečenici uzimajući u obzir i lijevi i desni kontekst. Svaki transformer blok sadrži *self-attention* mehanizam i *feedforward* mrežu, koji su ključni za učinkovito modeliranje dugih ovisnosti u tekstualnim podacima. BERT je treniran na ogromnom skupu podataka, a nakon pretreniranja, može se dalje prilagoditi za izvršavanje specifičnih zadataka putem *fine-tuninga* [17].

3.2. MTEB

MTEB (engl. *Massive Text Embedding Benchmark*) predstavlja sveobuhvatnu platformu za evaluaciju performansi različitih *text embedding* modela na raznovrsnim zadacima obrade prirodnog jezika. Ovaj *benchmark* sustav omogućuje standardiziranu usporedbu modela pružajući alate za ocjenjivanje i unaprjeđenje modela. MTEB pokriva širok raspon zadataka obrade prirodnog jezika koji testiraju različite aspekte performansi *embedding* modela. Ti zadaci uključuju procjenu modela u zadacima kao što su klasifikacija tema, analiza sentimenta i kategorizacija dokumenata. Cilj je ocijeniti koliko dobro model može prepoznati i razvrstati

tekstualne podatke u predodređene kategorije. Evaluacija sposobnosti modela da prepoznaje i kategorizira imenovane entitete kao što su osobe, lokacije ili organizacije u tekstu. Testiranje modela u zadacima koji zahtijevaju generiranje koherentnog i smislenog teksta na temelju zadanog upita. Procjena učinkovitosti teksta s jednog jezika na drugi. Evaluacija modela u zadacima pretraživanja i uspoređivanja tekstualnih podataka na temelju semantičke sličnosti. MTEB je dizajniran na način da bude otvoren i dostupan, također podržavajući evaluaciju različitih vrsta *embedding* modela. MTEB sadrži više od 112 različitih jezika [18]. Korištenjem standardiziranih API-ja, platforma omogućuje jednostavnu integraciju novih modela i tehnika u sustavu ocjenjivanja.



Slika 3.2. Pregled zadataka i skupova podataka u MTEB-u. Višejezični skupovi podataka označeni su ljubičastom bojom [18]

Slika 3.2. prikazuje strukturu Massive Text Embedding Benchmark-a (MTEB) i obuhvaća osam glavnih zadataka sa ukupno pedeset osam različitih skupova podataka. Klasteriranje (*Clustering*) sadrži skupove metoda koji se koriste za grupiranje različitih tekstova u klastere. Klasifikacija (engl. *Classification*) sadrži skupovi podataka koji se koriste za klasifikacijske zadatke gdje se tekstovi razvrstavaju u unaprijed definirane kategorije, poput analize sentimenta i klasifikacije recenzija. Rudarenje bi-teksta (engl. *Bitext Mining*) sadrže skupove podataka koji se koriste za prepoznavanje paralelnih tekstova u različitim jezicima, što je korisno za zadatke strojnog prevođenja. STS (engl. *Semantic Textual Similarity*) sadrži skupove podataka koji se e koriste za procjenu semantičke sličnosti između parova tekstova. Pronalaženje (engl. *Retrieval*) sadrži skupove podataka koji se koriste za zadatke pronalaženja informacija, gdje se traži najrelevantniji

tekstualni odgovor na postavljeno pitanje. Klasifikacija parova (engl. *Summarization*) sadrži skupove podataka koji se koriste zadatke sažimanja teksta, gdje se generira kraći sažetak dugog teksta. Preuređivanje (engl. *Reranking*) sadrži skupove podataka koji se koriste za preuređivanje rangiranih rezultata pretraživanja kako bi se najrelevantniji rezultati prikazali na vrhu [18].

3.2.1. Proces evaluacije u MTEB-u

Proces evaluacije započinje prikupljanjem relevantnih skupova podataka koji će se koristiti za testiranje modela. Ovi skupovi podataka trebaju biti pažljivo odabrani kako bi predstavljali raznovrsne zadatke obrade prirodnog jezika i realne scenarije upotrebe. Bitno je da skupovi podataka dolaze iz različitih domena, te uključuju testove na različitim jezicima. Nakon prikupljanja, podaci prolaze kroz pred procesiranje kako bi bili u standardiziranom formatu pogodnom za evaluaciju. Ono uključuje nekoliko postupaka poput tokenizacije, normalizacije i uklanjanja nepotrebnih podataka. Evaluacija modela uključuje treniranje i testiranje modela na prethodno prikupljenim skupovima podataka. Ovaj je korak ključan za procjenu performansi modela u različitim zadacima. Evaluacija uključuje trening na trening podacima, i testiranja na testnom skupu. Koristi se tehnika unakrsne validacije kako bi se osigurala robusnost i pouzdanost rezultata. Podaci se dijele na nekoliko dijelova, a model se trenira i testira nad različitim kombinacijama ovih dijelova kako bi se smanjila varijabilnost rezultata [18]. Rezultati se prikupljaju i prezentiraju na daljnju analizu i usporedbu u standardiziranom obliku.

3.2.2.MTEB standarde metrike za evaluaciju modela

MTEB točno definira standardne metrike za evaluaciju performansi modela kako bi osigurao konzistentne i usporedive rezultate među različitim modelima i eksperimentima.

Točnost (engl. *Accuracy*) je proporcija ispravno klasificiranih primjera u odnosu na ukupan broj primjera. Računa se kao kvocijent broja ispravno klasificiranih primjera i ukupnog broja primjera [19] . Koristi se kao osnovna metrika za klasifikacijske zadatke. Jednostavna je za razumijevanje i interpretaciju, ali može biti varljiva za neuravnotežene skupove podataka gdje jedna klasa dominira, jer visoka točnost može biti postignuta jednostavnim predviđanjem dominantne klase.

Preciznost (engl. *Precision*) je omjer ispravno predviđenih pozitivnih primjera u odnosu na ukupan broj predviđenih pozitivnih primjera. Računa se kao kvocijent broja ispravno predviđenih pozitivnih primjera i ukupnog broja predviđenih pozitivnih primjera. Preciznost je posebno važna u situacijama gdje je cijena lažno pozitivnih predikcija visoka, kao na primjer u medicinskoj dijagnostici [19]. Korisna je za zadatke gdje su lažno pozitivni primjeri nepoželjni, ali sama po sebi ne pruža informacije o lažno negativnim primjerima stoga se često koristi zajedno sa odzivom.

Odziv (engl. *Recall*) je omjer ispravno predviđenih pozitivnih primjera u odnosu na ukupan broj stvarnih pozitivnih primjera. Računa se kao kvocijent broja ispravno predviđenih pozitivnih primjera i ukupnog broja stvarnih pozitivnih primjera. Odziv je ključan u situacijama gdje je bitno prepoznati što veći broj pozitivnih primjera. Koristan je za zadatke gdje su lažno negativni primjeri nepoželjni, ali ograničen je na to da visoki odziv može dovesti do velikog broja lažno pozitivnih predikcija, što može biti problematično ako nije uravnoteženo s preciznošću [19].

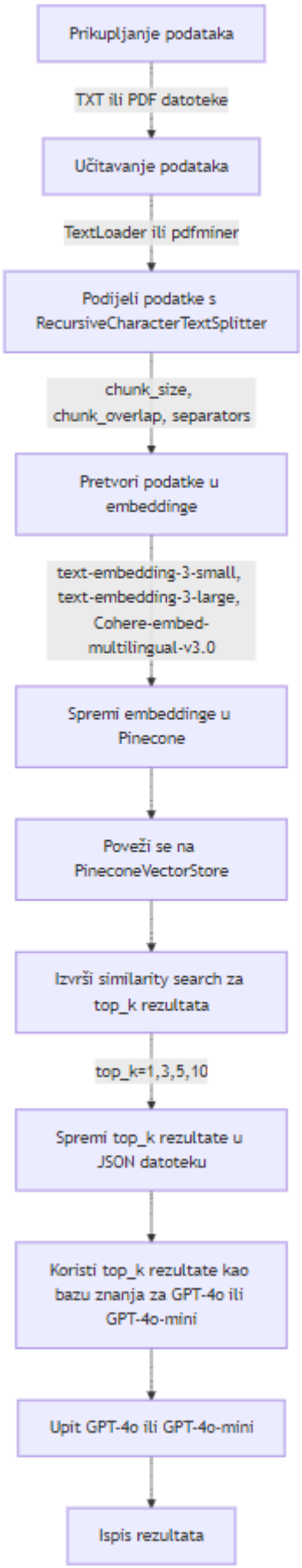
F1-*score* je harmonijska sredina preciznosti i odziva. Pruža uvid u omjer lažno pozitivnih i lažno negativnih predikcija. Računa se kao kvocijent preciznosti pomnožene sa odzivom i preciznosti zbrojene sa odzivom [19]. Posebno je koristan kada postoji potreba za balansiranjem preciznosti i odziva, međutim ne daje informacije o ukupnoj točnosti modela.

4. IMPLEMENTACIJA RAG CHATBOTA

U ovom je poglavlju detaljno objašnjena i opisana implementacija RAG *chatbota*. U početnoj fazi implementacije RAG (Retrieval-Augmented Generation) *chatbota*, ključan korak je pravilna priprema podataka koje će *chatbot* koristiti. Ova priprema podrazumijeva prikupljanje, čišćenje i strukturiranje podataka na način koji omogućuje efikasno pretraživanje i generiranje odgovora. Prikupljanje podataka obuhvaća sakupljanje relevantnih informacija koje će *chatbot* koristiti za odgovaranje na korisničke upite. Podaci mogu dolaziti iz različitih izvora, poput baza znanja, dokumenata, članaka, ili bilo kojeg drugog tekstualnog sadržaja koji je važan za područje na kojem *chatbot* djeluje. Kao baza znanja korištena je službena dokumentacija o studijima Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek dostupna na službenim stranicama fakulteta. Nakon prikupljanja, slijedi faza čišćenja podataka. Ovaj korak uključuje uklanjanje suvišnih informacija, ispravljanje grešaka i osiguravanje konzistentnosti podataka. Čisti podaci su temelj za stvaranje kvalitetnih vektorskih reprezentacija koje se koriste u pretraživanju. Učitati se podatke može iz više tipova datoteka. RAG *chatbot* u ovom završnom radu implementiran je učitavanjem podataka iz dva tipa datoteka, pdf datoteke i txt datoteke. Tekstualne datoteke su jednostavne, neformatirane datoteke koje sadrže čisti tekst. Učitavanje podataka iz ovih datoteka je relativno jednostavno jer ne postoji potreba za složenim *parsiranjem*. Svaka linija ili skup linija može se izravno pretvoriti u *string* i koristiti za izgradnju baza podataka. Međutim, zbog nedostatka formata, tekstualne datoteke nisu prikladne za dokumente s kompleksnom strukturom, poput tablica ili slika. U implementaciji ovog *chatbota* morao je biti korišten UTF 8-bit zbog podataka koji su na hrvatskom jeziku i sadrže znakove s dijakritičkim znakovima i znakove koji se ne nalaze u standardnom ASCII kodiranju koji koristi sedam bitova. PDF datoteke su složenije jer podržavaju različite formate, uključujući tekst, slike, tablice i multimediju. Potrebno je koristiti posebnu biblioteku u jeziku Python za učitavanje ovog tipa datoteka. Korištena je biblioteka PDFMiner jer omogućuje ekstrakciju podataka iz PDF dokumenta, ali i detaljno *parsiranje* i analizu dokumenta, uključujući složenije strukture poput fontova, stilova, tablica i multimedije.[11] Posljednji korak u pripremi podataka je njihovo strukturiranje, to jest organizacija podataka u format koji omogućuje brzu i preciznu pretragu. U ovom slučaju radi se o vektorskim reprezentacijama. U pripremi podataka za njihovu vektorizaciju korišten je RecursiveCharcterTextSplitter iz biblioteke LangChain. Koristi se za razbijanje velikih tekstualnih dokumenata na manje dijelove po određenim kriterijima kao što su paragrafi, razmaci ili drugi specifični znakovi. RecursiveCharacterTextSplitter pokušava podijeliti tekst na osnovu unaprijed definiranih parametara koji kontroliraju njegovo ponašanje kao što su *chunk_size* koji predstavlja

maksimalni broj znakova u jednom fragmentu, *chunk_overlap* predstavlja broj znakova koji se preklapaju između susjednih fragmenata, *separators* predstavlja listu znakova po kojima se tekst dijeli poredanih po prioritetu [20] . Svi ovi parametri će biti mijenjani te je cilj pronaći one najoptimalnije. Nakon što se tekst podjeli na manje fragmente, oni se spremaju u vektorsko spremište, u ovom slučaju radi se o Pinecone vektorskom spremištu. Pinecone je specijalizirana platforma za pohranu i pretraživanje vektorskih podataka, dizajnirana za visokoučinkovito pretraživanje vektora, čak i kada se radi o velikim skupovima podataka. Potrebno je stvoriti indeks i podesiti konfiguraciju koji odgovaraju *embedding* modelu koji će ga koristiti. Modeli koji će biti korišteni i evaluirani su text-embedding-3-small i text-embedding-3-large OpenAI, te Cohere-embed-multilingual-v3.0. Svaki model ima svoju dimenziju, metriku i tip te je potrebno konfigurirati indeks pri stvaranju tako da odgovara modelu koji će ga koristiti. Svi spomenuti modeli su tekst modeli, sa kosinus metrikom. Razlikuju se u dimenzijama gdje text-embedding-3-small ima 1536 dimenzija, text-embedding-3-large 3072 te Cohere-embed-multilingual-v3.0 1024 dimenzije. Nakon što je indeks konfiguriran i stvoren, s funkcijom PineconeVectorStore.from_documents() se stvara vektorsko spremište podataka iz skupa dokumenata. Parametri funkcije su dokumenti koji smo prethodno pripremili za vektorizaciju, *index_name* je ime Pinecone indeksa u koji će se pohraniti vektori, te *embedding* model koji smo prethodno odabrali. Kada se kod izvrši, indeks bi trebao sadržavati određeni broj vektora, ovisno koliki su bili veliki ulazni podaci i koliko su bili veliki fragmenti. Svaki vektor ima svoj jedinstveni identifikator ID, vrijednost, to jest sami podaci koji su spremljeni u obliku niza brojeva. Pinecone nudi i spremanje metapodataka, to su dodatne informacije koje se mogu pohraniti u sami vektor, kao što su na primjer oznake, opisi, datum, izvori, kategorija, autor i slično. Ovi podaci mogu pomoći u filtriranju i upravljanju vektorskim spremištem [21]. Slijedeća faza u implementaciji *chatbota* je *retrieval*. Nakon što su vektori uspješno stvoreni i pohranjeni u Pinecone indeksu, potrebno je omogućiti efikasno dohvaćanje relevantnih informacija iz tih vektora. *Retrieval* faza započinje procesom pretraživanja i dohvaćanja najrelevantnijih dokumenata koji su povezani s upitom korisnika. U ovoj fazi, *chatbot* koristi prethodno generirane vektore kako bi izračunao sličnost između korisničkog upita i pohranjenih dokumenata. Cilj je pronaći one dokumente koji najviše odgovaraju upitu korisnika i vratiti ih kao odgovor. Koristi se biblioteka langchain_pinecone koja s funkcijom PineconeVectorStore omogućava povezivanje s već postojećim Pinecone indeksom. Potom, koristeći funkciju similarity_search(), za svaki korisnički upit se izračunava sličnost i dohvaća se *top_k* najrelevantnijih rezultata. Metoda similarity_search() će ,u slučaju implementacije ovoga *chatbota*, računati kosinusnu sličnost jer korišteni *embedding* modeli su primarno dizajnirani za izračunavanje kosinusne sličnosti između

vektora. Najrelevantniji se rezultati zatim spremaju za daljnje korištenje u JSON datoteku koju će *chatbot* kasnije koristiti kao jedini i primarni izvor za generiranje svojih odgovora. Kako bi se olakšala evaluacija ove faze i kako bi se tekstualni rezultati filtrirali suvišnih razmaka, znakova i slično, korištena je funkcija filter_alphanumeric() iz Pythonove biblioteke *re* koja se koristi za rad s regularnim izrazima. Funkcija prima kao argument *string*. Povratni je tip također *string*, međutim s uklonjenim svim neželjenim znakovima koji nisu alfanumerički, te pretvaranja svih slova u mala slova. Ova faza je ključna u implementaciji *chabota* jer omogućuje pretraživanje velikih količina informacija i vraćanje samo onih najkorisnijih što pridonosi poboljšanju performansi, cijeni i brzini generiranja odgovora. U nastavku implementacije RAG *chatbota*, koristi se jezični model za generiranje odgovora iz konteksta prikupljenog u prijašnjoj fazi implementacije. Postavljeni upit se zajedno sa kontekstom, odnosno relevantnim informacijama šalje OpenAI modelu koji generira konačni odgovor. Za generiranje odgovora koristi se klasa ChatOpenAI iz langchain_openai modula, a sama interakcija s modelom obavlja se preko formatirane poruke HummanMessage(). Ovdje se kombinira upit s relevantnim kontekstom, čime se omogućuje da model koristi najbolje moguće informacije za generiranje odgovora. Konačni se rezultat prikazuje korisniku. Korišteni modeli za generiranje teksta, odnosno konačnog odgovora jesu GPT-4o i GPT-4o-mini te će u nastavku ovog rada biti opisana evaluacija rezultata oba modela i razlike u generiranju odgovora iz istog izvora, te usporedba u potrebnim novčanim resursima za korištenje pojedinog modela nad istim podacima, odnosno za isti broj tokena.



Sl. 4.1. Dijagram tijeka implementacije *chatbota*

5. EVALUACIJA KVALITETE EMBEDDINGA ZA HRVATSKI JEZIK

Evaluacija ovog *chatbota* provedena je kroz pažljivo prilagođavanje i testiranje različitih parametara ključnih funkcija. Prvi korak u evaluaciji uključivao je optimizaciju načina učitavanja podataka, kao i veličine fragmenata podataka koji se obrađuju. Različite veličine fragmenata isprobane su kako bi se postigla optimalna ravnoteža između preciznosti i učinkovitosti. Pored toga, analizirano je preklapanje između fragmenata kako bi se osiguralo da se svi relevantni podaci uzmu u obzir, bez nepotrebnog dupliciranja informacija. Nadalje, u *retrieval* fazi, posebna pažnja posvećena je broju najboljih k rezultata koji se vraćaju kao odgovor na korisnički upit. Ovaj parametar je ključan za postizanje optimalne točnosti odgovora, jer omogućuje povratak samo najrelevantnijih informacija iz baze podataka. Osim toga, evaluacija je uključivala ispitivanje različitih modela za stvaranje *embeddinga*. Ovi modeli imaju ključnu ulogu u prevođenju teksta u numeričke reprezentacije koje se mogu uspoređivati i obrađivati. Promjene u korištenom modelu za *embedding* utjecale su na kvalitetu rezultata pretrage i ukupnu performansu *chatbota*. Konačno, evaluacija je obuhvatila i ispitivanje različitih velikih jezičnih modela(engl. *Large Language Model,* LLM) modela, koji su korišteni za generiranje konačnih odgovora. Svaki od ovih modela ima svoje prednosti i slabosti, a njihova primjena je temeljito ispitana kako bi se odabrao onaj koji pruža najtočnije i najsmislenije odgovore korisnicima. Valja napomenuti da su pri testiranju korišteni isti podaci spremljeni u različitim tipovima datoteka, te da su kontrolna pitanja ostala ista za vrijeme cijelog testiranja kao i ostali parametri koji nisu ili neće biti spomenuti kao varijabilni. Što se tiče *dataseta* nad kojima se vršila evaluacija, on je pohranjen kao što je već spominjano u JSON datoteku u obliku pitanje i očekivani odgovor. Pitanja su bila raznolika, od jednostavnijih i onih sa kraćim i jednostavnijim odgovorom, do pitanja sa malo složenijim i dužim odgovorom kao što je vidljivo u tablici 5.1.

Tablica 5.1. Primjer pitanja i očekivanog odgovora iz *dataseta* nad kojim se vršila evaluacija

Pitanje	Očekivani odgovor
Koliko traje preddiplomski studij računarstva?	Sveučilišni preddiplomski studij računarstva traje 3 godine (šest semestara).
Koji su kriteriji i uvjeti prijenosa ECTS bodova?	Fakultet sudjeluje u organizaciji i provedbi Erasmus programa međunarodne mobilnosti. U okviru Erasmus programa međunarodne mobilnosti studenti mogu provesti jedan dio studija studirajući na visokom učilištu u inozemstvu ili obavljajući stručnu praksu što značajno pridonosi njihovoj samostalnosti, kulturnoj obogaćenosti, poznavanju stranih jezika i sposobnosti rada u multikulturalnim sredinama. Provedba i osnovna načela mobilnosti dolaznih i odlaznih studenata, prava i obveze studenta, prava i obveze Sveučilišnog povjerenstva za Program mobilnosti, , prava i obveze Erasmus koordinatora na sastavnicama Sveučilišta te druga pitanja značajna za provedbu Programa mobilnosti pobliže su određena Pravilnikom o Erasmus programu međunarodne mobilnosti.

Tablice u nastavku ove evaluacije će pokazivati postotak pronađenih očekivanih odgovora u segmentima koji su prema kosinusovoj sličnosti određeni kao najpogodniji odgovori za postavljena pitanja. *Dataset* sačinjava dvadeset pitanja, sa dvadeset očekivanih odgovora. Očekivani odgovori su prikupljeni na način da se u podacima pronalazio odgovor i bez ikakve izmjene pohranio u JSON datoteci u spomenutoj varijabli. Ukoliko se odgovor nalazio u vraćenih *top_k* segmenata, vrijednost brojača bila bi povećana. Na kraju postupka, ta bi se vrijednost podijelila s ukupnim brojem postavljenih pitanja, čime bi se dobio postotak koji je prikazan u tablicama koje slijede.

5.1. Učitavanje podataka iz PDF datoteka

PDF format je često korišten za pohranu dokumenata zbog svoje prenosivosti i mogućnosti zadržavanja formata na različitim uređajima. Za ekstrakciju teksta iz PDF datoteke korišten je pdfminer, čime je osigurano da se tekst ekstrahira iz izvornog dokumenta točno i precizno. *Embedding* modeli korišteni u evaluaciji su OpenAI modeli text-embedding-3-small, text-embedding-3-large i Cohere-ov model Cohere-embed-multilingual-v3.0. Parametri mijenjani pri evaluaciji su oni funkcije RecursiveCharacterTextSplitter(), odnosno *chunk_size, chunk_overlap*,

te *separators*. Prvi ključni parametar u usporedbi je *chunk_size,* odnosno broj tokena koji se koristi u jednom isječku teksta. Testirane su dvije različite vrijednosti – 500 i 1000 tokena po isječku. Veći *chunk_size* može zadržati više informacija u jednom isječku teksta, omogućujući modelu da bolje uhvati kontekst. Međutim, veći isječci također mogu dovesti do smanjenja preciznosti, ako je dulji tekst heterogen u sadržaju. S druge strane, manji *chunk_size* omogućuje detaljniju analizu kraćih dijelova teksta, no može izgubiti važan kontekst ako je podjela isječaka previše fragmentirana. Drugi ključni parametar je *chunk_overlap*, koji se odnosi na preklapanje između isječaka teksta. Preklapanje može biti korisno kada je važno očuvati kontinuitet informacija između dijelova teksta. U ovom istraživanju testirane su dvije postavke: bez preklapanja (*chunk_overl*ap=0) i s preklapanjem od deset posto veličine *chunk_size-a*. Preklapanje može poboljšati rezultate jer omogućuje očuvanje ključnih informacija koje prelaze granice između dva isječka. Treći ključni parametar jesu separatori. To su znakovi ili nizovi znakova koji se koriste za odvajanje dijelova teksta. Oni se navode u varijablu separator te se navode u redoslijedu važnosti, odnosno preferencije. Redom su navedeni kao novi paragraf, novi redak, točka, razmak. U tablici u nastavku su rezultati testiranja OpenAI modela text-embedding-3-small koji bi u teoriji i u pretpostavci trebao najbolje raditi na tipovima podataka koji će biti korišteni u ovom testu. Prvi stupac tablice nazvan *top_k* podrazumijeva broj rezultata koje vraćamo iz vektorskog spremišta po kosinusovoj sličnosti, gdje jedan predstavlja jedan vektor, odnosno fragment, tri predstavljaju tri vektora itd. Kao što je već rečeno brojevi u tablici predstavljaju postotak pronađenih odgovora, odnosno u koliko posto pitanja se očekivani odgovor zaista nalazio u segmentima koje smo dobili kao odgovor.

Tablica 5.2. Evaluacija text-embedding-3-small nad podacima učitanima iz PDF datoteke

top_k	Chunk_size=500 Chunk_overlap=0 Separators= -	Chunk_size=500 Chunk_overlap=50 Separators= -	Chunk_size=500 Chunk_overlap=50 Separators= +	Chunk_size=1000 Chunk_overlap=0 Separators= -	Chunk_size=1000 Chunk_overlap=100 Separators= -	Chunk_size=1000 Chunk_overlap=100 Separators= +
1	60%	50%	50%	50%	50%	50%
3	80%	80%	80%	80%	90%	90%
5	90%	90%	90%	90%	90%	90%
10	90%	90%	90%	90%	90%	90%

Segmenti veličine tisuću znakova generalno daju bolje rezultate u pretraživanju očekivanih odgovora, posebno kod većih *top_k* vrijednosti. Razlog tomu jesu pitanja i očekivani odgovori. Baza pitanja na kojima je model evaluiran se sastoje od jednostavnih odgovora od nekoliko znakova, ali u isto vrijeme postoje pitanja koja zahtijevaju duži odgovor, odnosno prelaze broj

znakova za pronalazak cijelog odgovora. Dodavanje preklapanja generalno poboljšava rezultate kod *chunk_size*=1000, dok s druge strane kod manjih segmenata, odnosno *chunk_size*=500, preklapanje od pedeset znakova ima manje značajan utjecaj. Korištenje separatora u ovom slučaju je gotovo zanemarivo. Kao što je bilo za pretpostaviti povećanjem broja vektora, u kojima provjeravamo prisutnost traženog odgovora, povećava šanse za pronalazak istog. Ova analiza pokazuje da uzimajući u obzir brzinu, potrebne resurse u obliku memorije, najbolje je koristiti segmente od tisuću znakova uz preklapanje od stotinu znakova, ne uzimajući u obzir separatore.

Tablica 5.3. Usporedba OpenAI text-embedding-3-large i Cohere-ov model Cohere-embed-multilingual-v3.0 nad podacima učitanim iz PDF datotekte

	OpenAI text-embedding-3-large		Cohere-embed-multilingual-v3.0	
Top_k	Chunk_size=500 Chunk_overlap=0 Separators= -	Chunk_size=1000 Chunk_overlap=100 Separators= -	Chunk_size=500 Chunk_overlap=0 Separators= -	Chunk_size=1000 Chunk_overlap=100 Separators= -
1	40%	50%	60%	70%
3	70%	70%	80%	80%
5	80%	90%	90%	90%
10	90%	90%	90%	90%

Evaluacija navedenih dvaju modela je odrađena na optimalnim parametrima OpenAI text-embedding-3-small modela. Kod OpenAI text-embedding-3-large kada je broj vektora, odnosno *top_k* mala vrijednost postotak točnosti je značajno niži u ovom segmentu, posebno s manjim isječcima bez preklapanja. To sugerira da je model manje uspješan u pronalaženju točnog odgovora među prvih jedan rezultata kada nema dovoljno konteksta ili kada su isječci premali. Također može se uočiti kako se točnost povećava kako se broj vektora, odnosno *top_k*, povećava, kod prethodnog OpenAI modela također je došlo do povećanja, ali ono nije bilo ako drastično kao u ovom slučaju. Optimalni parametri ostali su slični kao i kod prethodnog modela, iako uočavamo kako je potreban veći broj vektora da bi se postigla ista točnost. Cohere-embed-multilingual-v3.0 pokazuje znatno bolje performanse na malom broju vektora u kojima se traži odgovor. To pokazuje da je ovaj model efikasniji u pronalaženju odgovora, čak i kad su isječci manji ili bez preklapanja. To bi moglo biti posljedica bolje optimizacije za kontekstualno prepoznavanje u više jezika. Kada se *top_k* povećava, kao i kod prethodnih modela, točnost se povećava, međutim i kod malih vrijednosti *top_k* točnost je bolja nego li kod modela text-embedding-3-large, ali ipak lošija nego li kod modela text-embedding-3-small.

5.2. Učitavanje podataka iz tekstualnih datoteka

Tekstualne datoteke predstavljaju jedan od najčešćih formata za pohranu informacija. Tekstualne datoteke mogu sadržavati razne vrste podataka, od jednostavnog *plain texta* do složenijih struktura poput log datoteka, konfiguracijskih datoteka. Treba napomenuti da baza podataka ovoga *chatbota* predstavljaju podaci koji su prvotno bili pohranjeni u obliku pdf datoteke, ali su za potrebe eksperimenta i određivanje optimalnih parametara prebačeni i u tekstualnu datoteku. Za učitavanje tekstualne datoteke korištena je funkcija TextLoader(). Treba napomenuti i uzeti u obzir da je proces učitavanja podataka iz tekstualnih datoteka bio značajno brži nego li učitavanje PDF datoteke, što je bilo i za pretpostaviti s obzirom da učitavanje tekstualne datoteke podrazumijeva samo čitanje već strukturiranog teksta, nema potrebe za *parsiranjem*. Rukovanje s podacima u daljnjem postupku ostalo je jednako kao i u prethodnom slučaju, odnosno kod učitavanja podataka iz PDF datoteka. RecursiveCharacterTextSplitter() jer korišten kao i prije, sa istim parametrima i slučajevima. Kontrolna pitanja i očekivani odgovori su također, ostali isti.

Tablica 5.4. Evaluacija text-embedding-3-small nad podacima učitanima iz tekstualnih datoteka

Top_k	Chunk_size=500 Chunk_overlap=0 Separators= -	Chunk_size=500 Chunk_overlap=50 Separators= -	Chunk_size=500 Chunk_overlap=50 Separators= +	Chunk_size=1000 Chunk_overlap=0 Separators= -	Chunk_size=1000 Chunk_overlap=100 Separators= -	Chunk_size=1000 Chunk_overlap=100 Separators= +
1	30%	20%	20%	70%	70%	70%
3	60%	50%	50%	80%	90%	90%
5	80%	50%	50%	90%	90%	90%
10	80%	70%	70%	90%	90%	90%

Kao što je vidljivo u tablici 5.4 rezultati se dosta razlikuju od slučaja gdje su podaci učitavani iz PDF datoteke. Jako mali broj odgovora je pronađen kada se gleda mali broj vektora, mnogo manji nego li je to bio slučaj kod testiranja u poglavlju 5.1.. Preciznost se i više nego udvostručava kada se uzima veći broj vektora, što pokazuje da je u ovom pristupu potrebno razmatrati veći broj rezultata. Isto tako potrebno je ukazati kako kod segmenata veličine petsto znakova uvođenje dodatnih parametara smanjuje postotak rezultata, odnosno najoptimalniji su parametri bez *chunk_overlapa* i *separators*. Točnost znatno raste kada se koriste veći segmenti, odnosno u ovom slučaju *chunk_size* iznosi tisuću. Potreban je manji broj vektora kako bi se pronašao tražen odgovor. Isto tako dodavanjem parametara poput *chunk_overlapa* u iznosu deset posto *chunk_size-a* doprinosi povećavanju točnosti. Rezultati evaluacije pokazuju da je povećanje veličine segmenta teksta na tisuću znakova s malim preklapanjem najefikasnije za precizno dohvaćanje odgovora iz

tekstualnih datoteka. Nasuprot tome, manji segmenti s ili bez preklapanja, bez obzira na upotrebu separatora, imaju tendenciju smanjivanja točnosti, osobito kada se razmatra samo prvi rezultat.

Tablica 5.5. Evaluacija OpenAI text-embedding-3-large i Cohere-embed-multilingual-v3.0 nad podacima učitanim iz tekstualnih datoteka

	OpenAI text-embedding-3-large		Cohere-embed-multilingual-v3.0	
Top_k	Chunk_size=500 Chunk_overlap=0 Separators= -	Chunk_size=1000 Chunk_overlap=100 Separators= -	Chunk_size=500 Chunk_overlap=0 Separators= -	Chunk_size=1000 Chunk_overlap=100 Separators= -
1	40%	50%	50%	60%
3	70%	80%	70%	80%
5	80%	80%	70%	80%
10	90%	90%	80%	80%

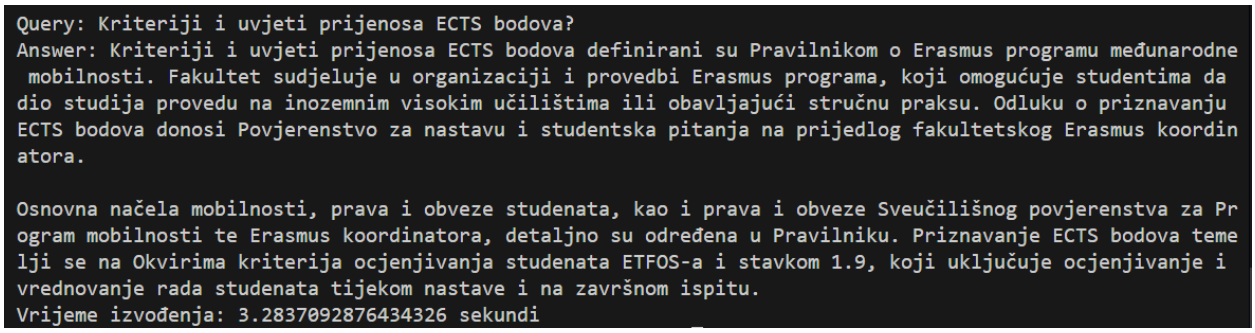
Za OpenAI text-embedding-3-large, kombinacija manjeg *chunk_size-a* i bez preklapanja rezultirala je malom točnošću za malu vrijednost *top_k*, dok za veće vrijednosti *top_k*, odnosno za najveću, točnost se više nego li udvostručila. Ovaj trend sugerira, kao i u prethodnim slučajevima, da ovaj model daje vrlo dobre rezultate kada mu se proširi raspon rezultata. Chohere-embed-multilingual-v3.0 daje bolje rezultate s manjom vrijednosti *top_k*, nego li prethodni model i trend povećanja točnosti postoji, ali nije u tolikoj mjeri izražen. U oba modela kombinacija većeg *chuk_size*-a i preklapanja pokazuje bolje rezultate, jer očuvanje konteksta kroz veće isječke i preklapanje omogućuje modelima bolji uvid u strukturu i značenje teksta.

Testiranjem različitih modela *embeddinga*, uključujući OpenAI text-embedding-3-small, OpenAI text-embedding-3-large i Cohere-embed-multilingual-v3.0, utvrđeno je da performanse svakog modela ovise o podešenim parametrima. Specifično, parametri poput *chunk_size* (veličina isječaka teksta), *chunk_overlap* (preklapanje isječaka) i *separators* (separatori) igraju važnu ulogu u preciznosti modela. Međutim, ključni parametar koji se pokazao kao presudan je *top_k*. Ovaj parametar određuje koliko se najrelevantnijih rezultata, tj. vektora ili segmenata, vraća u odgovoru na upit korisnika. S povećanjem vrijednosti *top_k*, raste broj segmenata koji se koriste za obogaćivanje konteksta prilikom generiranja odgovora. Time se direktno povećava količina informacija koje model može iskoristiti za preciznije i kontekstualno bogatije odgovore. Ipak, s povećanjem vrijednosti *top_k* dolazi i do nekoliko izazova. Prvo, veći broj segmenata zahtijeva dodatnu obradu, što može povećati vrijeme potrebno za obradu upita i generiranje odgovora. S obzirom na to da se svaki segment mora obraditi kako bi se model mogao koristiti za

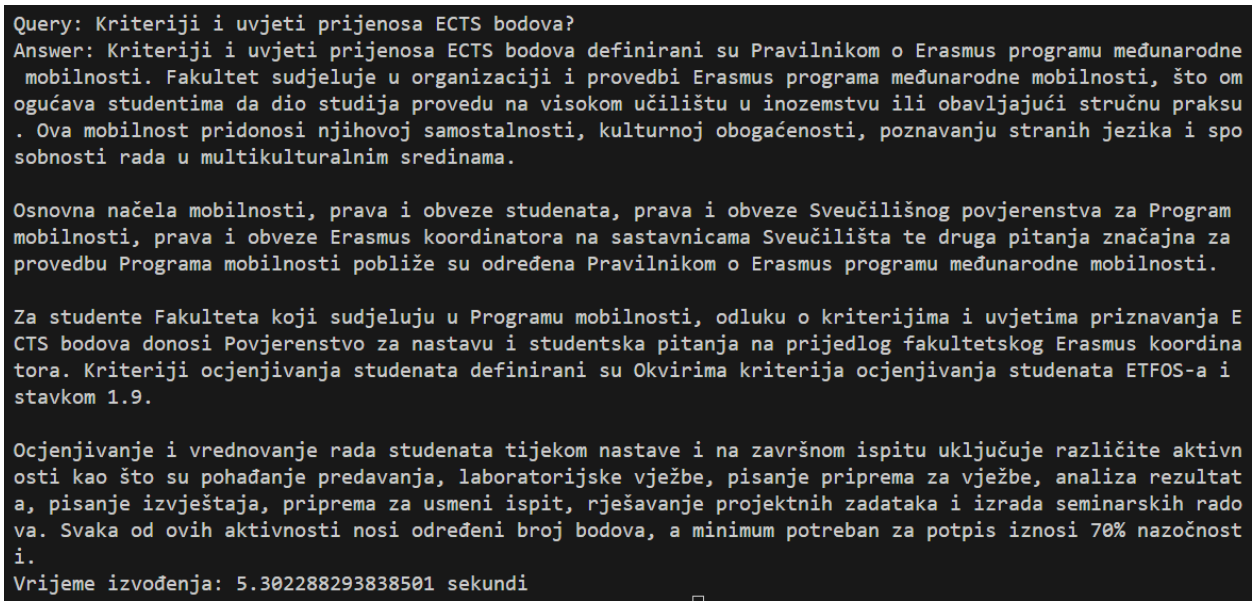
kontekstualizaciju i generiranje odgovora, raste računalna složenost obrade. Ovo povećanje obrade može opteretiti sistemske resurse, poput memorije i procesorske snage, posebno u okruženjima s velikim brojem istovremenih korisničkih upita. Optimalna vrijednost *top_k* ovisi o specifičnom scenariju primjene, veličini baze znanja te složenosti upita korisnika, i predstavlja balans između pružanja dovoljno konteksta i efikasnosti obrade podataka. Nad podacima kojima je implementiran ovaj *chatbot*, optimalne vrijednosti *top_k*, uzeći u obzir sve ranije rečeno, bi bile tri ili pet. U nastavku, odnosno evaluaciji LLM modela će biti prikazano i vrijeme potrebno da se generira odgovor i evaluacija kvalitete samog odgovora.

5.3. Usporedba i evaluacija LLM modela

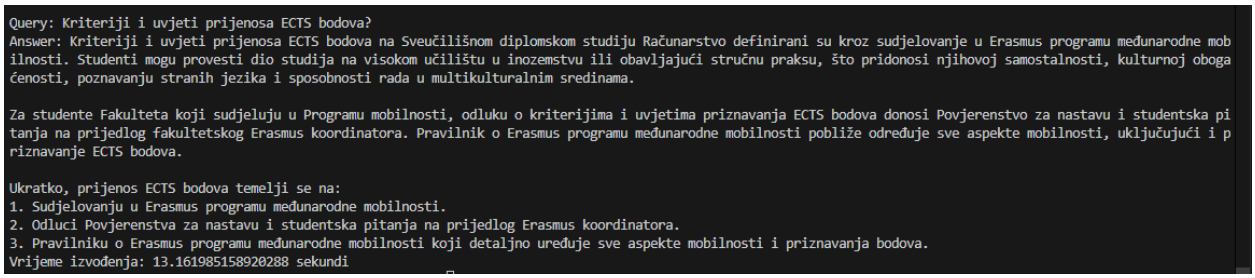
Kao što je već spomenuto, rezultati koji se vraćaju u *retrieval* fazi implementacije ovoga *chatbota* se spremaju u JSON datoteku, te se koriste kao baza podataka koju će jezični modeli koristiti kako bi odgovorili na postavljeno pitanje. Važno je pri implementaciji kombinirati upit i kontekst koji će formirati *prompt*, to jest ulazni tekst koji se šalje modelu za generiranje odgovora. *Prompt* je strukturiran tako da jasno daje do znanja modelu da odgovori na hrvatskom jeziku i postavlja pitanje u kontekstu. Važno je napomenuti da je način na koji se *prompt* formulira ključan za dobivanje željenih odgovora. U ovom slučaju, *prompt* jasno postavlja granice između pitanja i konteksta, što pomaže modelu da bolje razumije zadatak. U ovoj usporedbi i evaluaciji, fokusirat ćemo se na dva modela unutar GPT-4 serije: GPT-4o i GPT-4o-mini. Oba modela pripadaju istoj generaciji velikih jezičnih modela (LLM), ali se razlikuju u svojoj veličini, sposobnostima, te primjeni u kontekstu zadataka koji uključuju obradu prirodnog jezika. Usporedbu ćemo temeljiti na performansama ovih modela unutar prethodno objašnjenog koda, koji generira odgovore na temelju konteksta iz JSON datoteke. Podaci koji će biti predani modelu i koji će služiti kao izvor znanja će biti neki od slučajeva iz prethodne evaluacije. Pitanja nad kojima su modeli evaluirani također će biti manje kompleksna, sa kraćim odgovorom, kao i ona više kompleksna. Primjer u nastavku je onaj gdje je postavljeno pitanje općenito. Postoji cijeli jedan paragraf u podacima koji daje odgovor na postavljeno pitanje. Promatrano je kako će model generirati odgovor koji ne može biti konkretan, kratak i jednostavan. U nastavku su slike zaslona terminala uz optimalne parametre OpenAI text-embedding-3-small, uz mijenjani parametar *top_k*.



Sl. 5.1. Generirani odgovor modela GPT-4o kad je predan kontekst sa *top_k*=3



Sl. 5.2. Generirani odgovor modela GPT-4o kad je predan kontekst sa *top_k*=5



Sl. 5.3. Generirani odgovor modela GPT-4o kad je predan kontekst sa *top_k*=10

Odgovori u prva dva slučaja, kao što je prikazano na slikama 5.2 i 5.3, vrlo su slični, kako u pogledu konteksta tako i sintakse, ipak došlo je do blagog povećanja vremena. Ovi rezultati sugeriraju da povećanje broja segmenata prilikom pretrage unutar razumnog raspona ne utječe značajno na kvalitetu ili vrijeme generiranja odgovora. Međutim, u trećem slučaju, gdje je vrijednost parametra *top_k* postavljena na 10, prikazano na slici 5.3, oblikovanje odgovora značajno se razlikuje. Iako je u sadržajnom smislu kontekst ostao sličan prethodnim odgovorima,

zamjetno je povećanje složenosti u načinu na koji je odgovor oblikovan, što može ukazivati na veći i bogatiji kontekst koji model pokušava integrirati.. Vrijeme izvođenja se, međutim, značajno povećalo upravo zbog većeg broja segmenata koji se modelu predaju kao baza znanja. Ova veća količina informacija omogućuje modelu da odgovori detaljnije, ali ujedno povećava opterećenje sustava i produžuje vrijeme obrade.

```
Query: Kriteriji i uvjeti prijenosa ECTS bodova?
Answer: Kriteriji i uvjeti prijenosa ECTS bodova na Elektrotehničkom fakultetu Sveučilišta J.J. Strossmay
era u Osijeku definirani su u okviru Erasmus programa međunarodne mobilnosti. Studenti koji sudjeluju u o
vom programu mogu provesti dio svog studija na inozemnim visokim učilištima ili obavljati stručnu praksu,
 što doprinosi njihovoj samostalnosti i kulturnoj obogaćenosti.

Odluku o kriterijima i uvjetima priznavanja ECTS bodova donosi Povjerenstvo za nastavu i studentska pitan
ja na prijedlog fakultetskog Erasmus koordinatora. Ovi kriteriji su usklađeni s Okvirima kriterija ocjenj
ivanja studenata na fakultetu.

Ukratko, prijenos ECTS bodova ovisi o:

1. **Usklađenosti studijskih programa**: Studenti moraju osigurati da su predmeti koje su pohađali u inoz
emstvu ekvivalentni onima na matičnom fakultetu.
2. **Odluci Povjerenstva**: Povjerenstvo donosi odluku o priznavanju bodova na temelju predloženih progra
ma i postignutih rezultata.
3. **Pravilniku o Erasmus programu**: Svi postupci i prava studenata definirani su ovim pravilnikom.

Ovi uvjeti omogućuju studentima da nastave svoje obrazovanje i steknu međunarodno iskustvo, dok istovreme
no osiguravaju da njihovo znanje i vještine budu priznate i valorizirane.
Vrijeme izvođenja: 3.8467659950256348 sekundi
```

Sl. 5.4. Generirani odgovor modela GPT-4o-mini kad je predan kontekst sa *top_k*=3

```
Query: Kriteriji i uvjeti prijenosa ECTS bodova?
Answer: Kriteriji i uvjeti prijenosa ECTS bodova u okviru studijskog programa Računarstvo definirani su k
roz sudjelovanje fakulteta u Erasmus programu međunarodne mobilnosti. Studenti imaju mogućnost provesti d
io svog studija na inozemnim visokim učilištima ili obavljati stručnu praksu, što doprinosi njihovoj samo
stalnosti i kulturnoj obogaćenosti.

Prijenos ECTS bodova temelji se na sljedećim načelima:

1. **Odluka o priznavanju**: Odluku o kriterijima i uvjetima priznavanja ECTS bodova donosi Povjerenstvo
za nastavu i studentska pitanja na prijedlog fakultetskog Erasmus koordinatora.

2. **Pravilnik o Erasmus programu**: Detalji o provedbi mobilnosti, pravima i obvezama studenata, kao i o
bvezama fakultetskog povjerenstva i Erasmus koordinatora, definirani su Pravilnikom o Erasmus programu me
đunarodne mobilnosti.

3. **Usklađenost s programom**: Prijenos ECTS bodova moguć je samo ako su sadržaji predmeta koje student
polaže u inozemstvu ili na drugom visokom učilištu slični onima na matičnom fakultetu.

4. **Dokumentacija**: Studenti su dužni priložiti potrebnu dokumentaciju koja potvrđuje uspješno završen
dio studija ili praksu u inozemstvu.

5. **Savjetovanje**: Prije odlaska na mobilnost, studenti se savjetuju s Erasmus koordinatorom kako bi os
igurali da će im predmeti biti priznati.

Ovi kriteriji omogućuju studentima fleksibilnost i priliku za stjecanje međunarodnog iskustva, dok istovr
emeno osiguravaju da se njihovo obrazovanje i dalje usklađuje s akademskim standardima fakulteta.
Vrijeme izvođenja: 5.593539714813232 sekundi
```

Sl. 5.5. Generirani odgovor modela GPT-4o-mini kad je predan kontekst sa *top_k*=5

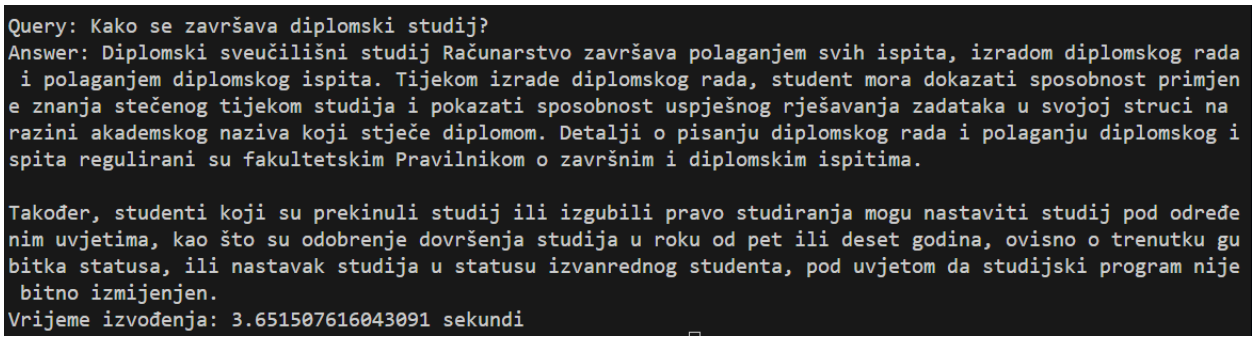
```
Query: Kriteriji i uvjeti prijenosa ECTS bodova?
Answer: Kriteriji i uvjeti prijenosa ECTS bodova definirani su u okviru Erasmus programa međunarodne mobi
lnosti, koji omogućava studentima da provedu dio svog studija na inozemnim visokim učilištima ili obavlja
ju stručnu praksu. Ovi kriteriji uključuju:

1. **Priznavanje ECTS bodova**: Prijenos ECTS bodova može se ostvariti za predmete koje studenti polože n
a inozemnim institucijama, pod uvjetom da su ti predmeti relevantni za njihov studijski program.

2. **Odluka Povjerenstva**: Odluku o priznavanju ECTS bodova donosi Povjerenstvo za nastavu i studentska
pitanja na prijedlog fakultetskog Erasmus koordinatora.

3. **Usklađenost s programom**: Predmeti koji se priznaju moraju biti u skladu s ciljevima i ishodima uče
nja studijskog programa na matičnom fakultetu.

4. **Dokumentacija**: Studenti su dužni dostaviti svu potrebnu dokumentaciju koja potvrđuje uspješno zavr
šene predmete, uključujući potvrde o položenim ispitima i opis predmeta.

5. **Kriteriji ocjenjivanja**: Priznavanje ECTS bodova također se temelji na kriterijima ocjenjivanja koj
i su definirani pravilnicima fakulteta.

Ovi uvjeti omogućuju studentima da iskoriste svoje međunarodno iskustvo i steknu dodatne kompetencije, a
istovremeno osiguravaju da njihovo obrazovanje ostane usklađeno s akademskim standardima domaće instituci
je.
Vrijeme izvođenja: 7.425034761428833 sekundi
```

Sl. 5.6. Generirani odgovor modela GPT-4o-mini kad je predan kontekst sa *top_k*=10

Na isto pitanje, sa istim podacima i parametrima testiran je GPT-4o-mini. Odgovori generirani ovim modelom ne razlikuju se pretjerano u kontekstu u usporedbi s drugim modelom. Promjena parametra *top_k* nije dovelo do značajnije promjene u oblikovanju odgovora. Što se tiče vremena izvođenja, najbolji rezultati su postignuti s parametrom *top_k* = 3. Ovaj parametar nudi optimalnu ravnotežu između relevantnosti odgovora i vremena obrade, s puno boljim vremenom izvođenja u usporedbi s ostalim testiranim slučajevima. Vrijeme obrade kod *top_k* = 3 je značajno kraće od ostala dva scenarija, što ukazuje da je ovaj parametar najefikasniji u pogledu performansi, bez negativnog utjecaja na kontekstualnu točnost odgovora. Sljedeće je pitanje kratko i jednostavno, kao i sam odgovor. Promatrano je kako će model generirati odgovor ukoliko je odgovor konkretan.

```
Query: Kako se završava diplomski studij?
Answer: Diplomski sveučilišni studij Računarstvo završava polaganjem svih ispita, izradom diplomskog rada
 i polaganjem diplomskog ispita. Diplomskim radom student mora dokazati da je sposoban primjenjivati znan
je stečeno tijekom studija i pokazati da može uspješno rješavati zadatke svoje struke na razini akademsko
g naziva kojeg stječe diplomom. Detalji vezani uz pisanje diplomskog rada i polaganje diplomskog ispita u
ređeni su fakultetskim Pravilnikom o završnim i diplomskim ispitima.
Vrijeme izvođenja: 2.754713773727417 sekundi
```

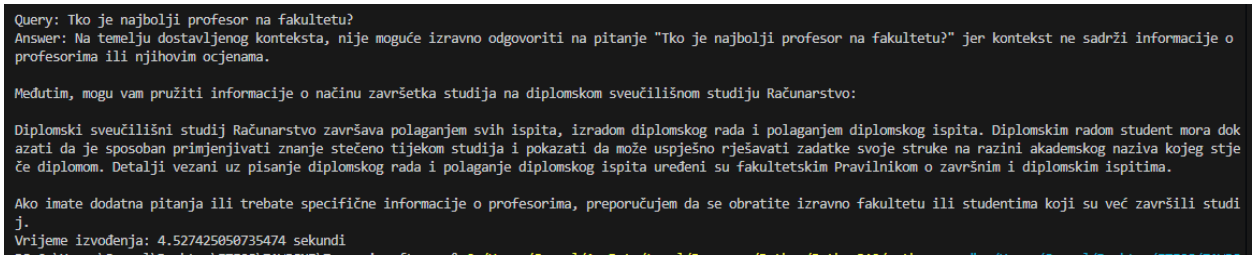
Sl. 5.7. Generirani odgovor modela GPT-4o na jednostavno pitanje kad je predan kontekst s *top_k*=3



Sl. 5.8. Generirani odgovor modela GPT-4o-mini na jednostavno pitanje kad je predan kontekst s *top_k*=3

Odgovor GPT-4o modela pruža točne, konkretne i tražene informacije, s jasnim opisom koraka. Vrijeme izvođenja je relativno brzo. GPT-4o-mini nudi nešto opširniji odgovor s dodatnim detaljima, ako je odgovor dulji i sadržajniji, osnovne informacije su slične kao kod GPT-4o. Vrijeme izvođenja je duže nego kod GPT-4o, što može biti posljedica dodatnog konteksta koji model generira i opširnijeg odgovora. Ovo produženje vremena može biti značajno u aplikacijama gdje je brzina ključna.

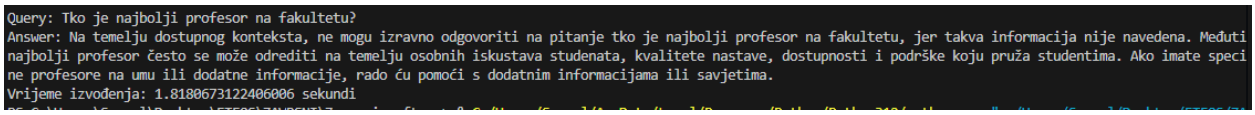
GPT-4o je značajno veći model u usporedbi s GPT-4o-mini, što ga čini sposobnijim za generiranje detaljnih, preciznih odgovora. On je treniran na opsežnim skupovima podataka, što rezultira njegovom mogućnošću da duboko analizira pitanja i pruži iscrpne odgovore. Nasuprot tome, GPT-4o-mini, iako temeljen na istoj arhitekturi, zbog svoje smanjene veličine daje odgovore brže, ali manje detaljno. Ova razlika u veličini modela odražava se ne samo u brzini, već i u sposobnosti modela da kontekstualizira složenije upite i pruži bogatije informacije. Jedan od ključnih aspekata evaluacije oba modela je način na koji postupaju u *retrieval* fazi, tj. kada pristupaju podatcima unutar JSON datoteke kako bi konstruirali odgovor. GPT-4o je sposoban detaljno analizirati podatke i pokušati pronaći bilo kakve relevantne informacije, čak i ako kontekst nije savršeno jasan. Model često generira proširene odgovore, nadmašujući očekivanja, uključujući dodatne informacije koje nisu direktno zatražene. Kao što je vidljivo u slici, vraća kontekst koji uopće nije zatražen u pitanju.



Sl. 5.9. Generirani odgovor modela GPT-4o na pitanje kada se odgovor ne nalazi u podacima

Ovaj način rada može biti iznimno koristan u situacijama gdje je potrebno dobiti širi pregled teme, ali također može dovesti do neželjenih komplikacija kada je potreban precizan, sažet odgovor.

GPT-4o-mini, s druge strane, ima tendenciju pružanja odgovora koji su uže vezani uz postavljeno pitanje. Iako manje sklon dodavanju dodatnih informacija koje nisu vezane uz temu, ovaj model pruža konkretnije odgovore, što ga čini pogodnim za zadatke gdje je sažetost i brzina ključna. U slučajevima kada *retrieval* faza ne uspije pronaći relevantan odgovor, GPT-4o-mini češće priznaje neznanje, navodeći da nema informacija o tom pitanju. Iako povremeno vraća neodređene odgovore kada sličnost upita i odgovora nije visoka, ova tendencija je rjeđa nego kod većeg modela, ali je i dalje prisutna u malom broju slučajeva.



Sl. 5.10. Generirani odgovor modela GPT-4o-mini na pitanje kada se odgovor ne nalazi u podacima

Oba modela omogućuju prilagodbu parametra temperature, što je ključna karakteristika koja utječe na stil generiranja odgovora. Temperatura kontrolira stupanj deterministočnosti i kreativnosti u odgovorima. Kada je temperatura postavljena na nulu, modeli generiraju konzistentne, deterministične odgovore koji su najvjerojatniji prema modelovoj obuci. To omogućuje da odgovori budu točno citirani iz konteksta i predvidljivi, što je ključno u situacijama gdje je potrebna preciznost. S druge strane, viša temperatura modelima omogućava veću kreativnost, čineći ih manje predvidljivima i fleksibilnijima u izražavanju. To može biti korisno za zadatke koji zahtijevaju inovativne pristupe, ali može rezultirati promjenama u formulaciji odgovora, uključujući promjene redoslijeda riječi ili uvođenje dodatnih fraza. Ovaj aspekt posebno je važan kada je redoslijed ili formulacija odgovora kritična, budući da bi čak i mala promjena mogla promijeniti smisao odgovora ili kompromitirati njegovu točnost. Jedan od najvažnijih faktora prilikom odabira između ovih modela je cijena korištenja. Veći model, GPT-4o, zbog svoje složenosti i bolje performanse, ima značajno višu cijenu u usporedbi s manjim modelima poput GPT-4o-mini. Cijena GPT-4o modela za input od jednog milijuna tokena iznosi pet dolara, dok za output iznosi petnaest dolara za milijun tokena. Dok je cijena GPT-4o-mini stotinu i pedeset centi za milijun input tokena, dok je za milijun output tokena cijena šesto centi u trenutku pisanja ovoga završnog rada. Broj znakova u milijun tokena može značajno varirati ovisno o jeziku, vrsti teksta, i konkretnoj tokenizaciji. Stoga je odabir modela često stvar kompromisa između kvalitete generiranih odgovora i financijskih troškova. Za zadatke gdje je potrebno generirati vrlo precizne i informativne odgovore, posebno kada postoji složenija baza podataka, GPT-4o će vjerojatno biti

bolji izbor, unatoč višim troškovima. S druge strane, za zadatke koji zahtijevaju brze i jednostavne odgovore, GPT-4o-mini nudi optimalan omjer cijene i performansi.

6. ZAKLJUČAK

Ovaj rad je istražio i implementirao kompleksan sustav *chatbota* koristeći OpenAI API, fokusirajući se na integraciju RAG metode za poboljšanje točnosti i relevantnosti odgovora. Kroz rad su razrađeni ključni koraci, uključujući pripremu podataka, izgradnju *embeddinga* i primjenu različitih jezičnih modela.

Embeddingzi predstavljaju ključni element u procesu obrade prirodnog jezika jer omogućuju pretvaranje tekstualnih podataka u numeričke vektore. Ovi vektori omogućuju modelima da razumiju i uspoređuju semantičko značenje riječi i fraza. Korištenjem različitih *embedding* modela kao što su text-embedding-3-small, text-embedding-3-large, i Cohere-embed-multilingual-v3.0 imali smo uvid u fleksibilnost i prilagodljivost u razvoju *chatbota* temeljenog na RAG metodi. Svaki od ovih modela ima svoje prednosti i nedostatke, a njihova učinkovitost značajno ovisi o prirodi podataka na kojima se primjenjuju. Razumijevanje kako različiti modeli funkcioniraju i kako njihova preciznost može varirati u odnosu na specifične tekstualne informacije ključ je za optimizaciju performansi *chatbota* i pružanje relevantnih i korisnih odgovora korisnicima.

Evaluacija performansi jezičnih modela predstavlja ključni korak u razumijevanju njihove učinkovitosti i prikladnosti za različite primjene. U našem istraživanju, detaljno smo analizirali dva modela iz OpenAI obitelji: GPT-4o i GPT-4o-mini. Ova evaluacija obuhvatila je razmatranje točnosti, efikasnosti, troškovne isplativosti i prilagodljivosti svakog modela u stvarnim scenarijima. Evaluacija GPT-4o i GPT-4o-mini otkriva ključne razlike u njihovim performansama i prikladnosti za različite primjene. GPT-4o, sa svojom složenom arhitekturom i velikim brojem parametara, pruža visoku razinu točnosti i detaljnosti, ali dolazi s većim troškovima i zahtjevima za računalne resurse tako da može biti neprikladan za scenarije s ograničenim resursima. S druge strane, GPT-4o-mini nudi povoljan omjer cijene i performansi, što ga čini privlačnim za mnoge aplikacije koje ne zahtijevaju vrhunsku preciznost i detaljnost. Konačan odabir između GPT-4o i GPT-4o-mini nije samo tehnički izbor, već i odluka koja odražava specifične potrebe, ciljeve i ograničenja projekta.

Kroz ovu analizu, jasno je da konačni izbor modela, vektorskih spremišta, *embeddinga* i jezičnih modela ovisi o pažljivom balansiranju između troškova, performansi i specifičnih potreba projekta. Svaki od ovih elemenata igra ključnu ulogu u oblikovanju učinkovitosti i uspješnosti sustava temeljenog na Retrieval-Augmented Generation (RAG) metodi.

LITERATURA

[1] A. Vaswani, N. Shazeer, N. Parmar, et al., "Attention is All You Need," in *Advances in Neural Information Processing Systems* (NeurIPS), 2017.

[2] TensorFlow, *Transformer Chatbot Tutorial with TensorFlow 2* [online], TensorFlow, SAD, 2019. Dostupno na: https://blog.tensorflow.org/2019/05/transformer-chatbot-tutorial-with-tensorflow-2.html, [09.2024].

[3] K. Cho, et al., "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," in *arXiv preprint* arXiv:1406.1078, 2014.

[4] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[5] A. Khalid, Understanding Vector Similarity [online], Medium, 2023. Dostupno na: https://medium.com/advanced-deep-learning/understanding-vector-similarity-b9c10f7506de [09.2024.].

[6] S. Robertson and H. Zaragoza, "The Probabilistic Relevance Framework: BM25 and Beyond," Foundations and Trends in Information Retrieval, vol. 3, no. 4, pp. 333-389, 2009.

[7] X. Rong, "Word2Vec Parameter Learning Explained," *arXiv preprint arXiv:1411.2738*, 2014.

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean, Efficient Estimation of Word Representations in Vector Space, 2013.

[9] Codethulo, *Understanding the Continuous Bag of Words (CBOW) Model Architecture, Working Mechanism and Math* [online], Medium, 2024. Dostupno na: https://medium.com/@codethulo/understanding-the-continuous-bag-of-words-cbow-model-architecture-working-mechanism-and-math-78c7284a8d5a [09.2024.].

[10] Word Embedding Using FastText, Medium, [online] Dostupno na: https://medium.com/@93Kryptonian/word-embedding-using-fasttext-62beb0209db9, [09.2024.].

[11] illiz, *Sparse vs Dense Embeddings* [online], Zilliz Learn, 2024. Dostupno na: https://zilliz.com/learn/sparse-and-dense-embeddings [09.2024.].

[12] Pinecone, Dense Vector Embeddings for NLP [online], Pinecone Learn, 2024. Dostupno na: https://www.pinecone.io/learn/series/nlp/dense-vector-embeddings-nlp/ [09.2024.].

[13] MakeUseOf, GPT Models Explained and Compared [online], MakeUseOf, 2024. Dostupno na: https://www.makeuseof.com/gpt-models-explained-and-compared/ [09.2024.].

[14] HGS, From GPT-1 to GPT-4: A Look at the Evolution of Generative AI [online], HGS, 2024, Dostupno na: https://hgs.cx/blog/from-gpt-1-to-gpt-4-a-look-at-the-evolution-of-generative-ai/ [09.2024.].

[15] OpenAI, Models Documentation [online], OpenAI, 2024. Dostupno na: https://platform.openai.com/docs/models [09.2024.].

[16] A. Helwan, ELMo [online], Medium, 2023. Dostupno na: https://abdulkaderhelwan.medium.com/elmo-e-c8e6fc068882 [09.2024.].

[17] Hugging Face, BERT Model Documentation [online], Hugging Face, 2024. Dostupno na: https://huggingface.co/docs/transformers/en/model_doc/bert [09.2024.].

[18] Hugging Face, Massive Text Embedding Benchmark (MTEB) [online], Hugging Face, 2024. Dostupno na: https://huggingface.co/blog/mteb [09.2024.].

[19] R. Baeza-Yates, B. Ribeiro-Neto, *Modern Information Retrieval: The Concepts and Technology behind Search*, Addison-Wesley, Boston, 2011.

[20] Langchain, Recursively Split by Character - Langchain Documentation [online], Langchain, 2024. Dostupno na: https://python.langchain.com/v0.1/docs/modules/data_connection/document_transformers/recursive_text_splitter/ [09.2024.].

[21] Pinecone, Namespaces - Langchain Integration Documentation [online], Pinecone, 2024. Dostupno na: https://docs.pinecone.io/integrations/langchain#namespaces [09.2024.].

SAŽETAK

Ovaj završni rad bavi se istraživanjem i implementacijom *chatbota* temeljenog na OpenAI API-ju, koristeći metodologiju Retrieval-Augmented Generation (RAG). Cilj rada bio je pokazati kako kombinacija generativnih jezičnih modela i metoda pretraživanja informacija može poboljšati točnost i relevantnost odgovora koje *chatbot* pruža korisnicima. Kroz rad su obrađeni ključni koraci poput pripreme podataka, izgradnje *embeddinga* te primjene različitih jezičnih modela. *Embeddingzi* predstavljaju ključni element obrade prirodnog jezika, omogućujući pretvaranje tekstualnih podataka u numeričke vektore, što pomaže modelima u razumijevanju i usporedbi semantičkog značenja riječi i fraza. Evaluacija je uključivala analizu OpenAI modela GPT-4o i GPT-4o-mini, razmatrajući točnost, učinkovitost, troškovnu isplativost i prilagodljivost. GPT-4o, iako vrlo točan i detaljan, dolazi s većim troškovima i zahtjevima za računalne resurse, dok GPT-4o-mini nudi povoljan omjer cijene i performansi za aplikacije s ograničenim resursima. Konačan odabir između ova dva modela ovisi o specifičnim potrebama i ciljevima projekta.

***Ključne riječi**: RAG chatbot, embedding, evaluacija, OpenAI modeli*

ABSTRACT

Development of a Chatbot Based on the OpenAI API Using RAG.

This thesis focuses on the research and implementation of a chatbot based on the OpenAI API, using the Retrieval-Augmented Generation (RAG) methodology. The goal of the thesis was to demonstrate how the combination of generative language models and information retrieval methods can improve the accuracy and relevance of the responses provided by the chatbot to users. The work covered key steps such as data preparation, the construction of embeddings, and the application of various language models. Embeddings represent a key element in natural language processing, enabling the transformation of textual data into numerical vectors, which helps models understand and compare the semantic meaning of words and phrases. The evaluation included an analysis of OpenAI models GPT-4o and GPT-4o-mini, considering accuracy, efficiency, cost-effectiveness, and adaptability. While GPT-4o is highly accurate and detailed, it comes with higher costs and computational resource requirements, whereas GPT-4o-mini offers a favorable balance between cost and performance for applications with limited resources. The final choice between these two models depends on the specific needs and goals of the project.

***Keywords**: RAG chatbot, embedding, evaluation, OpenAI models*