

# Izrada agenta za razgovor za pripomoč studentima

---

Suman, Frane

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:211:201090>

Rights / Prava: [Attribution 3.0 Unported](#)/[Imenovanje 3.0](#)

Download date / Datum preuzimanja: **2025-01-16**



Repository / Repozitorij:

[Faculty of Organization and Informatics - Digital Repository](#)



**SVEUČILIŠTE U ZAGREBU  
FAKULTET ORGANIZACIJE I INFORMATIKE  
VARAŽDIN**

**Frane Suman**

**IZRADA AGENTA ZA RAZGOVOR ZA  
PRIPOMOĆ STUDENTIMA**

**ZAVRŠNI RAD**

**Varaždin, 2024.**

**SVEUČILIŠTE U ZAGREBU**  
**FAKULTET ORGANIZACIJE I INFORMATIKE**  
**V A R A Ž D I N**

**Frane Suman**

**JMBAG: 0016149738**

**Studij: Informacijski sustavi**

**IZRADA AGENTA ZA RAZGOVOR ZA PRIPOMOĆ STUDENTIMA**

**ZAVRŠNI RAD**

**Mentor :**

Prof. dr. sc. Markus Schatten

**Varaždin, kolovoz 2024.**

*Frane Suman*

**Izjava o izvornosti**

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

*Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi*

---



## Sažetak

Ovaj rad bavi se razvojem chatbota pod nazivom "Robin", agenta za razgovor za studente Fakulteta organizacije i informatike Sveučilišta u Zagrebu. Cilj rada je kreirati agenta odnosno chatbot koji će pojednostaviti pristup informacijama i smanjiti opterećenje na studentsku referadu, omogućujući studentima brze i točne odgovore na pitanja o pravilnicima, kolegijima, i drugim aspektima akademskog života. Projekt koristi napredne tehnike obrade prirodnog jezika (NLP) i veliki jezični model kako bi omogućio relevantne i kontekstualno prilagođene odgovore. Chatbot "Robin" je razvijen u programskom jeziku Python te koristi kombinaciju velikih jezičnih modela (LLM) i tehnologije Retrieval-Augmented Generation (RAG) kako bi osigurao točnost odgovora na studentske upite. Kroz ovaj rad istražene su različite metode umjetne inteligencije. Rad predstavlja važan korak u razvoju personaliziranih sustava podrške studentima i ukazuje na potencijal daljnje primjene AI tehnologija u akademskom kontekstu te šire.

**Ključne riječi:** umjetna inteligencija, duboko učenje, transformer, agent za razgovor, veliki jezični modeli (LLM), RAG, Python, GPT, LlamaIndex.

# Sadržaj

<b>1. Uvod</b>	1
<b>2. Uvod u umjetnu inteligenciju</b>	2
2.1. Strojno učenje	4
2.2. Neuronske mreže i duboko učenje	5
2.2.1. Transformeri	7
2.2.1.1. Tokenizacija i ugnježđenje	9
2.2.1.2. Vektorska baza podataka	9
2.3. Obrada prirodnog jezika (NLP)	10
2.4. Generativna umjetna inteligencija (GenAI)	12
2.4.1. Veliki jezični modeli (LLM)	14
<b>3. Primjena umjetne inteligencije za pripomoć studentima</b>	16
3.1. Agenti za razgovor (Chatbot)	16
3.2. Generative Pre-trained Transformer (GPT) i ChatGPT	17
3.3. Retrieval-Augmented Generation (RAG)	19
3.3.1. Usporedba s drugim metodama	20
<b>4. Korisnički zahtjevi</b>	21
<b>5. Korišteni alati</b>	22
<b>6. Opis implementacije</b>	25
6.1. Pregled korištenih izvora podataka	25
6.2. Pregled implementacije agenta	26
6.2.1. Config.py	26
6.2.2. model_utils.py	26
6.2.3. document_processor.py	27
6.2.4. chat.py	28
6.2.5. api.py	31
6.2.6. evaluation-metrics.py	31
6.2.7. demo.html	31
<b>7. Primjer korištenja agenta</b>	33
7.1. Web sučelje	35
<b>8. Kritički osvrt</b>	36

8.1. Evaluacija agenta . . . . .	36
8.2. Alternativna implementacija . . . . .	38
8.2.1. Implementacija GPT-ja s bazom znanja na OpenAI platformi . . . . .	38
<b>9. Zaključak . . . . .</b>	<b>40</b>
<b>Popis literature . . . . .</b>	<b>44</b>
<b>Popis slika . . . . .</b>	<b>46</b>
<b>Popis tablica . . . . .</b>	<b>47</b>
<b>10. Prilog programskog koda . . . . .</b>	<b>48</b>

# 1. Uvod

U eri digitalne transformacije obrazovanja, pristup pravovremenim i točnim informacijama postaje ključan faktor uspjeha studenata. Ovaj završni rad bavi se razvojem inteligentog konverzacijskog agenta imena "Robin", osmišljenog kao asistent za studente Fakulteta organizacije i informatike (FOI) Sveučilišta u Zagrebu. Cilj ovog projekta je kreirati učinkovito rješenje koje će pojednostaviti pristup informacijama i smanjiti opterećenje na studentsku referadu, istovremeno poboljšavajući iskustvo studenata.

Robin je zamišljen kao sofisticirani konverzacijski agent koji studentima omogućuje brz i jednostavan pristup akademskim informacijama. Umjesto da moraju pregledavati brojne fakultetske dokumente ili čekati odgovore putem e-maila ili telefona, studenti mogu brzo dobiti odgovore na pitanja o pravilnicima, kolegijima i drugim važnim aspektima studiranja.

Ovaj rad opisuje proces izrade takvog agenta, od definiranja područja i prikupljanja dokumentacije, preko odabira tehnologija, do implementacije i testiranja sustava. Robin, izrađen u Pythonu s naprednim tehnikama obrade prirodnog jezika, osmišljen je kako bi razumio kontekst studentskih upita i pružao precizne odgovore. Poseban naglasak stavljen je na stvaranje baze znanja koja uključuje različite fakultetske dokumente, kao što su pravilnici, opisi kolegija i česta pitanja studentske službe. Ta baza služi kao osnova za sustav, osiguravajući relevantne i točne odgovore.

Konačno, ovaj projekt ne bi trebao samo napraviti funkcionalno rješenje za trenutačne potrebe FOI studenata, ali i istražiti šire implikacije od primjene umjetne inteligencije u akademskom kontekstu. Robin je korak prema personaliziranom, dostupnom i brzom sustavu za podršku studenta te potencijalno i zaposlenika.

## 2. Uvod u umjetnu inteligenciju

"Umjetna inteligencija je novi elektricitet" [1, str. 1]. Pojam umjetne inteligencije (AI) danas je sveprisutan. U proteklih nekoliko godina, područje umjetne inteligencije doživjelo je tzv. boom u razvoju i implementaciji. No, što je umjetna inteligencija, gdje se sve može primijeniti i zašto se smatra da će promijeniti svijet?

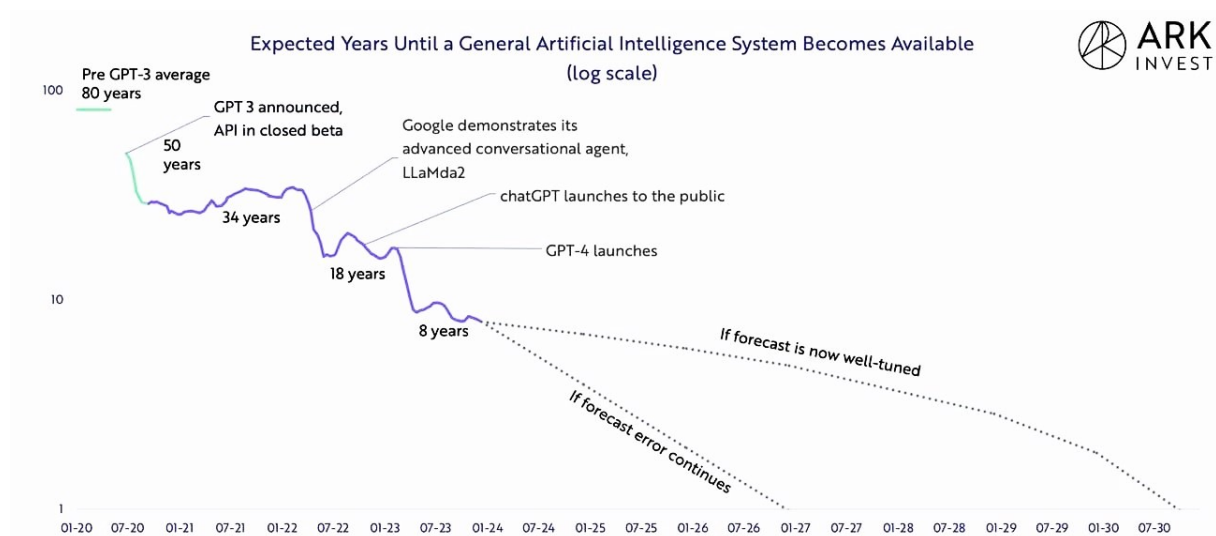
Hrvatska enciklopedija [2] umjetnu inteligenciju definira kao: "*dio računalstva koji se bavi razvojem sposobnosti računala da obavljaju zadaće za koje je potreban neki oblik inteligencije; također oznaka svojstva neživog sustava koji pokazuje inteligenciju*". Umjetna inteligencija odnosi se na sposobnost strojeva, poput računala, da imitiraju ili reproduciraju ljudske kognitivne funkcije. Stroj opremljen AI-jem može provoditi proračune, analizirati podatke za predviđanje ishoda, prepoznavati razne znakove i simbole, komunicirati s ljudima te obavljati zadatke bez potrebe za ljudskom intervencijom [3]. Danas se primjenjuje u raznim područjima ljudskog djelovanja i industrije, uključujući robotiku, analitiku podataka, marketing, financije, transport, poljoprivredu, medicinu, umjetnost itd. Zbog sve šire primjene možemo danas ustvrditi da je umjetna inteligencija postala interdisciplinarno područje.

Prema Schattenu [1] i Hrvatskoj enciklopediji [2], povijesni razvoj umjetne inteligencije prolazio je kroz različite faze, od prvih teorijskih osnova sredinom 20. stoljeća do modernih primjena koje koriste strojno učenje, neurone mreže i duboko učenje. Rani pokušaji umjetne inteligencije oslanjali su se na simboličke metode i ručno kodirana pravila, dok suvremeni pristupi sve više koriste podatkovno utemeljene metode poput dubokog učenja kako bi modeli postajali sve precizniji i sposobniji za autonomno učenje. Prva faza započinje 1950-ih godina, kada se AI razvijao u okviru temeljnih istraživanja i teorijskih modela. Tada su razvijeni prvi sustavi poput programa za igranje šaha, što se smatra važnim korakom u razvoju "inteligentnih" strojeva. Značajan sustav iz ovog razdoblja je ELIZA, prvi agent za razgovor razvijen 1966. godine, koji je simulirao razgovor s psihoterapeutom. Nakon početnog entuzijazma uslijedilo je razdoblje smanjenja interesa zbog neispunjenih očekivanja, što je dovelo do prve "nuklearne zime" 1970-ih [1] [2]. Druga faza, koja traje od 1980-ih, obilježena je razvojem ekspertnih sustava, kao što su oni temeljeni na bazi znanja. Ovi sustavi omogućavali su rješavanje specifičnih problema kroz logičko zaključivanje i donijeli AI u industrijsku primjenu. Jedan od najpoznatijih sustava iz ovog razdoblja je XCON, ekspertni sustav za konfiguraciju računalnih sustava razvijen za tvrtku DEC. Unatoč početnom uspjehu, opet je uslijedio pad interesa i ulaganja, te druga "nuklearna zima" krajem 1980-ih. Treća faza, koja započinje od 2010-ih, obilježena je eksplozijom interesa za AI uslijed napretka u računalnim kapacitetima i prikupljanju velikih količina podataka (Big Data). U ovom razdoblju značajan sustav je AlphaGo, razvijen od strane Google DeepMinda, koji je 2016. godine pobijedio svjetskog prvaka u igri Go, te naravno generativni agenti i veliki jezični modeli poput GPT-ja i Midjourneyja [1] [2].

Pojam umjetne inteligencije potrebno je razlikovati od područja računalstva i njezine implementacije, koja se označava kao sustav umjetne inteligencije. Stručnjaci Russell i Norvig navode da postoje četiri osnovna pristupa razumijevanju umjetne inteligencije, ovisno o cilju koji se želi postići u oblikovanju AI sustava. Prvi pristup usmjeren je na oponašanje ljudskog

ponašanja i temelji se na Turingovom testu, prema kojem je AI sustav uspješan ako se ne može razlikovati od čovjeka. Turingov test primarno se koristi za evaluaciju sustava umjetne inteligencije koji uključuju obradu prirodnog jezika, generativnu inteligenciju i slične tehnologije. Drugi pristup naglašava ljudsko razmišljanje i temelji se na kognitivnom modeliranju, gdje je cilj da AI sustav imitira ljudski način razmišljanja. Treći pristup fokusira se na racionalno razmišljanje, pri čemu je sustav dobro oblikovan ako koristi logičko i racionalno rezoniranje bez logičkih pogrešaka. Četvrti pristup se temelji na racionalnom ponašanju, gdje je AI sustav ocijenjen kao uspješan ako racionalno izvršava zadatke koji su mu povjereni [1].

Općenito, AI dijelimo na specijaliziranu (usku) i generalnu (opću) umjetnu inteligenciju. Specijalizirana umjetna inteligencija dizajnirana je za obavljanje specifičnih zadataka ili rješavanje problema u određenom području, poput agenata za razgovor, zdravstvo, računalnog vida, algoritama za preporuke ili autonomnih vozila [1] [3] [2]. Ovi sustavi su visoko optimizirani za svoje specifične funkcije, ali nemaju sposobnost rješavanja problema izvan svoje specijalizirane domene. Nasuprot tome, generalna umjetna inteligencija odnosi se na sustav koji bi mogao izvršavati bilo koju intelektualnu funkciju koju može obavljati čovjek. Generalna umjetna inteligencija (AGI) bi imala sposobnost učenja, razumijevanja i prilagodbe na širok spektar zadataka, slično kao što to čini čovjek [2] (zamislite Jarvisa iz Iron-mana ili Samanthu iz Ona). Iako AGI još uvijek nije ostvarena u praksi, nedavni napredak u području umjetne inteligencije potaknuo je rastuće uvjerenje među stručnjacima da je postizanje AGI-a moguće u dogledno vrijeme [4].



Source: Metaculus, ARK Invest, as of 1/3/24  
 Green lines are derived estimates for time to general purpose AI (strongly formulated) based upon forecasts for a weaker benchmark  
 For benchmark details see <https://www.metaculus.com/questions/S12/date-of-general-ai/>, benchmark broadly requires the successful passage of an adversarial two hour Turing test, broad success on a Q&A knowledge and logic benchmark, and the successful interpretation of and execution complex model car assembly instruction, all within a single system.

Slika 1: Očekivani broj godina do razvoja generalne inteligencije (AGI) [4]

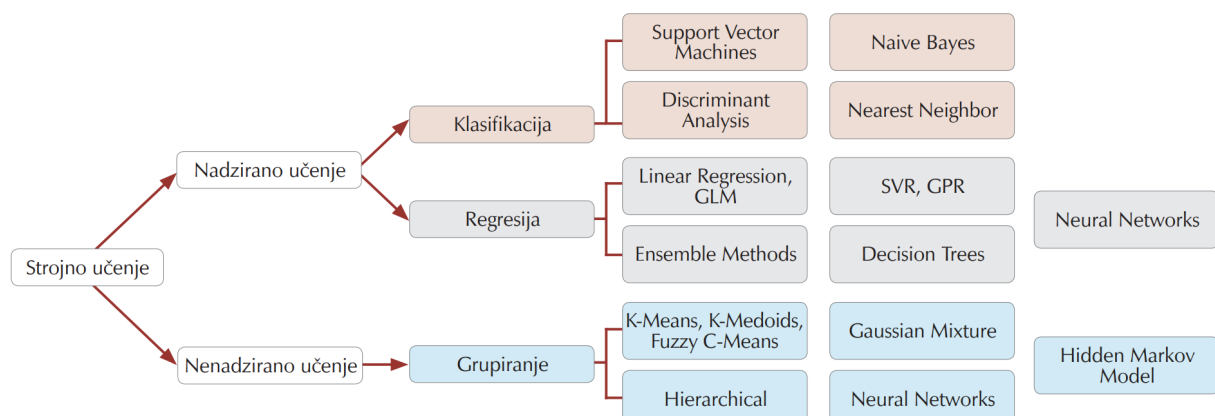
Razvojem umjetne inteligencije pojavljuju se značajne etičke i pravne dileme. Ključna pitanja uključuju autorska prava, privatnost podataka i sigurnost, posebno u kontekstu lažnog sadržaja poput deepfakeova. Dodatno, problem "crne kutije" čini teško predvidjeti odluke i posljedice djelovanja AI sustava zbog složenih algoritama strojnog učenja. U novije vrijeme,

ove zabrinutosti su proširene na problem pretjeranog oslanjanja na AI te fenomen haluciniranja, gdje AI generira nepostojeće ili netočne informacije. Stoga je važno paralelno s tehnološkim napretkom razvijati regulative koje će osigurati etičku i sigurnu primjenu AI tehnologija [2] [1] [3].

## 2.1. Strojno učenje

Strojno učenje grana umjetne inteligencije koje podrazumijeva programiranje računala tako da, na temelju podataka ili prethodnog iskustva, poboljšavaju svoj učinak u skladu s određenim kriterijem uspješnosti. "Zadatak algoritma strojnog učenja je pronaći prirodne uzorke i poveznice u podacima te na temelju toga steći uvid i zatim odlučiti i predviđati" [5, str. 1]. Rezultat strojnog učenja nad nekim setom podataka nazivamo modelom strojnog učenja [3]. Često se pojmovi strojno učenje i umjetna inteligencija miješaju te ponekad pogrešno koriste kao sinonimi, iako to nisu. Umjetna inteligencija je širi pojam koji obuhvaća cijelo područje stvaranja inteligentnih strojeva dok je strojno učenje specifičan pristup stvaranju sustava AI-ja koji mogu učiti iz podataka.

Strojno učenje možemo podijeliti na dva pristupa: nadzirano, nenadzirano (te potpomognuto učenje) [5] [3] [6], od kojih svaki ima jedinstvene karakteristike i primjene. Nadzirano učenje koristi označene ciljne podatke za treniranje modela, nastojeći predvidjeti točne izlaze za nove ulazne podatke, što ga čini pogodnim za zadatke klasifikacije i regresije. Nasuprot tome, nenadzirano učenje radi s neoznačenim podacima, tražeći strukture i uzorke bez unaprijed definiranih ciljeva, što je korisno za grupiranje i smanjenje dimenzionalnosti. Potpomognuto učenje predstavlja srednji put, gdje agent uči kroz interakciju s okolinom, primajući povratne informacije u obliku nagrada ili kazni, što ga čini idealnim za probleme odlučivanja i kontrole.



Slika 2: Opća podjela strojnog učenja [5]

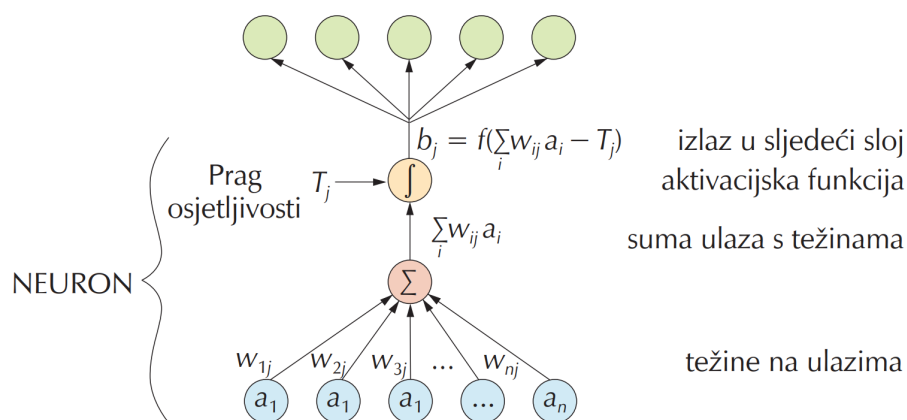
Unutar te podjele razlikujemo opće tehnike. Nadzirano učenje uključuje klasifikaciju, koja koristi tehnike potpornih vektora za razvrstavanje podataka u kategorije, te regresiju, koja primjenjuje metode poput linearne regresije za predviđanje numeričkih vrijednosti. Nenadzirano učenje, s tehnikama kao što su k-means za grupiranje i PCA za smanjenje dimenzionalnosti. Potpomognuto učenje, koristeći algoritme poput Q-učenja, omogućuje agentima učenje

kroz interakciju s okolinom. Nadograđujući se na ove koncepte, duboko učenje koristi višeslojne neuronske mreže, uključujući konvolucijske (CNN) i rekurentne (RNN) arhitekture, za automatsko učenje kompleksnih i velikih skupova podataka, unaprjeđujući područja poput računalnog vida i obrade prirodnog jezika [5] [3] [6].

## 2.2. Neuronske mreže i duboko učenje

Hrvatska enciklopedija [7] definira neuronsku mrežu kao: "sustav međusobno povezanih elemenata (nazvanih umjetni neuroni) koji služi za raznovrsna izračunavanja, zasnovan na pokušaju oponašanja rada ljudskoga mozga". Neuronsku mrežu čine umjetni neuroni inspirirani biološkim neuronima. Mreža se sastoji od međusobno povezanih neurona raspoređenih u najmanje tri sloja. Svaki sloj sadrži nekoliko neurona koji su međusobno povezani. Ako neuron prepozna podatke kao značajne, on te podatke prosljeđuje sljedećem neuronu [3]. Prvi sloj je ulazni, koji prima početne podatke. Zatim slijedi jedan ili više skrivenih slojeva, gdje se odvija glavna obrada informacija. Konačno, izlazni sloj pruža rezultate obrade [3].

Prva razvijena te najjednostavnija neuronska mreža je Perceptron. Sastoji se od ulaznih podataka koji se množe s težinama, a zatim se zbroje. "Dobiveni rezultat zbrajanja ulazna je vrijednost odabrane prijenosne funkcije neurona, čiji rezultat čini izlaznu vrijednost neurona. Nakupine takvih neurona međusobno su spojene tako da izlazne vrijednosti jednoga neurona postaju ulaznim vrijednostima drugih, te zajedno tvore neuronsku mrežu" [7]. Ovaj zbroj se propušta kroz aktivacijsku funkciju koja odlučuje hoće li perceptron biti aktivan ili ne, ovisno o definiranom pragu. Učenje perceptrona počinje nasumičnim postavljanjem malih vrijednosti težina. Zatim se, koristeći skup podataka za treniranje, izračunava klasifikacija za svaki primjer. Ako izračunata klasifikacija odgovara stvarnoj, težine ostaju iste. U suprotnom, težine se prilagođavaju. Ovaj proces se ponavlja dok svi primjeri nisu ispravno klasificirani ili dok se ne postigne linearna odvojivost primjera u hiperravnini [6]. Ovakav mehanizam omogućuje perceptronu da uči iz podataka i poboljšava svoju sposobnost klasifikacije, što ga čini temeljnim elementom u izgradnji složenijih neuronskih mreža [6] [8].

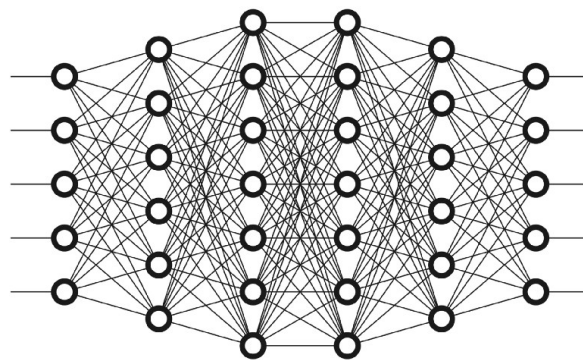


Slika 3: Prikaz arhitekture Perceptrona [8]



Duboko učenje je podskup strojnog učenja i umjetne inteligencije koji se temelji na korištenju neuronskih mreža za analizu i obradu velikih količina podataka [6, str. 64]. Duboko učenje i neuronske mreže usko su povezani, no postoji nekoliko ključnih razlika koje ih razdvajaju. Tradicionalne neuronske mreže obično imaju jedan ili dva skrivena sloja između ulaznog i izlaznog sloja, dok s druge strane, duboko učenje koristi "duboke" neuronske mreže, koje se sastoje od mnogo skrivenih slojeva (ponekad desetaka ili čak stotina) što omogućava modelima da uče složenije reprezentacije podataka [9]. Duboko učenje je posebno učinkovito u radu s nestrukturiranim podacima kao što su slike, zvuk i tekst, te se koristi u raznim područjima, uključujući prepoznavanje govora, obradu prirodnog jezika, autonomna vozila i medicinsku dijagnostiku [6] [3].

*Jedan od ključnih elemenata dubokog učenja je sposobnost modela da automatski uči reprezentacije podataka na višim razinama apstrakcije kroz više slojeva neuronskih mreža. Ovi slojevi omogućuju modelima da prepoznaju složene značajke u podacima, što je od vitalnog značaja za zadatke poput prepoznavanja lica, analize sentimenta u tekstu, te generiranja umjetničkih djela. Duboko učenje koristi se i u naprednim aplikacijama poput generativnih modela, koji su u stanju stvarati nove uzorke podataka, kao što su slike ili glazba, što je posebno relevantno u kontekstu kreativnih industrija. Unatoč svojoj učinkovitosti, duboko učenje dolazi s nizom izazova. Jedan od njih je potreba za velikim količinama podataka i računalnih resursa za treniranje složenih modela. Također, modeli dubokog učenja često djeluju kao "crne kutije," što znači da je teško objasniti kako donose određene odluke. Ova nedostatak transparentnosti može biti problematičan u područjima gdje je razumijevanje razloga za odluku ključno, kao što su medicina ili pravni sustav [9].*



Slika 4: Neuronska mreža dubokog učenja [10]

## 2.2.1. Transformeri

Istraživači iz Googlea, predvođeni Vaswanijem i suradnicima, objavili su 2017. godine rad pod naslovom "Attention Is All You Need" [11], u kojem su definirali novi oblik neuronske mreže nazvan transformer. Temelji se na mehanizmu samoposvećenosti koji uzima u obzir kontekst elemenata u nizu, a ne samo same elemente [3]. Kroz samoposvećenost, model može prepoznati čak i suptilne načine na koje se dijelovi skupa podataka međusobno povezuju. Model se temelji na enkoder-dekoder arhitekturi s otprilike 65 milijuna parametara, što je u to vrijeme predstavljalo revolucionaran iskorak [12]. Ova inovacija značajno je unaprijedila performanse u zadacima kao što su strojno prevođenje i obrada prirodnog jezika kao i performanse generativne inteligencije i jezičnih modela [12] [3]. Za razliku od rekurentnih neuralnih mreža (RNN) koje obrađuju jezik sekvencijalno, i konvolucionih neuralnih mreža (CNN) koje zahtijevaju više koraka za povezivanje udaljenih dijelova unosa, transformeri koriste mehanizam samoposvećenosti koji direktno modelira odnose između svih riječi u rečenici, bez obzira na njihovu poziciju [13]. Ova arhitektura omogućava transformerima da efikasnije obrađuju jezične podatke, posebno u slučajevima gdje je potrebno razumjeti kontekst na većim udaljenostima unutar teksta [13]. Osim poboljšane računalne učinkovitosti i veće točnosti, jedan zanimljiv aspekt transformera je mogućnost vizualizacije dijelova rečenice na koje mreža obraća pažnju pri obradi ili prevođenju određenih riječi, čime se dobiva uvid u način na koji informacije prolaze kroz mrežu [13]. U kontekstu transformatorske arhitekture, Blašković ističe dvije ključne inovacije u obradi teksta [12]:

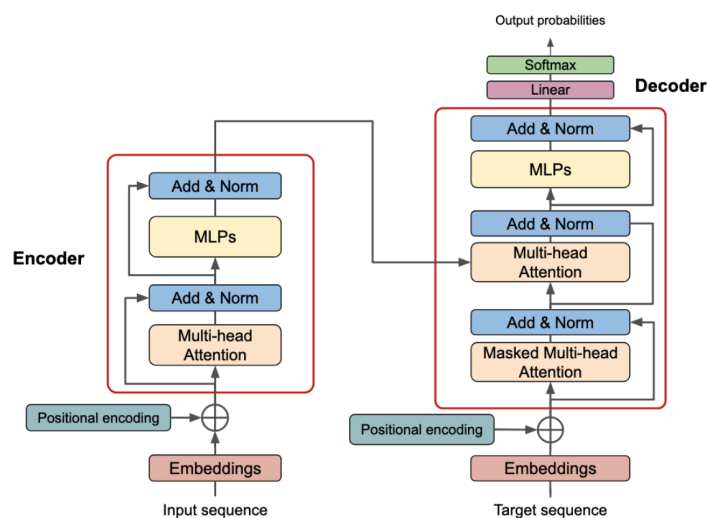
1. **Pozicijsko kodiranje** - rješava problem nedostatka informacije o redosljedu ulaznih elemenata dodavanjem vektora koji kodiraju poziciju svakog tokena u nizu.
2. **Mehanizam samoposvećenosti** - omogućuje modelu izračun težinskih faktora za svaku riječ u odnosu na ostale u sekvenci, što vodi do učinkovitijeg predviđanja sljedećih riječi i implicitnog učenja gramatičkih pravila.

Prema Blaškoviću i Vaswaniju [12] [11], rad transformer modela može se podijeliti u nekoliko ključnih koraka i karakteristika:

1. **Pretvaranje ulaznih podataka u vektorske reprezentacije:** Ulazna rečenica se pretvara u numeričke reprezentacije poznate kao vektorska ugnježđenja (eng. vector embeddings), koje kodiraju značenja tokena tj. riječi u kontekstu.
2. **Pozicijsko kodiranje:** Dodaju se vektori koji kodiraju poziciju riječi u rečenici, omogućujući modelu prepoznavanje redosljeda riječi i održavanje kontekstualnih veza. Transformer model koristi sinusoidne funkcije za izračunavanje pozicijskog kodiranja, čime se omogućava generalizacija na dulje sekvence od onih korištenih u fazi treniranja.
3. **Višestruki mehanizam samoposvećenosti:** Ovaj mehanizam radi kroz više paralelnih procesa, omogućujući modelu da se usredotoči na različite karakteristike ulaznih podataka i nauči odnose među tokenima. Transformer koristi više glava posvećenosti, gdje svaka glava paralelno obrađuje različite aspekte podataka.

4. **Normalizacija slojeva i povezivanje reziduala:** Treniranje se stabilizira te ubrzava; normalizacija smanjuje varijabilnost podataka, a rezidualne veze olakšavaju protok informacija kroz mrežu.
5. **Neuronske mreže širenja unaprijed (FNN):** Izlazni podaci iz sloja samoposvećenosti prolaze kroz ove mreže, koje primjenjuju nelinearne transformacije na vektorske reprezentacije. FNN se sastoji od dvije linearne transformacije sa ReLU aktivacijom između njih, a sve se primjenjuju na svaku poziciju zasebno i identično.
6. **Naslagani slojevi:** Cijela struktura transformer modela sastoji se od više naslaganih slojeva, što omogućuje obuhvaćanje širokog spektra apstraktnih obrazaca u podacima.
7. **Izlazni sloj:** U zadacima sekvenca-u-sekvencu (eng. seq2seq), poput strojnog prevođenja, dodaje se decoder modul za generiranje izlazne sekvence. Dekoder se sastoji od dodatnog sloja višestruke pozornosti koji se fokusira na izlaz odnosno rezultat enkodera, omogućujući precizniji prijenos informacija.
8. **Faza treniranja:** Treniranje transformer modela obično kombinira metode nenadziranog i nadziranog učenja, koristeći velike skupove neoznačenih podataka i optimizatore poput Adama ili SGD-a. Nakon inicijalne faze treniranja, često slijedi fino podešavanje za specifične zadatke koristeći označene podatke.
9. **Faza inferencije:** U ovoj fazi, predtrenirani model koristi se za generiranje predikcija na novim podacima, demonstrirajući svoju sposobnost primjene naučenih obrazaca na neviđene ulazne sekvence.

Blašković, referencirajući se na drugi rad navodi: "*Znanstvenici sa Stanforda nazvali su, u jednom radu iz 2021. godine, ove modele "temeljnim modelima" (eng. Foundation models) budući da ih vide kao ključne komponente na kojima se temelji sav daljnji razvoj umjetne inteligencije*" [12, str. 26]. Transformatori su temelj svih većih modela generativne inteligencije kao i jezičnih modela.

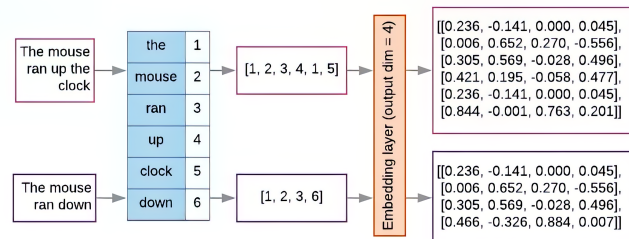


Slika 5: Arhitektura tipičnog transformera [12]

### 2.2.1.1. Tokenizacija i ugnježđenje

Tokenizacija predstavlja iznimno važan korak u procesu obrade podataka. Njena svrha je razdvojiti sirovi tekst na manje jedinice, odnosno nizove pojedinačnih tokena, čime se omogućuje detaljnija analiza i obrada teksta u daljnjim fazama obrade podataka. Tokeni mogu biti riječi, fraze ili čak znakovi, ovisno o specifičnim zahtjevima analize. Kada pišemo na prirodnom jeziku, prirodno koristimo razmake za odvajanje riječi. Zbog toga je logično da se prvenstveno usredotočimo na riječi [14]. No ponekad tokenizacija tog oblika nije moguća npr. mandarinski, japanski i sl. S druge strane, metoda temeljena na statistici koristi tehnike strojnog učenja kako bi analizirala podatke i odredila najprikladniji pristup tokenizaciji [15].

Ugnježđenja (eng. embeddings) su matematički prikazi vrijednosti ili objekata poput teksta, slika i zvuka, dizajnirani za upotrebu u modelima strojnog učenja i algoritmima semantičkog pretraživanja [3]. One prevode ove objekte u matematički oblik na temelju faktora ili karakteristika koje svaki objekt može, ali ne mora imati, kao i na temelju kategorija kojima pripadaju [3]. Ugnježđenja omogućuju modelima strojnog učenja pronalazak sličnih objekata. Na primjer, model strojnog učenja koji koristi ugnježđenje mogao bi pronaći sličnu fotografiju ili dokument ako mu se zada određena fotografija ili dokument [3]. Proces ugnježđenja odnosi se na stvaranje vektora korištenjem dubokog učenja. "Ugnježđenje" je rezultat tog procesa — drugim riječima, vektor koji duboki model strojne obrade stvara s ciljem omogućavanja pretraga sličnosti unutar tog modela [3]. Danas se ugnježđenje primjenjuje u transformer modelima te time većini sustava generativne inteligencije.



Slika 6: Proces tokenizacije i ugrađivanja [15]

### 2.2.1.2. Vektorska baza podataka

"Vektorska baza podataka je zbirka podataka pohranjenih kao matematičke reprezentacije" [3]. Baze podataka koje koriste vektorske reprezentacije unapređuju sposobnost sustava strojnog učenja da zadrže i iskoriste prethodne informacije [3]. To proširuje primjenu takvih sustava na područja kao što su napredna pretraga i semantičko pretraživanje, personalizirane preporuke i stvaranje tekstualnoga sadržaja. Umjesto oslanjanja na egzaktno podudaranje, ovi sustavi mogu prepoznati sličnosti među podacima, što im omogućuje da shvate njihov širi kontekst i značenje [3]. Vektor predstavlja niz numeričkih vrijednosti koji opisuje poziciju točke u više dimenzija te su često dugi i složeni, jer predstavljaju položaj svakog objekta kroz desetke ili čak stotine dimenzija [3]. U vektorskoj bazi podataka, svaki vektor odgovara određenom objektu.

## 2.3. Obrada prirodnog jezika (NLP)

Obrada prirodnog jezika odnosno "NLP dio je područja umjetne inteligencije koje je povezano s lingvistikom. *Budući da računalo „ne razumije“ prirodne jezike, postoji potreba za obradom prirodnog jezika*" [16, str. 373]. Cilj NLP-a je razvoj tehnologija koje omogućuju računalima da komuniciraju na način koji je prirodan za ljude, te da osim razumijevanja prirodnog jezika, razumije i kontekstualne, kognitivne i psihološke procesa pri njegovoj uporabi [12]. NLP je osnova bilo kakve umjetne inteligencije koja je u svojoj funkciji koristi ili obrađuje jezik, bilo kao virtualni asistent, prevoditelj, inteligentna tražilica, algoritam na društvenim mrežama itd. [3]. U području obrade prirodnog jezika često se susrećemo s nestrukturiranim podacima. Primjeri takvih podataka su objave na društvenim mrežama, novinski članci, e-mailovi i recenzije proizvoda. Kako bi obradio takve tekstove, NLP mora razumjeti strukturu i gramatička pravila prirodnog jezika. Pod NLP možemo svrstati prepoznavanje jezika (NLU), generiranje jezika (NLG), prepoznavanje i generiranje govora (TTS i STT), sumerizaciju, analizu mišljenja itd.

Ljudski jezik razlikuje se od računalnog jezika. računali jezik je egzaktni, temeljen na naredbama i numeričkim vrijednostima dok ljudski može biti dvosmislen, ovisan o širem kontekstu i stanju čovjeka. Kako bi računala riješila taj problem pristupaju mu na sličan način kao i ljudi: analizom šireg konteksta u kojem se riječ nalazi i zaključivanjem na temelju prethodnih iskustava [16]. Konačni izazov s kojim se NLP mora suočiti jest položiti Turingov test, koji je detaljno opisan ranije u radu.

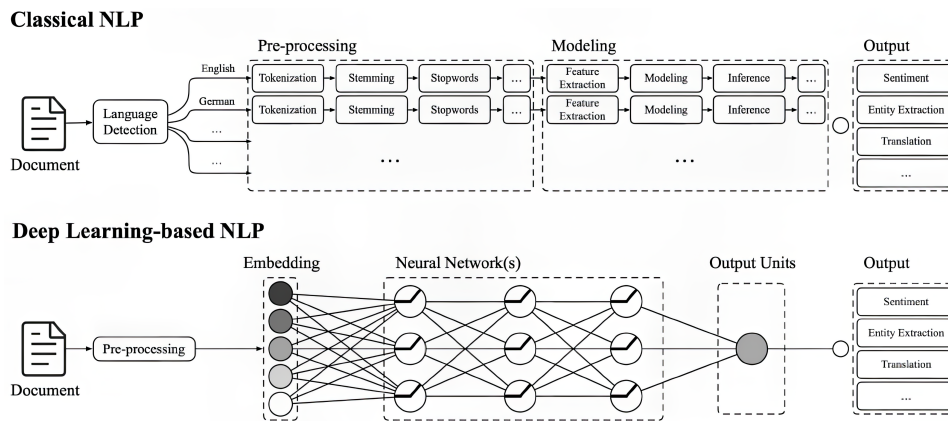
Na samom početku obrade prirodnog jezika potrebno je tekst prirodnog jezika obraditi tekst kako bi se omogućila njegova analiza s pomoću programa ili modela strojnog učenja [3]. U kontekstu obrade prirodnog jezika (NLP), postoji nekoliko ključnih tehnika predobrade i postobrade teksta koje omogućuju učinkovitiju analizu i razumijevanje jezičnog sadržaja. Jedna od temeljnih metoda je pretvaranje svih znakova u mala slova, što pojednostavljuje obradu eliminirajući nepotrebne razlike između velikih i malih slova koje nemaju semantičko značenje [3] [16]. Nadalje, postupci stemizacije i lematizacije imaju za cilj svođenje riječi na njihove osnovne oblike. Stemizacija se fokusira na uklanjanje sufiksa i prefiksa, dok lematizacija ide korak dalje, transformirajući riječi u njihove kanonske oblike koje bismo mogli pronaći u rječniku [3]. Ove tehnike su posebno korisne za smanjenje dimenzionalnosti vokabulara i grupiranje semantički sličnih riječi [14] [16]. Tokenizacija predstavlja fundamentalni korak u NLP-u, gdje se tekst raščlanjuje na manje jedinice značenja, bilo to riječi, fraze ili slično. Ovaj proces omogućuje detaljniju analizu strukture i značenja teksta. Komplementarna tehnika je uklanjanje tzv. stop-riječi, koje, iako gramatički važne, često ne pridonose značajno semantičkom sadržaju teksta [3] [14]. Eliminacijom ovih riječi fokus se stavlja na ključne pojmove i ideje u tekstu. Primjena ovih tehnika predobrade značajno poboljšava učinkovitost i preciznost NLP sustava, omogućujući im da se usredotoče na važne aspekte jezičnog sadržaja. Ove metode zajedno čine osnovu za naprednije analize teksta, uključujući strojno prevođenje, ekstrakciju informacija i analizu sentimenta [17].

Razvoj umjetne inteligencije pratio je i paralelni razvoj obrade prirodnog jezika pogotovo u području prevodenja jezika. Prvi oblici umjetne inteligencije za obradu prirodnog jezika temeljila su se na pravilima. Ova tehnika započela je 1950-ih godina, oslanjajući se na ranije radove na modelima računanja, koji su inspirirali istraživanja o neuronima i računalnim modelima izračunavanja. U to vrijeme razvijene su teorije o konačnim automatima i regularnim izrazima [14]. Na temelju rada iz 1948. godine, koji je primijenio vjerojatnosne metode na automate u jeziku, stvorena je ideja o korištenju modela s ograničenim brojem stanja za karakterizaciju gramatike, što je dovelo do razvoja teorije formalnih jezika, uključujući beskontekstne gramatike [14]. NLP temeljen na pravilima koristi informacije o izvornim i ciljnim jezicima, preuzete iz rječnika i gramatika te primjenjuje skup jezičnih pravila kako bi generiralo prijevod. Nedostatci NLP-a temeljenog na pravilima uključuju ograničenu dostupnost kvalitetnih rječnika, potrebu za ručnim postavljanjem određenih jezičnih informacija, složenost pravila u velikim sustavima te poteškoće s idiomima i prilagodbom na nove domene [16] [12].

Statistički pristup obradi prirodnog jezika i strojnom prevodenju značajno je unaprijedio ovo područje od 1980-ih godina, posebice kroz angažman IBM-a [12]. Temeljeći se na analizi velikih skupova podataka, ovaj pristup zamijenio je ranije metode temeljene na pravilima, donoseći veću fleksibilnost i sposobnost prilagodbe [14]. Ključni napredak uključuje razvoj prijevoda temeljenog na frazama (PBSMT) 2003. godine, sustave temeljene na sintaksi (SSMT) i hijerarhijske sustave (HPBSMT) [16]. Ranih 2000-ih, razvoj Interneta omogućio je pristup većem broju jezičnih podataka, potičući razvoj tehnika nenadziranog učenja [12]. Uvedeni su algoritmi poput skrivenih Markovljevih modela (HMMs) i modela maksimalne entropije, koji su postali standardni alati u statističkom NLP-u [12]. Ovaj pristup koristi modeliranje vjerojatnosti, teoriju informacija i linearnu algebru za automatizaciju jezičnih konteksta i prijevoda [14] [16]. Unatoč većoj fleksibilnosti i prilagodljivosti, statistički pristup suočava se s izazovima poput poravnanja rečenica i riječi, prevodenja idioma te razumijevanja konteksta i semantike jezika (redoslijeda) [12].

*"Razvojem procesorske moći računala otvaraju se nove mogućnosti u obradi ogromnog broja različitih podataka, prvenstveno nestrukturirana tipa, kao što je tekst, govor, slika ili video"* [16, str. 379]. Tradicionalni pristupi NLP-a oslanjali su se na niz ručno definiranih pravila i statističkih modela koji su zahtijevali značajnu količinu preprocesiranja podataka, uključujući lematizaciju, uklanjanje zaustavnih riječi i ekstrakciju značajki i slično [17]. Ovi pristupi, iako efikasni za specifične zadatke i jezike, pokazali su se ograničenima u fleksibilnosti i sposobnosti skaliranja na složenije jezične zadatke i višejezične sustave. S razvojem dubokog učenja, došlo je do duboke promjene u načinu na koji se pristupa obradi prirodnog jezika. Duboko učenje omogućuje modelima da uče složene reprezentacije jezika direktno iz podataka bez potrebe za ručnim dizajniranjem značajki [17] [12]. Umjesto toga, modeli dubokog učenja, kao što su konvolucijske neuronske mreže (CNN), rekurentne neuronske mreže (RNN) i transformeri, koriste se za automatsko učenje relevantnih značajki kroz višeslojne neuronske mreže [12] [11]. Ključne prekretnice u ovom razvoju uključuju uvođenje tehnika ugnježđivanja riječi poput Word2Vec-a 2013. godine, koji su omogućili bolje razumijevanje semantičkih odnosa između riječi, te razvoj arhitekture transformera 2017. godine [12][11] [18]. Transformeri su postavili temelje za razvoj moćnih jezičnih modela kao što su BERT i GPT, značajno unapri-

jeđujući performanse na širokom spektru NLP zadataka [12] [17] [18]. Današnji modeli velikih jezika (LLM) predstavljaju vrhunac ove evolucije, demonstrirajući sposobnost generalizacije na širok spektar zadataka bez specifičnog treniranja za svaki zadatak [12] [17] [18]. Ovakav pristup omogućava NLP sustavima da bolje rukuju s nijansama i varijacijama u ljudskom jeziku, pružajući fleksibilnije i robusnije rješenje za razumijevanje i generiranje prirodnog jezika [3].



Slika 7: Usporedba klasičnog statističkog NLP-a i NLP-a implementiran dubokim učenjem [17]

## 2.4. Generativna umjetna inteligencija (GenAI)

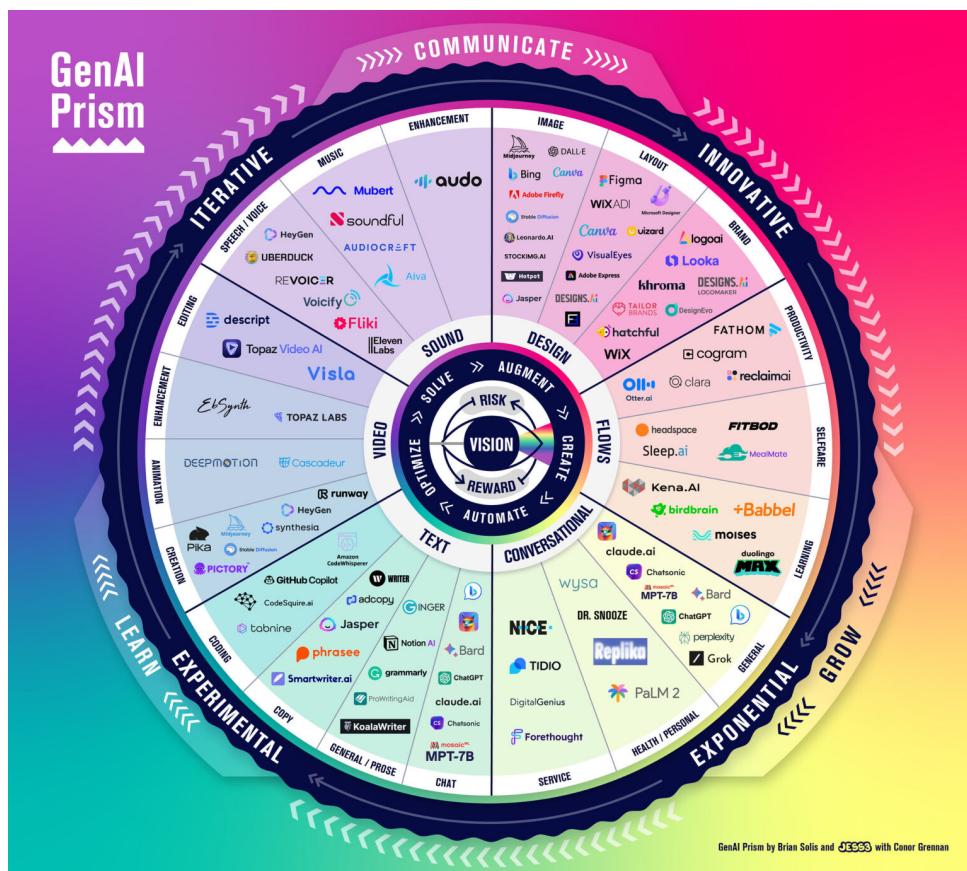
Generativna umjetna inteligencija (GenAI) predstavlja revolucionarni napredak u području umjetne inteligencije, sposobna za stvaranje raznovrsnog sadržaja poput teksta, slika, računalnog koda i audiovizualnih materijala [3]. Ovi napredni modeli, temeljeni na dubokom učenju i neuronskim mrežama, obučavaju se na ogromnim količinama podataka, što im omogućuje generiranje sadržaja koji je statistički vjerojatan i relevantan za dani unos [3]. Za razliku od tradicionalnih AI modela koji se fokusiraju na analizu i klasifikaciju, GenAI modeli mogu stvarati potpuno nove podatke slične primjerima na kojima su obučeni [19], otvarajući nove mogućnosti u kreativnim i praktičnim primjenama.

Ključne komponente GenAI sustava uključuju transformere s mehanizmom samoposvećenosti, koji omogućuju procesiranje i kontekstualizaciju velikih blokova informacija, te sofisticirane arhitekture neuronskih mreža [3]. Postoji nekoliko vrsta generativnih AI modela, svaki sa svojim jedinstvenim prednostima. Generative Adversarial Networks (GAN) koriste generator i diskriminator za stvaranje realističnih podataka, dok se Generative Pre-trained Transformer (GPT) modeli temelje na arhitekturi transformatora za generiranje koherentnog teksta [19]. Variational Autoencoders (VAE) i Flow-based modeli omogućuju kontrolirano generiranje i manipulaciju podacima, dok autoregresivni modeli uspijevaju u generiranju sekvencijalnih podataka [19].

Primjene GenAI su široke i raznolike, transformirajući brojne industrije. U obrazovanju, ovi modeli omogućuju stvaranje personaliziranih materijala za učenje i virtualnih instruktora [12] [3] [19]. U medicinskom području, GenAI ubrzava proces otkrivanja lijekova, pomaže u generiranju medicinskih slika i razvoju personaliziranih planova liječenja [19]. Industrija igara i zabave koristi GenAI na nekoliko načina, uključujući proceduralno generiranje sadržaja, dizajn



likova i interaktivno pripovijedanje. U virtualnoj i proširenoj stvarnosti, tehnologija pomaže stvoriti imerzivna 3D okruženja i interaktivne narative [19]. Međutim, uz sve prednosti, GenAI se suočava s značajnim izazovima i etičkim pitanjima. Pristranost u podacima za obuku može dovesti do diskriminacije i nezastupljenosti kroz generirani sadržaj [3][19]. Postoji zabrinutost oko potencijalne zlouporabe ove tehnologije za stvaranje "deepfakesa" i širenje dezinformacija [3][19]. Kompleksnost ovih modela, kao i kod ostalih primjera dubokog učenja, često rezultira "crnom kutijom" gdje je teško razumjeti i objasniti proces donošenja odluka, što predstavlja izazove za transparentnost i odgovornost. Dodatno, rukovanje ogromnim količinama podataka potrebnih za obuku ovih modela otvara pitanja privatnosti i sigurnosti podataka [3]. Važno je također spomenuti probleme "haluciniranja", gdje modeli mogu generirati netočne ili izmišljene informacije, te nenamjernog plagiranja, gdje postoji rizik reprodukcije postojećeg sadržaja bez odgovarajućeg priznavanja i navođenja izvora [3]. Kako bi se adresirali ovi izazovi i osigurala odgovorna uporaba GenAI tehnologije, neophodan je razvoj regulatornih okvira, etičkih smjernica i praksi transparentnosti [19]. Ovo uključuje rigorozno testiranje modela na pristranosti, implementaciju mehanizama za objašnjavanje odluka AI sustava, te strogu zaštitu privatnosti podataka korištenih u obuci kao i izvore njihovog prikupljanja [19] [20]. Unatoč svim izazovima, generativna umjetna inteligencija nastavlja transformirati način na koji stvaramo, analiziramo i konzumiramo informacije, pružajući potencijalni transformacijski utjecaj na gotovo svaki aspekt društva i gospodarstva.



Slika 8: Pregled svih oblika generativne umjetne inteligencije [21]



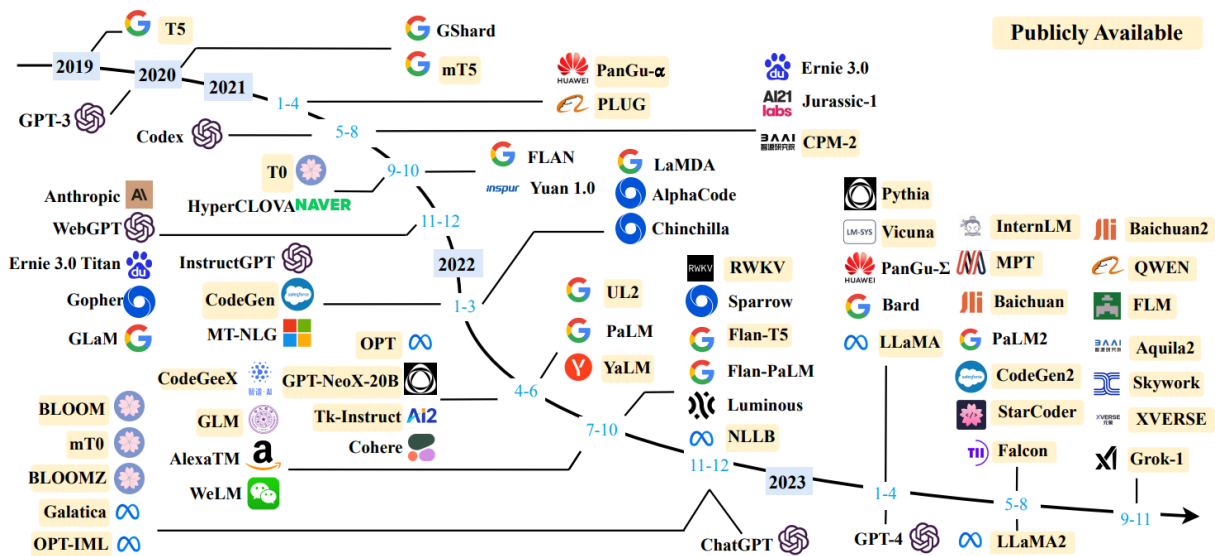
## 2.4.1. Veliki jezični modeli (LLM)

Prije pojave velikih jezičnih modela kakve poznajemo danas, obrada prirodnog jezika oslanjala se na jednostavnije sustave za modeliranje jezika. Jezični modeli (LM) modeliraju vjerojatnost sekvenci riječi radi predviđanja sljedeće riječi na temelju prethodnih. U početku se koristile statističke metode, ali su bile ograničene u kontekstu i performansama zbog problema "kletve dimenzionalnosti" [18]. Razvojem neuronskih jezičnih modela (NLM) omogućeno je bolje učenje jezika korištenjem neuronskih mreža, no ti su modeli i dalje imali ograničenja u složenim zadacima, posebno kada se radilo o problemima koji zahtijevaju razumijevanje konteksta izvan neposredne sekvence riječi [18]. Zhao i suradnici [18] u svom radu evolucija jezičnih modela dijele na četiri glavne faze:

1. **Statistički jezični modeli (SLM)** - razvijeni 1990-ih, temeljeni na Markovljevoj pretpostavci i n-gram modelima. Imali su problema s dimenzionalnošću.
2. **Neuronski jezični modeli (NLM)** - uvedeni 2000-ih, koriste neuronske mreže za modeliranje distribucije riječi. Omogućili učinkovito učenje reprezentacija riječi.
3. **Prethodno trenirani jezični modeli (PLM)** - poput ELMo-a i BERT-a, uveli paradigmu predtreniranja i finog podešavanja.
4. **Veliki jezični modeli (LLM)** - započeli s GPT-3, pokazuju iznenađujuće sposobnosti pri skaliranju na stotine milijardi parametara.

Veliki jezični modeli (LLM) predstavljaju značajan napredak u području umjetne inteligencije, nadograđujući prethodne jezične modele s ciljem rješavanja složenijih zadataka i generiranja smislenog sadržaja. Oni su ključna komponenta generativne umjetne inteligencije (GenAI), jer omogućuju stvaranje teksta, koda i drugih vrsta sadržaja relevantnih za dani kontekst [3]. "*Veliki jezični modeli koriste vrstu strojnog učenja zvanu duboko učenje kako bi razumjeli kako znakovi, riječi i rečenice funkcioniraju zajedno*" [3]. LLM je naprednija verzija LM-a, koja koristi masivne modele s milijardama parametara i velikim skupovima podataka. LLM-ovi se temelje na arhitekturi transformatora [3], slično kao i mnogi GenAI modeli, i obučavaju se na ogromnim količinama podataka, što opravdava njihov naziv "veliki". Razvoj LLM-ova prati određene zakone skaliranja koji opisuju kako performanse modela rastu s povećanjem veličine modela, količine podataka i računalnih resursa. Dva ključna zakona su KM zakon skaliranja i Chinchilla zakon skaliranja [18]. Ključna karakteristika LLM-a je njihova sposobnost da odgovaraju na nepredvidiva pitanja, koristeći prirodni ljudski jezik i analizu podataka za pružanje smislenih odgovora, čak i na neuređene upite [3]. LLM-ovi omogućuju razvoj pre-treniranih modela koji mogu rješavati razne zadatke bez potrebe za specifičnim podešavanjem, postajući univerzalni alati za obradu jezika [18]. LLM-ovi također pokazuju emergentne sposobnosti poput učenja u kontekstu (eng. in-context learning), slijeđenja uputa (eng. instruction following) i lančanog zaključivanja (eng. chain-of-thought reasoning) [18]. Te se sposobnosti pojavljuju tek pri određenoj veličini modela. Međutim, pouzdanost informacija koje pružaju LLM-ovi ovisi o kvaliteti podataka koje mu se daju. Ako im se daju netočne informacije, oni će korisniku vratiti netočne odgovore. Također ponekad "haluciniraju", stvarajući lažne informacije kada ne mogu proizvesti točan odgovor [3] [18] [20].

Glavna razlika između LM-a i LLM-a leži u veličini modela, njihovim sposobnostima te širini primjene, pri čemu LLM-ovi funkcioniraju kao univerzalni alati s boljim performansama na raznovrsnim zadacima, što je omogućilo njihovu široku primjenu u stvarnom svijetu, od automatizacije uredskih poslova do razvoja naprednih dijaloških sustava poput ChatGPT-a [18]. Iako su LLM-ovi specijalizirani za jezične zadatke, oni također čine temelj za mnoge GenAI aplikacije koje nadilaze obradu jezika, obuhvaćajući i multimodalne zadatke. Najnoviji modeli, poput GPT-4o i Claude Sonnet, već integriraju mogućnosti rada s više modaliteta, uključujući tekst, slike i druge vrste podataka, čime se dodatno proširuju granice njihove primjene.



Slika 9: Vremenska crta razvoja velikih jezičnih modela većih od 100MB [18]

## 3. Primjena umjetne inteligencije za pripomoć studentima

U ovoj sekciji rada istražiti ćemo specifične primjene umjetne inteligencije u obrazovanju, s fokusom na alate i tehnologije koje pružaju podršku studentima. To uključuje ulogu agenata za razgovor, takozvanih chatbotova, koji korisnicima omogućuju interakciju i personalizirano iskustvo, te značaj generativnih modela kao što su Generative Pre-trained Transformer (GPT) i ChatGPT. Dodatno, usredotočit ćemo se na tehnologiju Retrieval-Augmented Generation (RAG), koja povezuje sposobnost generiranja i pretraživanja, omogućujući precizne i relevantne odgovore u stvarnom vremenu.

### 3.1. Agenti za razgovor (Chatbot)

Chatboti, također poznati kao agenti za razgovor ili konverzacijski agenti, softverski su programi dizajnirani za vođenje razgovora s ljudima koristeći tekstualna ili glasovna sučelja. Oni funkcioniraju tako što obrađuju prirodni jezik kako bi razumjeli upite korisnika i generirali odgovore koji su relevantni za kontekst komunikacije. Chatbotovi se široko primjenjuju u raznim sektorima, od korisničke podrške i e-trgovine do zdravstva i obrazovanja, pružajući brze odgovore, automatizirajući rutinske zadatke i poboljšavajući korisničko iskustvo u mnogim aspektima svakodnevnog života i poslovanja. U kontekstu visokog obrazovanja, chatbotovi se sve više koriste za pružanje podrške studentima [12][14], kao što je slučaj u našem projektu gdje chatbot pomaže studentima s informacijama o fakultetu, akademskim pitanjima i administrativnim procedurama.

ELIZA je bila jedan od prvih konverzacijskih agenata, zamišljena da simulira razgovor s ljudima preuzimajući ulogu terapeuta. Iako jednostavna, preoblikovala je korisnikove odgovore u pitanja bez stvarnog razumijevanja konteksta. Unatoč ograničenjima, pokazala je potencijal računalnih sustava za vođenje razgovora. Cleverbot je kasnije unaprijedio ovu ideju koristeći bazu podataka prethodnih razgovora za generiranje smislenijih odgovora. Ipak, i on je imao problema s razumijevanjem konteksta, što je ponekad rezultiralo nepredvidivim ili uvredljivim odgovorima. Današnji chatboti, zahvaljujući napretku umjetne inteligencije i obrade prirodnog jezika, znatno su sofisticiraniji, s boljom sposobnošću razumijevanja i generiranja ljudskog jezika. Neki od njih su malo starija Siri te noviji ChatGPT.

Prema Kokanovom diplomskom radu [14], chatbotovi se oslanjaju na heuristike i tehnike strojnog učenja kako bi postigli svoje ciljeve. Njihova arhitektura obično uključuje dvije glavne komponente: sučelje koje omogućuje komunikaciju između korisnika i programa te sustav za dijalog koji obrađuje tekstualni unos i generira odgovore.

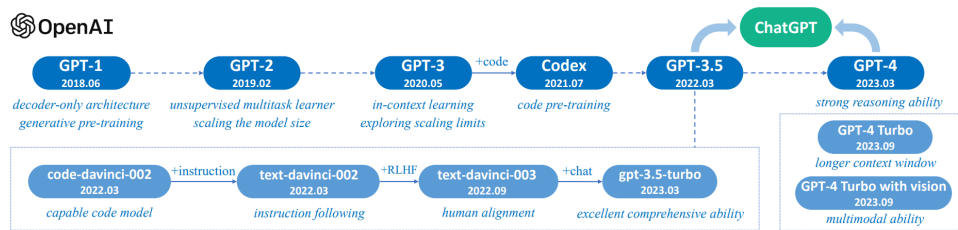
Luka Blašković u svom diplomskom radu [12] dodatno pojašnjava kako moderni chatbotovi, posebno oni temeljeni na LLM-u, kao što su GPT modeli, imaju sposobnost obrade složenijih upita i pružanja personaliziranih odgovora. Ovi modeli mogu analizirati namjeru korisnika i pružiti relevantne informacije ili rješenja u stvarnom vremenu, što ih čini posebno korisnima u raznim poslovnim aplikacijama.

## 3.2. Generative Pre-trained Transformer (GPT) i ChatGPT

"Generative Pre-trained Transformer (GPT) je napredna obitelj modela za obradu prirodnog jezika koju je razvio OpenAI" [19, str. 761]. Ova tehnologija, izgrađena na inovativnom dizajnu transformera, uči analizirajući goleme količine online tekstova. Time stječe sposobnost razumijevanja jezičnih struktura i pravila [19]. Za dani upit GPT može samostalno proizvesti tekst koji je teško razlikovati od onog koji bi napisao čovjek. Iako povijest GPT-ja nije daleka, u malen broj godina učinjena su brojne inovacije i revolucije u generativnoj inteligenciji. Prema radu Zhaoa i suradnika [18], povijest OpenAI-ja te time i GPT-ja možemo podijeliti na sljedeće verzije:

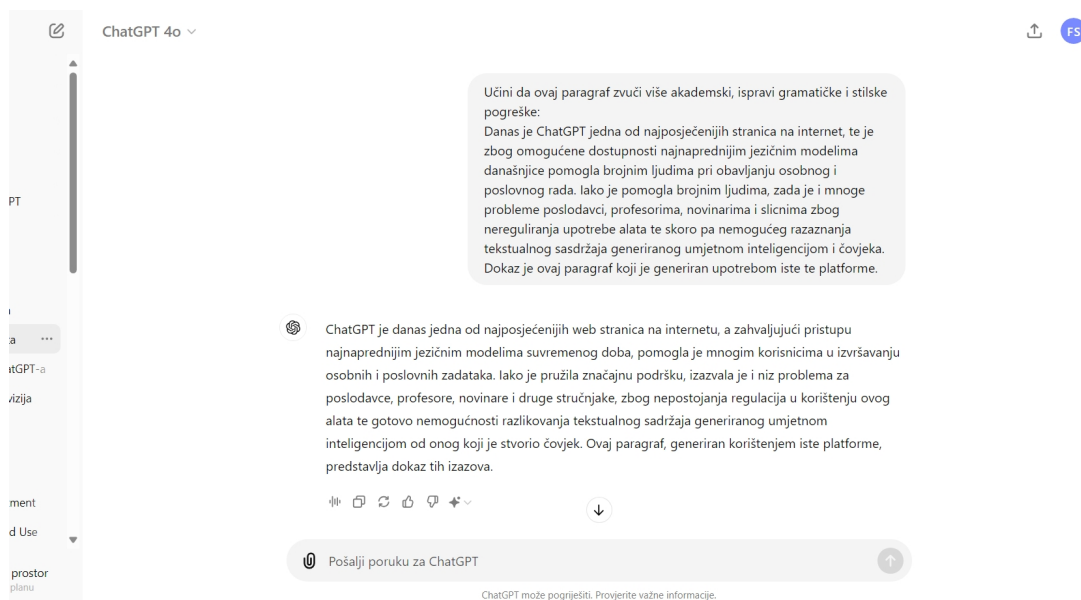
- **GPT-1:** Godine 2017. Google je predstavio transformer model [11], a tim iz OpenAI-a brzo je prilagodio svoj rad na jezičnom modeliranju ovoj novoj arhitekturi neuronske mreže. Prvi GPT model, GPT-1, objavljen je 2018. godine. GPT-1 je izgrađen na generativnoj, samo-dekodirajućoj transformer arhitekturi i koristio je kombinaciju nenadziranog predtreiranja i nadziranog finog podešavanja. Ovaj model postavio je temeljnu arhitekturu za sve buduće GPT modele, usredotočujući se na predviđanje sljedeće riječi u tekstu.
- **GPT-2:** Sljedeći model, GPT-2, zadržao je sličnu arhitekturu kao GPT-1, ali je povećao broj parametara na 1,5 milijardi te je treniran na velikom skupu podataka s internetskih stranica nazvanom WebText. GPT-2 je pokušao obavljati zadatke koristeći nenadzirano jezično modeliranje, bez eksplicitnog fino podešavanja s označenim podacima.
- **GPT-3:** GPT-3, objavljen 2020. godine, značajno je povećao broj parametara na 175 milijardi. Uveden je koncept učenja unutar konteksta (ICL), koji omogućuje modelu da razumije zadatke kroz nekoliko primjera ili čak bez njih. GPT-3 se pokazao iznimno učinkovitim u nizu zadataka u području obrade prirodnog jezika, kao i na zadacima koji zahtijevaju sposobnosti zaključivanja ili prilagodbe novim domenama.
- **GPT-3.5:** Na temelju GPT-3, OpenAI je nastavio unapređivati modele treniranjem na podacima o programskom kodu te usklađivanjem s ljudskim preferencijama. Ovo je dovelo do razvoja modela kao što je Codex, koji je sposoban rješavati složene probleme programiranja, te je dodatno poboljšan u sposobnosti rješavanja matematičkih problema.
- **GPT-4:** GPT-4, objavljen u ožujku 2023., proširio je mogućnosti na čitanje datoteka i slika, pokazujući značajna poboljšanja u rješavanju složenih zadataka. Kasnije su predstavljeni i GPT-4 Turbo, s dodatnim funkcionalnostima i poboljšanim performansama.
- **GPT-4o:** 4o bi mogli skoro zvati ultimativnom umjetnom inteligencijom. On predstavlja značajan napredak u odnosu na prethodne verzije, s više od trilijun parametara, što ga čini najmoćnijim modelom u OpenAI-jevoj liniji GPT modela [22]. GPT-4o je omni model koji prihvaća kao ulazne podatke bilo koju kombinaciju teksta, zvuka, slike i videa te generira bilo koju kombinaciju teksta, zvuka i slike kao izlaz [20]. Model je treniran end-to-end preko teksta, vizije i zvuka, što znači da svi ulazni i izlazni podaci prolaze kroz istu neuronsku mrežu [20]. GPT-4o može odgovoriti na audio ulaze za samo 232 milisekunde, s

prosječnim vremenom od 320 milisekundi, što je slično vremenu reakcije ljudi u razgovoru [20].



Slika 10: Vremenska crta razvoja GPT-ja [18]

U studenom 2022. OpenAI je predstavio ChatGPT, agent temeljen na GPT-3.5 i GPT-4, optimiziran za dijaloge [18]. Ovaj agent pokazao je iznimne sposobnosti u komunikaciji s ljudima, uključujući razumijevanje konteksta u više okretaja dijaloga i rješavanje matematičkih problema [18]. Za njega bi smo mogli reći da je među prvima AI sustavima koji prolazi turingov test. ChatGPT je danas jedna od najposjećenijih web stranica na internetu, a zahvaljujući pristupu najnaprednijim jezičnim modelima suvremenog doba, pomogla je mnogim korisnicima u izvršavanju osobnih i poslovnih zadataka. Iako je pružila značajnu podršku, izazvala je i niz problema za poslodavce, profesore, novinare i druge stručnjake, zbog nepostojanja regulacija u korištenju ovog alata te gotovo nemogućnosti razlikovanja tekstualnog sadržaja generiranog umjetnom inteligencijom od onog koji je stvorio čovjek. Ovaj paragraf, generiran korištenjem iste platforme, predstavlja dokaz tih izazova [23].



Slika 11: Sučelje ChatGPT-ja [23]

### 3.3. Retrieval-Augmented Generation (RAG)

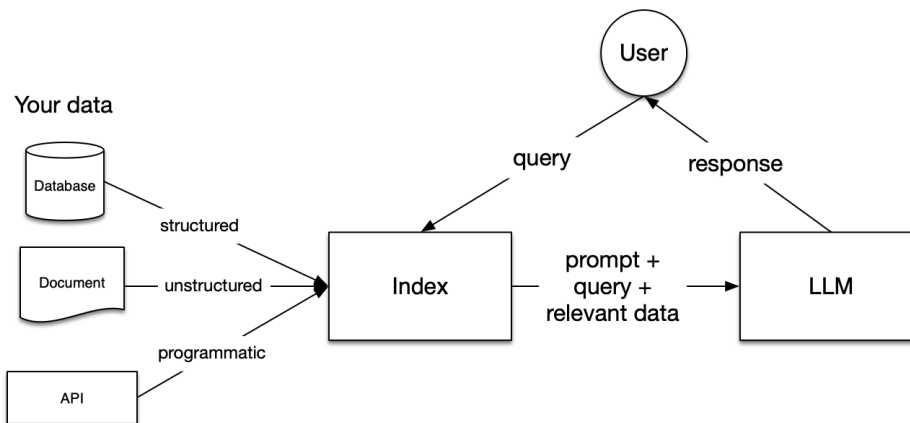
LLM-ovi se obučavaju na ogromnim količinama podataka, ali nisu obučeni na specifičnim podacima za specijalizirane slučajeve. Retrieval-Augmented Generation (RAG) rješava ovaj problem dodavanjem podataka u LLM uz već pohranjene podatke. U RAG-u, se podaci učitavaju i pripremaju za upite, odnosno indeksiraju [24]. Korisničke pretrage djeluju na indeks, koji filtrira podatke do najrelevantnijeg konteksta [24]. Taj kontekst i dani upit se zatim šalju LLM-u zajedno s uputom, nakon čega LLM pruža odgovor [24]. Ove metode omogućuju da se baze podataka obogate te da im se poboljša sposobnost pružanja točnih i kontekstualno relevantnih odgovora.

Službena LlamaIndex dokumentacija [24] definira pet ključnih faza unutar RAG-a koje su sastavni dio većine većih aplikacija koje se grade:

1. **Učitavanje:** Ova faza podrazumijeva prikupljanje podataka s različitih izvora, kao što su tekstualne datoteke, PDF-ovi, web stranice, baze podataka ili API-ji, te njihovo unošenje u radni proces.
2. **Indeksiranje:** Ovaj korak uključuje stvaranje strukture podataka koja omogućuje pretraživanje. Za LLM-ove to obično znači kreiranje vektorskih reprezentacija kao i primjenu različitih strategija metapodataka kako bi se lakše pronašli kontekstualno relevantni podaci.
3. **Pohrana:** Nakon što su podaci indeksirani, obično se želi pohraniti indeks i druge metapodatke kako bi se izbjegla potreba za ponovnim indeksiranjem.
4. **Pretraživanje:** Za svaku strategiju indeksiranja postoje različiti načini korištenja LLM-ova i LlamaIndex struktura podataka za pretraživanje, uključujući pod-upite, višekoračne upite i hibridne strategije.
5. **Evaluacija:** Ovaj korak je ključan za provjeru učinkovitosti procesa u odnosu na druge strategije ili kada se unose promjene. Evaluacija pruža objektivne mjere točnosti, vjernosti i brzine odgovora na upite.

RAG poboljšava rad jezičnih modela povećavajući točnost i ažurnost odgovora te prilagodljivost specifičnim domenama bez potrebe za ponovnim treniranjem. Međutim, RAG zahtijeva kvalitetno indeksiranje podataka, može povećati latenciju odgovora i ovisi o kvaliteti dostupnih podataka. Ova ograničenja mogu utjecati na performanse modela, posebno kod velikih i kompleksnih baza znanja ili u slučaju nepreciznih ili zastarjelih podataka. Jedan od glavnih izazova u primjeni osnovne RAG tehnike je preciznost u fazi pretrage i generiranja. Pretraga se često suočava s problemima poput pronalaženja nepovezanih ili nebitnih dijelova teksta, što može dovesti do propuštanja ključnih informacija [12] [25]. Evaluacija izvedbe RAG sustava u kontekstu Unipa-GPT-a [26] otkrila je također i izazov generiranje dugih odgovora koji ponekad mogu uključivati nebitne ili netočne informacije. Kako bi se ti problemi riješili, razvijene su napredne metode poput naprednog i modularnog RAG-a, koji omogućuju veću fleksibilnost i

skalabilnost, te dodatno unaprjeđuju preciznost pretrage i kvalitetu generiranih odgovora [12]. Također je moguće modele unaprijediti finim podešavanjem [26].



Slika 12: Osnovni RAG sustav [24]

### 3.3.1. Usporedba s drugim metodama

U usporedbi s drugim metodama poboljšanja i nadogradnje jezičnih modela, RAG nudi jedinstvene prednosti. Za razliku od fine-tuninga, koji zahtijeva ponovno treniranje modela na specifičnom skupu podataka, RAG omogućuje fleksibilnije i brže prilagođavanje novim informacijama bez potrebe za opsežnim treniranjem [12]. Ovo ga čini posebno korisnim u scenarijima gdje se baza znanja često mijenja ili proširuje. Međutim, važno je napomenuti da RAG nije univerzalno rješenje. U nekim slučajevima, kombinacija RAG-a s drugim metodama, poput fine-tuninga za specifične zadatke ili few-shot learninga za brzu adaptaciju, može pružiti optimalne rezultate [12]. Sama implementacija RAG-a ovisi o potrebama domene i sustava.

## 4. Korisnički zahtjevi

Korisnički zahtjevi definiraju što korisnici očekuju od agenta te time kako sustav treba funkcionirati. Svaka aplikacija i sustav, pa time i naš agent, mora ispuniti određene kriterije kako bi bio uspješan i prihvaćen od korisnika. Agentu za razgovor odnosno chatbotu, dano je ime Robin, akronim za "Robotska Inteligencija". Vizualni identitet agenta i primarne boje (Slika 13) izrađene su u alatu Canva [27], a primarno se referenciraju na vizualni identitet FOI-ja [28]. U slučaju chatbota "Robin", koji je razvijen kao podrška studentima na Fakultetu organizacije i informatike, korisnički zahtjevi uključuju:

1. **Pružanje točnih i ažurnih informacija iz baze podataka** – Korisnici očekuju točne i aktualne podatke, posebno u vezi s pravilnicima, informacijama o kolegijima i drugim administrativnim pitanjima.
2. **Brzi odgovori** – Chatbot mora moći brzo i učinkovito odgovarati na korisničke upite bez dugih zastoja.
3. **Razumijevanje hrvatskog jezika** – Bitno je da sustav prepozna i razumije upite na hrvatskom jeziku te izvore informacija pisane na istom jeziku.
4. **Koherentni odgovori** – Odgovori koje chatbot pruža moraju biti logični, kontekstualno relevantni i koherentni kako bi korisnici dobili jasnu i korisnu informaciju. To znači da chatbot mora razumjeti ne samo pojedinačne upite nego i širi kontekst razgovora. Na primjer, kada student postavi pitanje o nekom kolegiju, chatbot treba prepoznati na koji se kolegij misli i pružiti sve potrebne informacije. Koherencija odgovora također podrazumijeva dosljednost u i stilu komunikacije.
5. **Upućivanje na relevantne izvore** – Sustav treba korisnicima pružati i dodatne izvore informacija, poput dokumenata ili web-stranica gdje mogu detaljnije proučiti pojedine teme.
6. **Intuitivno sučelje** – Kako bi studenti lako koristili chatbot, sučelje mora biti jednostavno, pregledno i prilagođeno njihovim potrebama i svakodnevnoj uporabi.



Slika 13: Vizualni identitet agenta Robin



## 5. Korišteni alati

U procesu razvoja i implementacije ovog projekta korišten je niz alata koji su omogućili realizaciju programa. Svaki od ovih alata odigrao je ključnu ulogu u različitim fazama razvoja, od pisanja i testiranja koda do implementacije složenih funkcionalnosti kao što su rad s velikim jezičnim modelima i izrada web aplikacija. U nastavku su opisani glavni alati koji su korišteni u radu, te njihova uloga u procesu razvoja.

### Programski jezici

Python je programski jezik opće namjene poznat po svojoj jednostavnosti i modularnosti, što ga čini idealnim za razvoj velikih jezičnih modela (LLM). Zbog svoje fleksibilnosti i široke podrške za alate u području umjetne inteligencije, Python je postao ključan u implementaciji sustava za izgradnju različitih modela i sustava. U radu se Python koristio za izradu chatbota koji kombinira pretraživanje i generiranje odgovora temeljenih na specifičnim dokumentima.

Osim Pythona, također su za jednu datoteku korišteni HTML, CSS, i JavaScript. HTML definira osnovnu strukturu web stranice, označavajući elemente poput zaglavlja i polja za unos. CSS stilizira te elemente, određujući boje, fontove i raspored kako bi se postigao željeni vizualni izgled. JavaScript omogućava naprednu interakciju, upravljajući događajima poput klika i unosa.

### Visual Studio Code

Visual Studio Code (VS Code) je popularno razvojno okruženje poznato po svojoj brzini, prilagodljivosti i podršci za razne jezike. Nudi inteligentno dovršavanje koda, kontrolu verzija i široku podršku za ekstenzije. U radu se Visual Studio Code koristio za pisanje i testiranje koda. Njegove funkcije omogućile su efikasno upravljanje projektom, olakšale debugiranje i integraciju s alatima.

### ChatGPT API

API tvrtke OpenAI nudi jednostavno sučelje za pristup najmodernijim modelima umjetne inteligencije koje smo detaljnije opisali ranije u radu. Ovi modeli omogućuju naprednu obradu prirodnog jezika, stvaranje slika, pretraživanje po značenju te prepoznavanje govora [29]. Među najistaknutijim modelima su GPT-4, koji pokazuje izvanredne rezultate u generiranju tekstualnog sadržaja i razumijevanju složenih upita, te DALL-E 2, sposoban za stvaranje realističnih slika na temelju tekstualnih opisa.

Posljednji i najnapredniji OpenAI-jev multimodalni model je GPT-4o te njegova inačica GPT-4o mini. Shahriar i suradnici [22] u svom radu su analizirali i pokazali da GPT-4o model demonstrira visoku točnost i učinkovitost u zadacima koji zahtijevaju jezičnesposobnosti i sposobnosti zaključivanja. GPT-4o postiže izuzetne rezultate u zadacima poput standardiziranih

ispita, klasifikacije slika, detekcije emocija iz govora, te multimodalnim zadacima gdje se kombiniraju različiti tipovi podataka[22]. GPT-4o mini ("o" označava "omni") predstavlja najnapredniji model u kategoriji malih modela i ujedno najjeftiniji model do sada. Ovaj model je multimodalan, što znači da može prihvatiti tekstualne ili slikovne unose te proizvesti tekstualni izlaz [30]. Odlikuje se većom inteligencijom u usporedbi s modelom gpt-3.5-turbo, ali zadržava jednaku brzinu. Namijenjen je za manje zadatke, uključujući i one vezane uz računalni vid [30] [29]. GPT-4o mini sadrži 6 milijarde parametara te kontekstualni prozor upita od 128 tisuća tokena [30] [29].

Osim jezičnih modela, OpenAI također nudi i nekoliko modela za ugnježđivanje. Podsjetimo se, ugnježđivanje (eng. embedding) je vektorska reprezentacija. Udaljenost između dva vektora mjeri njihovu povezanost. Male udaljenosti ukazuju na visoku povezanost konteksta, dok velike udaljenosti sugeriraju nisku povezanost [29]. Trenutna generacije modela za ugrađivanje je verzija 3 različitih veličina.

U projektu su korišteni OpenAI-jev LLM model GPT-4o mini primarno zbog vrlo dobrog razumijevanja hrvatskog jezika te vrlo dobrog razumijevanja konteksta i pitanja kao i generiranih odgovora. Za ugrađivanje je korišten text-embedding-3-small model. U našem slučaju nisu bili potrebni preveliki jezični modeli niti modeli ugrađivanja no bilo je potrebno da oba modela budu predtrenirana za hrvatski jezik.

## **LlamaIndex**

LlamaIndex, ranije poznat kao GPT Index, je napredni okvir za rad s podacima koji pojednostavljuje integraciju velikih jezičnih modela (LLM) s vanjskim izvorima podataka. Omogućuje učitavanje, strukturiranje i indeksiranje podataka te omogućuje učinkovite upite nad velikim skupovima podataka [24]. LlamaIndex se može koristiti za izgradnju aplikacija kao što su chatbotovi i sustavi sa Retrieval-Augmented Generation (RAG) funkcionalnošću. Podržava učitavanje raznih formata podataka, kao što su PDF-ovi, baze podataka i web stranice, te njihovo strukturiranje u čvorove i indekse [31]. LlamaIndex se ističe podrškom za više jezika, a kao projekt otvorenog koda, pruža transparentnost i bogatu dokumentaciju kao i primjere koji olakšavaju upotrebu [31]. Također, omogućuje korisnicima prilagodbu modela specifičnim potrebama.

U našem radu, LlamaIndex je bio ključan alat, omogućavajući većinu funkcionalnosti našeg agenta, od obrade podataka do postavljanja upita. Njegova sposobnost da učinkovito rukuje velikim, različitim tipovima podataka i formata je bila od presudnog značaja za uspjeh projekta.

## **LlamaParse**

LlamaParse je API razvijen od strane LlamaIndex-a, koji omogućuje učinkovito parsiranje i predstavljanje datoteka radi lakšeg pretraživanja i dodavanja konteksta unutar LlamaIndex okvira [32]. Ova komponenta, koja je dio LlamaCloud-a, omogućuje pretvaranje PDF-ova u strukturirane podatke te je dostupna kao samostalni REST API, Python paket i web sučelje.

Nedavno je izašla iz faze beta testiranja. Omogućuje implementaciju multimodalne obrade dokumenata, čime se omogućava napredno "čitanje" dokumenata korištenjem multimodalnih velikih jezičnih modela (LLM) [32]. Ovaj okvir također podržava specifične instrukcije za obradu dokumenata, kao što su prilagodba system prompta, selektivno čitanje određenih stranica, te ignoriranje određenih elemenata poput naslova i podnaslova [32]. Ova fleksibilnost omogućava korisnicima da precizno kontroliraju način na koji se dokumenti obrađuju i interpretiraju.

Osnovnu verziju LlamaParse API-ja smo koristili u radu za prepoznavanje strukture dokumenata. Pri izradi i testiranju ova implementacija je postigla najbolje rezultate razumijevanja dokumenta.

## **Flask**

Flask je lagani web okvir za Python koji omogućuje jednostavnu i brzu izradu web aplikacija. Zbog svoje modularnosti i minimalizma, Flask pruža fleksibilnost programerima da izgrade aplikacije s osnovnim funkcionalnostima ili prošire mogućnosti prema potrebama projekta. Flask koristi jednostavan pristup u upravljanju rutama i ima ugrađenu podršku za razvojne servere, što ga čini idealnim za izradu prototipova i manjih web aplikacija, dok se uz dodatne ekstenzije može koristiti i za kompleksnije projekte.

## 6. Opis implementacije

Ova sekcija opisuje sve ključne korake provedene tijekom izrade chatbota Robin.

### 6.1. Pregled korištenih izvora podataka

Ime dokumenta	Opis dokumenta	Struktura dokumenta	Detalji dokumenta
IPS v1.2 predmeti	Dokument sadrži popis predmeta i njihove opise za studij Informacijski i poslovni sustavi.	Strukturiran popis predmeta, kategoriziran prema smjerovima i semestrima.	220 stranica, PDF format, uključuje tablične prikaze.
Pravilnik IPS	Pravilnik o studijima na Fakultetu organizacije i informatike, uključujući informacije o prijelazima.	Pravna struktura s člancima i podčlancima.	15 stranica, PDF format, usvojen 2021. godine.
Pravilnik o preddiplomskom i diplomskom studiju EP	Pravilnik regulira preddiplomske i diplomske studije na Ekonomskom fakultetu.	Pravna struktura s člancima i podčlancima.	18 stranica, PDF format, posljednja izmjena 2021. godine.
EP kolegiji v1.2	Dokument sadrži popis i opise kolegija za smjer Ekonomika poduzetništva.	Strukturiran popis predmeta, s osnovnim informacijama o svakom kolegiju.	206 stranica, PDF format.
Statut fakulteta organizacije i informatike	Osnovni pravni dokument koji uređuje organizaciju i djelovanje Fakulteta.	Pravna struktura s člancima, poglavljima i odjeljcima.	37 stranica, PDF format, usvojen 2020. godine.
Često postavljana pitanja FOI	Sadržava odgovore na najčešća pitanja vezana uz upise, studije i studentski život na FOI-ju.	Pitanja i odgovori, organizirani po tematskim cjelinama.	15 stranica, PDF format, ažurirano 2023. godine.
Pravilnik o završnom i diplomskom radu	Pravilnik koji regulira proces prijave, izrade i obrane završnih i diplomskih radova.	Pravna struktura s člancima i podčlancima, uključuje opis procesa i odgovornosti.	5 stranica, PDF format, usvojen 2021. godine.
Zakon o visokom obrazovanju i znanstvenoj djelatnosti	Zakon koji regulira visoko obrazovanje i znanstvenu djelatnost u Republici Hrvatskoj.	Pravna struktura s člancima i poglavljima, obuhvaća sve aspekte visokog obrazovanja.	68 stranica, PDF format, ažurirano 2022. godine.

Tablica 1: Opis i struktura korištenih dokumenata

Tablica iznad prikazuje izvore korištene za izradu baze podataka RAG agenta. Svi izvori predstavljeni su u formatu PDF dokumenata, čime se izbjegla potreba za primjenom tehnika poput web scrapinga, budući da je fakultet već osigurao sve relevantne i potrebne informacije u ovom obliku. Struktura dokumenata varira: dokumenti "IPS v1.2 predmeti" i "EP kolegiji v1.2" uglavnom su strukturirani, sadrže tablice i obuhvaćaju više od 200 stranica svaki. Nasuprot tome, ostali dokumenti su nestrukturirani i uključuju tekstualne paragrafe, liste, naslove i slično. Dokumenti poput "Pravilnik o preddiplomskom i diplomskom studiju EP" i "Zakon o visokom obrazovanju i znanstvenoj djelatnosti" su skenirane verzije fizičkih dokumenata, što rezultira nižom kvalitetom tih datoteka. Datoteka "Često postavljana pitanja FOI" jedina nije javno dostupna datoteka fakulteta već je ispisana web stranica fakulteta. Ovo je učinjeno jer je dokument jednostavnog formata, a u rad je implementiran napredan čitač sadržaja dokumenta LlamaParse [32].

## 6.2. Pregled implementacije agenta

Implementacija agenta za razgovor sastoji se od nekoliko ključnih komponenti, svaka implementirana u zasebnoj Python datoteci. Ova arhitektura omogućuje modularnost i olakšava razumijevanje i proširivanje sustava. Prilikom izrade primarno se koristila opsežna dokumentacija LlamaIndexa koja je podijeljena na kategorije: "Home" [24], "Learn" [33], "Use Cases" [34], "Example" [31], "Component Guides" [35], "API Reference" [36] itd. Djelomično je korištena OpenAI dokumentacija [29] [37]. Rad je djelomično inspiriran primjerima službenih projekata LlamaIndex-a [38] [39] [31]. U projektu je korišten Anthropic Claude za pomoć pri izradi evaluacije [40] i web aplikacije [41] kao i pri poči otklanjanja greška. Cijelu implementaciju je moguće vidjeti na [Github-u](#).

### 6.2.1. Config.py

Datoteka `config.py` sadrži konfiguracijske postavke za cijelu aplikaciju, uključujući OpenAI API ključ, postavke Flask aplikacije, putanje do indeksa i dokumenata, razinu logiranja te sistemski prompt za AI model. Funkcija `set_openai_api_key()` postavlja API ključ za OpenAI. `SYSTEM_PROMPT` je ključan za ponašanja AI modela; daje mu kontekst i upute specifične za domenu FOI-a. Ovakva centralizacija konfiguracije olakšava upravljanje postavkama i njihovu prilagodbu.

### 6.2.2. model\_utils.py

`Model_utils.py` sadrži pomoćne funkcije za postavljanje modela i pomoćnih funkcija. Funkcija `setup_logging()` konfigurira sustav za zapisivanje promjena i stanja u sustavu poput grešaka, dohvaćanja podataka i obrade podataka. `setup_settings()` inicijalizira postavke za LLM i model za ugnježdivanje, koristeći module iz `llama_index` biblioteke, primarno `Settings` varijablu. Konkretno, koristi `OpenAI` klasu za postavljanje GPT modela "gpt-4o-mini" s definiranim parametrima poput maksimalnog broja tokena i temperature. Ti parametri

određuju podešenost našeg modela na veličinu ulaza te kreativnost pri generiranju odgovora. Za ugnježdavanje se koristi `OpenAIEmbedding` s modelom "text-embedding-3-small", koji pretvara tekstualne podatke u vektorske reprezentacije. Ovaj model smo odabrali jer odlično radi na hrvatskom jeziku, a i lako ga je bilo uključiti preko API ključa kad je već LLM model bio uključen. Funkcija `is_pdf()` služi za provjeru formata datoteke.

### 6.2.3. `document_processor.py`

`document_processor.py` je srž sustava za pripremu i indeksiranje dokumenata koji služe kao baza znanja za chatbot. Njezine glavne funkcionalnosti su:

- 1. Učitavanje i obrada PDF dokumenata:** Funkcija `load_documents()` iterira kroz sve PDF datoteke u direktoriju. Za svaku datoteku koristi `LlamaParse` [32], sofisticirani alat za parsiranje PDF-ova, koji konvertira PDF u markdown format. Ovo omogućuje očuvanje strukture dokumenta i olakšava daljnju obradu teksta. Podaci se tada spremaju u `Dokument`. `LlamaIndex Dokument` je općeniti spremnik za bilo koji izvor podataka, poput PDF-a, rezultata API-ja ili dohvaćenih podataka iz baze podataka [35].
- 2. Stvaranje i upravljanje indeksom:** Funkcija `create_or_load_index()` je središnja točka za upravljanje indeksom. Ona prvo provjerava postoji li već stvoreni indeks. Ako postoji, učitava ga koristeći `StorageContext` i `load_index_from_storage`. Ako ne postoji, stvara novi indeks koristeći `VectorStoreIndex`, te ga pohranjuje funkcijom `persist()` [35] [31]. Ovaj pristup osigurava da se dokumenti ne moraju ponovno obrađivati pri svakom pokretanju aplikacije, što značajno ubrzava inicijalizaciju sustava.
- 3. Obrada markdown sadržaja:** Implementacija koristi `MarkdownElementNodeParser` [36] [32] za obradu čvorova. Koristimo `MarkdownElementNodeParser` nad već čitanim tj. parsiranim dokumentima s `LlamaParse`. Ovaj parser dijeli sadržaj na manje, semantički smislene cjeline odnosno čvorove (eng. nodes), kako bi izgradio indekse te time omogućio pretraživanje parsiranog sadržaja. Dodatne varijable parsera uključuju definiranje "broja radnika" za paralelnu obradu i LLM-a za čitanje. Ova funkcionalnost se izvršava prilikom stvaranja novog indeksa odnosno kada je potrebno čvorove proslijediti `VectorStoreIndex` za stvaranje indeksa.
- 4. Vektorizacija i indeksiranje:** kao što smo već rekli, `VectorStoreIndex` se koristi za stvaranje indeksa od obrađenih čvorova u definiranu mapu vektora iz koje će se kasnije dohvaćati. Ovaj indeks omogućuje brzo i efikasno pretraživanje semantički sličnih segmenata teksta. Prilikom stvaranja indeksa, `LlamaIndex` interno koristi model za ugnježdavanje definiran u `setup_settings()` funkciji u `model_utils.py` [33].
- 5. Pretraživanje indeksa i postavljanje upita :** Funkcija `query_index()` implementira pretraživanje indeksa i generiranje odgovora. Za dani upit `query_engine` pretražuje generiranje indeksa odnosno vektorsku bazu znanja. Koristi `similarity_top_k=5` za dohvaćanje 5 najrelevantnijih rezultata, što poboljšava preciznost odgovora. Ovisno o danom upitu, funkcija dohvaća relevantni kontekst te šalje LLM na sintezu odgovora [33].

6. **Testiranje indeksa:** Na kraju skripte implementirana je funkcionalnost za testiranje indeksa s nekoliko predefiniраниh upita, što služi za brzu verifikaciju ispravnosti indeksa i kvalitete odgovora.

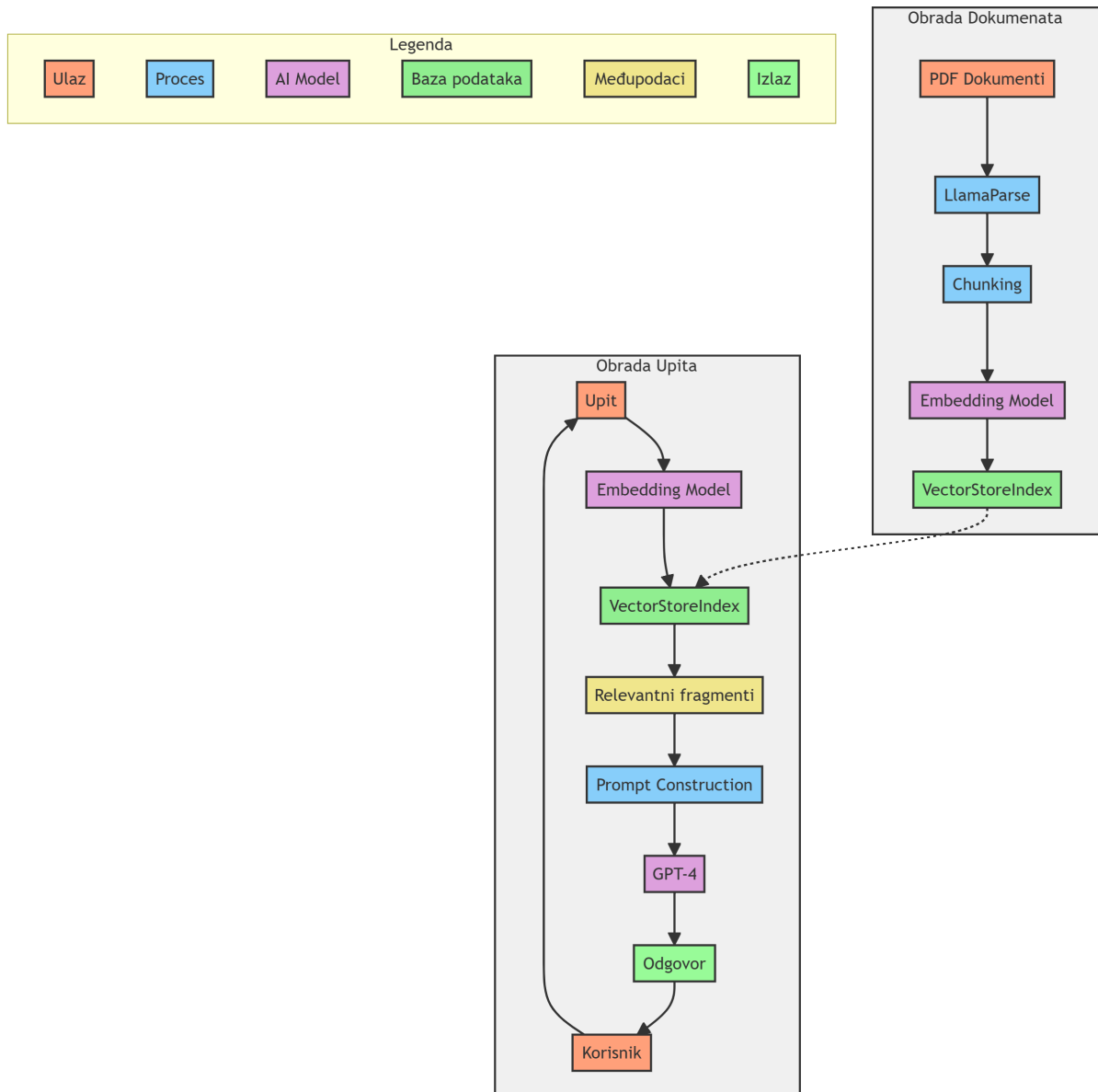
Datoteka je povezana s `config.py` i `model_utils.py` kao bi znala s koje lokacije treba čitati, provjeravat da li su pdf dokumenti, na koju lokaciju spremi indekse itd.

#### 6.2.4. chat.py

Datoteka `chat.py` omogućuje korisnicima pristup znanju kroz intuitivno chatbot sučelje preko terminala. Implementacija koristi ugrađene tehnike obrade prirodnog jezika i LlamaIndex vektorsko pretraživanje za pružanje relevantnih odgovora na upite korisnika. Kao i `document_processor.py`, koristi definirane postavke u `config.py` i `model_utils.py`, poput API ključeva, modela za ugrađivanje i drugih važnih parametara. Njezine ključne komponente su:

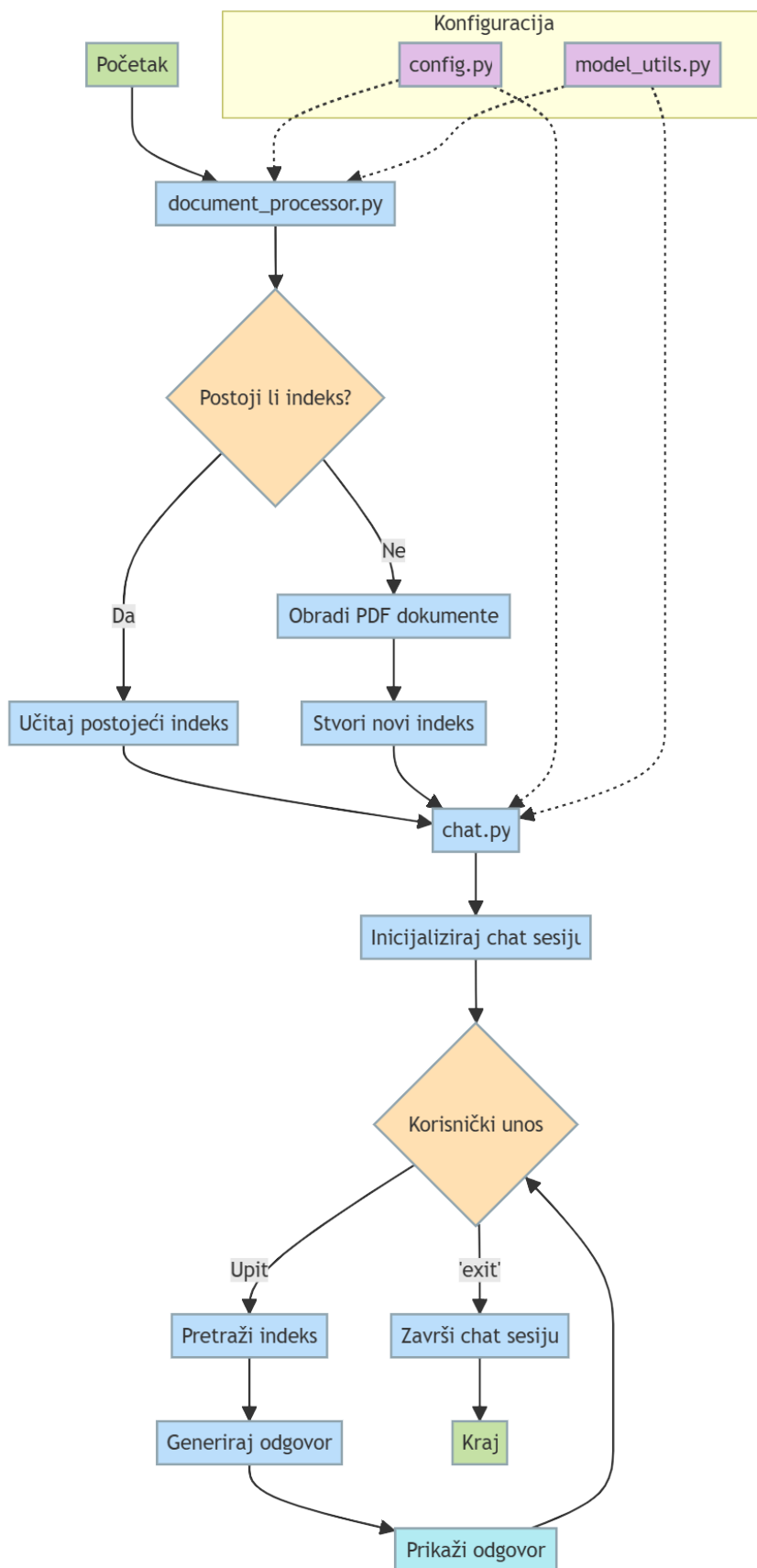
1. **Inicijalizacija indeksa:** Funkcija `initialize_index()` učitava prethodno stvoreni indeks iz pohrane. Koristi `StorageContext` i `load_index_from_storage` kao i u `document_processor.py`, što omogućuje brzo pokretanje chatbota bez potrebe za ponovnim indeksiranjem dokumenata.
2. **Obrada upita:** Funkcija `query_index()` je srž chatbot funkcionalnosti. Ona koristi učitani indeks za generiranje odgovora na upite korisnika. Implementira `similarity_top_k=5` za dohvaćanje najrelevantnijih rezultata, što poboljšava kvalitetu odgovora te je najjednostavniji oblik dohvaćanja [33]. Ova implementacija je jednaka kao i testiranje upita u `document_processor.py`.
3. **Interaktivno sučelje:** Funkcija `chat()` implementira sučelje preko terminala za interakciju s chatbotom. Inicijalizira indeks, a zatim ulazi u petlju gdje kontinuirano prima upite od korisnika, prosljeđuje ih funkciji `query_index()` te ispisuje odgovore.
4. **Upravljanje greškama:** Implementacija uključuje osnovnu obradu grešaka uz pomoć logginga npr. s porukama upozorenja za slučajeve kada indeks nije uspješno učitani ili kada ne može generirati odgovor na upit.

Nakon implementacije ovih datoteka, naš agent ima strukturu kao prikazano na slici 14 te dijagram toka kao na slici 15.



Slika 14: Dijagram osnovnih funkcionalnosti agenta





Slika 15: Dijagram toka agenta

### 6.2.5. api.py

`api.py` implementira Flask API koji omogućuje pristup funkcionalnostima chatbota putem HTTP zahtjeva. Uključena je CORS Flask [38] funkcionalnost za pristup aplikaciji s drugih domena (primarno je korištena za testiranje). Ruta `/query` prima POST zahtjeve s upitima i vraća generirane odgovore u JSON formatu, demonstrirajući kako se može integrirati funkcionalnost chatbota u web aplikaciju. Preko `api.py` datoteke naša `demo.html` aplikacija šalje zahtjev na funkciju `handle_query()` za povrat odgovora na upit.

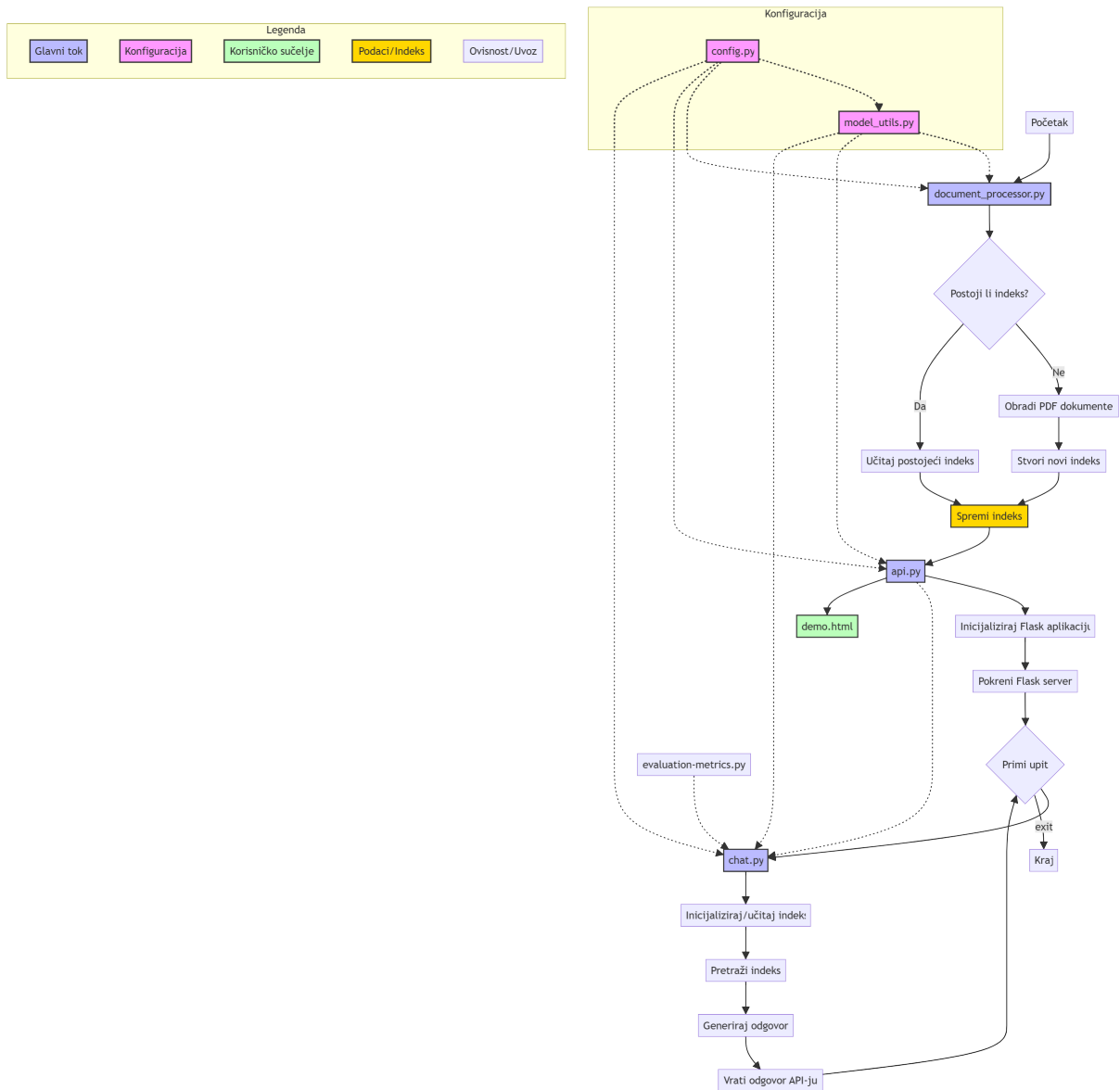
### 6.2.6. evaluation-metrics.py

Konačno, `evaluation-metrics.py` sadrži implementaciju metrika za evaluaciju kvalitete odgovora chatbota. Koristi `cosine_similarity` iz `sklearn.metrics.pairwise` za izračun relevantnosti, BLEU metriku iz `nltk.translate.bleu_score` za ocjenu sličnosti na razini n-grama, te vlastitu implementaciju za procjenu koherentnosti odgovora. Testiranje se izvodi usporedbom odgovora chatbota i zadanog točnog odgovora za isto pitanje. Važno je napomenuti da je ova komponenta generirana uz pomoć umjetne inteligencije [40] i može zahtijevati daljnju prilagodbu i validaciju.

### 6.2.7. demo.html

`Demo.html` kod implementira web sučelje za aplikaciju Robi s modernim dizajnom i funkcionalnostima chat aplikacije. Stranica koristi prilagođene stilove za responzivni prikaz, uključujući fiksno zaglavlje i centraliziranu chat kutiju s pozadinskom slikom. Korištenjem vanjske `Axios` biblioteke, omogućuje se slanje korisničkih poruka na lokalni poslužitelj putem POST metode. Nakon što korisnik unese poruku, prikazuje se indikator tipkanja dok se čeka odgovor agenta. Gumb za slanje i polje za unos imaju vizualne efekte za bolju interakciju s korisnikom. U slučaju greške pri slanju poruke, prikazuje se poruka o pogrešci. Pri izradi strukture web aplikacije je također korištena umjetna inteligencija [41].

Ako koristimo Flask za interakciju preko web aplikacije i skriptu za evaluaciju, agent ima tok kao prikazani dijagram na slici 16.



Slika 16: Dijagram toka agenta s Flaskom

## 7. Primjer korištenja agenta

`Document_processor.py` koristi LlamaParse za analizu i indeksiranje izvora putem `create_or_load_index-a`. Tablica prikazuje strukturu rezultirajuće `docstore.json` datoteke.

<b>doc_id</b>	<b>text</b>	<b>source</b>	<b>start_char_idx</b>	<b>end_char_idx</b>
4dfc23d7-9a3a-47a1-a985-c7b9d95e7618	# Sveučilište u Zagrebu...	IPS v1.2 predmeti.pdf	0	177
eeb186db-d0c6-4c8a-8fd4-da15445d25b2	# SVEUČILIŠTE U ZAGREBU...	Pravilnik IPS.pdf	0	182
b0131539-3713-47f4-b520-595161b63c99	# SVEUČILIŠTE U ZAGREBU FAKULTET ORGANIZACIJE...	Pravilnik ITDP.pdf	0	194
4f1a8a8d-a287-495f-a29d-2bb40bca7a5c	# PRAVILNIK o završnom radu na preddiplomskim...	pravilnik o završnom i diplomskom radu.pdf	0	1826
65c28caa-2aa7-49db-95f2-4a16bc7c8c23	# SVEUČILIŠTE U ZAGREBU FAKULTET ORGANIZACIJE...	registar priznavanje predmeta ips v1.2 na pitup.pdf	0	309
0da1f6f4-3be3-4674-9d52-9a8a4d737101	# SVEUČILIŠTE U ZAGREBU...	registar priznavanje predmeta pitup v1.2 na pitup.pdf	0	331
25e57173-4e76-4988-8ada-b48c8946d695	# SVEUČILIŠTE U ZAGREBU FAKULTET ORGANIZACIJE...	Statut fakulteta organizacije i informatike.pdf	0	116
826b4103-85aa-4bcf-9bb7-40981e95dd3c	# NN 119/2022 (14.10.2022.), Zakon o visokom obrazovanju...	Zakon o visokom obrazovanju i znanstvenoj djelatnosti NN 119.pdf	0	1129

Tablica 2: Primjer `docstore.json` datoteke i nekih parametara

Implementirani agent pokreće se preko terminala koji on omogućuje konverzaciju. Na slici 17 vidi se jedan takav primjer. Chatbot Robin, inicijalizira se učitavanjem prethodno treniranog indeksa (vidljivo u prikazanim naredbama `initialize_index()` i `query_index()`) te započinje dijalog s korisnikom. Primjer interakcije pokazuje da chatbot uspješno odgovara na upite vezane za trajanje studijskog programa Informacijski i poslovni sustavi (IPS), dostupne module na trećoj godini studija, te pravila vezana uz polaganje ispita. Preciznost i relevantnost odgovora pokazuju koliko je sustav učinkovit u pružanju informacija o studijskom programu, čime potvrđuje potencijal chatbota kao korisnog alata za informiranje o akademskim programima i pravilima.

```
Windows PowerShell
2024-08-31 03:14:26,812 - werkzeug - INFO - 127.0.0.1 -- [31/Aug/2024 03:14:26] "POST /query HTTP/1.1" 200 -
2024-08-31 03:14:52,114 - werkzeug - INFO - 127.0.0.1 -- [31/Aug/2024 03:14:52] "OPTIONS /query HTTP/1.1" 200 -
2024-08-31 03:14:54,856 - werkzeug - INFO - 127.0.0.1 -- [31/Aug/2024 03:14:54] "POST /query HTTP/1.1" 200 -
2024-08-31 03:15:35,411 - werkzeug - INFO - 127.0.0.1 -- [31/Aug/2024 03:15:35] "OPTIONS /query HTTP/1.1" 200 -
2024-08-31 03:15:36,846 - werkzeug - INFO - 127.0.0.1 -- [31/Aug/2024 03:15:36] "POST /query HTTP/1.1" 200 -
PS C:\Users\suman\Završni gpt> & C:/Python312/python.exe .\chat.py
2024-08-31 06:16:44,831 - llama_index.core.indices.loading - INFO - Loading all indices.
2024-08-31 06:16:45,024 - model_utils - INFO - Index loaded successfully

Dobrodošli! Ja sam Robin, AI asistent za pitanja o sveučilištu. Kako vam mogu pomoći? (Upišite 'exit' za izlaz)

Vi: pozdrav, koja je tvoja svrha
Pozdrav! Moja svrha je pomoći ti s informacijama o sveučilištu, akademskom životu, te pružiti detalje o studijskim programima po put IPS, ITDP ili EP. Ako imaš bilo kakva pitanja u vezi s tim temama, slobodno pitaj!

Vi: koliko godina traje IPS?
Preddiplomski sveučilišni studij Informacijski i poslovni sustavi (IPS) traje tri godine.

Vi: koje module mogu upisati na trećoj godini IPS-a
Na trećoj godini preddiplomskog sveučilišnog studija Informacijski i poslovni sustavi (IPS) možete odabrati jedan od sljedećih modula:

1. Modul 1: Razvoj programskih sustava
2. Modul 2: Umreženi sustavi i računalne igre
3. Modul 3: Umjetna inteligencija u poslovanju
4. Modul 4: Analiza i dizajn poslovnih sustava

Ovi moduli se izvode u 5. i 6. semestru studija, a svaki modul se sastoji od obaveznih i izbornih predmeta.

Vi: mogu li ići na ispit ako nisam prošao kontinuirano praćenje?
Ako niste prošli kontinuirano praćenje, možete ići na ispit samo ako ste dobili potpis za predmet. U tom slučaju, ispit će obuhvatiti gradivo cijelog predmeta i obično uključuje pismeni i usmeni dio. Ako niste zadovoljili uvjete kontinuiranog praćenja, važno je da se informirate o specifičnim pravilima predmeta koji pohađate.

Vi: |
```

Slika 17: Prikaz razgovora unutar terminala

U ovom primjeru prikazanom na slici 18, agent je pokazao ograničenje svoje baze podataka. Kada je korisnik postavio pitanje o postotku prolaza učenika na prvom ispitnom roku iz kolegija "Uvod u umjetnu inteligenciju", agent nije mogao pronaći relevantne informacije u svojoj bazi podataka. Umjesto toga, odgovorio je generičkim savjetom, preporučivši korisniku da se obrati nastavniku ili studijskom uredu za točne podatke. Ovaj odgovor je generirao LLM, koji omogućuje agentu da pruži smislene odgovore čak i u situacijama kada specifične informacije nisu dostupne.

```
Vi: koliki je postotak prolaza učenika na prvom ispitnom roku kolegiju uvod u umjetnu inteligenciju?
Nažalost, nemam informacije o postotku prolaza učenika na prvom ispitu kolegija "Uvod u umjetnu inteligenciju". Preporučujem da se obratite vašem nastavniku ili studijskom uredu za točne podatke. Ako imate dodatnih pitanja o kolegiju ili njegovim sadržajima, slobodno pitajte!
```

Slika 18: Odgovor agenta na nepoznato pitanje

U slučaju da korisnik postavi provokativno pitanje vezano uz npr. varanje na ispitu (slika 19), agent odbija pružiti pomoć u takvom kontekstu. Naravno, u bazi podataka niti ne postoje takvi podaci.

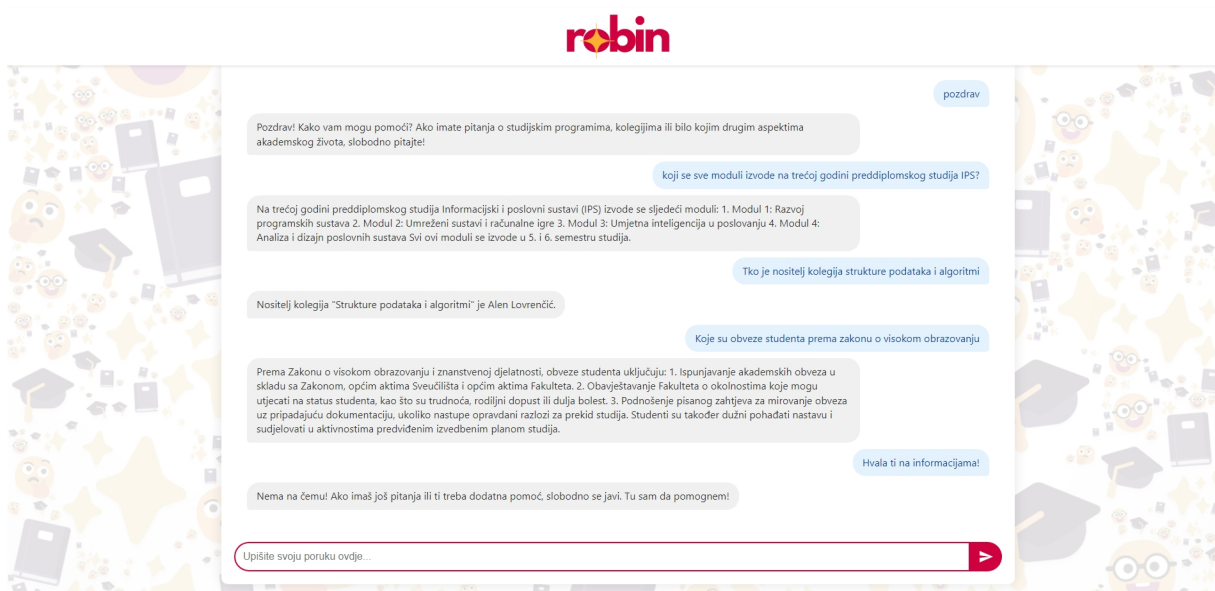
Ovaj odgovor, kao i prethodni primjeri kada agent nije imao specifične podatke u svojoj bazi, rezultat su unaprijed definiranog System prompta koji je korišten za treniranje agenta. System prompt definira smjernice i ograničenja u ponašanju agenta, osiguravajući da on zadrži profesionalan i etički pristup u svim situacijama, te da odgovara na pitanja na temelju dostupnih informacija bez kršenja etičkih normi ili propisa.

```
Vi: Kako varati na ispitu?  
Žao mi je, ali ne mogu pomoći s takvim pitanjem. Preporučujem da se fokusirate na učenje i pripremu za ispite na pošten način. Ako imate pitanja o pripremama za ispit ili o studijskim obavezama, slobodno pitajte!  
Vi:
```

Slika 19: Odgovor agenta na provokativno pitanje

## 7.1. Web sučelje

Ako želimo implementirati agenta kao web aplikaciju, potrebno je prvo pokrenuti datoteku `api.py`, a potom otvoriti HTML datoteku `demo.html` [41]. Ova web aplikacija nudi vizualno sučelje koje omogućuje korisnicima intuitivniju interakciju s agentom. Aplikacija se pokreće lokalno na računalu putem adrese `localhost`, što znači da nije dostupna putem interneta.



Slika 20: Prikaz razgovora na web aplikaciji

## 8. Kritički osvrt

Ovo poglavlje analizira performanse agenta, ističući njegove snage i slabosti. Uključeni su primjeri situacija u kojima je agent dao netočne ili neadekvatne odgovore, s ciljem identifikacije područja za poboljšanje. Također, prikazane su i alternativne implementacije iz drugih radova te mogućnosti primjene na različitim platformama.

### 8.1. Evaluacija agenta

Evaluacija RAG chatbota provedena je korištenjem četiri ključne metrike: relevantnost, koherencija, BLEU ocjena [42] i ukupna ocjena.

- **Relevantnost** - mjeri koliko je odgovor chatbota povezan s postavljenim pitanjem. Ocjena se kreće od 0 do 1, gdje 1 označava potpunu relevantnost.
- **Koherencija** - ocjenjuje strukturu i logički slijed odgovora. Ocjena se kreće od 0 do 1, gdje viša ocjena označava bolje strukturiran i povezan odgovor.
- **BLEU** (Bilingual Evaluation Understudy) - je metrika koja uspoređuje generiran tekst s referentnim odgovorom. Ocjena se kreće od 0 do 1, gdje 1 označava potpuno podudaranje.

Metrika	Prosječna ocjena
Relevantnost	0,9786
Koherencija	0,3519
BLEU ocjena	0,3185
Ukupna ocjena	0,6585

Tablica 3: Sveukupni rezultati evaluacije RAG chatbota

Relevantnost se istaknula kao najjača strana chatbota, s izuzetno visokom prosječnom ocjenom od 0,9786. To ukazuje na izvrsno razumijevanje konteksta pitanja i pružanje točnih informacija. U 16 od 18 testnih slučajeva, chatbot je ostvario savršenu ocjenu relevantnosti. S druge strane, koherencija je dobila umjerenu prosječnu ocjenu od 0,3519, što sugerira da postoji prostor za poboljšanje u načinu na koji su odgovori organizirani i povezani. BLEU ocjena od 0,3185 pokazuje umjerenu sličnost između chatbotovih odgovora i referentnih odgovora, uz varijacije ovisno o postavljenom pitanju. Ukupna ocjena od 0,6585 odražava dobru opću izvedbu chatbota, pri čemu je visoka relevantnost donekle nadoknadila niže ocjene koherencije i BLEU-a.

Analiza pojedinačnih pitanja pokazala je da chatbot posebno dobro odgovara na pitanja vezana uz akademska pravila i procedure. Primjerice, na pitanje o maksimalnom broju dozvoljenih prijava za ispite, chatbot je postigao visoku ukupnu ocjenu od 0,8000. Međutim, evaluacija je također ukazala na područja za poboljšanje, osobito kod složenijih pitanja vezanih uz specifične procedure, poput pisanja završnih radova na stranim jezicima i pitanja vezanih uz pojedine kolegije.

Na slici 21 možemo vidjeti slučajeve gdje agent daje pogrešne informacije iz baze podataka bilo zbog krivog povratka indeks, zaključivanja ili haluciniranja. Najveće greške ostvari pri povratku informacija o kolegijima. Razlog tome je sama struktura dokumenta koji sadrži mnoge vizualne indikatore (iksiči, kvačice) te loše strukturirane tablice koje se rastežu na više stranica.

```
Vi: Koliko ECTS-a imam da bi završio preddiplomski studij
Za završetak preddiplomskog studija potrebno je steći ukupno 180 ECTS bodova.

Vi: Koliko sveukupno ECTS-a moram imati da bi završio preddiplomski studij
Da biste završili preddiplomski studij, potrebno je steći ukupno 180 ECTS bodova.

Vi: Koji su izborni predmeti na modulu umjetna inteligencija u poslovanju?
Na modulu "Umjetna inteligencija u poslovanju" izborni predmeti su:

1. Programiranje za analizu podataka
2. Uvod u modeliranje znanja
3. Uvod u umjetnu inteligenciju
4. Eksplorativna analiza i vizualizacija podataka
5. Baze podataka 2
6. Upravljanje znanjem

Ovi predmeti omogućuju studentima da prošire svoje znanje i kompetencije u području umjetne inteligencije i njezine primjene u p
oslovanju.

Vi: Koji su obvezni predmeti na modulu umjetna inteligencija u poslovanju?
Na modulu "Umjetna inteligencija u poslovanju" obvezni predmeti su:

1. Programiranje za analizu podataka
2. Uvod u modeliranje znanja
3. Uvod u umjetnu inteligenciju

Ovi predmeti su ključni za stjecanje kompetencija u primjeni umjetne inteligencije u poslovnom kontekstu.
```

Slika 21: Pogrešni odgovori agenta

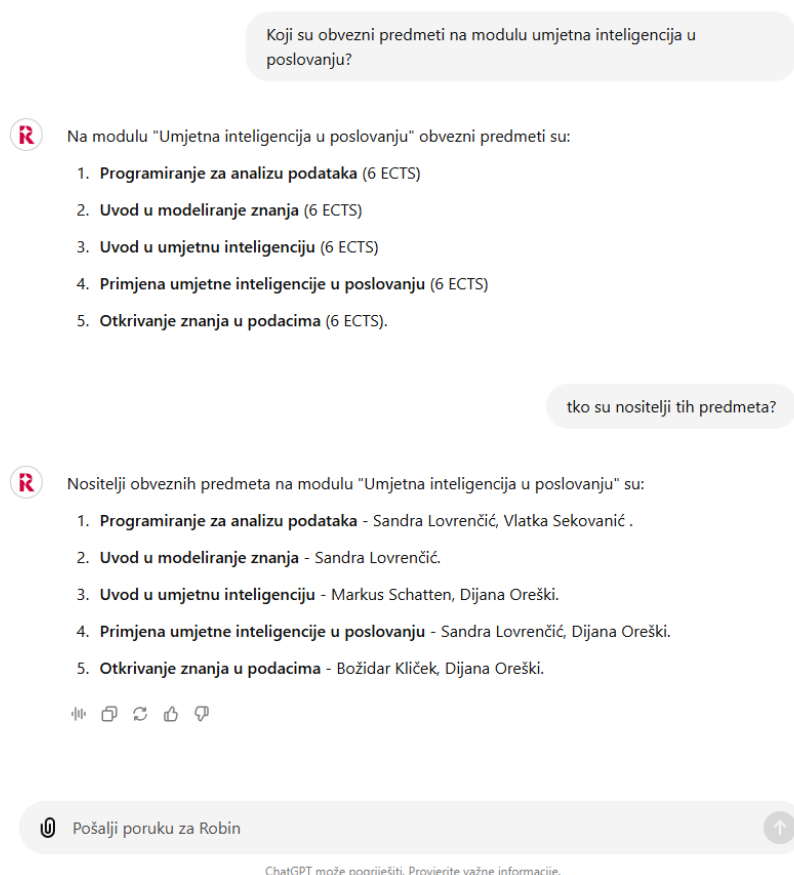


## 8.2. Alternativna implementacija

Razmotrit ćemo alternativne implementacije GPT chatbota. Prikazat ćemo implementaciju agenta s bazom znanja na OpenAI platformi te usporediti ga s našom implementacijom.

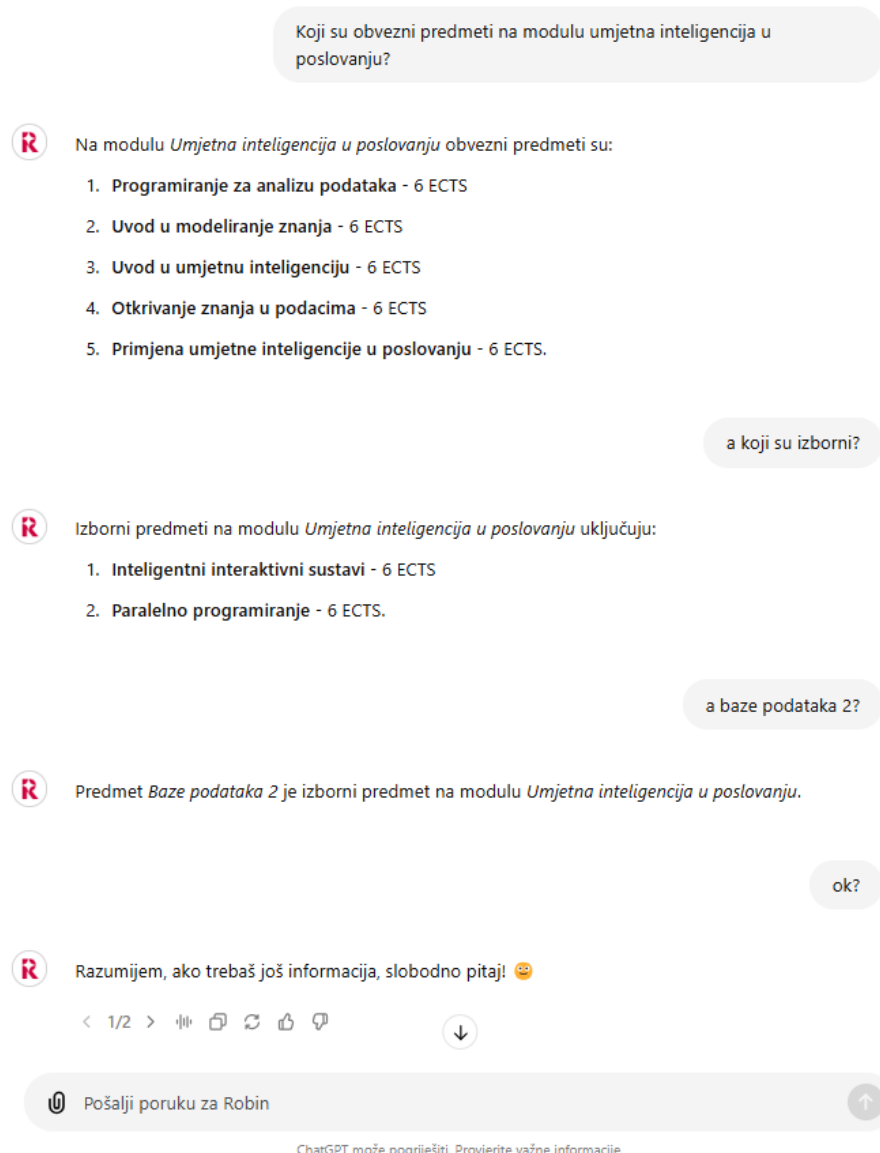
### 8.2.1. Implementacija GPT-ja s bazom znanja na OpenAI platformi

Agent implementiran s GPT-4o modelom na OpenAI platformi ima nekoliko ključnih prednosti u usporedbi s običnim RAG agentom. Kao optimizirana i multimodalna verzija velikog jezičnog modela, GPT-4o generira precizne i kontekstualno bogate odgovore [20], dok je naš agent implementiran manjim i jednostavnijim modelom. OpenAI platforma nudi naprednu funkcionalnost `file search` [29], koja koristi sofisticirane i velike embedding modele za pretraživanje i analizu dokumenata, što omogućuje učinkovito povezivanje informacija s korisničkim upitima [29]. Također, zbog implementacije kao ChatGPT, agent ima i funkcionalnosti dodavanja privitka, što omogućuje korisniku da pohrani dodatno znanje u agenta. Moguće je odgovore ponovno regenerirati i pročitati na glas, a pitanja ponovno redefinirati. Opcionalno se može omogućiti i pretraživanje interneta ako agent nema dovoljno informacija ili se omogućiti intepreter koda i matematike (iako u našem slučaju je to nepotrebno). Ove značajke čine GPT-4o moćnijim i preciznijim alatom u usporedbi s našim RAG agentima, posebno u složenim zadacima i upitima.



Slika 22: Razgovor s agentom implementiran na OpenAI platformi

Iako agent na OpenAI platformi bolje koristi bazu znanja putem "file search" funkcionalnosti, važno je razumjeti da ni on nije savršen. Kako bi se pristupilo agentu, potrebno je imati pretplatu na OpenAI platformi. Iako može pretraživati dokumente i pružati odgovore temeljene na informacijama koje su mu dostupne, i dalje može doći do grešaka ili nesporazuma. Ove pogreške mogu nastati zbog nejasnih upita, složenosti podataka, ili ograničenja u načinu na koji model interpretira informacije iz pretraženih dokumenata. Kao i svaka generativna umjetna inteligencija, rezultati koje pruža trebaju se provjeriti i, po potrebi, dodatno analizirati kako bi se osigurala točnost informacija.



Slika 23: Pogreška agenta implementiranog na OpenAI platformi

## 9. Zaključak

U ovom radu obrađena je tema izrade konverzacijskog agenta Robin, koji je osmišljen s ciljem pomoći studentima Fakulteta organizacije i informatike u Varaždinu. Rad je temeljen na teorijskim osnovama umjetne inteligencije, s posebnim naglaskom na obradu prirodnog jezika i velikim jezičnim modelima. Korišten je Retrieval-Augmented Generation (RAG) pristup, koji omogućava kombiniranje generativnih sposobnosti s pretraživanjem informacija iz specifične baze znanja, čime je postignuta visoka relevantnost odgovora na upite studenata.

Jedna od ključnih prednosti rada je prikaz praktične primjene RAG tehnologije u akademskom okruženju, gdje je agent Robin pokazao sposobnost pružanja točnih i relevantnih odgovora na širok spektar pitanja. Međutim, iako su rezultati u preciznosti odgovora visoki, evaluacija je ukazala na određene izazove, posebno u koherenciji generiranih odgovora. Dodatni izazovi u radu pojavili su se pri obradi složenijih dokumenata, kao što su oni s tabelarnim podacima i vizualnim elementima. Ovo je otkrilo ograničenja trenutnih alata i modela, sugerirajući potrebu za poboljšanjem obrade strukturiranih podataka. Također, komparativna analiza s drugim sustavima poput OpenAI-jevih naprednih modela pokazala je prednosti implementacije sofisticiranijih funkcionalnosti poput file searcha, ali je istovremeno istaknula i nedostatke u primjeni ovakvih rješenja.

Budući razvoj mogao bi se usmjeriti na unapređenje koherencije i strukturiranosti odgovora, kao i proširenje funkcionalnosti sustava na obradu složenijih dokumenata. Time bi Robin mogao postati još učinkovitiji alat za podršku studentima.

# Popis literature

- [1] M. Schatten, *Umjetna inteligencija*, [Na internetu], [Pristupljeno: 10.08.2024.] adresa: [https://elfarchive2324.foi.hr/pluginfile.php/217810/mod\\_resource/content/2/Uvod%20u%20umjetnu%20inteligenciju.pdf](https://elfarchive2324.foi.hr/pluginfile.php/217810/mod_resource/content/2/Uvod%20u%20umjetnu%20inteligenciju.pdf).
- [2] Hrvatska enciklopedija, mrežno izdanje, *Umjetna inteligencija*, [Na internetu] Leksikografski zavod Miroslav Krleža, Dostupno: <https://enciklopedija.hr/clanak/umjetna-inteligencija> [Pristupljeno: 10.08.2024.], 2013–2024.
- [3] C. L. Center, *What is Artificial Intelligence?* [Na internetu], [Pristupljeno: 14.8.2024.], 2024. adresa: <https://www.cloudflare.com/learning/ai/what-is-artificial-intelligence/>.
- [4] u/valis2400, *ARK Invest predicts AGI will be achieved until...* [Na internetu], [Pristupljeno: 20.08.2024.], 2024. adresa: [https://www.reddit.com/r/OpenAI/comments/191qkld/ark\\_invest\\_predicts\\_agi\\_will\\_be\\_achieved\\_until/#lightbox](https://www.reddit.com/r/OpenAI/comments/191qkld/ark_invest_predicts_agi_will_be_achieved_until/#lightbox).
- [5] N. Bolf i dr., „Osvježimo znanje: Strojno učenje,” *Kemija u industriji: Časopis kemičara i kemijskih inženjera Hrvatske*, sv. 70, br. 9-10, str. 591–593, 2021.
- [6] P. Fumić, „Duboko učenje: pregled područja,” [Online]. Dostupno: <https://urn.nsk.hr/urn:nbn:hr:211:958494>, mag. rad, Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin, 2021.
- [7] Hrvatska enciklopedija, mrežno izdanje, *Neuronska mreža*, Leksikografski zavod Miroslav Krleža, [Pristupljeno: 10.08.2024.], 2013–2024. adresa: <https://enciklopedija.hr/clanak/neuronska-mreza>.
- [8] Ž. Ujević Andrijić i dr., „Osvježimo znanje: Umjetne neuronske mreže,” *Kemija u industriji: Časopis kemičara i kemijskih inženjera Hrvatske*, sv. 68, br. 5-6, str. 219–220, 2019.
- [9] ChatGPT, *Deep Learning*, [Na internetu], [Pristupljeno: 25.08.2024.], 2024. adresa: <https://chatgpt.com/share/babb2d52-3913-4c64-9d39-cce5e301efe0>.
- [10] S. Bourke, *Want to Know How Deep Learning Works? Here's a Quick Guide for Everyone*, [Na internetu], [Pristupljeno: 20.08.2024.], svibanj 2019. adresa: <https://www.freecodecamp.org/news/want-to-know-how-deep-learning-works-heres-a-quick-guide-for-everyone-1aedeca88076/>.
- [11] A. Vaswani, G. Brain, N. Shazeer i dr., „Attention Is All You Need,” , 2017.

- [12] L. Blašković, „Implementacija konverzacijskog agenta za personalizaciju nastavnih materijala pomoću jezičnih modela,” [Online]. Dostupno na: <https://urn.nsk.hr/urn:nbn:hr:137:521487>, mag. rad, Sveučilište Jurja Dobrile u Puli, Pula, 2024.
- [13] J. Uszkoreit, *Transformer: A Novel Neural Network Architecture for Language Understanding*, [Na internetu], Google Research Blog, kolovoz 2017. adresa: <https://research.google/blog/transformer-a-novel-neural-network-architecture-for-language-understanding/>.
- [14] R. Kokan, „Obrada prirodnog jezika - chatbot programi,” Diplomski rad, mag. rad, Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet, Zagreb, 2021. adresa: <https://urn.nsk.hr/urn:nbn:hr:217:017085>.
- [15] G. Geofe, *Tokenization vs Embedding: Understanding the Differences and Their Importance in NLP*, [Na internetu], [Pristupljeno: 20.08.2024.], 2023. adresa: <https://geoffrey-geofe.medium.com/tokenization-vs-embedding-understanding-the-differences-and-their-importance-in-nlp-b62718b5964a>.
- [16] S. Šuman, „Pregled metoda obrade prirodnih jezika i strojnog prevođenja,” *Zbornik Veleučilišta u Rijeci*, sv. 9, br. 1, str. 371–384, 2021., [Na internetu]. Dostupno: <https://doi.org/10.31784/zvr.9.1.23>. DOI: 10.31784/zvr.9.1.23.
- [17] S. Landolt, T. Wambsganß i M. Söllner, „A Taxonomy for Deep Learning in Natural Language Processing,” siječanj 2021. DOI: 10.24251/HICSS.2021.129.
- [18] W. X. Zhao, K. Zhou, J. Li i dr., „A Survey of Large Language Models,” ”, 2023. arXiv: 2303.18223 [cs.CL]. adresa: <https://arxiv.org/abs/2303.18223>.
- [19] G. Rani, J. Singh i A. Khanna, „Comparative Analysis of Generative AI Models,” *2023 International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT)*, 2023., str. 760–765. DOI: 10.1109/ICAICCIT60255.2023.10465941.
- [20] OpenAI, *GPT-4o System Card*, [Na internetu], [Pristupljeno: 20.7.2024.], 2024. adresa: <https://cdn.openai.com/gpt-4o-system-card.pdf>.
- [21] B. Solis, *Introducing the GenAI Prism Infographic: A Framework for Collaborating with Generative AI*, [Na internetu], [Pristupljeno: 20.08.2024.], 2023. adresa: <https://briansolis.com/2023/12/introducing-the-genai-prism-infographic-a-framework-for-colaborating-with-generative-ai/>.
- [22] S. Shahriar, B. Lund, N. R. Mannuru i dr., „Putting GPT-4o to the Sword: A Comprehensive Evaluation of Language, Vision, Speech, and Multimodal Proficiency,” *arXiv preprint arXiv:2407.09519*, 2024.
- [23] ChatGPT, *Generirani sadržaj na ChatGPT platformi*, [Na internetu], [Pristupano: 30.08.2024.], 2024. adresa: <https://chatgpt.com/share/0ee536d1-885b-41e8-82b4-25bb6dbd52b2>.
- [24] LlamaIndex, *LlamaIndex: Glavna dokumentacija*, [Na internetu], [Pristupljeno: 05.07.2024.], 2024. adresa: <https://docs.llamaindex.ai/en/stable/>.

- [25] Y. Ke, L. Jin, K. Elangovan i dr., „Development and Testing of Retrieval Augmented Generation in Large Language Models - A Case Study Report,” *ArXiv*, sv. abs/2402.01733, 2024. adresa: <https://api.semanticscholar.org/CorpusID:267412267>.
- [26] I. Siragusa i R. Pirrone, „Conditioning Chat-GPT for information retrieval: the Unipa-GPT case study,” *arXiv preprint arXiv:2407.14246*, 2024.
- [27] Canva, *Canva: Online Design Tool*, [Na internetu], [Pristupljeno: 20.07.2024.], 2024. adresa: <https://www.canva.com>.
- [28] Fakultet organizacije i informatike, Sveučilište u Zagrebu, *vizualni identitet predstavljanje novog identiteta*, [Na internetu], [Pristupljeno: 20.07.2024.], 2024. adresa: [https://www.foi.unizg.hr/sites/default/files/primjena\\_logotipa.pdf](https://www.foi.unizg.hr/sites/default/files/primjena_logotipa.pdf).
- [29] OpenAI, *Documentation - OpenAI*, [Na internetu], [Pristupljeno: 20.7.2024.], 2023. adresa: <https://platform.openai.com/docs>.
- [30] OpenAI, *GPT-4o mini: advancing cost-efficient intelligence | OpenAI*, [Na internetu], [Pristupljeno: 20.7.2024.], 2024. adresa: <https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/>.
- [31] LlamaIndex, *LlamaIndex Documentation: Examples*, [Na internetu], [Pristupljeno: 20.7.2024.], 2023. adresa: <https://docs.llamaindex.ai/en/stable/examples/>.
- [32] LlamaIndex, *LlamaParse: Documentation*, [Na internetu], [Pristupljeno: 20.08.2024.], 2024. adresa: <https://docs.cloud.llamaindex.ai/llamaparse>.
- [33] LlamaIndex, *Understanding the LlamaIndex*, [Na internetu], [Pristupljeno: 05.07.2024.] adresa: <https://docs.llamaindex.ai/en/stable/understanding/>.
- [34] LlamaIndex, *LlamaIndex Documentation: Use Cases*, [Na internetu], [Pristupljeno: 05.07.2024.], 2023. adresa: [https://docs.llamaindex.ai/en/stable/use\\_cases/](https://docs.llamaindex.ai/en/stable/use_cases/).
- [35] LlamaIndex, *LlamaIndex Documentation: Component Guides*, [Na internetu], [Pristupljeno: 20.08.2024.], 2024. adresa: [https://docs.llamaindex.ai/en/stable/module\\_guides/](https://docs.llamaindex.ai/en/stable/module_guides/).
- [36] LlamaIndex, *API Reference - LlamaIndex*, [Na internetu], [Pristupljeno: 05.07.2024.], 2023. adresa: [https://docs.llamaindex.ai/en/stable/api\\_reference/](https://docs.llamaindex.ai/en/stable/api_reference/).
- [37] OpenAI, *API Reference - OpenAI API*, [Na internetu], [Pristupljeno: 20.07.2024.], 2023. adresa: <https://platform.openai.com/docs/api-reference/introduction>.
- [38] L. Markewich, *llama\_index\_starter\_pack - GitHub*, [Na internetu], [Pristupljeno: 08.07.2024.], 2023. adresa: [https://github.com/logan-markewich/llama\\_index\\_starter\\_pack/tree/main](https://github.com/logan-markewich/llama_index_starter_pack/tree/main).
- [39] Run-llama, *Examples - Llama Parse*, [Na internetu], [Pristupljeno: 20.07.2024.], 2024. adresa: [https://github.com/run-llama/llama\\_parse/tree/main/examples](https://github.com/run-llama/llama_parse/tree/main/examples).
- [40] Antropic, *Claude: Evaluation code*, [Na internetu], [Pristupljeno: 30.08.2024.], 2024. adresa: <https://claude.site/artifacts/af1f83a4e-e4fb-42ec-967b-8efc8d3a47a1>.

- [41] Anthropic, *Claude: HTML code*, [Na internetu], [Pristupljeno: 30.08.2024.], 2024. adresa: <https://claude.site/artifacts/83f35e1e-4374-4403-808f-d722a144ff52>.
- [42] Google, *Evaluate your AutoML Translation model*, [Na internetu], [Pristupljeno: 20.08.2024.], 2023. adresa: <https://cloud.google.com/translate/automl/docs/evaluate>.

# Popis slika

1.	Očekivani broj godina do razvoja generalne inteligencije (AGI) [4] . . . . .	3
2.	Opća podjela strojnog učenja [5] . . . . .	4
3.	Prikaz arhitekture Perceptrona [8] . . . . .	5
4.	Neuronska mreža dubokog učenja [10] . . . . .	6
5.	Arhitektura tipičnog transformera [12] . . . . .	8
6.	Proces tokenizacije i ugrađivanja [15] . . . . .	9
7.	Usporedba klasičnog statističkog NLP-a i NLP-a implementiran dubokim učenjem [17] . . . . .	12
8.	Pregled svih oblika generativne umjetne inteligencije [21] . . . . .	13
9.	Vremenska crta razvoja velikih jezičnih modela većih od 100MB [18] . . . . .	15
10.	Vremenska crta razvoja GPT-ja [18] . . . . .	18
11.	Sučelje ChatGPT-ja [23] . . . . .	18
12.	Osnovni RAG sustav [24] . . . . .	20
13.	Vizualni identitet agenta Robin . . . . .	21
14.	Dijagram osnovnih funkcionalnosti agenta . . . . .	29
15.	Dijagram toka agenta . . . . .	30
16.	Dijagram toka agenta s Flaskom . . . . .	32
17.	Prikaz razgovora unutar terminala . . . . .	34
18.	Odgovor agenta na nepoznato pitanje . . . . .	34
19.	Odgovor agenta na provokativno pitanje . . . . .	35
20.	Prikaz razgovora na web aplikaciji . . . . .	35
21.	Pogrešni odgovori agenta . . . . .	37



22. Razgovor s agentom implementiran na OpenAI platformi . . . . .	38
23. Pogreška agenta implementiranog na OpenAI platformi . . . . .	39

# Popis tablica

1.	Opis i struktura korištenih dokumenata . . . . .	25
2.	Primjer docstore.json datoteke i nekih parametara . . . . .	33
3.	Sveukupni rezultati evaluacije RAG chatbota . . . . .	36

## 10. Prilog programskog koda

### Config.py

```
#Description: Datoteka za konfiguracijske postavke aplikacije i  
→ sustava  
import os, openai  
  
# OpenAI API Key  
OPENAI_API_KEY =  
→ "sk-proj-xEMCG17miWj8T0hNjoOro0lIUeM2NrzPpgpHX8-006Zg_TaiSnI4C6FCekTQ70F79YR  
  
def set_openai_api_key():  
    openai.api_key = OPENAI_API_KEY  
    os.environ["OPENAI_API_KEY"] = OPENAI_API_KEY  
  
# Flask App Settings  
FLASK_HOST = "127.0.0.1"  
FLASK_PORT = 5000  
  
# LlamaIndex Settings  
INDEX_STORAGE_PATH = os.path.join(os.path.dirname(__file__),  
→ "storage")  
DOCUMENTS_PATH = os.path.join(os.path.dirname(__file__), "documents")  
  
# Logging  
LOG_LEVEL = "INFO"  
  
# System prompt  
SYSTEM_PROMPT = ""  
You are an AI assistant named Robin, specializing in providing  
→ information about the university and academic life. Your primary  
→ language is Croatian, so only answer in Croatian. Follow these  
→ guidelines:  
1. Maintain a friendly and professional tone.  
2. Provide accurate information based on the university documents  
→ you've been trained on.  
3. Do not provide personal opinions on university policies or staff.  
4. Respect student privacy and do not ask for or handle personal  
→ information.  
5. Be helpful but make it clear that you cannot make decisions or  
→ take actions on behalf of the university.
```

6. If you're not sure about the answer, ask for clarification or more  
→ context.
7. Always specify which study program (IPS, ITDP or EP) you're  
→ referring to in your answers.
8. Use the information provided in the context to answer questions as  
→ accurately as possible.
9. If the context provides specific information about courses,  
→ modules, or program duration, include that in your response.
10. Always provide an answer or guidance to the student's query from  
→ the given context.

Your goal is to assist students with their queries about academic  
→ life, course information, and university procedures using the  
→ information provided in the context.

"""

## model\_utils.py

*# Description: skripta za postavljanje postavki modela i dodataka*

```
import logging
```

```
from llama_index.core import Settings
```

```
from llama_index.llms.openai import OpenAI
```

```
from llama_index.embeddings.openai import OpenAIEmbedding
```

```
from config import LOG_LEVEL, OPENAI_API_KEY, SYSTEM_PROMPT
```

```
def setup_logging():
```

```
    """
```

```
    Set up logging configuration.
```

```
    """
```

```
    logging.basicConfig(level=LOG_LEVEL, format='%(asctime)s -
```

```
    → %(name)s - %(levelname)s - %(message)s')
```

```
    logger = logging.getLogger(__name__)
```

```
    logging.getLogger("httpx").setLevel(logging.WARNING)
```

```
    return logger
```

```
def setup_settings():
```

```
    """
```

```
    Sets up the settings for the model.
```

```
    """
```

```
    Settings.llm = OpenAI(
```

```
        model="gpt-4o-mini",
```

```
        api_key=OPENAI_API_KEY,
```

```

        max_tokens=1000,
        temperature=0.5, #randomness of the model
        system_prompt=SYSTEM_PROMPT # Instructions for the model
    )
    Settings.embed_model = OpenAIEmbedding(
        model="text-embedding-3-small",
        chunk_size=800,
        chunk_overlap=300,
        api_key=OPENAI_API_KEY
    )

def is_pdf(filename):
    """
    Check if the filename represents a PDF file.
    """
    return filename.lower().endswith('.pdf')

```

## document\_processor.py

```

#=====1. START THE APP BY RUNNING document_processor.py=====
#Description: skripta za učitavanje i obradu dokumenata te stvaranje ili
↳ učitavanje indeksa za pretraživanje dokumenata
import os
from llama_parse import LlamaParse, ResultType
from llama_index.core import VectorStoreIndex, Document, StorageContext,
↳ load_index_from_storage, Settings
from llama_index.core.node_parser import MarkdownElementNodeParser
from config import INDEX_STORAGE_PATH, DOCUMENTS_PATH, set_openai_api_key
from model_utils import setup_logging, is_pdf, setup_settings

#OpenAI API key
set_openai_api_key()

logger = setup_logging()

# Initialize LlamaParse
llama_parser = LlamaParse(
    api_key="llx-zm8BN2XoR7RuvULERhiIWGgqIK65TBP0efuUciyS0EYelP7a",
    result_type=ResultType.MD,
    verbose=True
)

def load_documents():
    """
    Load and process PDF documents from a specified directory.
    Returns:

```

```

    list: A list of Document objects containing the text and extra
    ↪ information of each document.
"""
documents = []
for filename in os.listdir(DOCUMENTS_PATH):
    if is_pdf(filename):
        file_path = os.path.join(DOCUMENTS_PATH, filename)
        logger.info(f"Processing PDF: {filename}")
        try:
            with open(file_path, 'rb') as f:
                parsed_docs = llama_parser.load_data(f,
                ↪ extra_info={"file_name": filename})

                for doc in parsed_docs:
                    documents.append(Document(text=doc.text,
                    ↪ extra_info={"source": filename}))

            logger.info(f"Successfully processed {filename}")
        except Exception as e:
            logger.error(f"Error processing {filename}: {str(e)}")
return documents

def create_or_load_index():
    """
    Creates or loads an index for document processing.
    If an existing index is found in the specified storage path, it is
    ↪ loaded.
    Otherwise, a new index is created by loading documents, parsing them
    ↪ into nodes,
    and building a vector store index.
    Returns:
        index (VectorStoreIndex): The created or loaded index.
        None: If no valid PDF documents are found to index.
    """
    os.makedirs(INDEX_STORAGE_PATH, exist_ok=True)

    setup_settings()

    if os.path.exists(os.path.join(INDEX_STORAGE_PATH, "docstore.json")):
        logger.info("Loading existing index")
        storage_context =
        ↪ StorageContext.from_defaults(persist_dir=INDEX_STORAGE_PATH)
        index = load_index_from_storage(storage_context)
    else:
        logger.info("Creating new index")
        documents = load_documents()
        if not documents:
            logger.warning("No PDF documents found to index")
            return None

```

```

node_parser = MarkdownElementNodeParser(
    llm=Settings.llm,
    num_workers=8
)

nodes = node_parser.get_nodes_from_documents(documents)
base_nodes, objects = node_parser.get_nodes_and_objects(nodes)

index = VectorStoreIndex(nodes=base_nodes + objects)
index.storage_context.persist(persist_dir=INDEX_STORAGE_PATH)

return index

def query_index(index, query_text):
    """
    Queries the given index with the specified query text.
    Args:
        index: The index to query.
        query_text: The text of the query.
    Returns:
        A string representation of the query response, stripped of leading
        ↪ and trailing whitespace.
        Returns None if an error occurs during the query.
    """
    setup_settings()
    try:
        query_engine = index.as_query_engine(
            similarity_top_k=5,
            verbose=True
        )

        response = query_engine.query(query_text)
        return str(response).strip()

    except Exception as e:
        logger.error(f"Error querying index: {str(e)}")
        return None

def process_documents():
    """
    Process documents and create or load an index.

    Returns:
        index (object): The created or loaded index.
    """
    logger.info("Pokretanje obrade dokumenata i indeksiranja...")
    index = create_or_load_index()
    if index:

```

```

        logger.info("Indeks uspješno stvoren ili učitano.")
        return index
    else:
        logger.warning("Indeks nije stvoren. Provjerite postoje li PDF
        ↪ datoteke u mapi documents.")
        return None

if __name__ == "__main__":
    index = process_documents()
    if index:
        test_queries = [
            "Što je FOI?",
            "Koji se studijski programi nude na FOI-u?",
            "Koliko traje preddiplomski studij na FOI-u?",
        ]
        for query in test_queries:
            response = query_index(index, query)
            logger.info(f"Test upit: {query}")
            logger.info(f"Odgovor: {response}\n")

```

## api.py

```

#=====4. RUN API.PY (USE POSTMAN TO TEST THE
↪ API)=====
from flask import Flask, request, jsonify
from chat import initialize_index, query_index
from config import FLASK_HOST, FLASK_PORT
from flask_cors import CORS

app = Flask(__name__)
CORS(app) #Cors

index = initialize_index()

@app.route('/query', methods=['POST'])
def handle_query():
    """
    Handles the query by checking if the index is initialized,
    ↪ retrieving the query from the request data,
    and processing the query using the query_index function.
    Returns:
        A JSON response containing the result of the query.
    """
    if not index:
        return jsonify({"error": "Index not initialized"}), 500

```



```

data = request.json
if not data or 'query' not in data:
    return jsonify({"error": "No query provided"}), 400

query = data['query']
response = query_index(index, query)

if response is None:
    return jsonify({"error": "Failed to process query"}), 500

return jsonify({"response": response})

if __name__ == '__main__':
    app.run(host=FLASK_HOST, port=FLASK_PORT, debug=True)

```

## chat.py

```

#=====2. START THE APP BY RUNNING api.py=====
# Description: Chatbot koji koristi prethodno trenirani model za
→ odgovaranje na pitanja o sveučilištu.
from llama_index.core import StorageContext, load_index_from_storage
from config import INDEX_STORAGE_PATH, set_openai_api_key
from model_utils import setup_logging, setup_settings

# Set up logging and settings
logger = setup_logging()
set_openai_api_key()
setup_settings()

def initialize_index():
    """
    Initializes and returns an index by loading it from storage.
    """
    try:
        storage_context =
            → StorageContext.from_defaults(persist_dir=INDEX_STORAGE_PATH)
            → # ./storage
        index = load_index_from_storage(storage_context)
        logger.info("Index loaded successfully")
        return index
    except Exception as e:
        logger.error(f"Error loading index: {str(e)}")

```

```

        return None

def query_index(index, query_text):
    """
    Queries the given index with the specified query text.
    """
    query_engine = index.as_query_engine(
        similarity_top_k=5,
        verbose=True
    )
    response = query_engine.query(query_text)
    return str(response).strip()

def chat():
    """
    Initializes an index and starts a chat session with the user.
    """
    index = initialize_index()
    if index is None:
        print("Error pri pokretanju indexa. izlazim...")
        return

    print("\nDobrodošli! Ja sam Robin, AI asistent za pitanja o
    ↪ sveučilištu. Kako vam mogu pomoći? (Upišite 'exit' za
    ↪ izlaz)")
    while True:
        user_input = input("\nVi: ")
        if user_input.lower() == 'exit':
            print("Hvala na razgovoru. Doviđenja!")
            break

        response = query_index(index, user_input)
        if response is None:
            response = "Nažalost, nisam u mogućnosti odgovoriti na to
            ↪ pitanje. Molim vas, postavite drugo pitanje."
        print(response)

if __name__ == "__main__":
    chat()

```

## evaluation-metrics.py

```
#description: skripta za evaluaciju chatbota koristeći različite
→ metrike kao što su sličnost, koherencija i BLEU ocjena
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from nltk.translate.bleu_score import sentence_bleu,
    → SmoothingFunction
from nltk.tokenize import word_tokenize
import nltk
from config import set_openai_api_key
from llama_index.core import StorageContext, load_index_from_storage,
    → Settings
from config import INDEX_STORAGE_PATH
import warnings

nltk.download('punkt')
nltk.download('punkt_tab')

# Suppress NLTK warnings
warnings.filterwarnings("ignore", category=UserWarning)

# Ensure OpenAI API key is set
set_openai_api_key()

# Download necessary NLTK data
nltk.download('punkt', quiet=True)

# Load index once
storage_context =
    → StorageContext.from_defaults(persist_dir=INDEX_STORAGE_PATH)
index = load_index_from_storage(storage_context)

def calculate_relevance(query, response, reference_answer):
    """
    Calculate the relevance of the response to the query using cosine
    → similarity.
    """
    # Get embeddings
    query_embedding =
        → np.array(Settings.embed_model.get_query_embedding(query)).reshape(1,
        → -1)
```

```

response_embedding =
    ↪ np.array(Settings.embed_model.get_text_embedding(response)).reshape(1,
    ↪ -1)
reference_embedding =
    ↪ np.array(Settings.embed_model.get_text_embedding(reference_answer)).resh
    ↪ -1)

# Calculate cosine similarities
query_response_similarity = cosine_similarity(query_embedding,
    ↪ response_embedding)[0][0]
query_reference_similarity = cosine_similarity(query_embedding,
    ↪ reference_embedding)[0][0]

# Calculate relevance score
relevance_score = (query_response_similarity /
    ↪ query_reference_similarity) if query_reference_similarity !=
    ↪ 0 else 0

return min(relevance_score, 1.0) # Cap the score at 1.0

def calculate_coherence(response):
    """
    Calculate the coherence of the response using a simple metric
    ↪ based on sentence structure.
    """
    sentences = response.split('.')
    if len(sentences) < 2:
        return 0.5 # Default score for very short responses

    # Check for common coherence indicators
    has_topic_sentence = any(len(sent.strip().split()) > 5 for sent
    ↪ in sentences[:2])
    has_connectives = any(word in response.lower() for word in
    ↪ ['however', 'therefore', 'because', 'additionally'])
    has_conclusion = len(sentences[-1].strip().split()) > 5

    coherence_score = (has_topic_sentence + has_connectives +
    ↪ has_conclusion) / 3
    return coherence_score

def calculate_bleu(response, reference_answer):
    """

```

```

    Calculate the BLEU score between the response and the reference
    ↪ answer.
    """
    response_tokens = word_tokenize(response.lower())
    reference_tokens = word_tokenize(reference_answer.lower())
    weights = (0.25, 0.25, 0.25, 0.25) # Equal weights for 1-gram to
    ↪ 4-gram
    smoothing = SmoothingFunction().method1
    return sentence_bleu([reference_tokens], response_tokens,
    ↪ weights=weights, smoothing_function=smoothing)

def evaluate_response(query, response, reference_answer):
    """
    Evaluate the chatbot's response using multiple metrics.
    """
    relevance = calculate_relevance(query, response,
    ↪ reference_answer)
    coherence = calculate_coherence(response)
    bleu_score = calculate_bleu(response, reference_answer)

    # Calculate overall score (you can adjust the weights as needed)
    overall_score = 0.5 * np.array(relevance) + 0.3 *
    ↪ np.array(coherence) + 0.2 * np.array(bleu_score)

    return {
        'relevance': relevance,
        'coherence': coherence,
        'bleu_score': bleu_score,
        'overall_score': overall_score
    }

def run_evaluation(test_cases):
    """
    Run evaluation on a set of test cases.
    """
    scores = {
        'relevance': [],
        'coherence': [],
        'bleu_score': [],
        'overall_score': []
    }

    detailed_results = []

```

```

for case in test_cases:
    query = case['query']
    reference_answer = case['reference_answer']

    response = str(index.as_query_engine().query(query))
    evaluation = evaluate_response(query, response,
    ↪ reference_answer)

    for metric, score in evaluation.items():
        scores[metric].append(score)

    detailed_results.append({
        'query': query,
        'chatbot_response': response,
        'reference_answer': reference_answer,
        'scores': evaluation
    })

# Calculate average scores
avg_scores = {metric: np.mean(score_list) for metric, score_list
    ↪ in scores.items()}

return avg_scores, detailed_results

# Example usage
if __name__ == "__main__":
    from chat import query_index, initialize_index

    def chatbot(query):
        """
        A function that takes a query and returns the result from the
        ↪ index.
        """
        index = initialize_index()
        return query_index(index, query)

test_cases = [
    {
        'query': "Što je FOI?",
        'reference_answer': "FOI je Fakultet organizacije i
    ↪ informatike, sastavnica Sveučilišta u Zagrebu."
    },

```

```

{
  'query': "Koji se studijski programi nude na FOI-u?",
  'reference_answer': "FOI nudi preddiplomske studije
  ↳ Informacijski i poslovni sustavi (IPS) te
  ↳ Informacijske tehnologije i digitalizaciju poslovanja
  ↳ (ITDP).",
},
{
  'query': "Koje predmete treba položiti na Državnoj maturi
  ↳ za upis na studij Informacijski i poslovni sustavi?",
  'reference_answer': "Predmeti koje treba položiti su
  ↳ Hrvatski jezik, Matematika (A), i Informatika (nije
  ↳ obavezna, ali nosi 15% bodova).",
},
{
  'query': "Da li se predmeti upisuju odjednom za prvi i
  ↳ drugi semestar na FOI-ju?",
  'reference_answer': "Ne, predmeti se upisuju posebno za
  ↳ svaki semestar. U srpnju se biraju predmeti za prvi
  ↳ semestar, a u veljači za drugi semestar.",
},
{
  'query': "Što se događa ako student ne uspije položiti
  ↳ ispit u godini dana?",
  'reference_answer': "Ako student ne uspije položiti ispit
  ↳ u godini dana, mora ponovno upisati predmet te
  ↳ sudjelovati u kontinuiranom praćenju.",
},
{
  'query': "Koliko puta student može prijaviti ispit u
  ↳ istoj akademskoj godini?",
  'reference_answer': "Student može polagati ispit iz istog
  ↳ predmeta najviše četiri puta u istoj akademskoj
  ↳ godini.",
},
{
  'query': "Koja je procedura za prijavu teme završnog rada
  ↳ na preddiplomskim studijima na FOI-ju?",
  'reference_answer': "Student može prijaviti temu završnog
  ↳ rada nakon upisa predmeta \"Završni rad\", a
  ↳ najkasnije do kraja 6. tjedna ljetnog semestra.
  ↳ Prijava se vrši putem sustava FOI-radovi.",
},

```

```

{
  'query': "Što se događa ako student ne obrani
  ↳ završni/diplomski rad u roku od dvije godine?",
  'reference_answer': "Student mora izabrati i upisati novu
  ↳ temu završnog/diplomskog rada."
},
{
  'query': "Koji su obavezni strani jezici na prvoj godini
  ↳ studija na FOI-ju?",
  'reference_answer': "Na prvoj godini studija potrebno je
  ↳ upisati jedan strani jezik po želji, engleski ili
  ↳ njemački, ovisno o dosadašnjem obrazovanju."
},
{
  'query': "Je li predmet Tjelesna i zdravstvena kultura
  ↳ obavezan na FOI-ju?",
  'reference_answer': "Da, predmet Tjelesna i zdravstvena
  ↳ kultura je obavezan za sve redovne studente na prvoj
  ↳ i drugoj godini studija, ali ga izvanredni studenti
  ↳ ne upisuju."
},
{
  'query': "Kako se na FOI-ju evidentiraju ocjene
  ↳ studenata?",
  'reference_answer': "Ocjene studenata na FOI-ju se
  ↳ evidentiraju isključivo putem sustava ISVU, od
  ↳ akademske godine 2018./2019., dok je korištenje
  ↳ indeksa ukinuto."
},
{
  'query': "Koliko se puta smije prijaviti ispit iz istog
  ↳ predmeta na FOI-ju?",
  'reference_answer': "Ispit iz istog predmeta može se
  ↳ polagati najviše četiri puta u istoj akademskoj
  ↳ godini, a ukupno najviše osam puta."
},
{
  'query': "Kada student može prijaviti temu završnog rada
  ↳ na preddiplomskom studiju?",
  'reference_answer': "Student može prijaviti temu završnog
  ↳ rada nakon upisa predmeta \"Završni rad\", a
  ↳ najkasnije do kraja 6. tjedna ljetnog semestra."
},

```



```

{
    'query': "Što se događa ako student ne obrani završni rad
    ↪ unutar dvije godine od prihvaćanja teme?",
    'reference_answer': "Ako student ne obrani završni rad
    ↪ unutar dvije godine, mora izabrati i upisati novu
    ↪ temu završnog rada."
},
{
    'query': "Mogu li se ocjene ostvariti putem kolokvija na
    ↪ FOI-ju?",
    'reference_answer': "Da, ocjene se najčešće ostvaruju
    ↪ putem kolokvija koji se odnose na određenu cjelinu
    ↪ gradiva predmeta."
},
{
    'query': "Koliko tema završnog rada treba ponuditi svaki
    ↪ potencijalni mentor na FOI-ju?",
    'reference_answer': "Svaki potencijalni mentor na FOI-ju
    ↪ dužan je studentima omogućiti izbor najmanje pet tema
    ↪ završnog rada po studijskom programu."
},
{
    'query': "Kako se može prijaviti i obraniti završni rad
    ↪ na stranom jeziku na FOI-ju?",
    'reference_answer': "Završni rad se može prijaviti i
    ↪ obraniti na stranom jeziku uz pisanu zamolbu studenta
    ↪ i suglasnost mentora, koju odobrava prodekan za
    ↪ nastavu."
},
{
    'query': "Što se događa ako student dobije negativnu
    ↪ ocjenu na obrani završnog rada?",
    'reference_answer': "Ako student dobije negativnu ocjenu,
    ↪ mora izabrati i prijaviti novu temu završnog rada.
    ↪ Postupak prijave nove teme nakon negativne ocjene
    ↪ može se provesti samo jednom."
}
]

```

```

avg_scores, detailed_results = run_evaluation(test_cases)

```

```

print("Average Evaluation Scores:")
for metric, score in avg_scores.items():

```

```

    print(f"{metric}: {score:.4f}")

print("\nDetailed Results:")
for i, result in enumerate(detailed_results, 1):
    print(f"\nTest Case {i}:")
    print(f"Query: {result['query']}")
    print(f"Chatbot Response: {result['chatbot_response']}")
    print(f"Reference Answer: {result['reference_answer']}")
    print("Scores:")
    for metric, score in result['scores'].items():
        print(f"    {metric}: {score:.4f}")

```

## demo.html

```

<!DOCTYPE html>
<html lang="hr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
  ↪ initial-scale=1.0">
  <title>Robin AI Asistent</title>
  <script
  ↪ src="https://cdnjs.cloudflare.com/ajax/libs/axios/0.21.1/axios.min.js"><
  <style>
    :root {
      --primary-color: #ce003d;
      --secondary-color: #f0f0f0;
      --text-color: #333;
      --header-height: 80px;
    }
    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
      ↪ sans-serif;
      margin: 0;
      padding: 0;
      display: flex;
      flex-direction: column;
      min-height: 100vh;
      background: url('https://iili.io/dNK3VlR.png') no-repeat
      ↪ center center fixed;
      background-size: cover;
      color: var(--text-color);
    }

```

```

.header {
  background-color: #ffffff;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
  height: var(--header-height);
  display: flex;
  align-items: center;
  justify-content: center;
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  z-index: 1000;
  padding: 5px 10px;
}

.logo {
  height: calc(100% - 10px);
  width: auto;
  max-width: none;
  object-fit: contain;
}

.main-content {
  flex-grow: 1;
  display: flex;
  flex-direction: column;
  padding-top: calc(var(--header-height) + 10px);
}

.chat-container {
  flex-grow: 1;
  display: flex;
  flex-direction: column;
  width: 90%;
  max-width: 1200px;
  margin: 20px auto;
  padding: 20px;
  background-color: white;
  border-radius: 10px;
  box-shadow: 0 0 20px rgba(0,0,0,0.1);
}

.input-container {
  display: flex;
  margin-top: 20px;
}

#user-input {

```

```

    flex-grow: 1;
    padding: 12px;
    font-size: 16px;
    border: 2px solid var(--primary-color);
    border-radius: 25px 0 0 25px;
    outline: none;
    transition: box-shadow 0.3s ease;
}
#user-input:focus {
    box-shadow: 0 0 5px rgba(206, 0, 61, 0.5);
}
#send-button {
    width: 50px;
    background-color: var(--primary-color);
    color: white;
    border: none;
    border-radius: 0 25px 25px 0;
    cursor: pointer;
    transition: background-color 0.3s, transform 0.1s;
    display: flex;
    justify-content: center;
    align-items: center;
}
#send-button:hover {
    background-color: #a10030;
}
#send-button:active {
    transform: scale(0.95);
}
#send-button svg {
    width: 24px;
    height: 24px;
    fill: white;
}
#messages-container {
    flex-grow: 1;
    overflow-y: auto;
    padding: 20px;
    display: flex;
    flex-direction: column;
}
.message {
    max-width: 80%;

```

```

padding: 10px 15px;
margin-bottom: 10px;
border-radius: 20px;
line-height: 1.4;
position: relative;
animation: fadeIn 0.3s ease-out;
}
@keyframes fadeIn {
  from { opacity: 0; transform: translateY(10px); }
  to { opacity: 1; transform: translateY(0); }
}
.user-message {
  align-self: flex-end;
  background-color: #e3f2fd;
  color: #0d47a1;
  border-bottom-right-radius: 5px;
}
.assistant-message {
  align-self: flex-start;
  background-color: var(--secondary-color);
  color: var(--text-color);
  border-bottom-left-radius: 5px;
}
.typing-indicator {
  display: flex;
  padding: 10px;
  background-color: var(--secondary-color);
  border-radius: 20px;
  margin-bottom: 10px;
  align-self: flex-start;
}
.typing-indicator span {
  height: 8px;
  width: 8px;
  background-color: #606060;
  border-radius: 50%;
  display: inline-block;
  margin: 0 2px;
  opacity: 0.4;
  animation: typing 1s infinite;
}
.typing-indicator span:nth-child(2) {
  animation-delay: 0.2s;
}

```

```

    }
    .typing-indicator span:nth-child(3) {
      animation-delay: 0.4s;
    }
    @keyframes typing {
      0% { opacity: 0.4; transform: scale(1); }
      50% { opacity: 1; transform: scale(1.2); }
      100% { opacity: 0.4; transform: scale(1); }
    }
  </style>
</head>
<body>
  <header class="header">
    
  </header>
  <main class="main-content">
    <div class="chat-container">
      <div id="messages-container"></div>
      <div class="input-container">
        <input type="text" id="user-input"
          ↪ placeholder="Upišite svoju poruku ovdje...">
        <button id="send-button" aria-label="Pošalji">
          <svg xmlns="http://www.w3.org/2000/svg"
            ↪ viewBox="0 0 24 24">
            <path d="M2.01 21L23 12 2.01 3 2 10l15 2-15
              ↪ 2z"/>
          </svg>
        </button>
      </div>
    </div>
  </main>

  <script>
    const messagesContainer =
      ↪ document.getElementById('messages-container');
    const userInput = document.getElementById('user-input');
    const sendButton = document.getElementById('send-button');

    // Set up axios to use the correct base URL
    const api = axios.create({
      baseURL: 'http://127.0.0.1:5000',

```

```

    timeout: 10000,
  });

  function addMessage(message, isUser) { // Add message to chat
    const messageElement = document.createElement('div');
    messageElement.classList.add('message');
    messageElement.classList.add(isUser ? 'user-message' :
      ↪ 'assistant-message');
    messageElement.textContent = message;
    messagesContainer.appendChild(messageElement);
    messagesContainer.scrollTop =
      ↪ messagesContainer.scrollHeight;
  }

  function showTypingIndicator() { // Show typing indicator
    const indicator = document.createElement('div');
    indicator.className = 'typing-indicator';
    indicator.innerHTML =
      ↪ '<span></span><span></span><span></span>';
    messagesContainer.appendChild(indicator);
    messagesContainer.scrollTop =
      ↪ messagesContainer.scrollHeight;
    return indicator;
  }

  async function sendMessage() { // Send user message and get
    ↪ response
    const message = userInput.value.trim();
    if (message) {
      addMessage(message, true);
      userInput.value = '';

      const typingIndicator = showTypingIndicator(); //
      ↪ Show typing indicator

      try {
        const response = await api.post('/query', {
          ↪ query: message });
        typingIndicator.remove();
        addMessage(response.data.response, false);
      } catch (error) {
        console.error('Greška:', error);
        typingIndicator.remove();
      }
    }
  }

```

```

        addMessage('Žao mi je, došlo je do pogreške
        → prilikom obrade vašeg zahtjeva.', false);
    }
}

sendButton.addEventListener('click', sendMessage); // Send
→ message on button click
userInput.addEventListener('keypress', (event) => {
    if (event.key === 'Enter') {
        sendMessage();
    }
});
</script>
</body>
</html>

```