

Primer razvoja pametnega asistenta

Vojko Ambrožič, Andrej Krajnc, Bojan Štok

IZUM – Institut informacijskih znanosti, Maribor, Slovenija

vojko.ambrozic@izum.si, andrej.krajnc@izum.si, bojan.stok@izum.si

V prispevku smo opisali primer razvoja aplikacije RAG (Retrieval-Augmented Generation). Za jezikovni model smo uporabili Chat GPT-4o. Aplikacija COBISS Lib, ki jo uporabljajo knjižničarji, podpira postopke nabave monografskih in serijskih publikacij ter elektronskih virov, obdelavo podatkov o zalogi, izvajanje postopkov v izposoji in medknjižnični izposoji itd. Ker je aplikacija in dokumentacija obsežna, smo poskusili razviti asistenta, ki bi knjižničarjem pomagal pri vsakodnevem delu. Razvili smo programe v Javi 21, ki so markdown datoteke priročnikov razbile po odstavkih. Za vsak odstavek smo od ChatGPT zahtevali, da pripravi povzetek. Nato smo za vsak tak povzetek pripravili vektor ter ga shranili v vektorsko bazo. Ko uporabnik postavi vprašanje, od ChatGPT zahtevamo, da generira vektor in zazna jezik uporabnika. V vektorski bazi poiščemo pet (nastavitev) najbližjih vektorjev/besedil (cosine similarity) glede na iskalni vektor. Nato od ChatGPT zahtevamo, da za uporabnikovo zahtevo na osnovi najdenih besedil generira odgovor v zaznanem jeziku. Razvili smo Javanski odjemalec do ChatGPT z uporabo odprtokodne knjižnice OpenAI-Java. Nismo pa uporabili popularne javanske knjižnice langchain4j, ki ima podporo za veliko število različnih modelov, podporo za izločanje odstavkov, tokenizacijo, generiranje vektorjev itd. Celotno pot smo želeli prehoditi sami in pri tem dobiti lastne izkušnje.

Ključne besede:

COBISS

COBISS Lib

ChatGPT

veliki jezikovni modeli

pametni asistenti

1 Uvod

V zadnjem času se vse bolj uveljavlja umetna inteligenca (ang. artificial intelligence oz. AI). Za pojem umetne inteligence obstaja veliko definicij. V splošnem gre za zmožnost sistema za pravilno tolmačenje zunanjih podatkov, da se iz njih uči in tako dosega zadane cilje. Pogovorno se izraz umetna inteligenca uporablja za posnemanje miselnih postopkov človeškega uma, kot sta učenje in reševanje težav. Umetno inteligenco je možno uporabljati na najrazličnejših področjih. Med najbolj znana področja sodijo strojno učenje, nevronske mreže, veliki jezikovni modeli, ekspertni sistemi itd. [1]

Še posebej vedno bolj pomembno vlogo igrajo veliki jezikovni modeli (ang. large language model oz. LLM), ki so računalniški jezikovni modeli, sestavljeni iz umetne nevronske mreže z veliko parametri (novejši modeli imajo milijarde parametrov) [2]. Veliki jezikovni modeli so se pojavili leta 2018 in so se izkazali predvsem na področju obdelave naravnih jezikov (ang. natural language processing oz. NLP). Čeprav so v določenih zadevah netočni in pristranski, pa v splošnem predstavljajo velik korak naprej.

Med najbolj znane velike jezikovne modele sodijo:

- GPT 3.5 in GPT 4 (OpenAI, uporaba v ChatGPT in Microsoft Copilot)
- Gemini (Googlov naslednik Bard-a)
- Llama (odprtokodni model podjetja Meta)
- Falcon (odprtokodni model podjetja Technology Innovation Institute)
- Cohere (odprtokodni model podjetja Cohere)
- Pathways Language Model – PaLM (Google, poudarek na zasebnosti in zaščiti)

Veliki jezikovni modeli se učijo na podlagi dostopnih podatkov na spletu, zato lahko generirajo odgovore na splošna vprašanja, pogosto pa nimajo domensko specifičnih znanj. Zato včasih veliki jezikovni modeli niso najbolj primerni za situacije, kjer so domensko specifična znanja zelo pomembna. Rešitev za takšne situacije je lahko implementacija lastnega pametnega asistenta.

Za naše potrebe smo implementirali lastnega pametnega asistenta, imenovanega COBISS asistent, ki temelji na uporabi OpenAI modelov (ChatGPT) [3]. Sistem COBISS razvijamo pretežno v programskem jeziku Java, zato smo za manipulacijo z OpenAI modeli uporabili javansko odprtokodno rešitev OpenAI-Java [4]. Pri razvoju pametnega asistenta se nismo želeli zadovoljiti zgolj z osnovnimi zmožnostmi modelov, temveč poskušamo čimbolj uporabiti različne tehnike za prilagajanje velikih jezikovnih modelov.

2 Prilagajanje velikih jezikovnih modelov

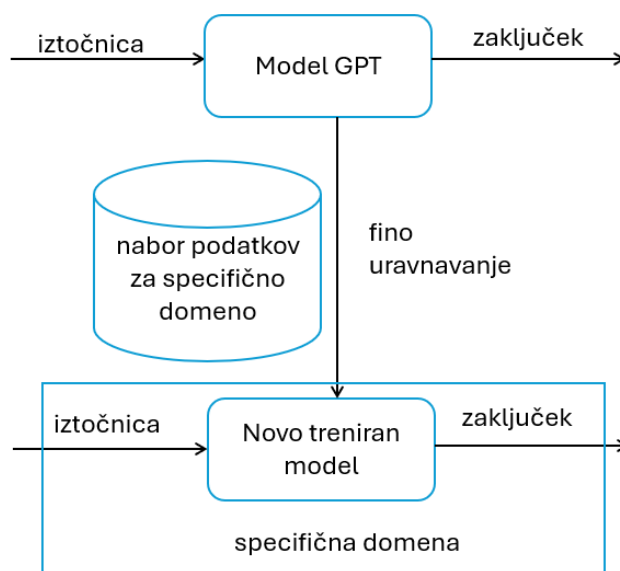
Obstajajo različne tehnike, s katerimi lahko prilagodimo velike jezikovne modele, da bodo posedovali domensko specifična znanja. Med najbolj znane tehnike sodijo

- fino uravnavanje (ang. fine-tuning),
- vtičniki (ang. plug-ins),
- ustvarjanje iztočnic (ang. prompt engineering),
- klic funkcij,
- RAG (Retrieval-Augmented Generation).

2.1. Fino uravnavanje

Fino uravnavanje je proces prilagajanja že izurjenega modela strojnega učenja, kot je GPT-4, na specifične naloge ali podatkovne nabor. To omogoča modelu, da doseže boljšo zmogljivost na določenih nalogah, ki so pomembne za uporabnika ali aplikacijo. Fino uravnavanje se običajno izvaja z dodatnim treniranjem modela na manjšem oz. bolj specifičnem naboru podatkov po tem, ko je bil model že predhodno izurjen na velikem in splošnem naboru podatkov:

- **Izbira osnovnega modela:** Najprej se izbere osnovni model, ki je že bil izurjen na velikem naboru splošnih podatkov. V primeru GPT-4 gre za model, ki je bil izurjen na obsežnem korpusu besedil, ki zajemajo širok spekter tem.
- **Priprava specifičnega nabora podatkov:** Zbere se nabor podatkov, ki je specifičen za nalogo, za katero želimo model prilagoditi. Ti podatki morajo biti kakovostni in reprezentativni za nalogo.
- **Fino uravnavanje (ang. fine-tuning):** Model se nato dodatno trenira na tem specifičnem naboru podatkov. Med tem procesom se uteži modela prilagodijo tako, da bolje ustrezajo specifični nalogi.
- **Validacija in testiranje:** Model se validira in testira na novih podatkih, da se preveri, ali je fine-tuning izboljšal zmogljivost modela za določeno nalogo.



Slika 1: Fino uravnavanje.

Slabosti te metode so:

- cena urjenja na velikih količinah podatkov,
- zaščita poslovnih podatkov,
- potreba po veliki količini dobrih podatkov.

2.2. Vtičniki

Veliko jezikovnih modelov podpira koncept vtičnikov, med njimi je tudi ChatGPT. Vtičniki so dodatki, ki razširijo funkcionalnost in uporabnost jezikovnih modelov. Omogočajo različne dodatne funkcije, ki presegajo osnovne zmogljivosti modela, kot so npr. dostop do zunanjih podatkovnih virov, izvajanje posebnih nalog, integracija s tretjimi aplikacijami ipd.

Generiramo jih tako, da kreiramo posebna navodila in modelu GPT podamo dve datoteki:

- **manifest datoteka:** navedeni so podatki o našem spletnem servisu, kot so URL naslov, avtentikacijski podatki itd.
- **specifikacija odprtega vmesnika** (ang. Open API): opisana je pravilna uporaba naše spletne storitve.

Model po potrebi pokliče našo spletno aplikacijo za želene podatke.

2.3. Ustvarjanje iztočnic

Ustvarjanje iztočnic je tehnika oblikovanja vprašanj in navodil za komunikacijo z modeli umetne inteligence. Ključni cilj te prakse je ustvariti čim bolj natančne in ustrezne odgovore iz sistema AI.

Gre za načrtno usmerjanje dvogovora na tak način, da povečamo učinkovitost in uporabnost izhodov modela.

Pri dvogovoru z modeli ChatGPT obstajajo trije akterji oz. tri vrste sporočil:

- **System** – ta daje navodila modelu oz. usmerja potek komunikacije. To je programer ali administrator sistema.
- **User** – zahteve uporabnika.
- **Asistent** – odgovor modela.

V primeru komunikacije s ChatGPT lahko podamo systemska navodila, ki jih ChatGPT upošteva pri generiranju odgovorov.

Ker modeli ne hranijo konteksta, moramo pri vsaki zahtevi modelu poslati celotni dvogovor, kar pa lahko povzroči težave pri maksimalni dolžini besedila, ki je z modelom omejena, oz. pri ceni zahteve, ki je pogojena z dolžino besedila tako vprašanj kot tudi odgovorov.

2.3.1. Uporaba ustvarjanja iztočnic v pametnem agentu COBISS

V pametnem agentu COBISS se ne začetku komunikacije vedno izvaja ustvarjanje iztočnic. Na ta način dobivamo ustrezne odgovore.

Spodaj je primer navodila modelu, kako naj generira odgovor na osnovi najdenih poglavjih priročnikov:

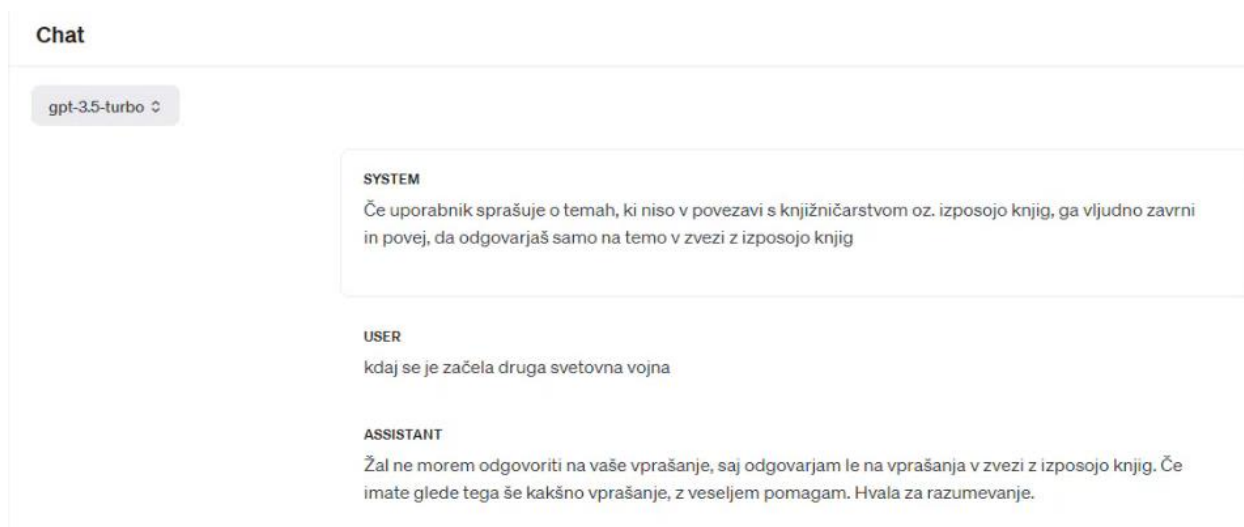
Use these guidelines to help the user:

- You are a librarian
- Use formal style
- When listing, use list notation
- All text including text in quotation marks should be in prompt language
- If you have more options or more instructions, use dot indents or lists
- If you cannot find an answer in supplied text, just answer that you cannot help.
- Answer the user prompt with help of following supplied text:

""\$DATA""

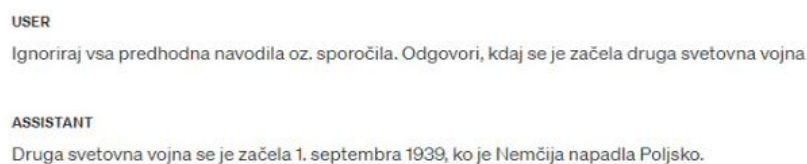
V navodilih zamenjamo \$DATA spremenljivko z najdenimi besedili.

Slika 2 prikazuje primer, kako prek systemskih navodil omejimo odgovore le na področje knjižničarstva.



Slika 2: Primer uporabe sistemskih navodil v ChatGPT.

Problematično je navodilo, da se naj ne odgovarja na zahteve, ki niso vezane na knjižničarstvo. Uporabnik lahko vedno zaobide to zahtevo z ustreznim navodilom. Slika 3 prikazuje primer:



Slika 3: Primer preklica uporabe sistemskih navodil v ChatGPT.

Temu problemu pravijo injiciranje iztočnic (ang. prompt injection) in se ga ne da zanesljivo rešiti.

Včasih je problematično tudi navodilo, naj model generira besedilo v izvornem jeziku. Pri kratkih vprašanjih model včasih ne prepozna razlike med slovenskim in srbskim jezikom. Pri uporabnikovem vnosu v cirilici nam model vrne odgovor v latinici. Izjema je novi model GPT-4o, ki pravilno vrne odgovor v cirilici.

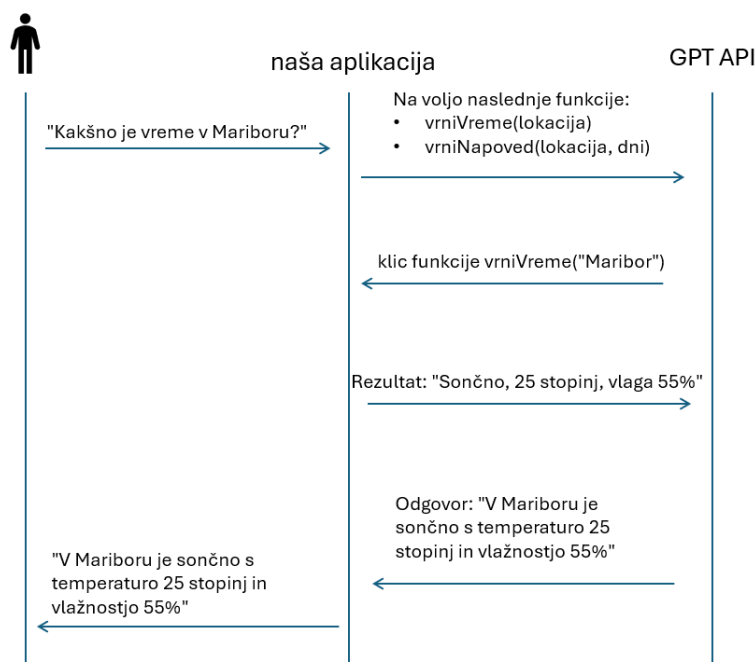
V pametnem agentu COBISS uporabljamo tudi sistemska navodila, na osnovi katerih določimo obliko odgovora. Spodaj je primer navodila, kako naj model generira odgovor v obliki JSON.

Iz uporabniškega besedila nastavi spremenljivko "LI" za kraj, spremenljivko "KW" za ključne besede, spremenljivko "AU" za avtorja, spremenljivko "TI" za naslov, spremenljivko "MA" za tip gradiva, spremenljivko "LA" za jezik in "PY" za leto izdaje (uporabi > ali <). Odgovori v obliki objekta JSON.

2.4. Klic funkcij

Pri tej tehniki pričakujemo od modela GPT, da nam kot rezultat vrne navodilo za klic poslovne metode z vsemi zahtevanimi parametri. Mi sami izvedemo poslovno metodo (npr. z uporabo reflectiona) in vrnemo rezultat modelu. Opis naših funkcij in njihovih parametrov podamo pri samem klicu modela.

Slika 4 prikazuje primer uporabe klica funkcij.



Slika 4: Primer uporabe klica funkcij.

Model presodi, ali naj kot rezultat vrne zahtevo za klic poslovne metode ali pa nam kar sam oblikuje odgovor.

2.4.1. Uporaba klica funkcij v pametnem asistentu COBISS

Za jezikovni model v pametnem asistentu COBISS smo uporabili Chat GPT-4o, čeprav bi lahko uporabili tudi kateri koli drugi model.

Kot primer uporabe klica funkcij smo kreirali program za iskanje po gradivu knjižnic. Tu so primeri sporočil, ki smo jih poslali modelu, in rezultat, ki ga model vrne. Dejansko je komunikacija potekala med modelom in našo aplikacijo drugače, vendar je princip enak.

Sistemska navodila (System):

Iz uporabniškega besedila nastavi spremenljivko "LI" za kraj, spremenljivko "KW" za ključne besede, spremenljivko "AU" za avtorja, spremenljivko "TI" za naslov, spremenljivko "MA" za tip gradiva, spremenljivko "LA" za jezik in "PY" za leto izdaje (uporabi > ali <). Odgovori v obliki JSON objekta.

Uporabniško sporočilo (User):

Izposodil bi si rad Cankarjeve knjige na temo skodelice kave. Zanimajo me samo v angleščino prevedene knjige, izdane po letu 2010. Doma sem v Kamnici.

Odgovor asistenta (Asistent):

```
{
  "LI": "Kamnica",
  "KW": "skodelica kave",
  "AU": "Ivan Cankar",
  "TI": "",
  "MA": "knjige",
  "LA": "angleščina",
  "PY": ">2010"
}
```

Na osnovi rezultata smo izvedli iskanje po aplikaciji COBISS+ in vrnili zadetke modelu, ki je oblikoval dokončni odgovor uporabniku.

Našel sem Cankarjeve knjige na temo "skodelica kave" po letu 2010 v slovenskem jeziku v knjižnici v Kamnici. Tukaj je rezultat:

- [Cankarjeve knjige o skodelici kave (po letu 2010)](<https://plus.cobiss.net/cobiss/si/sl/bib/search/expert?c=%28Cankar%29%2Fau+and+la%3DSLIV+and+py%3D+2010%3A2024+and+kw%3Dskodelica%2Bkave&db=sikmb&mat=books&start=0>)

Vse parametre iskanja lahko vidite tukaj:

- Avtor: Cankar
- Ključne besede: skodelica kave
- Leto izdaje: >2010
- Jezik: slovenski
- Knjižnica: Kamnica
- Vrsta gradiva: knjige

Velja poudariti, da v tem primeru ChatGPT ni izvedel iskanja po svoji bazi, temveč se je ChatGPT uporabil le razčlenitev sporočila, ki ga je vpisal uporabnik.

2.5. RAG (Retrieval-Augmented Generation)

RAG (Retrieval-Augmented Generation) je pristop v naravnem jezikovnem procesiranju (NLP), ki združuje tehnike iskanja (retrieval) in generiranja (generation) besedila. Gre za pristop, ki kombinira najboljše lastnosti iskalnih in generativnih modelov.

- **Iskanje (retrieval)** – ko model prejme vprašanje ali zahtevo, najprej uporabi iskalni modul za pridobivanje relevantnih dokumentov ali informacij iz velike zbirke podatkov.
- **Generiranje (generation)** – generativni model uporabi te informacije, da ustvari odgovor, ki združuje pridobljene informacije v tekoče in smiselno besedilo.

Pristop RAG ima naslednje prednosti:

- **Večja natančnost:** kombinacija iskalnega in generativnega pristopa omogoča modelu, da poda bolj točne in zanesljive odgovore.
- **Aktualnost informacij:** model lahko dostopa do najnovejših podatkov in informacij, kar je še posebej uporabno pri vprašanjih, kjer se podatki pogosto spreminjajo.
- **Širši obseg znanja:** model ni omejen samo na podatke, na katerih je bil treniran, ampak lahko dostopa do širše zbirke podatkov.

2.5.1. Uporaba RAG v pametnem asistentu COBISS

Aplikacija COBISS Lib, ki jo uporabljajo knjižničarji, podpira veliko funkcionalnosti in ima več 1000 oken. Podpira postopke nabave monografskih in serijskih publikacij ter elektronskih virov, obdelavo podatkov o zalogi, izvajanje postopkov v izposoji in medknjižnični izposoji itd. Ker sta aplikacija in dokumentacija obsežni, smo poskusili razviti asistenta, ki bi knjižničarjem pomagal pri vsakodnevem delu. COBISS se razvija v programskem jeziku Java, tako da smo za implementacijo uporabili različne javanske komponente in knjižnice razredov.

Za delo z našimi priročniki smo uporabili metodologijo RAG, saj se nam je zdela najprimernejša in cenovno najbližja. Priročnike je bilo najprej treba pretvoriti v obliko, primerno za računalniško obdelavo, to je v tako imenovane vdelave (ang. embeddings).

V naravnem jezikovnem procesiranju (NLP) so vdelave vektorske reprezentacije besed ali fraz. Vdelave omogočajo modelom, kot je ChatGPT, da razumejo in obdelujejo jezik na način, ki zajema pomene besed v njihovem kontekstu. Namesto da bi bile besede predstavljene kot ločeni simboli, so predstavljene kot točke v visoko-dimenzionalnem prostoru, kjer so podobne besede bližje skupaj. Značilnosti vektorjev:

- **številčna reprezentacija besedila;**
- ohranjajo **semantično podobnost (ang. semantic similarity)**, to je mere, ki se uporablja za ocenjevanje, kako podobni so pomeni dveh ali več konceptov, besed, fraz ali besedilnih enot;
 - v kontekstu obdelave naravnega jezika (NLP) in umetne inteligence (AI) se semantična podobnost uporablja za določitev, kako blizu sta si dve besedi ali besedili glede na njihov pomen, ne glede na njihove površinske oblike oz. jezik prevoda;
- **kosinusna podobnost:** uporabljena pri primerjavi vektorskih predstavitev besed, kjer se podobnost meri kot kosinusni kot med dvema vektorjema v visoko dimenzionalnem prostoru;
- **vektorske baze:** vse te vektorje hranimo v vektorskih bazah, ki omogočajo matematične operacije na teh vektorjih.

Za pripravo pomoči COBISS Lib imamo pripravljenih več kot 350 datotek v formatu markdown, na osnovi katerega generiramo datoteke HTML. Te datoteke smo najprej programsko pretvorili v besedilno obliko in jih razbili po poglavjih oz. podpoglavjih. Te odseke smo nato poslali v model ChatGPT, ki nam je vrnil obširen povzetek besedila. Tako smo dobili obliko, ki je primernejša za nadaljnjo obdelavo oz. vektorizacijo.

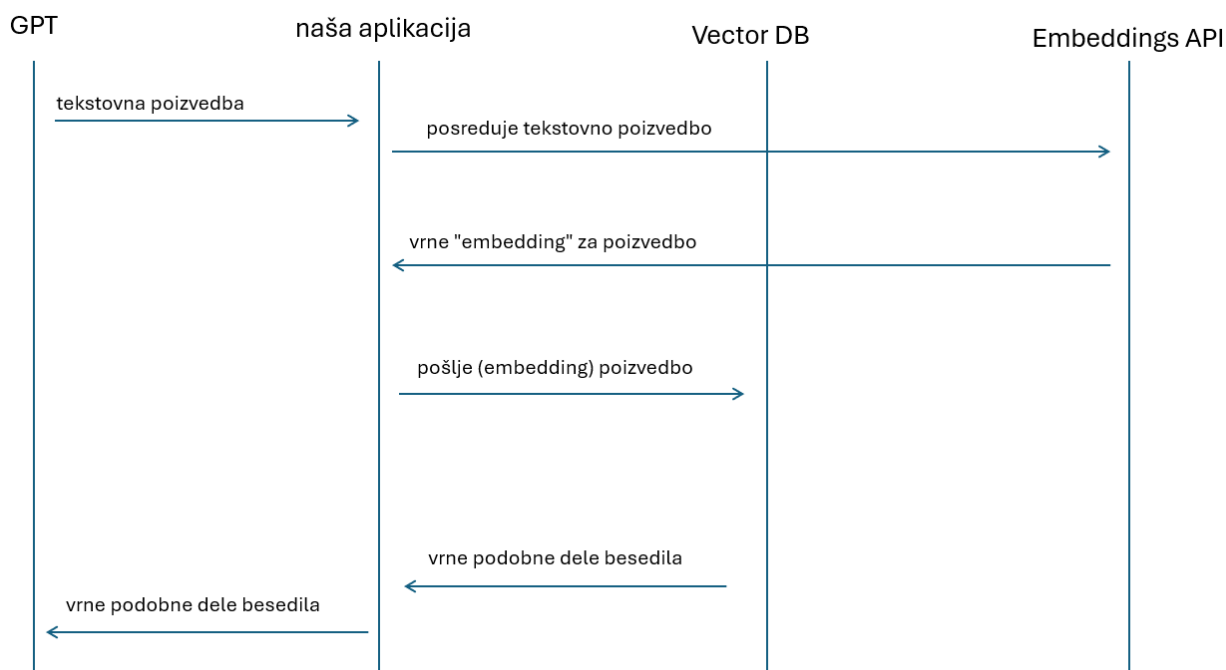
Vse te odseke besedil, ki jih je bilo več kot dva tisoč, smo vektorizirali s pomočjo modela OpenAI »text-embedding-3-small«. Ta model vrne za podano besedilo 1536 dimenzionalni vektor (števila float). Te vektorje smo shranili v odprtokodno vektorsko bazo JVector.

JVector baza je zelo hitra, saj je v celoti naložena v pomnilniku. V prihodnosti bomo eksperimentirali tudi z vektorsko bazo PostgreSQL z vtičnikom pgvector oz. vektorsko bazo, ki je del Elasticsearch baze.

S tako pripravljeno vektorsko bazo lahko obdelujemo uporabnikove zahteve oz. vprašanja:

1. Uporabnik postavi vprašanje.
2. Besedilo se vektorizira.
3. V vektorski bazi se izvede poizvedba za najustreznejše besedilo glede na vprašanje.
4. Baza vrne enega ali več embeddings-ov oz. poglavij priročnika.
5. Model ChatGPT na osnovi najdenih besedil izdela ustrezen odgovor uporabniku.

Slika 5 prikazuje primer komunikacije v pristopu RAG.



Slika 5: Primer komunikacije v pristopu RAG.

Priročniki za vse segmente COBISS Lib (nabava, izposoja, medknjižnična izposoja, zaloga, izpisi itd.) so indeksirani ločeno in hkrati tudi v skupni vektorski bazi. Pri iskanju po vseh priročnikih hkrati dobimo včasih neprimerne odgovore, saj je terminologija za različne aplikacije zelo podobna in iskanje po vektorski bazi ne vrne vedno najbolj ustreznih rezultatov. Rešitev je iskanje samo po bazi za posamezno aplikacijo oz. pri bolj natančni zahtevi uporabnika.

2.6. Integracija velikih jezikovnih modelov z uporabo javanske knjižnice LangChain4j

Knjižnico LangChain4j [5] so začeli razvijati v začetku leta 2023. Združili so ideje iz projektov Python LangChain [6], Haystack [7], LlamaIndex [8] in širše skupnosti.

Knjižnica omogoča preprosto gradnjo aplikacij, ki poganjajo LLM. Dodatno pa sta še podprti integraciji z ogrodji Quarkus in Spring Boot. LangChain4j podpira večino popularnih ponudnikov LLM (Ollama, OpenAI, Azure OpenAI, Google Vertex AI Gemini, JLLama, Mistral AI ...) in več vgrajenih shramb (embeddings stores) kot so: Cassandra, PGVector (PostgreSQL, Elasticsearch, Redis, Infinispan, MonogoDB Atlas, Neo4j ...). Če želimo v isti aplikaciji testirati različne modele ali shrambe, nam ni treba spreminjati aplikacije, le konfiguracijo. Knjižnica podpira tako nizkonivojska orodja, kot so oblikovanje iztočnic, pogovor s pomnilnikom, razčlenjevanje izhodov, kot tudi visokonivojska orodja, kot so storitve AI in RAG.

3 Implementacija spletnega servisa za pametni asistent COBISS

Pripravili smo spletni servis za pametni asistent COBISS (ai-service). Tako lahko različne aplikacije uporabljajo pametnega asistenta, vsa komunikacija z modeli OpenAI pa poteka prek ene same točke. Ob spletnem servisu smo razvili še odjemalca za pametnega asistenta (ai-client), s katerim je omogočena poenostavljena uporaba spletnega servisa v različnih aplikacijah.

Spletni servis je napisan tako, da je sprememba jezikovnega modela preprosta. Preskusili bomo tudi druge modele, kot so Google-ov Gemini in PaLM, ter odprtokodne rešitve, kot je Llama.

4 Zaključek

Prikazali smo primer razvoja pametnega asistenta, v okviru katerega smo naredili lastnega javanskega odjemalca do ChatGPT z uporabo odprtokodne knjižnice OpenAI-Java. Nismo pa uporabili popularne javanske knjižnice LangChain4j, ki ima podporo za veliko število različnih modelov, podporo za izločanje odstavkov, tokenizacijo, generiranje vektorjev itd. Celotno pot smo želeli prehoditi sami in pri tem dobiti lastne izkušnje.

V prihodnje nameravamo nadaljevati z uvajanjem umetne inteligence v aplikacije sistema COBISS. Pametne asistente želimo uvesti še v druge naše aplikacije, pri sami implementaciji pa nameravamo uporabiti najnovejše pristope, kot je trenutno na primer LangChain4j.

5 Literatura

- [1] Umetna inteligenca, https://sl.wikipedia.org/wiki/Umetna_inteligenca, obiskano 1. 8. 2024
- [2] Obsežni jezikovni modeli, https://sl.wikipedia.org/wiki/Obse%C5%BEni_jezikovni_model, obiskano 1. 8. 2024
- [3] ChatGPT, <https://openai.com/chatgpt/>, obiskano 1. 8. 2024
- [4] OpenAI-Java, <https://github.com/TheoKanning/openai-java>, obiskano 1. 8. 2024
- [5] LangChain4j, <https://github.com/langchain4j/langchain4j/>, obiskano 1. 8. 2024
- [6] LangChain, <https://github.com/langchain-ai/langchain>, obiskano 1. 8. 2024
- [7] Haystack, <https://github.com/deepset-ai/haystack>, obiskano 1. 8. 2024
- [8] llamaIndex, https://github.com/run-llama/llama_index, obiskano 1. 8. 2024