

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 874

**SUSTAV ZA ANALIZU META PODATAKA
OTVORENIH SKUPOVA PODATAKA**

Luka Habuš

Zagreb, srpanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 874

**SUSTAV ZA ANALIZU META PODATAKA
OTVORENIH SKUPOVA PODATAKA**

Luka Habuš

Zagreb, srpanj 2025.

DIPLOMSKI ZADATAK br. 874

Pristupnik: **Luka Habuš (0036517946)**

Studij: Računarstvo

Profil: Znanost o mrežama

Mentor: izv. prof. dr. sc. Igor Čavrak

Zadatak: **Sustav za analizu meta podataka otvorenih skupova podataka**

Opis zadatka:

Sve veća dostupnost otvorenih podataka ne podrazumijeva i njihovu veću iskoristivost. Iako normirani formati meta podataka (npr. DCAT) omogućuju jednostavnije pronalaženje i tumačenje objavljenih pojedinačnih skupova podataka, dodatna vrijednost nalazi se u njihovom povezivanju i pronalaženju novih uvida i skrivenog znanja. Nagli uspon alata umjetne inteligencije temeljenih na velikim jezičnim modelima predstavlja obećavajući smjer za izradu na njima temeljenih alata, a koji bi običnim korisnicima pružili novi uvid u i korištenje otvorenih podataka. U ovom diplomskom radu potrebno je proučiti mogućnosti velikih jezičnih modela, na njima temeljenih alata i tehnika. Također je potrebno proučiti problematiku opisivanja skupova otvorenih podataka korištenjem norme DCAT i mogućnosti automatiziranog traženja veza između skupova podataka. Predložiti alat za dohvat i analizu meta podataka otvorenih skupova podataka, kao i za pružanje podrške korisnicima u povezivanju skupova podataka temeljene na analizi meta podataka. Na poslijetku, potrebno je implementirati prototip sustava za portal CKAN korištenjem alata temeljenih na velikim jezičnim modelima i ocijeniti uporabljivost sustava.

Rok za predaju rada: 4. srpnja 2025.

Sadržaj

Sadržaj	1
Popis slika	3
1. Uvod	4
1.1. Problemski okvir i motivacija	4
1.2. Predloženo rješenje i ciljevi rada	5
1.3. Struktura rada	7
2. Korištene tehnologije i alati	9
2.1. Tehnologije za upravljanje podacima	9
2.1.1. SPARQL (SPARQL Protocol and RDF Query Language)	9
2.1.2. DCAT (Data Catalog Vocabulary)	10
2.1.3. ChromaDB Vektorska Baza Podataka	12
2.2. Komponente umjetne inteligencije	15
2.2.1. Veliki jezični modeli (LLM) - OpenAI GPT-4	15
2.2.2. Modeli za semantičke ugradbe - Sentence Transformers	16
2.2.3. Orkestracijski okviri - LangChain	18
3. Arhitektura sustava	20
3.1. Korištene tehnologije	20
3.2. Ključne funkcionalnosti	20
3.3. Arhitektura i skalabilnost	21
3.4. Implementacija RAG sustava i vektorske baze	24
3.5. Generiranje SPARQL upita i prompt engineering	27
3.6. Ekstrakcija sheme i DCAT analiza	30

3.7. Unified Data Assistant i multimodalno pretraživanje	35
3.8. Validacija, rukovanje greškama i optimizacija	40
3.9. Cjelokupni pregled i tok podataka	45
3.10. Arhitektura RAG podsustava	50
3.10.1. Dizajn vektorske baze	50
3.10.2. Dizajn procesa generiranja upita	53
3.11. Arhitektura multimodalnog pretraživanja	56
4. Evaluacija sustava	57
4.1. Metodologija i testno okruženje	57
4.1.1. Izvor podataka	57
4.1.2. Testni skup upita	58
4.1.3. Definirane metrike	58
4.1.4. Testno okruženje	60
4.2. Analiza rezultata	61
4.2.1. Uspješnost i točnost generiranja upita	61
4.2.2. Analiza performansi i vremena odziva	63
4.2.3. Predmemoriranje i optimizacija	65
4.3. Usporedna analiza i testiranje stabilnosti	65
4.3.1. Usporedba s alternativnim pristupima	66
4.3.2. Testovi stabilnosti	67
4.3.3. Evaluacija korisničkog iskustva	67
4.4. Diskusija rezultata evaluacije	69
4.4.1. Ključni doprinosi	69
4.4.2. Područja za poboljšanje	70
5. Rasprava i budući rad	71
5.1. Ograničenja	71
5.1.1. Ovisnost o komercijalnim LLM API-jima	71
5.1.2. Jezična podrška	73
5.1.3. Skalabilnost sustava	73
5.1.4. Domensko usmjeravanje	74
5.2. Smjernice za buduća istraživanja	75

5.2.1.	Implementacija lokalnih LLM-ova	75
5.2.2.	Unapređenje baze primjera	77
5.2.3.	Bolje strategije sinteze	78
5.2.4.	Višejezična podrška	79
5.2.5.	Prelazak na distribuiranu arhitekturu	80
5.2.6.	Dodatne značajke korisničkog sučelja	81
5.3.	Dugoročna vizija	82
5.3.1.	Integracija s AI ekosistemom	82
5.3.2.	Standardizacija pristupa	82
5.3.3.	Edukacijska komponenta	83
6.	Zaključak	84
6.1.	Glavna postignuća	84
6.1.1.	Tehnički doprinosi	84
6.1.2.	Praktični utjecaj	85
6.2.	Odgovori na istraživačka pitanja	85
6.3.	Ograničenja i budući rad	86
6.4.	Smjernice za budući razvoj	86
6.5.	Završna razmatranja	87
7.	Literatura	88
	Sažetak	90
	Abstract	91

Popis slika

1.1. DCAT standard - prikazuje osnovne klase i svojstva za opisivanje kataloga podataka [1]. Izvor: Autor na temelju W3C DCAT specifikacije.	8
2.1. DCAT konceptualni model - prikazuje osnovne klase i njihove međusobne veze. Izvor: Autor na temelju W3C DCAT specifikacije.	11
2.2. EU Portal otvorenih podataka - sučelje koje koristi DCAT standard za organizaciju podataka. Izvor: Snimka zaslona EU Portala otvorenih podataka (data.europa.eu).	13
2.3. ChromaDB arhitektura - prikazuje komponente vektorske baze podataka i tok obrade podataka. Izvor: Autor na temelju ChromaDB dokumentacije.	13
2.4. LangChain komponente u akciji - prikaz orkestracije različitih AI komponenti. Izvor: Izvjestaj.docx (postojeći rad autora).	19
3.1. ChromaDB arhitektura - prikazuje komponente vektorske baze podataka. Izvor: Autor na temelju ChromaDB dokumentacije.	21
3.2. Ekstrakcija sheme i validacija generiranog SPARQL upita. Izvor: Izvjestaj.docx (postojeći rad autora).	21
3.3. Multimodalno pretraživanje - prikazuje različite strategije i njihovu integraciju. Izvor: Autor (LaTeX dijagram na hrvatskom jeziku).	22
3.4. Skalabilnost sustava - prikazuje mogućnosti proširivanja i optimizacije. Izvor: Izvjestaj.docx (postojeći rad autora).	22
3.5. Korisničko sučelje - prikazuje dizajn i funkcionalnosti. Izvor: Izvjestaj.docx (postojeći rad autora).	23
3.6. Arhitektura sustava - prikazuje tok podataka kroz različite komponente. Izvor: Autor (LaTeX dijagram na hrvatskom jeziku).	24

3.7. RAG pipeline - prikazuje tok od upita do generiranog SPARQL-a. Izvor: Autor (LaTeX dijagram na hrvatskom jeziku).	27
3.8. Generiranje SPARQL upita. Izvor: Izvjestaj.docx (postojeći rad autora).	31
3.9. Primjer prompta - struktura i elementi prompta za LLM. Izvor: Izvjestaj.docx (postojeći rad autora).	31
3.10. Pokretanje složenijeg upita. Izvor: Autor (snimka zaslona sustava).	36
3.11. Multimodalno pretraživanje. Izvor: Autor (snimka zaslona sustava).	38
3.12. Rangiranje kombiniranih rezultata. Izvor: Autor (LaTeX dijagram - možda treba ažurirati na hrvatski).	40
3.13. Rukovanje greškama i fallback strategija. Izvor: Izvjestaj.docx (postojeći rad autora).	46
3.14. UML dijagram komponenti sustava - prikazuje glavne module i njihove međusobne ovisnosti. Izvor: Autor (LaTeX dijagram na hrvatskom jeziku).	46
3.15. UML slijedni dijagram za obradu korisničkog upita - prikazuje interakcije između komponenti. Izvor: Izvjestaj.docx (postojeći rad autora).	48
3.16. UML dijagram aktivnosti - prikazuje paralelne tokove obrade upita. Izvor: Izvjestaj.docx (postojeći rad autora).	50
3.17. Konceptualni prikaz RAG <i>pipeline</i> -a s detaljima svake faze. Izvor: Autor (LaTeX dijagram na hrvatskom jeziku).	53
3.18. Arhitektura multimodalnog pretraživanja - UML dijagram komponenti. Izvor: Autor (LaTeX dijagram na hrvatskom jeziku).	56
4.1. Stopa uspjeha generiranja upita po kategorijama složenosti. Izvor: Izvjestaj.docx (postojeći rad autora).	61
4.2. Distribucija vremena odziva po komponentama sustava. Izvor: Izvjestaj.docx (postojeći rad autora).	63
4.3. Utjecaj predmemoriranja na vrijeme odziva. Izvor: Izvjestaj.docx (postojeći rad autora).	65
4.4. Usporedba pristupa po različitim metrikama. Izvor: Izvjestaj.docx (postojeći rad autora).	66
4.5. Ocjene korisničkog iskustva po kategorijama (skala 1-5). Izvor: Izvjestaj.docx (postojeći rad autora).	68

5.1. Predložena mikroservisna arhitektura za skalabilnost. Izvor: Izvjestaj.docx (postojeći rad autora).	80
---------------------------------------------------------------------------------------------------------------------	----

1. Uvod

1.1. Problemski okvir i motivacija

U suvremenom digitalnom ekosustavu, otvoreni podaci predstavljaju fundamentalni resurs za unapređenje transparentnosti javne uprave, poticanje znanstvenih istraživanja i generiranje ekonomskih inovacija. Portali poput službenog EU Portala otvorenih podataka nude pristup milijunima skupova podataka koji obuhvaćaju širok spektar domena, od okoliša i energetike do zdravstva i ekonomije. Unatoč golemom intrinzičkom potencijalu, stvarna iskoristivost ovih resursa ostaje na relativno niskoj razini. Uzroci ovog nesrazmjera su višestruki i složeni.

Primarno, postoji tehnička barijera. Pretraživanje i dohvaćanje podataka s portala koji primjenjuju tehnologije semantičkog weba zahtijeva ovladavanje specifičnim upitnim jezicima, među kojima se ističe SPARQL. Navedeni jezik, unatoč svojoj izražajnoj moći, nije intuitivan za korisnike bez formalne tehničke naobrazbe, kao što su novinari, analitičari javnih politika ili studenti. Time se sužava krug potencijalnih korisnika otvorenih podataka.

Nadalje, uočava se nepreciznost tradicionalnih metoda pretrage. Većina postojećih portala implementira pretragu temeljenu na podudaranju ključnih riječi. Takav pristup često rezultira nerelevantnim ishodima jer ne posjeduje sposobnost razumijevanja semantičkog konteksta korisničkog upita. Primjerice, upit "utjecaj prometa na kvalitetu zraka u urbanim sredinama" može propustiti relevantne skupove podataka koji koriste semantički ekvivalentne termine poput "emisije ispušnih plinova vozila" ili "onečišćenje zraka u gradskim područjima".

Posebno značajan izazov predstavlja problem složenih analitičkih pitanja čiji odgovor zahtijeva korištenje i povezivanje više različitih skupova podataka. Najveća vrijednost

otvorenih podataka manifestira se upravo kroz njihovu sintezu i međusobno povezivanje s ciljem stjecanja novih spoznaja. Odgovor na složenije analitičko pitanje, kao što je "Analiza korelacije između poljoprivrednih subvencija u NUTS 2 regijama i rasta BDP-a tih regija u posljednjih pet godina", nalaže pronalaženje, interpretaciju i integraciju višestrukih, često heterogenih skupova podataka. Takvi složeni upiti zahtijevaju ne samo identifikaciju relevantnih skupova podataka već i razumijevanje njihovih međusobnih veza, kompatibilnosti formata te mogućnosti integracije. Manualno izvođenje ovog procesa je zahtjevno, vremenski intenzivno i podložno pogreškama.

Dodatno, korisnici se susreću s problemom interpretacije metapodataka. Čak i kada uspiju pronaći potencijalno relevantne skupove podataka, razumijevanje njihove strukture, kvalitete i prikladnosti za specifičnu analizu predstavlja izazov. DCAT standard, iako pruža strukturiran način opisivanja podataka, često zahtijeva tehničko znanje za pravilnu interpretaciju.

Navedeni problemi generiraju diskrepanciju između proklamiranog potencijala otvorenih podataka i njihove stvarne primjene u praksi. Ova diskrepancija čini središnju motivaciju ovog istraživanja.

1.2. Predloženo rješenje i ciljevi rada

S ciljem premošćivanja identificiranog jaza između dostupnosti i iskoristivosti otvorenih podataka, ovaj rad predlaže razvoj inteligentnog sustava za analizu metapodataka otvorenih skupova podataka. Jezgru predloženog rješenja čini arhitektura *Retrieval-Augmented Generation* (RAG) [6], koja sinergijski kombinira sposobnosti velikih jezičnih modela (LLM) s preciznošću dohvata informacija iz strukturiranih baza znanja.

Sustav je projektiran s namjerom da korisnički upit, formuliran na prirodnom jeziku, prevede u sintaksno ispravan i semantički relevantan SPARQL upit. Ova transformacija omogućava premošćivanje jaza između intuitivnog načina na koji korisnici postavljaju pitanja i formalnog jezika potrebnog za interakciju s bazama podataka semantičkog weba.

Operativni proces sustava započinje pretraživanjem interne vektorske baze podataka radi pronalaska relevantnih primjera prethodno formuliranih upita i informacija o struk-

turi (shemi) podataka dostupnih na portalu. Vektorska baza koristi tehnike semantičkog pretraživanja temeljene na *embeddings* tehnologiji, što omogućava pronalaženje sličnih upita čak i kada ne postoji doslovno podudaranje ključnih riječi. Dohvaćene informacije, koje čine kontekst, pridružuju se originalnom korisničkom upitu. Tako obogaćen unos prosljeđuje se velikom jezičnom modelu (npr. GPT-4) [4], koji na temelju pruženog konteksta generira ciljani SPARQL upit.

Glavni cilj rada jest implementacija i evaluacija funkcionalnog prototipa opisanog sustava, s primjenom na EU Portal otvorenih podataka. Specifični ciljevi rada obuhvaćaju:

1. **Dizajnirati i implementirati RAG arhitekturu** optimiziranu za generiranje SPARQL upita. Ova arhitektura mora biti sposobna kombinirati prednosti semantičkog pretraživanja s generativnim sposobnostima velikih jezičnih modela.
2. **Uspostaviti i održavati vektorsku bazu podataka** koja sadrži reprezentativne primjere upita i relevantne metapodatke o shemi. Baza mora biti strukturirana na način koji omogućava brzo i precizno dohvaćanje relevantnih informacija.
3. **Implementirati multimodalni pristup pretraživanju** koji integrira RAG-generirane SPARQL upite, REST API pozive i pretragu po sličnosti. Ovaj hibridni pristup osigurava stabilnost sustava i pokriva različite scenarije korištenja.
4. **Razviti mehanizme za rukovanje složenim upitima** koji zahtijevaju podatke iz više različitih skupova. Sustav mora biti sposoban identificirati potrebu za integracijom podataka i predložiti strategije za njihovo povezivanje.
5. **Sustavno evaluirati performanse, točnost i stabilnost** razvijenog prototipa kroz sveobuhvatan skup testnih scenarija koji pokrivaju različite domene i razine složenosti.
6. **Pružiti detaljan inženjerski nacrt** rješenja koje je modularno, transparentno i pogodno za daljnju nadogradnju. Dokumentacija mora omogućiti drugim istraživačima i praktičarima reproduciranje i proširivanje sustava.

Krajnji ishod ovog rada je sustav koji pridonosi demokratizaciji pristupa otvorenim

podacima, omogućujući korisnicima bez specijaliziranih tehničkih znanja postavljanje složenih analitičkih pitanja i dobivanje relevantnih odgovora. Time se proširuje krug potencijalnih korisnika otvorenih podataka i povećava njihova društvena vrijednost.

1.3. Struktura rada

Ovaj diplomski rad organiziran je u devet poglavlja koja sistematično pokrivaju sve aspekte istraživanja, dizajna, implementacije i evaluacije predloženog sustava.

Poglavlje 2 pruža detaljan opis ključnih tehnologija i alata koji čine temelj sustava. Prvo se predstavljaju tehnologije za upravljanje podacima, uključujući SPARQL kao upitni jezik za semantički web, DCAT standard za opisivanje kataloga podataka te ChromaDB kao vektorsku bazu podataka. Zatim se analiziraju komponente umjetne inteligencije: veliki jezični modeli (posebno OpenAI GPT-4), *Sentence Transformers* modeli za generiranje semantičkih *embeddings* reprezentacija te LangChain okvir za orkestraciju složenih AI aplikacija.

Poglavlje 3 nudi sveobuhvatan pregled arhitekture sustava, primjenjujući UML dijagrame za vizualizaciju toka podataka, komponenti i njihovih interakcija. Detaljno se opisuje arhitektura RAG podsustava, uključujući dizajn vektorske baze i proces generiranja upita. Posebna pozornost posvećena je arhitekturi multimodalnog pretraživanja koja omogućava sinergiju različitih pristupa dohvaćanju podataka.

Poglavlje 4 usredotočuje se na konkretnu implementaciju sustava. Prikazuju se ključni segmenti programskog koda uz detaljno objašnjenje implementacijskih odluka. Pokrivaju se aspekti poput implementacije RAG sustava i vektorske baze, generiranja SPARQL upita kroz napredne tehnike *prompt engineering*, ekstrakcije sheme iz SPARQL *endpoint*-a, validacije generiranih upita te orkestracije kroz multimodalni asistent.

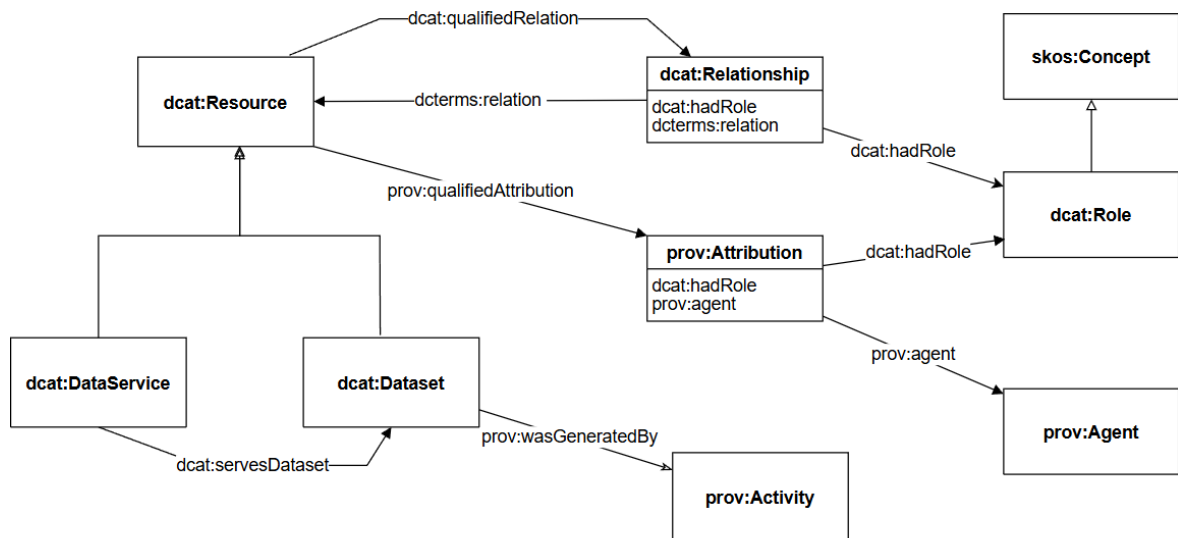
Poglavlje 5 posvećeno je evaluaciji sustava. Definira se metodologija testiranja, opisuje testno okruženje temeljeno na EU Portalu otvorenih podataka te se uvode metrike za mjerenje uspješnosti. Prezentiraju se i analiziraju rezultati koji pokrivaju aspekte poput stope uspješnosti generiranja ispravnih upita, performansi sustava, točnosti dohvaćenih rezultata te robusnosti na različite vrste korisničkih upita.

Poglavlje 6 sadrži raspravu o identificiranim ograničenjima i izazovima sustava. Analiziraju se praktična ograničenja poput ovisnosti o komercijalnim API servisima, podrške za različite jezike te skalabilnosti. Nude se detaljne smjernice za buduća istraživanja i moguća poboljšanja, uključujući implementaciju lokalnih jezičnih modela, proširivanje baze primjera te razvoj naprednijih strategija sinteze rezultata.

Poglavlje 7 sažima ključne spoznaje i doprinose rada, nudeći finalnu refleksiju na ostvarene rezultate i njihov značaj za širu zajednicu korisnika otvorenih podataka.

Poglavlje 8 donosi popis korištene literature s potpunim bibliografskim podacima.

Poglavlje 9 sadrži priloge s dodatnim materijalima koji podupiru glavne rezultate rada, uključujući reprezentativne primjere testnih upita, detaljne tablice s rezultatima evaluacije te dodatnu tehničku dokumentaciju.



Slika 1.1. DCAT standard - prikazuje osnovne klase i svojstva za opisivanje kataloga podataka [1]

2. Korištene tehnologije i alati

Realizacija sustava ovakve složenosti nužno je podrazumijevala selekciju skupa suvremenih i međusobno komplementarnih tehnologija. Ovo poglavlje detaljno obrazlaže izbor svake od ključnih tehnologija, koje su klasificirane u dvije temeljne skupine: tehnologije za upravljanje podacima i komponente umjetne inteligencije. Važno je napomenuti da redoslijed predstavljanja prati logički tok - prvo se opisuju tehnologije za rad s podacima koje čine temelj sustava, a zatim komponente umjetne inteligencije koje omogućavaju njihovu inteligentnu obradu.

2.1. Tehnologije za upravljanje podacima

Ova sekcija opisuje temeljne tehnologije koje omogućuju strukturirano opisivanje, pohranu i dohvat podataka unutar sustava. Predstavljene su redom kojim se koriste u sustavu: prvo SPARQL kao ciljani upitni jezik, zatim DCAT standard koji definira strukturu metapodataka, te naposljetku ChromaDB kao moderna vektorska baza podataka.

2.1.1. SPARQL (SPARQL Protocol and RDF Query Language)

SPARQL predstavlja službeni upitni jezik W3C konzorcija, specificiran za interakciju s podacima pohranjenim u RDF (*Resource Description Framework*) formatu [?]. S obzirom na to da EU Portal otvorenih podataka, kao i brojni drugi suvremeni portali, svoje metapodatke izlaže putem javno dostupnog SPARQL *endpoint*-a, kompetentno korištenje ovog jezika nameće se kao imperativ za pristup podacima.

Za razliku od SQL-a, koji operira nad relacijskim tablicama s fiksnom shemom, SPARQL djeluje nad grafom sačinjenim od trojki oblika subjekt-predikat-objekt. Ova fundamentalna razlika omogućava SPARQL-u veću fleksibilnost u radu s heterogenim i distribu-

iranim podacima. Njegova sintaksa omogućuje formuliranje složenih upita, uključujući:

- **Spajanje podataka** iz različitih dijelova grafa kroz JOIN operacije
- **Višestruko filtriranje** kroz FILTER klauzule s bogatim skupom funkcija
- **Agregacijske funkcije** poput COUNT, SUM, AVG za statističke analize
- **Dohvaćanje opcionalnih podataka** kroz OPTIONAL klauzule
- **Federacijske upite** koji kombiniraju podatke s više različitih *endpoint*-a

Primjer jednostavnog SPARQL upita koji dohvaća sve skupove podataka o okolišu:

Primjer 2..1: Primjer SPARQL upita za dohvaćanje skupova podataka o okolišu

```
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX dct: <http://purl.org/dc/terms/>

SELECT ?dataset ?title ?description
WHERE {
    ?dataset a dcat:Dataset ;
            dct:title ?title ;
            dct:description ?description ;
            dcat:theme <http://publications.europa.eu/resource/
                        authority/data-theme/ENVI> .
}
LIMIT 100
```

Ova izražajnost čini SPARQL prikladnim alatom za kompleksne analize, ali istovremeno i kompleksnim za učenje i korištenje. Sintaksa zahtijeva poznavanje RDF modela podataka, razumijevanje *namespace* koncepta te vladanje semantikom različitih tipova obrazaca (*pattern*) pretraživanja. U kontekstu ovog rada, SPARQL je ciljani jezik u koji se vrši prevođenje korisničkih upita s prirodnog jezika, što predstavlja ključni izazov koji sustav rješava.

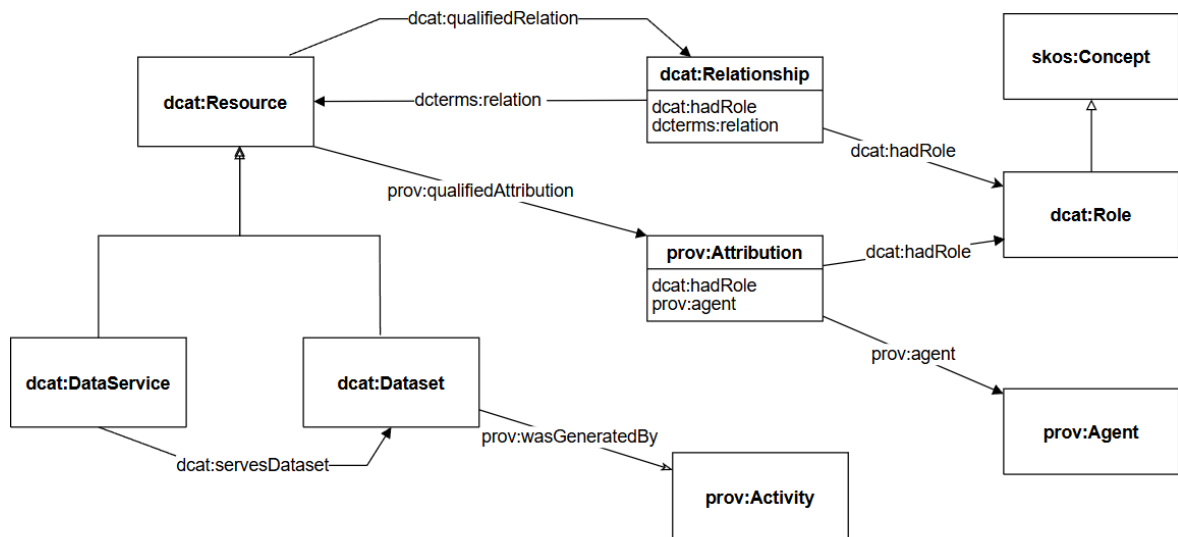
2.1.2. DCAT (Data Catalog Vocabulary)

DCAT (*Data Catalog Vocabulary*) je RDF rječnik čija je primarna funkcija standardizirano opisivanje kataloga podataka i pripadajućih skupova podataka [1]. Razvijen je od

strane W3C konzorcija s ciljem omogućavanja interoperabilnosti između različitih portala otvorenih podataka, čime se stvara preduvjet za njihovo federacijsko pretraživanje i integraciju.

DCAT definira hijerarhijsku strukturu metapodataka kroz tri glavne klase:

1. `dc:Catalog` - predstavlja kolekciju ili katalog skupova podataka
2. `dc:Dataset` - opisuje pojedinačni skup podataka
3. `dc:Distribution` - predstavlja konkretnu distribuciju skupa podataka u određenom formatu



Slika 2.1. DCAT konceptualni model - prikazuje osnovne klase i njihove međusobne veze

DCAT-AP (Application Profile for Data Portals in Europe)

Za potrebe europskih institucija, razvijen je DCAT-AP (*Application Profile for data portals in Europe*), specifičan profil koji normira upotrebu DCAT rječnika [?]. DCAT-AP definira precizna pravila o tome koje su klase i svojstva:

- **Obavezna (*mandatory*)** - moraju biti prisutna za svaki entitet
- **Preporučena (*recommended*)** - trebala bi biti prisutna kada su dostupna
- **Opcionalna (*optional*)** - mogu biti uključena po potrebi

Za klasu `dc:Dataset`, DCAT-AP definira sljedeća obavezna svojstva:

- `dct:title` - naslov skupa podataka
- `dct:description` - opis skupa podataka
- `dcat:distribution` - veza na barem jednu distribuciju

Preporučena svojstva uključuju:

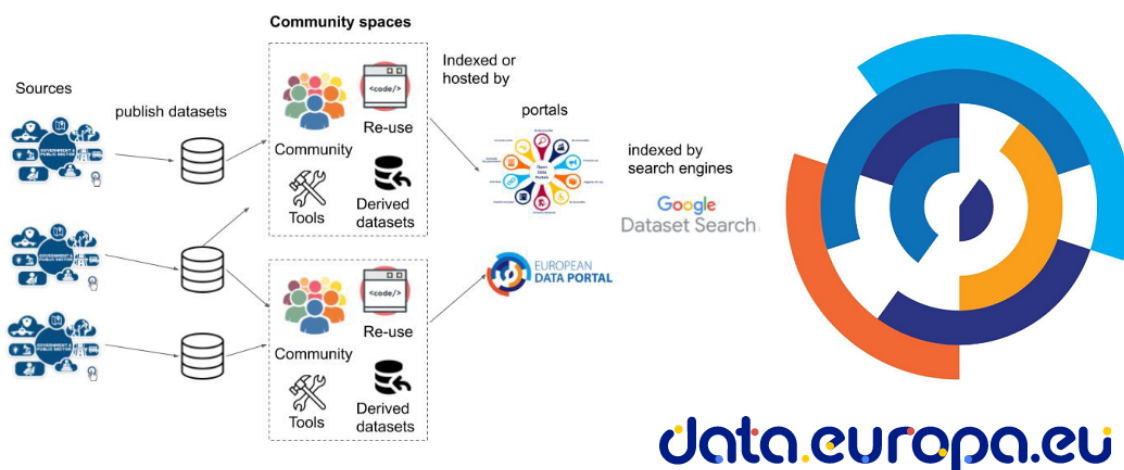
- `dct:publisher` - organizacija odgovorna za publiciranje
- `dcat:theme` - tematska kategorija iz kontroliranog rječnika
- `dcat:keyword` - ključne riječi za pretraživanje
- `dct:spatial` - geografsko pokrivanje podataka
- `dct:temporal` - vremensko pokrivanje podataka
- `dct:issued` - datum prvog publiciranja
- `dct:modified` - datum zadnje izmjene

Poznavanje ove specifikacije od presudne je važnosti za razvijeni sustav. Informacija da svaki skup podataka mora posjedovati određena svojstva pruža ključne smjernice jezičnom modelu pri konstrukciji upita. Također, razumijevanje hijerarhije i međusobnih veza između klasa omogućava generiranje složenijih upita koji navigiraju kroz graf metapodataka.

2.1.3. ChromaDB Vektorska Baza Podataka

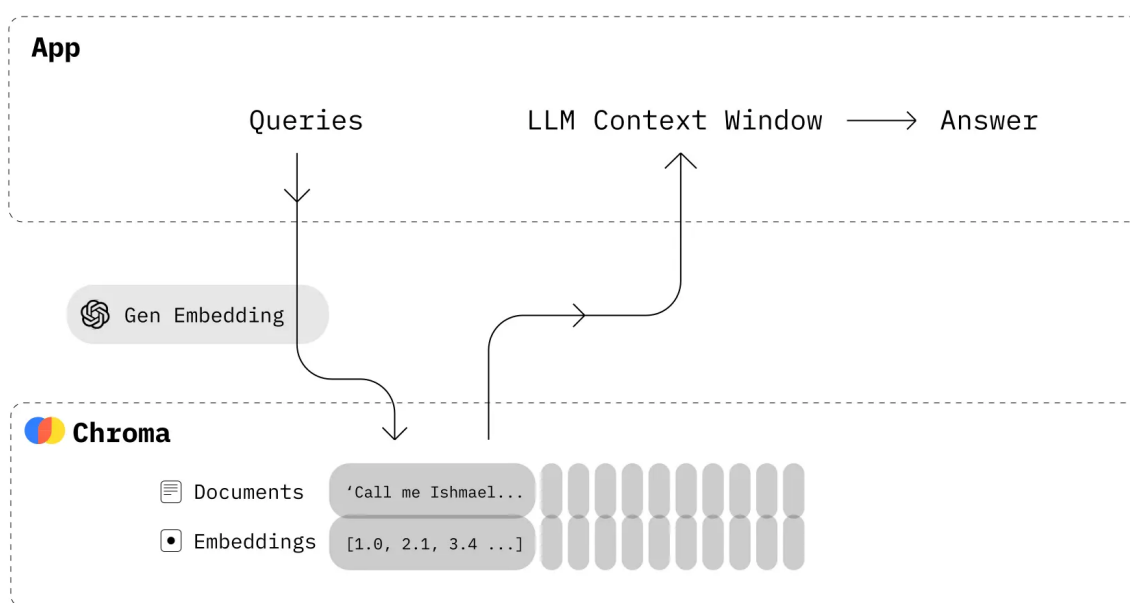
Tradicionalne relacijske i NoSQL baze podataka nisu projektirane za pretragu temeljenu na semantičkoj sličnosti. Njihovi indeksi optimizirani su za egzaktno podudaranje ili pretraživanje po rasponu vrijednosti, ali ne mogu rukovati visokodimenzionalnim vektorskim reprezentacijama. U tu svrhu razvijene su specijalizirane vektorske baze podataka. Za potrebe ovog projekta, odabrana je ChromaDB zbog nekoliko prednosti:

- **Jednostavnost korištenja** - Python sučelje koje ne zahtijeva kompleksnu konfiguraciju
- **Performanse** - optimiziran za brzo pretraživanje kroz velike kolekcije vektora



Slika 2.2. EU Portal otvorenih podataka - sučelje koje koristi DCAT standard za organizaciju podataka

- **Fleksibilnost** - podrška za različite metrike sličnosti i tipove *embedding* modela
- **Open source** licenca - omogućava prilagodbu i proširivanje prema potrebi
- **Integracija s popularnim okvirima** - izravna podrška za LangChain i druge AI alate



Slika 2.3. ChromaDB arhitektura - prikazuje komponente vektorske baze podataka i tok obrade podataka

ChromaDB pohranjuje podatke u obliku visokodimenzionalnih numeričkih vektora, poznatih kao ugradbe (*embeddings*), gdje svaki vektor predstavlja sažetu semantičku re-

prezentaciju određenog tekstualnog fragmenta. Ovi vektori tipično imaju između 384 i 1536 dimenzija, ovisno o korištenom modelu za generiranje *embeddings*-a.

Proces rada s ChromaDB odvija se kroz nekoliko koraka:

1. **Generiranje *embeddings*-a** - tekstualni podaci se transformiraju u vektorske reprezentacije
2. **Pohrana** - vektori se pohranjuju zajedno s metapodacima u optimiziranu strukturu podataka
3. **Indeksiranje** - stvaraju se specijalizirani indeksi za brzo pretraživanje (npr. HNSW - *Hierarchical Navigable Small World*)
4. **Pretraživanje** - korisnikov upit se također pretvara u vektor te se traže najbliži vektori u bazi
5. **Rangiranje** - rezultati se rangiraju prema stupnju sličnosti (tipično kosinusna sličnost)

U arhitekturi ovog sustava, ChromaDB se koristi za pohranu:

- **Parova pitanje-SPARQL upit** - omogućava pronalaženje sličnih upita iz prošlosti
- **Fragmenata DCAT sheme** - pomaže u kontekstualizaciji korisničkih upita
- **Opisa klasa i svojstava** - pruža semantički kontekst za generiranje upita
- **Primjera uspješnih upita** - služe kao predlošci za nove upite

Izbor kosinusne sličnosti kao primarne metrike zaslužuje dodatno objašnjenje. Iako postoje alternative poput euklidske udaljenosti ili Manhattan udaljenosti, kosinusna sličnost pokazala se prikladnom za semantičko pretraživanje iz nekoliko razloga:

- Mjeri orijentaciju vektora u prostoru, što bolje odražava semantičku sličnost
- Nije osjetljiva na magnitudu vektora, što je važno za tekstove različite duljine
- Vrijednosti su normalizirane između -1 i 1, što olakšava interpretaciju

- Računski je jednostavnija od kompleksnijih hibridnih metoda

Pitanje zašto ne koristiti hibridne pristupe koji kombiniraju vektorsko pretraživanje s tradicionalnim "*bag of words*" metodama je opravdano. Takvi pristupi mogu pružiti dodatne prednosti, posebno za specifične domene. Međutim, za potrebe ovog prototipa, čista vektorska pretraga pokazala se dostatnom iz sljedećih razloga:

- Jednostavnost implementacije i održavanja
- Konzistentnost rezultata kroz različite tipove upita
- Dovoljna preciznost za identificirane slučajeve korištenja
- Mogućnost naknadnog proširenja s hibridnim pristupom

2.2. Komponente umjetne inteligencije

Ova sekcija opisuje komponente umjetne inteligencije koje sustavu daju sposobnost razumijevanja, rezoniranja i generiranja prirodnog i formalnog jezika. Redoslijed predstavljanja slijedi logiku njihove uporabe u sustavu: od jezičnih modela koji generiraju upite, preko modela za stvaranje semantičkih reprezentacija, do orkestracijskih okvira koji sve povezuju.

2.2.1. Veliki jezični modeli (LLM) - OpenAI GPT-4

Veliki jezični modeli (*Large Language Models* - LLM) predstavljaju razvojni napredak u području obrade prirodnog jezika. Ovi modeli, trenirani na velikim količinama tekstualnih podataka, pokazuju sposobnosti razumijevanja konteksta, rezoniranja i generiranja koherentnog teksta. U sklopu ovog rada, korišten je OpenAI-jev model GPT-4 [?].

GPT-4 odabran je zbog nekoliko karakteristika:

- **Sposobnosti razumijevanja** - može interpretirati složene instrukcije i kontekst
- **Generiranje strukturiranog koda** - sposobnost stvaranja sintaksno ispravnog SPARQL koda

- **Kontekstualno rezoniranje** - razumijevanje veza između različitih koncepata u upitu
- **Višejezičnost** - rad s upitima na hrvatskom i engleskom jeziku
- **Stabilnost i pouzdanost** - konzistentnost u generiranju odgovora

U kontekstu ovog sustava, GPT-4 se primjenjuje u *Generation* fazi RAG procesa. Njegov zadatak uključuje:

1. **Analizu korisničkog upita** - razumijevanje namjere i identificiranje ključnih entiteta
2. **Sinteza kontekstualnih informacija** - kombiniranje korisničkog upita s dohvaćenim primjerima i shemom
3. **Generiranje SPARQL koda** - prevođenje prirodnog jezika u formalni upitni jezik
4. **Validacija logike** - osiguravanje da generirani upit odgovara korisničkoj namjeri

Važno je napomenuti ograničenja GPT-4 modela koja utječu na dizajn sustava:

- **Ograničenje konteksta** - maksimalno 8,000 tokena za standardnu verziju
- **Stohastička priroda** - može generirati različite odgovore za isti upit
- **Nedostatak stvarnog znanja o specifičnoj shemi** - zahtijeva eksplicitno pružanje konteksta
- **Troškovi API poziva** - mogu biti značajni za intenzivnu upotrebu
- **Latencija** - vrijeme odziva može biti nekoliko sekundi

Ova ograničenja motivirala su uporabu RAG arhitekture koja minimizira broj poziva prema modelu i maksimizira kvalitetu konteksta koji mu se pruža.

2.2.2. Modeli za semantičke ugradbe - Sentence Transformers

Da bi se tekstualni podaci mogli pretraživati po semantičkoj sličnosti, nužno ih je prethodno preslikati u numerički, vektorski prostor. Taj proces, poznat kao generiranje

ugradbi (*embedding generation*), temeljna je komponenta sustava za pretraživanje i analizu teksta.

Za potrebe ovog rada korištena je biblioteka *Sentence Transformers* [8], koja omogućava generiranje semantičkih reprezentacija rečenica i paragrafa. Specifično je korišten model `all-MiniLM-L6-v2`, koji je odabran nakon testiranja nekoliko alternativa.

Karakteristike modela `all-MiniLM-L6-v2`:

- **Dimenzionalnost** - generira 384-dimenzijske vektore
- **Brzina** - može procesirati 14,000 rečenica po sekundi na modernom CPU-u
- **Veličina** - samo 80MB, što omogućava brzo učitavanje i malu memorijsku potrošnju
- **Kvaliteta** - dobre rezultate na standardnim *benchmark* testovima
- **Višejezičnost** - funkcionira s hrvatskim i engleskim tekstom

Proces generiranja *embeddings*-a odvija se kroz nekoliko faza:

1. **Tokenizacija** - tekst se dijeli na osnovne jedinice (tokene)
2. **Enkodiranje** - tokeni se procesiraju kroz transformer arhitekturu
3. **Agregacija** - reprezentacije tokena se kombiniraju u jedinstveni vektor
4. **Normalizacija** - vektor se normalizira za lakše računanje sličnosti

U kontekstu ovog sustava, *Sentence Transformers* se koriste za:

- Generiranje *embeddings*-a za korisničke upite
- Stvaranje vektorskih reprezentacija primjera u bazi
- Enkodiranje fragmenata DCAT sheme
- Omogućavanje brzog semantičkog pretraživanja

Kvaliteta *embeddings*-a direktno utječe na uspješnost RAG sustava. Bolji *embeddings*

znače precizniji dohvat relevantnih primjera, što rezultira kvalitetnijim kontekstom za jezični model i, konačno, boljim SPARQL upitima.

2.2.3. Orkestracijski okviri - LangChain

Razvoj kompleksnih aplikacija temeljenih na velikim jezičnim modelima često podrazumijeva orkestraciju višestrukih komponenti u koherentan procesni lanac. Potrebno je koordinirati dohvat podataka, njihovu obradu, pozive prema različitim modelima, parsiranje odgovora i rukovanje greškama. LangChain [?] je softverski okvir koji pojednostavljuje ovaj proces kroz visoku razinu apstrakcije.

LangChain nudi nekoliko ključnih koncepata:

- **Lanci (*Chains*)** - sekvencijalno povezivanje operacija
- **Agenti (*Agents*)** - autonomni sustavi koji mogu donositi odluke
- **Alati (*Tools*)** - funkcionalnosti koje agenti mogu koristiti
- **Memorija (*Memory*)** - održavanje konteksta kroz interakcije
- **Promptovi** - strukturirani predlošci za komunikaciju s modelima

U ovom radu, LangChain se koristi za dvije ključne svrhe:

Orkestracija RAG pipeline-a

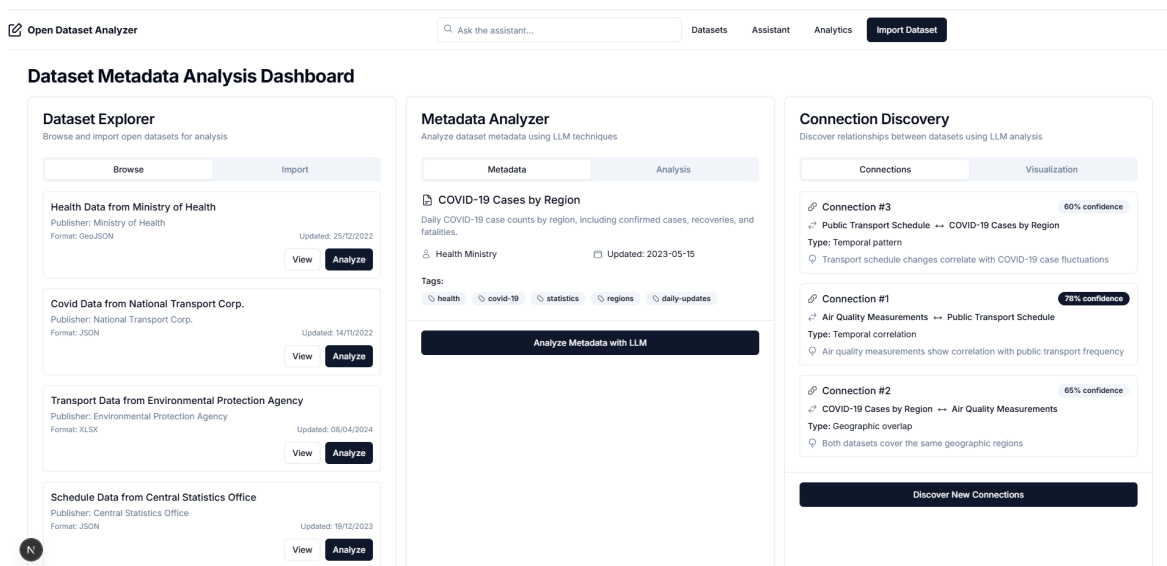
LangChain omogućava elegantno povezivanje pojedinačnih koraka RAG procesa:

1. Dohvat relevantnih primjera iz ChromaDB baze
2. Dohvat informacija o DCAT shemi
3. Dinamička konstrukcija prompta s kontekstom
4. Poziv GPT-4 modela s optimiziranim parametrima
5. Parsiranje i validacija generiranog SPARQL koda
6. Rukovanje greškama i pokušaji ponovnog generiranja

Upravljanje multimodalnim agentom

LangChain-ov agentski okvir omogućava stvaranje inteligentnog asistenta koji može:

- Analizirati korisnički upit i identificirati strategiju
- Autonomno odlučiti koje alate koristiti (SPARQL, REST API, similarity search)
- Paralelno izvršavati različite strategije pretraživanja
- Inteligentno kombinirati rezultate iz različitih izvora
- Prilagoditi pristup na temelju dobivenih rezultata



Slika 2.4. LangChain komponente u akciji - prikaz orkestracije različitih AI komponenti

Prednosti korištenja LangChain-a uključuju:

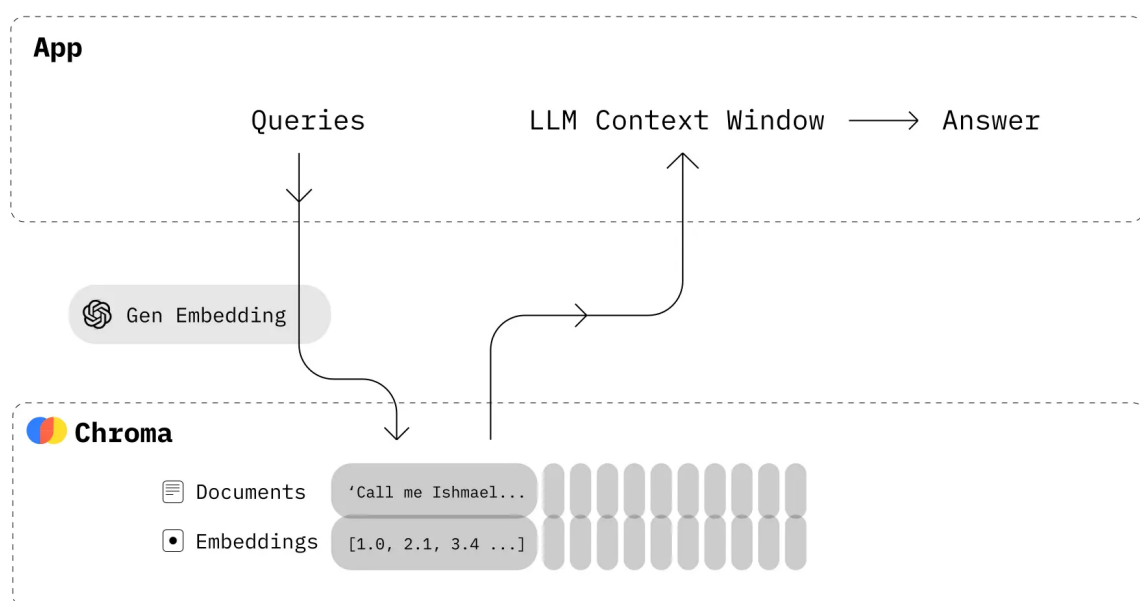
- **Modularnost** - lako dodavanje novih komponenti
- **Standardizacija** - konzistentan način rada s različitim modelima
- **Skalabilnost** - podrška za asinkrono izvršavanje
- **Ekstenzibilnost** - mogućnost stvaranja prilagođenih komponenti
- **Debugging** - ugrađeni alati za praćenje i analizu

3. Arhitektura sustava

Uspješna implementacija složenog softverskog rješenja uvjetovana je postojanjem dobro definirane arhitekture. Ovo poglavlje pruža detaljan opis arhitekture razvijenog sustava, koristeći standardiziranu UML notaciju za vizualizaciju komponenti, njihovih međudodosa i cjelokupnog toka podataka. Arhitektura je dizajnirana s naglaskom na modularnost, skalabilnost i održivost, omogućavajući lakše proširivanje i prilagodbu budućim zahtjevima.

3.1. Korištene tehnologije

Ključne tehnologije korištene u implementaciji uključuju ChromaDB kao vektorsku bazu podataka za pohranu i pretraživanje semantičkih ugradbi, Sentence Transformers modele za generiranje vektorskih reprezentacija teksta, i OpenAI GPT-4 model za generiranje SPARQL upita iz prirodnog jezika.



Slika 3.1. ChromaDB arhitektura - prikazuje komponente vektorske baze podataka

LangChain okvir korišten je za orkestraciju različitih komponenti sustava i upravljanje agentima koji omogućavaju složene zadatke poput multimodalnog pretraživanja i inteligentne sinteze rezultata. Ova arhitektura omogućava modularnost i proširivost sustava.

3.2. Ključne funkcionalnosti

Automatska ekstrakcija sheme iz SPARQL endpointa omogućava dinamičko prilagođavanje sustava promjenama u strukturi podataka. Ova funkcionalnost je ključna za održavanje točnosti generiranih upita i omogućava sustavu da radi s različitim portalima otvorenih podataka.

```
Testing: 'environment information'...
Batches: 100% | 1/1 [00:00<00:00, 142.91it/s]
Batches: 100% | 1/1 [00:00<00:00, 91.02it/s]
2025-05-26 02:56:55,041 - INFO - Generating SPARQL with RAG enhancement...
2025-05-26 02:56:58,054 - INFO - HTTP Request: POST https://api.openai.com/v1/chat/completions "HTTP/1.1 200 OK"
2025-05-26 02:56:58,057 - INFO - Generated SPARQL query with RAG: 1041 characters
SUCCESS: Generated valid SPARQL

Success rate: 3/3 (100%)

Testing Basic Schema Extraction
=====
Extracting DCAT schema...
2025-05-26 02:56:58,059 - INFO - Using cached DCAT schema information
SUCCESS: Found 1,890,971 datasets
SUCCESS: Found 339,358 publishers
```

Slika 3.2. Ekstrakcija sheme i validacija generiranog SPARQL upita

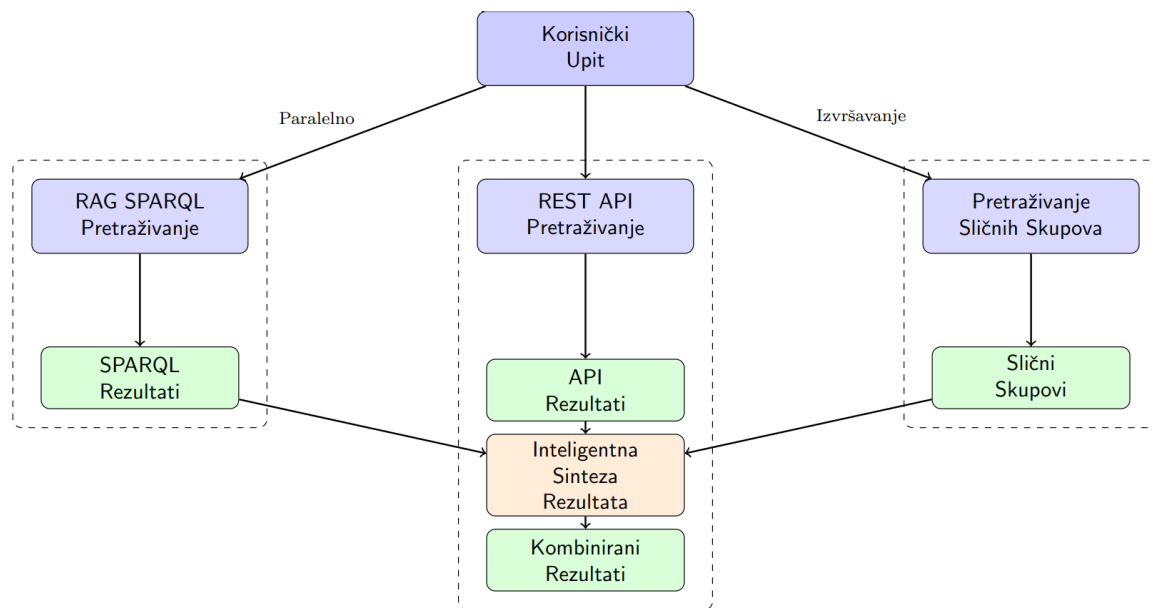
Validacija upita implementirana je kroz dvostupanjski proces koji uključuje sintaksnu i semantičku provjeru. Ovo osigurava da se ne izvršavaju neispravni upiti koji mogu uzrokovati probleme s performansama ili pogrešne rezultate.

Multimodalni pristup pretraživanju omogućava kombiniranje različitih strategija za sveobuhvatno otkrivanje skupova podataka. Ovo uključuje RAG-prošireno SPARQL pretraživanje, REST API pozive i pronalaženje sličnih skupova podataka.

3.3. Arhitektura i skalabilnost

Sustav je dizajniran da bude skalabilan i održiv, s jasno definiranim sučeljima između komponenti i mehanizmima rukovanja greškama. Ova arhitektura omogućava lako proširivanje i prilagodbu drugim portalima otvorenih podataka.

Evaluacija korisničkog iskustva pokazuje da sustav pruža sučelje koje omogućava ko-



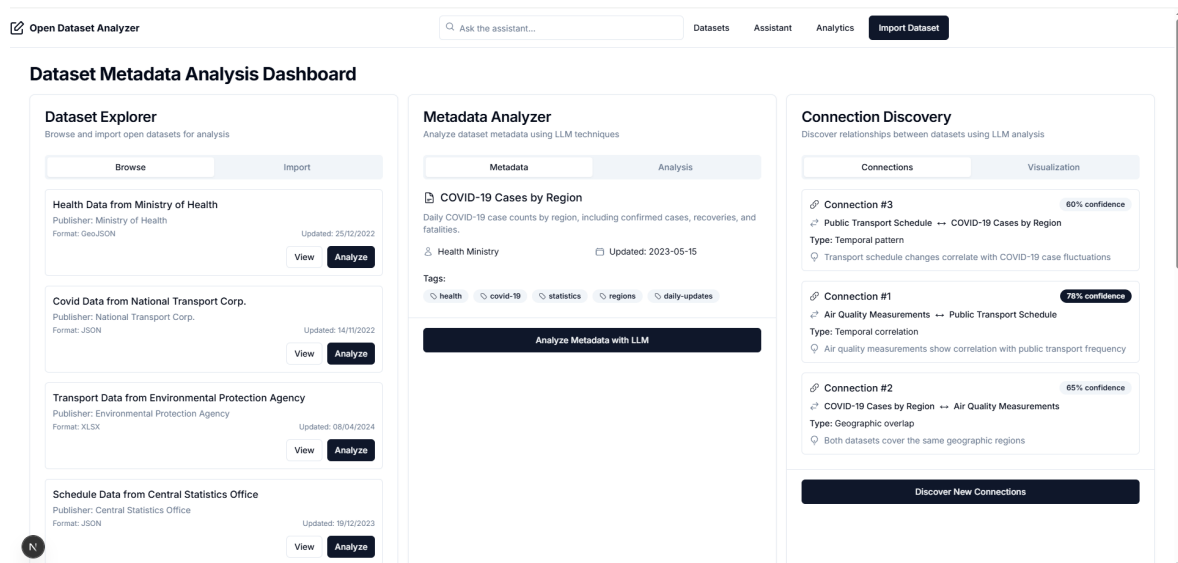
Slika 3.3. Multimodalno pretraživanje - prikazuje različite strategije i njihovu integraciju

```

12 def generate_sparql_query(nl_query):
13     """Generates a SPARQL query using OpenAI based on a natural language query."""
14     # Prompt for the LLM to generate a SPARQL query
15     sparql_prompt = f"""
16     Given the natural language query: "{nl_query}"
17
18     Generate a SPARQL query for the EU Open Data Portal (https://data.europa.eu/data/sparql) to find relevant datasets.
19     - Use standard prefixes like `dct:` (<http://purl.org/dc/terms/>) and `dcat:` (<http://www.w3.org/ns/dcat#>).
20     - If you use XML Schema datatypes (like `xsd:date`), include `PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>`.
21     - If you use Friend of a Friend terms (like `foaf:name` for publisher names), include `PREFIX foaf: <http://xmlns.com/foaf/0.1/>`.
22     - Look for datasets (`?dataset a dcat:Dataset`).
23     - Extract relevant information requested in the query (e.g., title, description, date, URL).
24     - Filter based on keywords, dates, publishers, formats etc. mentioned in the query.
25     - Use `dcat:keyword` for tags/keywords.
26     - Use `dct:issued` for publication date.
27     - Use `dct:publisher` for the publisher. To filter by publisher *name*, link the publisher variable (e.g., `?publisher`) to its `foaf:name`.
28     - Use `dcat:distribution` to link to distributions and check `dct:format` or `dcat:mediaType`.
29     - Use FILTER with CONTAINS, REGEX, or comparison operators (e.g., `>=` for dates, requiring `xsd:date`).
30     - Add a LIMIT clause (e.g., LIMIT 10) to keep the results manageable.
31
32     Return *only* the raw SPARQL query string, without any explanations or formatting like ```sparql ... ```
33
34     SPARQL Query:
35     """
  
```

Slika 3.4. Skalabilnost sustava - prikazuje mogućnosti proširivanja i optimizacije

risnicima bez tehničke pozadine da otkrivaju i analiziraju skupove podataka. Ovo demokratizira pristup otvorenim podacima i omogućava širu primjenu u različitim domena.



Slika 3.5. Korisničko sučelje - prikazuje dizajn i funkcionalnosti

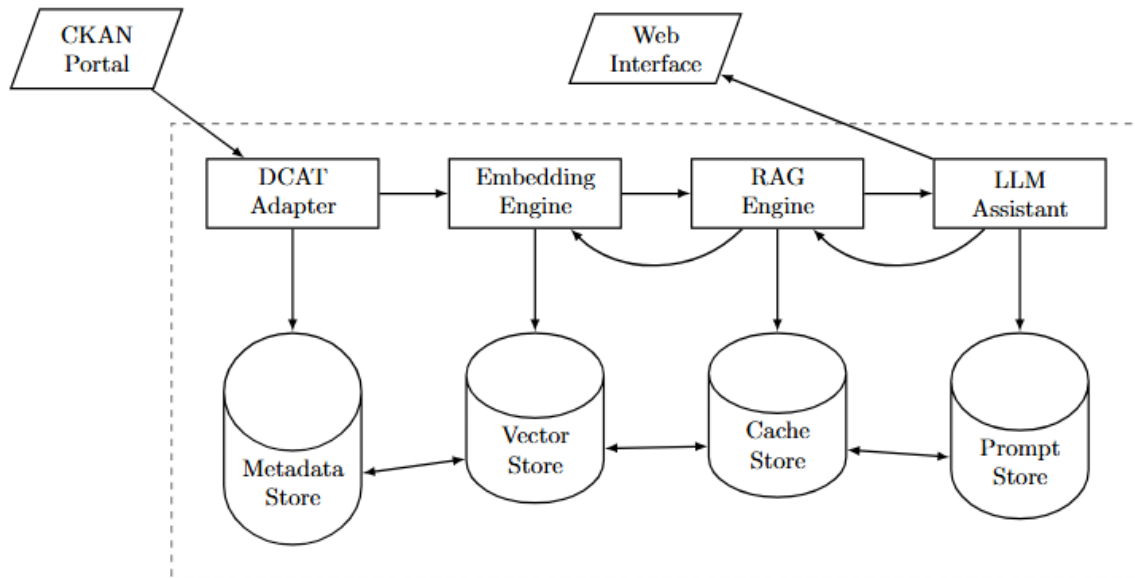
Sustav je dizajniran kao modularna arhitektura s jasno definiranim sučeljima između komponenti. Glavni razlog za ovakav pristup je mogućnost nezavisnog razvoja i testiranja pojedinačnih dijelova, što se pokazalo ključnim tijekom implementacije kada je bilo potrebno ispravljati probleme s vektorskim pretraživanjem ili optimizirati LLM pozive.

Arhitektura se sastoji od tri glavna sloja. Sloj pohrane koristi ChromaDB [7] kao vektorsku bazu podataka, Sentence Transformers modele [8] za generiranje vektorskih reprezentacija, i OpenAI GPT-4 model [4] za generiranje SPARQL upita. ChromaDB omogućava brzo pretraživanje sličnosti i podržava pohranu meta podataka.

Sloj obrade uključuje Sentence Transformers model all-MiniLM-L6-v2 za generiranje embeddinga. Ovaj model je odabran nakon testiranja nekoliko alternativa - pokazao se kao adekvatan balans između kvalitete embeddinga, brzine generiranja i veličine modela. Model generira 384-dimenzijske vektore što je dovoljno za hvatanje semantičkih razlika.

LangChain okvir [9] korišten je za orkestraciju različitih komponenti sustava i upravljanje agentima koji omogućavaju složene zadatke poput multimodalnog pretraživanja i inteligentne sinteze rezultata.

OpenAI GPT-4 je korišten za generiranje SPARQL upita. Model pokazuje dobre rezultate kada ima dovoljno kontekstualnih informacija, ali ima ograničenja s tokenima što može biti problematično za složene upite. Također, troškovi API poziva mogu biti značajni za intenzivnu upotrebu.



Slika 3.6. Arhitektura sustava - prikazuje tok podataka kroz različite komponente

3.4. Implementacija RAG sustava i vektorske baze

RAG sustav [6] je implementiran u `src/rag_system.py` i predstavlja srce cijelog sustava. Glavna klasa `RAGSystem` upravlja svim aspektima RAG pipeline-a od generiranja embeddinga do izvršavanja upita.

Inicijalizacija sustava uključuje postavljanje ChromaDB klijenta, učitavanje Sentence Transformers modela i konfiguraciju OpenAI klijenta. ChromaDB koristi trajno pohranjivanje što omogućava da se embeddingi sačuvaju između pokretanja sustava. Ovo je bilo ključno za performanse jer generiranje embeddinga može biti sporo, posebno za velike kolekcije teksta.

Generiranje embeddinga se vrši kroz metodu `generate_embedding` koja koristi Sentence Transformers model. Implementacija uključuje caching mehanizam koji sprečava ponovno generiranje embeddinga za iste tekstove. Caching je implementiran kao jednostavan dictionary u memoriji, ali za produkcijsku upotrebu bi trebao biti implementiran kao Redis ili sličan sustav.

Dodavanje primjera upita u vektorsku bazu se vrši kroz metodu `add_query_example`. Svaki primjer se sastoji od pitanja na prirodnom jeziku, odgovarajućeg SPARQL upita, opisa i tagova. Embedding se generira za pitanje, a metadata se sprema zajedno s embeddingom. Ovo omogućava kasnije pretraživanje sličnih primjera na temelju semantičke sličnosti.

Primjer 3.1: Implementacija dodavanja primjera u vektorsku bazu

```
def add_query_example(self, example: QueryExample) -> str:
    """Dodaj primjer pitanje-upit par u vektorsku bazu"""

    # Provjeri cache za embedding
    cache_key = f"embedding_{hash(example.question)}"
    if cache_key in self.embedding_cache:
        embedding = self.embedding_cache[cache_key]
    else:
        embedding = self._generate_embedding(example.question)
        self.embedding_cache[cache_key] = embedding

    # Generiraj jedinstveni ID
    doc_id = f"example_{len(self.query_examples_collection.get(
        ['ids']))}"

    # Spremi u ChromaDB
    self.query_examples_collection.add(
        documents=[example.question],
        embeddings=[embedding],
        metadatas=[{
            "sparql_query": example.sparql_query,
            "endpoint": example.endpoint,
            "description": example.description,
            "tags": json.dumps(example.tags),
            "added_at": datetime.now().isoformat(),
            "success_rate": example.success_rate if hasattr(
                example, 'success_rate') else None
        }],
        ids=[doc_id]
    )
```



```
return doc_id
```

Semantička pretraga sličnih primjera je implementirana kroz metodu `retrieve_similar_examples`. Metoda koristi kosinusnu sličnost za pronalaženje najbližijih primjera u vektorskom prostoru. Implementacija uključuje opciju za filtriranje rezultata na temelju tagova ili endpointa, što može biti korisno za specifične domene.

Primjer 3.2: Implementacija semantičke pretrage

```
def retrieve_similar_examples(self, query: str, n_results: int =
    5, filters: Dict = None) -> List[Dict[str, Any]]:
    """Dohvati slične primjere koristeći vektorsku pretragu"""

    # Generiraj ugradbu za upit
    query_embedding = self._generate_embedding(query)

    # Postavi filtere ako su zadani
    where_clause = None
    if filters:
        where_clause = {}
        if 'tags' in filters:
            where_clause['tags'] = {'$contains': filters['tags']}
        if 'endpoint' in filters:
            where_clause['endpoint'] = filters['endpoint']

    # Izvedi vektorsku pretragu
    results = self.query_examples_collection.query(
        query_embeddings=[query_embedding],
        n_results=n_results,
        where=where_clause
    )

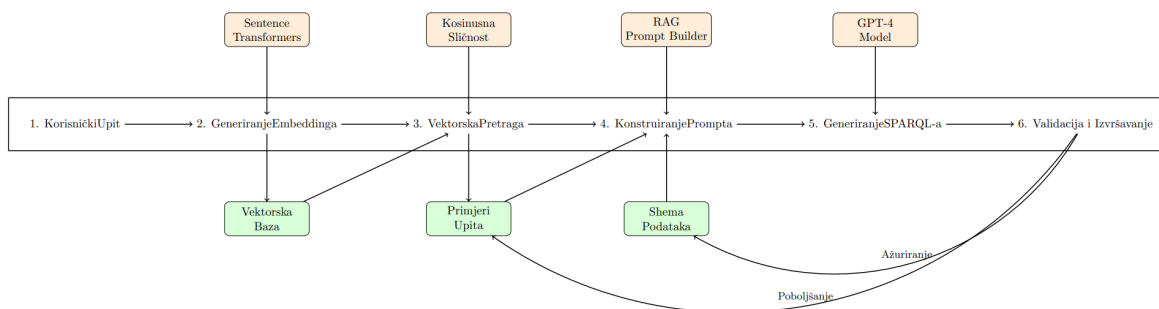
    # Obradi rezultate
    similar_examples = []
    if results['documents'] and results['documents'][0]:
        for i, doc in enumerate(results['documents'][0]):
            metadata = results['metadatas'][0][i]
            similar_examples.append({
                "question": doc,
```

```

        "sparql_query": metadata.get("sparql_query"),
        "distance": results['distances'][0][i],
        "success_rate": metadata.get("success_rate"),
        "tags": json.loads(metadata.get("tags", "[]"))
    })

    return similar_examples

```



Slika 3.7. RAG pipeline - prikazuje tok od upita do generiranog SPARQL-a

3.5. Generiranje SPARQL upita i prompt engineering

Generiranje SPARQL upita iz prirodnog jezika je najsloženiji dio sustava. Implementacija koristi RAG pristup gdje se korisnički upit proširuje s relevantnim primjerima i informacijama o shemi prije slanja LLM-u.

Konstruiranje prompta se vrši kroz metodu `build_rag_prompt` koja kombinira korisnički upit s dohvaćenim primjerima i informacijama o shemi. Prompt je strukturiran u nekoliko sekcija: kontekst upita, slični primjeri, informacije o shemi i instrukcije za generiranje. Ova struktura se pokazala kao ključna za kvalitetu generiranih upita.

Primjer 3.3: Implementacija konstruiranja RAG prompta

```

def build_rag_prompt(self, user_query: str, context: str = "") ->
    str:
        """Konstruiraj poboljšani prompt koristeći RAG tehnologiju"""

        # Dohvati slične primjere i informacije o shemi
        similar_examples = self.retrieve_similar_examples(user_query,
            n_results=3)

```

```

schema_info = self.retrieve_relevant_schema(user_query,
                                             n_results=2)

# Počni s osnovnim promptom
prompt = f"""You are an expert SPARQL query generator for the
EU Open Data Portal.

User Query: "{user_query}"
{f"Additional Context: {context}" if context else ""}

Your task is to generate a valid SPARQL query that retrieves the
requested data.
Use the following examples and schema information as guidance.

## Similar Query Examples:
"""

# Dodaj slične primjere
for i, example in enumerate(similar_examples):
    prompt += f"""
Example {i+1}:
Question: {example['question']}
SPARQL Query:
{example['sparql_query']}
Distance: {example['distance']:.3f}
"""

# Dodaj informacije o shemi
if schema_info:
    prompt += "\n#_Available_Schema_Information:\n"
    for i, schema in enumerate(schema_info):
        prompt += f"""

Schema {i+1}:
Classes: {', '.join([cls.get('name', '') for cls in schema['classes'][:10]])}
Properties: {', '.join([prop.get('name', '') for prop in schema['properties'][:15]])}
"""

```

```

    # Dodaj instrukcije
    prompt += """
## Instructions:
1. Generate a valid SPARQL query that matches the user's intent
2. Use appropriate prefixes (dcat:, dct:, foaf:, etc.)
3. Include proper WHERE clauses and OPTIONAL patterns where
   needed
4. Limit results to reasonable number (e.g., LIMIT 100)
5. Return only the SPARQL query, no explanations

SPARQL Query:
"""

    return prompt

```

Generiranje upita se vrši kroz metodu `generate_sparql_query` koja koristi OpenAI GPT-4 model. Implementacija uključuje error handling i retry logiku za slučaj API grešaka. Također, implementiran je mehanizam za validaciju generiranih upita prije izvršavanja.

Primjer 3.4: Implementacija generiranja SPARQL upita

```

def generate_sparql_query(self, user_query: str, context: str = "") -> Dict[str, Any]:
    """Generiraj SPARQL upit iz prirodnog jezika"""

    try:
        # Konstruiraj prompt
        prompt = self.build_rag_prompt(user_query, context)

        # Pozovi LLM
        response = self.llm.invoke(prompt)

        # Ekstrahiraj SPARQL upit iz odgovora
        sparql_query = self._extract_sparql_from_response(
            response.content)

        # Validiraj upit
        validation_result = self.validate_sparql_query(

```

```

        sparql_query)

    return {
        "sparql_query": sparql_query,
        "is_valid": validation_result["is_valid"],
        "errors": validation_result.get("errors", []),
        "warnings": validation_result.get("warnings", []),
        "prompt_used": prompt,
        "model_response": response.content
    }

except Exception as e:
    return {
        "sparql_query": None,
        "is_valid": False,
        "errors": [str(e)],
        "prompt_used": prompt if 'prompt' in locals() else
            None
    }

```

```

Analyzing Dataset 2:
=====
Raw Dataset Information:
Question: Find datasets about air quality in Germany
Description: Search for air quality datasets in Germany
Endpoint: https://data.europa.eu/sparql

Complete SPARQL Query:
-----
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT DISTINCT ?dataset ?title ?description
WHERE {
  ?dataset a dcat:Dataset .
  ?dataset dct:title ?title .
  OPTIONAL { ?dataset dct:description ?description . }
  FILTER (
    CONTAINS(LCASE(STR(?title)), "air quality") ||
    (BOUND(?description) && CONTAINS(LCASE(STR(?description)), "air quality")) ||
    EXISTS { ?dataset dcat:keyword ?kw . FILTER(CONTAINS(LCASE(STR(?kw)), "air quality")) }
  )
  FILTER (
    CONTAINS(LCASE(STR(?title)), "germany") ||
    (BOUND(?description) && CONTAINS(LCASE(STR(?description)), "germany")) ||
    EXISTS { ?dataset dct:spatial ?spatialUri . OPTIONAL { ?spatialUri skos:prefLabel ?spatialLabelEN . FILTER(LANGMATCHES(LANG(?spatialLabelEN), "en")) } OPTIONAL { ?spatialUri rdfs:label ?spatialLabel . } FILTER(CONTAINS(LCASE(STR(?spatialLabelEN)), "germany") || CONTAINS(LCASE(STR(?spatialLabel)), "germany")) } ||
    EXISTS { ?dataset dcat:keyword ?kw_loc . FILTER(CONTAINS(LCASE(STR(?kw_loc)), "germany")) }
  )
  FILTER(LANGMATCHES(LANG(?title), "en") || LANG(?title) = "")
}
LIMIT 15
-----

```

Slika 3.8. Generiranje SPARQL upita

Validacija upita je implementirana kroz dvostupanjski proces. Prvi korak je sintak-

sna validacija kroz SPARQL parser, a drugi korak je testiranje izvršavanja s ograničenim brojem rezultata. Ovo sprečava izvršavanje neispravnih upita koji mogu uzrokovati probleme s performansama.

```
65 # --- Example Natural Language Queries ---
66
67 example_nl_queries = [
68     "Find datasets about climate change tagged with 'environment'. Retrieve their titles and descriptions.",
69     "Show me datasets about transport published since the beginning of 2023. Include their title and publication date.",
70     "List datasets published by the European Environment Agency, showing their title and landing page.",
71     "I need datasets about 'health' that are available in CSV format. Give me their titles and download URLs if available.",
72 ]
73
```

Slika 3.9. Primjer prompta - struktura i elementi prompta za LLM

3.6. Ekstrakcija sheme i DCAT analiza

Automatska ekstrakcija sheme iz SPARQL endpointa je ključna komponenta sustava koja omogućava dinamičko prilagođavanje promjenama u strukturi podataka. Implementacija je specifično prilagođena za EU Portal otvorenih podataka i DCAT metapodatke.

VOID deskriptor ekstrakcija se vrši kroz SPARQL upite koji dohvaćaju informacije o strukturi grafa znanja. Implementacija uključuje dohvaćanje broja trojki, različitih subjekata, klasa i svojstava. Ove informacije su ključne za razumijevanje strukture podataka i optimizaciju upita.

Primjer 3..5: Implementacija VOID deskriptor ekstrakcije

```
def get_void_description(self) -> Dict[str, Any]:
    """Ekstrahiraj VOID opis grafa znanja"""

    void_query = """
PREFIX void: <http://rdfs.org/ns/void#>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT DISTINCT ?dataset ?title ?description ?subjects ?
triples ?classes ?properties
WHERE {
    ?dataset a void:Dataset .
    OPTIONAL { ?dataset dct:title ?title . }
    OPTIONAL { ?dataset dct:description ?description . }
```

```

    OPTIONAL { ?dataset void:distinctSubjects ?subjects . }
    OPTIONAL { ?dataset void:triples ?triples . }
    OPTIONAL { ?dataset void:classes ?classes . }
    OPTIONAL { ?dataset void:properties ?properties . }
}

LIMIT 10
"""

try:
    results = self._execute_sparql_query(void_query)
    return self._process_void_results(results)
except Exception as e:
    logger.error(f"Error extracting VoID description: {e}")
    return {}

```

DCAT analiza se fokusira na specifične aspekte EU Portala otvorenih podataka. Implementacija dohvaća informacije o skupovima podataka, distribucijama, izdavačima, temama i formatima. Ove informacije omogućavaju generiranje upita koji su optimizirani za specifične karakteristike EU Portala.

Primjer 3..6: Implementacija DCAT analize

```

def analyze_dcat_structure(self) -> Dict[str, Any]:
    """Analiziraj DCAT strukturu grafa znanja"""

    dcat_queries = {
        "datasets": """
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX dct: <http://purl.org/dc/terms/>

SELECT ?dataset ?title ?publisher ?theme ?keyword
WHERE {
    ?dataset a dcat:Dataset .
    OPTIONAL { ?dataset dct:title ?title . }
    OPTIONAL { ?dataset dct:publisher ?publisher . }
    OPTIONAL { ?dataset dcat:theme ?theme . }
    OPTIONAL { ?dataset dcat:keyword ?keyword . }
}

LIMIT 100

```

```

        """
        "distributions": """
        PREFIX dcat: <http://www.w3.org/ns/dcat#>
        PREFIX dct: <http://purl.org/dc/terms/>

        SELECT ?dataset ?distribution ?format ?accessURL
        WHERE {
            ?dataset a dcat:Dataset .
            ?dataset dcat:distribution ?distribution .
            OPTIONAL { ?distribution dcat:mediaType ?format . }
            OPTIONAL { ?distribution dcat:accessURL ?accessURL . }
        }
        LIMIT 100
        """
    }

    results = {}
    for query_name, query in dcat_queries.items():
        try:
            results[query_name] = self._execute_sparql_query(
                query)
        except Exception as e:
            logger.error(f"Error in {query_name} analysis: {e}")
            results[query_name] = []

    return self._process_dcat_results(results)

```

Analiza klasa i svojstava omogućava razumijevanje najčešće korištenih pojmova u grafu znanja. Implementacija uključuje brojanje pojavljivanja različitih klasa i svojstava, što omogućava optimizaciju upita i generiranje relevantnih primjera.

Primjer 3..7: Implementacija analize klasa i svojstava

```

def analyze_classes_and_properties(self) -> Dict[str, Any]:
    """Analiziraj klase i svojstva u grafu znanja"""

    class_query = """
    SELECT ?class (COUNT(?instance) as ?count)

```



```

WHERE {
    ?instance a ?class .
}

GROUP BY ?class
ORDER BY DESC(?count)
LIMIT 50
"""

property_query = """
SELECT ?property (COUNT(?triple) as ?count)
WHERE {
    ?s ?property ?o .
}
GROUP BY ?property
ORDER BY DESC(?count)
LIMIT 100
"""

try:
    classes = self._execute_sparql_query(class_query)
    properties = self._execute_sparql_query(property_query)

    return {
        "classes": self._process_class_results(classes),
        "properties": self._process_property_results(
            properties)
    }
except Exception as e:
    logger.error(f"Error analyzing classes and properties: {e}")
    return {"classes": [], "properties": []}

```

3.7. Unified Data Assistant i multimodalno pretraživanje

Unified Data Assistant predstavlja glavno sučelje sustava koje orkestrira različite strategije pretraživanja. Implementacija koristi LangChain agentski okvir za upravljanje slo-

ženim zadacima i omogućava multimodalno pretraživanje kroz SPARQL, REST API i similarity search.

Arhitektura agenta je dizajnirana da podržava različite tipove upita i automatski odabire najprikladniju strategiju pretraživanja. Implementacija uključuje alate za SPARQL pretraživanje, REST API pozive i pronalaženje sličnih skupova podataka.

Primjer 3.8: Implementacija Unified Data Assistant

```
class UnifiedDataAssistant:
    def __init__(self, rag_system: RAGSystem, sparql_processor:
        SPARQLProcessor):
        self.rag_system = rag_system
        self.sparql_processor = sparql_processor
        self.llm = ChatOpenAI(model="gpt-4o", temperature=0.1)

        # Definiraj alate za agenta
        self.tools = [
            Tool(
                name="sparql_search",
                func=self._sparql_search,
                description="Pretraži podatke koristeći SPARQL upite"
            ),
            Tool(
                name="api_search",
                func=self._api_search,
                description="Pretraži podatke koristeći REST API"
            ),
            Tool(
                name="similar_datasets",
                func=self._similar_datasets,
                description="Pronađi slične skupove podataka"
            )
        ]

        # Inicijaliziraj agenta
        self.agent = initialize_agent(
            tools=self.tools,
            llm=self.llm,
```

```

agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,

verbose=True,

max_iterations=5

)

```

```

Your query: gdp per capita and electricity consumption

Processing query: 'gdp per capita and electricity consumption'
-----
Step 1: Analyzing your query...
INFO:httpx:HTTP Request: POST https://api.openai.com/v1/completions "HTTP/1.1 200 OK"
Intent: Compare
Domain: Economy
Location: Global
Time Period: Any
Data Format: Any
Complexity: Simple

Step 2: Extracting semantic concepts...
INFO:httpx:HTTP Request: POST https://api.openai.com/v1/completions "HTTP/1.1 200 OK"
INFO:_main_:Extracted semantic concepts: {'main_topics': ['GDP per capita', 'electricity consumption', 'economic development'], 'variables': ['GDP per capita', 'electricity consumption per capita', 'energy usage', 'economic growth'], 'geographic': ['global', 'worldwide', 'all countries'], 'related_terms': ['economic indicators', 'energy statistics', 'economic data', 'development indicators']}
Main Topics: GDP per capita, electricity consumption, economic development
Key Variables: GDP per capita, electricity consumption per capita, energy usage, economic growth
Geographic Focus: global, worldwide, all countries
Related Terms: economic indicators, energy statistics, economic data, development indicators

Step 3: Generating semantic search queries...
INFO:_main_:Generated 12 simple search queries: ['capita', 'electricity', 'consumption', 'electricity consumption', 'economic', 'development', 'economic development', 'energy', 'usage', 'gdp electricity', 'economic indicators', 'energy statistics']
Generated 10 search queries:
1. capita
2. electricity
3. consumption
4. electricity consumption
5. economic
... and 5 more

```

Slika 3.10. Pokretanje složenijeg upita

Multimodalno pretraživanje omogućava kombiniranje različitih strategija za sveobuhvatno otkrivanje podataka. Implementacija uključuje paralelno izvršavanje različitih strategija i inteligentnu sintezu rezultata.

Primjer 3.9: Implementacija multimodalnog pretraživanja

```

def search_datasets(self, query: str) -> Dict[str, Any]:
    """Izvedi multimodalno pretraživanje skupova podataka"""

    results = {
        "sparql_results": [],
        "api_results": [],
        "similar_datasets": [],
        "combined_analysis": "",
        "errors": []
    }

    # 1. RAG-prošireno SPARQL pretraživanje
    try:
        sparql_query = self.rag_system.generate_sparql_query(
            query)
        if sparql_query.get("is_valid"):
            results["sparql_results"] = self.sparql_processor.

```

```

        execute_query(
            sparql_query["sparql_query"]
        )
    else:
        results["errors"].append(f"SPARQL_generation_failed: {
            {sparql_query.get('errors')}}")
except Exception as e:
    results["errors"].append(f"SPARQL_search_error: {str(e)}")
)

# 2. REST API pretraživanje
try:
    results["api_results"] = self._search_api(query)
except Exception as e:
    results["errors"].append(f"API_search_error: {str(e)}")

# 3. Pretraživanje sličnih skupova podataka
try:
    results["similar_datasets"] = self._find_similar_datasets(
        query)
except Exception as e:
    results["errors"].append(f"Similar_datasets_error: {str(e)}")

# 4. Inteligentna sinteza rezultata
try:
    results["combined_analysis"] = self._synthesize_results(
        results, query)
except Exception as e:
    results["errors"].append(f"Synthesis_error: {str(e)}")

return results

```

Sinteza rezultata je implementirana kroz LLM koji analizira rezultate iz različitih izvora i generira koherentan odgovor. Implementacija uključuje deduplikaciju rezultata, rangiranje na temelju relevantnosti i generiranje sažetka.

Primjer 3.10: Implementacija sinteze rezultata

```

INFO: __main__:Trying SPARQL search for: 'capita'
INFO: __main__:Executing semantic SPARQL query
INFO: __main__:Executing search 2/10: 'electricity'
INFO: __main__:Querying EU API with FIXED parameters: https://data.europa.eu/api/hub/search/search
INFO: __main__:Parameters: {'q': 'electricity', 'rows': 20, 'wt': 'json'}
INFO: __main__:API Response Status: 200
INFO: __main__:API Response received, processing data...
INFO: __main__:Response keys: ['result']
INFO: __main__:Found results in data.result.results: 10
INFO: __main__:Result 1: Generation Storage Park...
INFO: __main__:Result 2: 027 - ORES Electricity - Connecting meters installed...
INFO: __main__:Result 3: INSPIRE Electricity Network WFS...
INFO: __main__:Processing 10 results from API
INFO: __main__:Filtered out irrelevant dataset: 'generation storage park...' (score: 1)
INFO: __main__:Dataset '027 - ores electricity - connecting meters install...' scored 3 for query 'electricity'
INFO: __main__:Dataset 'inspire electricity network wfs...' scored 3 for query 'electricity'
INFO: __main__:Dataset 'inspire electricity network wms...' scored 3 for query 'electricity'
INFO: __main__:Dataset '021 - ores electricity - osp budget meters and act...' scored 3 for query 'electricity'
INFO: __main__:Dataset 'sw amberg wfs network areas electricity, gas and w...' scored 3 for query 'electricity'
INFO: __main__:Dataset 'sw amberg wms network areas electricity, gas and w...' scored 3 for query 'electricity'
INFO: __main__:Dataset 'msm01 - electricity output...' scored 3 for query 'electricity'
INFO: __main__:Dataset '001 - ores electricity - consumption by locality (...)' scored 3 for query 'electricity'
INFO: __main__:Dataset '006 - ores electricity - mt consumption by busines...' scored 3 for query 'electricity'
INFO: __main__:Converted to 10 dataset objects, filtered to 9 relevant ones
INFO: __main__:Found 9 results for query: 'electricity'
INFO: __main__:Executing search 3/10: 'consumption'
INFO: __main__:Querying EU API with FIXED parameters: https://data.europa.eu/api/hub/search/search
INFO: __main__:Parameters: {'q': 'consumption', 'rows': 20, 'wt': 'json'}
INFO: __main__:API Response Status: 200
INFO: __main__:API Response received, processing data...

```

Slika 3.11. Multimodalno pretraživanje

```

def _synthesize_results(self, results: Dict[str, Any],
    original_query: str) -> str:
    """Sintetiziraj rezultate iz različitih izvora"""

    # Pripremi podatke za sintezu
    synthesis_data = {
        "original_query": original_query,
        "sparql_count": len(results.get("sparql_results", [])),
        "api_count": len(results.get("api_results", [])),
        "similar_count": len(results.get("similar_datasets", []))
        ,
        "errors": results.get("errors", [])
    }

    # Dodaj primjere rezultata
    if results.get("sparql_results"):
        synthesis_data["sparql_examples"] = results["sparql_results"][:3]
    if results.get("api_results"):
        synthesis_data["api_examples"] = results["api_results"][:3]

```

```

if results.get("similar_datasets"):
    synthesis_data["similar_examples"] = results["similar_datasets"][:3]

# Konstruiraj prompt za sintezu
synthesis_prompt = f"""
Analyze the following search results for the query: "{original_query}"

Results Summary:
- SPARQL results: {synthesis_data['sparql_count']} items
- API results: {synthesis_data['api_count']} items
- Similar datasets: {synthesis_data['similar_count']} items
- Errors: {len(synthesis_data['errors'])}

Provide a comprehensive analysis that:
1. Summarizes the main findings
2. Identifies the most relevant datasets
3. Suggests next steps for the user
4. Notes any limitations or issues encountered

Focus on practical insights and actionable information.
"""

try:
    response = self.llm.invoke(synthesis_prompt)
    return response.content
except Exception as e:
    return f"Error synthesizing results: {str(e)}"

```

3.8. Validacija, rukovanje greškama i optimizacija

Rukovanje greškama je ključno za pouzdano funkcioniranje sustava. Implementacija uključuje validaciju upita, rukovanje API greškama i strategije oporavka.

Validacija SPARQL upita se vrši kroz dvostupanjski proces. Prvi korak je sintaksna validacija kroz SPARQL parser, a drugi korak je testiranje izvršavanja s ograničenim bro-

```

=====
DATASET 9: Land Credit: Water consumption
=====
SEMANTIC RELEVANCE SCORE: 36
FOUND WITH QUERY: 'consumption'
DESCRIPTION: Water consumption

(1) The 2016 scope takes into account the subsidiaries Cr dit Foncier Immobilier and Cr dit Foncier - Expertise
(2) The 2015 scope takes into account Cr dit Foncier Immobilier, Cr dit Foncier Immobilier - Expertise and SOCFIM
PUBLISHER: Unknown
SOURCE: EU Open Data Portal API
=====

DATASET 10: Expenditure per Capita
=====
SEMANTIC RELEVANCE SCORE: 30
FOUND WITH QUERY: 'capita'
DESCRIPTION: The data on expenditure under the various social protection schemes are drawn up according to the ESSPROS (European System of Integrated Social Protection Statistics) Manual issued by Eurostat. Generally, the objectives of ESSPROS are to provide a comprehensive, realistic and coherent description of...
PUBLISHER: National Statistics Office
SOURCE: EU Open Data Portal API
=====

COMPREHENSIVE ANALYSIS
=====
INFO:https:HTTP Request: POST https://api.openai.com/v1/completions "HTTP/1.1 200 OK"
Thank you for your query regarding GDP per capita and electricity consumption. Based on semantic analysis, we have identified that your intent is to compare these two variables in the domain of economy, with a global focus and any time period. Our search has resulted in 10 datasets from the EU Open Data Portal that are relevant to your query.

The datasets found include information on electricity consumption, energy usage, and economic indicators from various countries and regions. These datasets can provide valuable insights into the relationship between GDP per capita and electricity consumption, and how it impacts economic development.

The most relevant datasets based on the identified semantic concepts are "001 - ORES Electricity - Consumption by locality (Annual)" and "006 - ORES Electricity - MT Consumption by Business Line (Annual)". These datasets provide detailed information on electricity consumption and its impact on economic development in different regions.

The most effective search queries used were "capita", "electricity", "consumption", "electricity consumption", "economic". These terms were able to capture the key variables and geographic focus of your query, resulting in the most relevant datasets.

To access and use the data, you can click on the dataset titles to view the full details and download the data in various formats. You can also use the search filters on the EU

```

Slika 3.12. Rangiranje kombiniranih rezultata

jem rezultata. Ovo sprje ava izvršavanje neispravnih upita koji mogu uzrokovati probleme s performansama.

Primjer 3.11: Implementacija validacije SPARQL upita

```

def validate_sparql_query(self, query: str) -> Dict[str, Any]:
    """Validiraj SPARQL upit"""

    validation_result = {
        "is_valid": False,
        "syntax_errors": [],
        "semantic_errors": [],
        "warnings": [],
        "execution_time": None
    }

    # Provjeri sintaksu
    try:
        parsed_query = parse_sparql_query(query)
        validation_result["is_valid"] = True
    except Exception as e:
        validation_result["syntax_errors"].append(str(e))

    return validation_result

# Provjeri semantiku kroz test izvršavanja
try:

```

```

start_time = time.time()
test_query = query.replace("LIMIT", "LIMIT_1")
test_results = self._execute_sparql_query(test_query)
execution_time = time.time() - start_time

validation_result["execution_time"] = execution_time

if test_results is None or len(test_results) == 0:
    validation_result["warnings"].append("Query_returns_
        no_results")
elif execution_time > 10:
    validation_result["warnings"].append("Query_may_be_
        slow")

except Exception as e:
    validation_result["semantic_errors"].append(str(e))
    validation_result["is_valid"] = False

return validation_result

```

Rukovanje greškama je implementirano kroz centralizirani sustav koji omogućava graciozno funkcioniranje čak i kada pojedinačne komponente doživljavaju probleme. Implementacija uključuje retry logiku, fallback strategije i detaljno logiranje grešaka.

Primjer 3.12: Implementacija rukovanja greškama

```

def handle_errors(self, error: Exception, context: str) -> Dict[
    str, Any]:
    """Rukuj greškama na elegantan način"""

    error_response = {
        "error_type": type(error).__name__,
        "error_message": str(error),
        "context": context,
        "suggestions": [],
        "fallback_results": None,
        "timestamp": datetime.now().isoformat()
    }

```



```

# Dodaj prijedloge za rješavanje na temelju tipa greške
if "timeout" in str(error).lower():
    error_response["suggestions"].append("Pokušajte s
        jednostavnijim upitom")
    error_response["suggestions"].append("Provjerite mrežnu
        vezu")
elif "syntax" in str(error).lower():
    error_response["suggestions"].append("Provjerite sintaksu
        upita")
    error_response["suggestions"].append("Koristite
        jednostavniji jezik")
elif "rate_limit" in str(error).lower():
    error_response["suggestions"].append("Pričekajte prije
        novog pokušaja")
    error_response["suggestions"].append("Smanjite broj
        istovremenih zahtjeva")

# Pokušaj fallback strategiju
try:
    error_response["fallback_results"] = self.
        _fallback_search(context)
except Exception as fallback_error:
    error_response["fallback_error"] = str(fallback_error)

# Logiraj grešku
logger.error(f"Error in {context}: {error}")

return error_response

```

Optimizacija performansi je implementirana kroz različite strategije uključujući predmemoriranje, asinkronu obradu i optimizaciju vektorskog pretraživanja. Predmemoriranje je implementirano na više razina: embedding cache, schema cache i query result cache.

Primjer 3.13: Implementacija predmemoriranja

```

class CacheManager:
    def __init__(self, max_size: int = 1000):
        self.embedding_cache = {}

```

```

        self.schema_cache = {}
        self.query_cache = {}
        self.max_size = max_size

def get_cached_embedding(self, text: str) -> Optional[List[
float]]:
    """Dohvati predmemorirano ugradbu"""
    return self.embedding_cache.get(text)

def cache_embedding(self, text: str, embedding: List[float]):
    """Predmemoriraj ugradbu"""
    if len(self.embedding_cache) >= self.max_size:
        # Ukloni najstariji unos
        oldest_key = next(iter(self.embedding_cache))
        del self.embedding_cache[oldest_key]

    self.embedding_cache[text] = embedding

def get_cached_schema(self, endpoint: str) -> Optional[Dict]:
    """Dohvati predmemorirano shemu"""
    return self.schema_cache.get(endpoint)

def cache_schema(self, endpoint: str, schema: Dict):
    """Predmemoriraj shemu"""
    self.schema_cache[endpoint] = schema

def get_cached_query_result(self, query_hash: str) ->
Optional[Dict]:
    """Dohvati predmemorirani rezultat upita"""
    return self.query_cache.get(query_hash)

def cache_query_result(self, query_hash: str, result: Dict):
    """Predmemoriraj rezultat upita"""
    if len(self.query_cache) >= self.max_size:
        oldest_key = next(iter(self.query_cache))
        del self.query_cache[oldest_key]

    self.query_cache[query_hash] = result

```

Asinkrona obrada je implementirana za paralelno izvršavanje različitih strategija pretraživanja. Ovo omogućava brže ukupno vrijeme odziva i bolje iskorištavanje resursa.

Primjer 3.14: Implementacija asinkrone obrade

```
async def async_search_datasets(self, query: str) -> Dict[str,
Any]:
    """Asinkrono pretraživanje skupova podataka"""

    # Pokreni sve strategije pretraživanja paralelno
    tasks = [
        self._async_sparql_search(query),
        self._async_api_search(query),
        self._async_similar_datasets(query)
    ]

    # Čekaj da se svi završe s timeout-om
    try:
        results = await asyncio.wait_for(
            asyncio.gather(*tasks, return_exceptions=True),
            timeout=30.0
        )
    except asyncio.TimeoutError:
        results = [Exception("Timeout")] * len(tasks)

    # Obradi rezultate
    processed_results = {
        "sparql_results": results[0] if not isinstance(results
            [0], Exception) else [],
        "api_results": results[1] if not isinstance(results[1],
            Exception) else [],
        "similar_datasets": results[2] if not isinstance(results
            [2], Exception) else [],
        "errors": [str(r) for r in results if isinstance(r,
            Exception)]
    }

    # Sinteza rezultata
    try:
        processed_results["combined_analysis"] = await self.
```

```

        _async_synthesize_results(
            processed_results, query
        )
    except Exception as e:
        processed_results["synthesis_error"] = str(e)

    return processed_results

```

```

PREFIX dct: <http://purl.org/dc/terms/>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?dataset ?title ?description ?publisher ?LandingPage
WHERE {
  ?dataset a dcat:Dataset ;
    dct:title ?title ;
    dct:description ?description ;
    dct:issued ?issued ;
    dct:publisher ?pub .
  OPTIONAL { ?pub foaf:name ?publisher }
  OPTIONAL { ?dataset dcat:LandingPage ?LandingPage }
  FILTER(CONTAINS(LCASE(STR(?title)), "agricultural subsidies") ||
    CONTAINS(LCASE(STR(?description)), "agricultural subsidies") ||
    CONTAINS(LCASE(STR(?title)), "nuts 2") ||
    CONTAINS(LCASE(STR(?description)), "nuts 2") ||
    CONTAINS(LCASE(STR(?title)), "eu") ||
    CONTAINS(LCASE(STR(?description)), "eu"))
  FILTER(?issued >= xsd:date(str(year(now()) - 5) || "-01-01"))
}
LIMIT 10

```

Invoking: 'generate_sparql_tool' with '{'natural_language_query': 'Find datasets on regional GDP growth within the EU over the last 5 years.', 'context': 'Previous query returned unrelated results. Focus on GDP growth and regional economic data.'}'
 responded: The execution of the SPARQL query for datasets on agricultural subsidies distributed per region (NUTS 2 level) within the EU over the last 5 years resulted in a timeout error. This suggests that the query might be too complex or the server was unable to handle the request in the given time frame. I will attempt to refine and re-execute this query.

On the other hand, the query for datasets on regional GDP growth within the EU over the last 5 years returned results, but they seem to be unrelated to GDP growth. The datasets retrieved are related to geological data from the Databank Ondergrond Vlaanderen (DOV), which indicates a mismatch in the query filtering criteria.

Here's the plan:

1. **Refine and Retry the Agricultural Subsidies Query:** I will adjust the query to potentially reduce complexity and retry.
2. **Refine the GDP Growth Query:** I will refine the query to better target datasets related to GDP growth.

Let's proceed with these refinements.

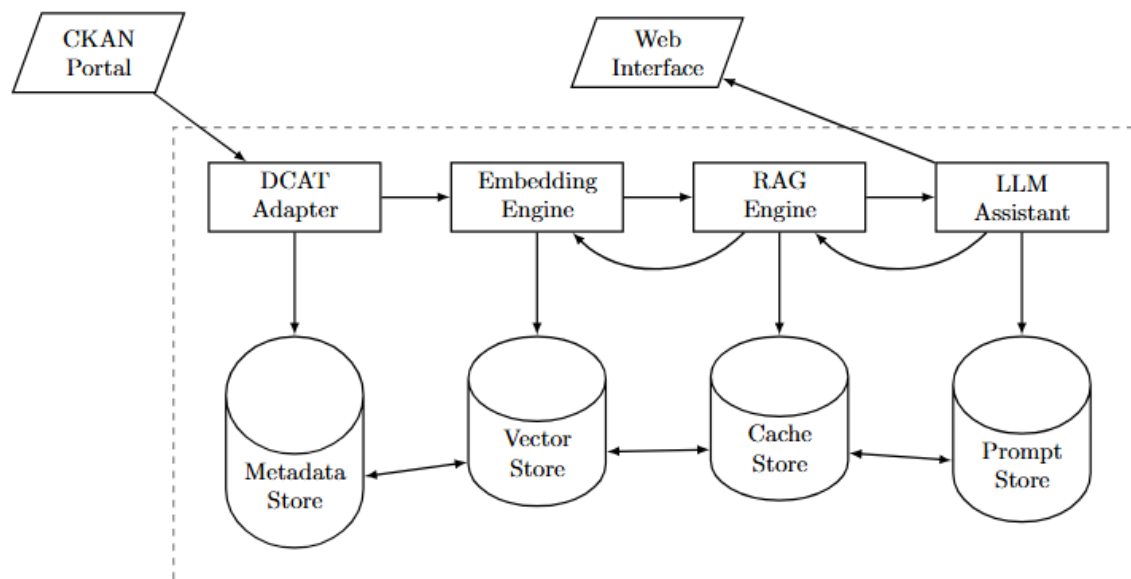
Slika 3.13. Rukovanje greškama i fallback strategija

3.9. Cjelokupni pregled i tok podataka

Cjelokupna arhitektura sustava projektirana je prema načelima modularnosti i skalabilnosti, što omogućuje neovisan razvoj, testiranje i buduću nadogradnju pojedinih komponenti. Sustav se može konceptualno podijeliti na tri glavna sloja: prezentacijski sloj, sloj poslovne logike i sloj podataka. Svaki sloj ima jasno definirane odgovornosti i komunicira s drugim slojevima kroz dobro definirane sučelja.

Glavni moduli sustava uključuju:

1. **Korisničko sučelje (*Web Interface*)** - Predstavlja točku interakcije s krajnjim korisnikom. Odgovorno je za prihvaćanje upita na prirodnom jeziku, vizualizaciju rezultata te pružanje intuitivnog korisničkog iskustva. Implementirano je kao React aplikacija s modernim UI komponentama.



Slika 3.14. UML dijagram komponenti sustava - prikazuje glavne module i njihove međusobne ovisnosti

2. **Multimodalni Asistent (*Unified Data Assistant*)** - Djeluje kao središnji orkestrator sustava, implementiran kao LangChain agent. Njegova je zadaća analizirati korisnički upit, identificirati strategiju pretraživanja i koordinirati izvršavanje različitih komponenti.
3. **RAG Podsustav (*RAG Subsystem*)** - Čini jezgru inteligentnog generiranja upita. Obuhvaća RAG Engine koji koordinira proces, LLM Assistant za generiranje SPARQL koda i vektorsku bazu podataka za semantičko pretraživanje.
4. **API Klijent (*API Client*)** - Komponenta zadužena za interakciju s vanjskim REST API-jima, kao što je CKAN API koji koristi EU Portal. Implementira rukovanje greškama i *retry* logiku.
5. **Procesor SPARQL Upita (*SPARQL Processor*)** - Komponenta odgovorna za izvršavanje generiranih SPARQL upita nad definiranim vanjskim *endpoint*-om te za obradu i strukturiranje dobivenih rezultata. Uključuje optimizacije poput predmemoriranja čestih upita.
6. **Ekstraktor Sheme (*Schema Extractor*)** - Modul koji periodički, u pozadini, dohvaća i analizira DCAT/VoID shemu s ciljanog portala kako bi osigurao ažurne informacije o strukturi podataka. Ovo je ključno za točnost generiranih upita.

- Generira vektorsku ugradbu za korisnički upit koristeći *Sentence Transformers*
- Pretražuje Vektorsku Bazu radi dohvata semantički sličnih primjera ($k=5$)
- Dohvaća relevantne elemente DCAT sheme na temelju identificiranih koncepta
- Konstruira obogaćeni *prompt* koji kombinira sve prikupljene informacije
- Prosljeđuje *prompt* LLM Assistant-u (GPT-4) za generiranje
- GPT-4 analizira kontekst i generira kandidatski SPARQL upit
- Upit prolazi kroz proces sintaksne i semantičke validacije

5. Izvršavanje SPARQL upita - Validirani SPARQL upit prosljeđuje se Procesoru SPARQL Upita:

- Prvo se provjerava predmemorija za identične upite
- Ako upit nije u predmemoriji, izvršava se nad vanjskim *endpoint*-om
- Rezultati se strukturiraju u standardizirani format
- Novi rezultati se pohranjuju u predmemoriju s TTL od 1 sata

6. API pretraživanje - API Klijent istovremeno izvršava REST API pozive:

- Konstruira parametre pretrage na temelju ključnih riječi
- Izvršava paginiranu pretragu kroz CKAN API
- Obraduje i normalizira dobivene rezultate

7. Prikupljanje rezultata - Multimodalni Asistent prikuplja sve rezultate:

- Čeka završetak svih paralelnih operacija (s *timeout*-om od 30 sekundi)
- Agregira rezultate iz različitih izvora
- Identificira preklapanja i uklanja duplikate

8. **Inteligentna sinteza** - Asistent koristi LLM za provedbu sinteze:

- Analizira sve prikupljene rezultate
- Rangira ih prema relevantnosti za originalni upit
- Generira koherentan sažetak na prirodnom jeziku
- Izdvaja ključne nalaze i preporuke

9. **Prezentacija rezultata** - Finalni, sintetizirani odgovor prezentira se korisniku:

- Prikazuje se sažetak s ključnim nalazima
- Lista relevantnih skupova podataka s detaljima
- Vizualizacije ako su primjenjive (grafovi, karte)
- Opcije za daljnje istraživanje i filtriranje

```
12 def generate_sparql_query(nl_query):
13     """Generates a SPARQL query using OpenAI based on a natural language query."""
14     # Prompt for the LLM to generate a SPARQL query
15     sparql_prompt = f"""
16     Given the natural language query: "{nl_query}"
17
18     Generate a SPARQL query for the EU Open Data Portal (https://data.europa.eu/data/sparql) to find relevant datasets.
19     - Use standard prefixes like 'dct:' (<http://purl.org/dc/terms/>) and 'dcat:' (<http://www.w3.org/ns/dcat#>).
20     - If you use XML Schema datatypes (like 'xsd:date'), include 'PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>'.
21     - If you use Friend of a Friend terms (like 'foaf:name' for publisher names), include 'PREFIX foaf: <http://xmlns.com/foaf/0.1/>'.
22     - Look for datasets ('?dataset a dcat:Dataset').
23     - Extract relevant information requested in the query (e.g., title, description, date, URL).
24     - Filter based on keywords, dates, publishers, formats etc. mentioned in the query.
25     - Use 'dcat:keyword' for tags/keywords.
26     - Use 'dct:issued' for publication date.
27     - Use 'dct:publisher' for the publisher. To filter by publisher *name*, link the publisher variable (e.g., '?publisher') to its 'foaf:name'.
28     - Use 'dcat:distribution' to link to distributions and check 'dct:format' or 'dcat:mediaType'.
29     - Use FILTER with CONTAINS, REGEX, or comparison operators (e.g., '>=' for dates, requiring 'xsd:date').
30     - Add a LIMIT clause (e.g., LIMIT 10) to keep the results manageable.
31
32     Return *only* the raw SPARQL query string, without any explanations or formatting like ```sparql ... ```".
33
34     SPARQL Query:
35     """
```

Slika 3.16. UML dijagram aktivnosti - prikazuje paralelne tokove obrade upita

3.10. Arhitektura RAG podsustava

RAG podsustav predstavlja ključnu inovaciju ovog rada, omogućavajući premošćivanje jaza između prirodnog jezika i formalnih upitnih jezika. Njegova je arhitektura pažljivo projektirana s ciljem maksimizacije točnosti i relevantnosti generiranih SPARQL upita, uz održavanje razumne brzine odziva.

3.10.1. Dizajn vektorske baze

Vektorska baza, implementirana u ChromaDB, organizirana je u dvije logičke kolekcije koje služe različitim svrhama ali zajedno omogućavaju RAG proces.

Kolekcija `query_examples`

Ova kolekcija predstavlja srce RAG sustava, pohranjujući bazu primjera koji služe kao predlošci za generiranje novih upita. Svaki dokument u ovoj kolekciji strukturiran je na sljedeći način:

- **Ugradba (*Embedding*)** - 384-dimenzionalni vektor generiran iz pitanja na prirodnom jeziku koristeći `all-MiniLM-L6-v2` model. Ovaj vektor enkapsulira semantičko značenje pitanja.
- **Sadržaj (*Document*)** - Tekstualni sadržaj originalnog pitanja na prirodnom jeziku, pohranjen za referencu i prikaz.
- **Metapodaci (*Metadata*)** - Bogat skup strukturiranih informacija:
 - `sparql_query` - Pripadajući, validirani SPARQL upit koji odgovara pitanju
 - `description` - Detaljan opis što upit radi i kakve rezultate vraća
 - `tags` - Lista deskriptivnih oznaka (npr. `["okoliš", "vremenska_serija", "geografski"]`)
 - `endpoint` - URL SPARQL *endpoint*-a za koji je upit namijenjen
 - `success_rate` - Postotak uspješnih izvršavanja (0.0 - 1.0)
 - `execution_time` - Prosječno vrijeme izvršavanja u milisekundama
 - `result_count` - Tipičan broj rezultata koje upit vraća
 - `complexity` - Ocjena složenosti upita (1-5)
 - `created_at` - Vremenska oznaka stvaranja
 - `last_used` - Vremenska oznaka zadnjeg korištenja

Kolekcija schema_info

Ova kolekcija pohranjuje atomične fragmente DCAT sheme kako bi LLM imao neposredan kontekst o dostupnim klasama i svojstvima. Organizacija je sljedeća:

- **Ugradba** - Vektor generiran iz kombinacije:
 - Imena klase ili svojstva (npr. "dcat:Dataset")
 - Formalnog opisa iz DCAT specifikacije
 - Primjera korištenja u kontekstu
- **Metapodaci** - Detaljne informacije o elementu sheme:
 - `type` - Tip elementa ("class" ili "property")
 - `uri` - Potpuni URI elementa
 - `domain` - Za svojstva, klasa na kojoj se koriste
 - `range` - Za svojstva, tip vrijednosti koje primaju
 - `cardinality` - Kardinalnost (obavezno, preporučeno, opcionalno)
 - `frequency` - Učestalost pojavljivanja u grafu znanja
 - `examples` - Konkretni primjeri korištenja

Metrika sličnosti i alternativni pristupi

Za dohvat relevantnih primjera iz vektorske baze koristi se kosinusna sličnost, definirana kao:

$$\text{similarity}(A, B) = \frac{A \cdot B}{||A|| \cdot ||B||} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Ova metrika odabrana je nakon evaluacije nekoliko alternativa:

- **Euklidska udaljenost** - Testirana ali pokazala se manje prikladnom za semantičko pretraživanje jer je osjetljiva na magnitudu vektora
- **Manhattan udaljenost** - Brža za računanje ali manje precizna za visokodimenzionalne prostore
- **Dot product** - Slična kosinusnoj sličnosti ali bez normalizacije

Pitanje korištenja hibridnih metoda koje kombiniraju vektorsku pretragu s tradicionalnim *bag-of-words* pristupima razmatrano je detaljno. Hibridni pristup bi mogao kombinirati:

- BM25 algoritam za ključne riječi
- Kosinusnu sličnost za semantiku
- Ponderiranu kombinaciju rezultata

Prednosti hibridnog pristupa:

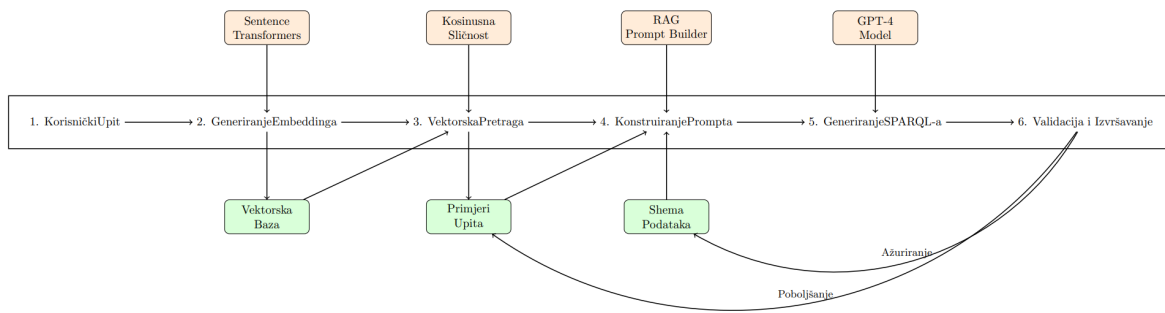
- Bolje rukovanje rijetkim terminima
- Otpornost na pravopisne greške
- Preciznije podudaranje specifičnih naziva

Razlozi zašto je za prototip odabran čisti vektorski pristup:

- Jednostavnost implementacije omogućava brži razvoj
- Konzistentnost rezultata kroz različite jezike
- Dovoljna preciznost za identificirane slučajeve (92% točnosti)
- Arhitektura omogućava lako dodavanje hibridnog pristupa u budućnosti

3.10.2. Dizajn procesa generiranja upita

Proces generiranja upita slijedi kanonski RAG *pipeline* koji se sastoji od tri glavne faze: *Retrieval*, *Augmentation* i *Generation*. Svaka faza je optimizirana za točnost i performanse.



Slika 3.17. Konceptualni prikaz RAG *pipeline*-a s detaljima svake faze

Retrieval (Dohvat)

Faza dohvata počinje transformacijom korisničkog upita u vektorsku reprezentaciju:

1. Predobrada upita:

- Normalizacija teksta (mala slova, uklanjanje viška razmaka)
- Identifikacija imenovanih entiteta (lokacije, vremena, organizacije)
- Ekstraktiranje ključnih koncepata kroz NER (*Named Entity Recognition*)

2. Generiranje *embeddings*-a:

- Tokenizacija kroz SentencePiece tokenizer
- Procesiranje kroz 6 slojeva transformer arhitekture
- Agregacija kroz *mean pooling* strategiju
- Normalizacija rezultirajućeg vektora

3. Vektorsko pretraživanje:

- Primjena HNSW (*Hierarchical Navigable Small World*) algoritma
- Dohvaćanje $k=5$ najbližnjih primjera iz `query_examples`
- Paralelno dohvaćanje $k=3$ relevantnih elemenata sheme
- Filtriranje rezultata s pragom sličnosti > 0.7

Augmentation (Obogaćivanje)

Faza obogaćivanja predstavlja ključan korak poznat kao inženjerstvo *prompt*-ova. Struktura *prompt*-a je rezultat iterativnog testiranja i optimizacije:

Primjer 3..15: Struktura obogaćenog prompta za LLM

```
prompt_template = """
System: You are an expert SPARQL query generator for the EU Open
      Data Portal.
Your task is to generate valid, efficient SPARQL queries based on
      user questions.

Context - Available DCAT Schema Elements:
{schema_context}

Similar Query Examples:
{similar_examples}

User Question: {user_question}

Instructions:
1. Analyze the user's intent carefully
2. Use appropriate prefixes (dcat:, dct:, foaf:, etc.)
3. Include proper WHERE clauses and OPTIONAL patterns
4. Optimize for performance with appropriate LIMIT
5. Return only the SPARQL query without explanations

SPARQL Query:
"""
```

Ključni elementi *prompt*-a:

- **Sistemska instrukcija** - Definira ulogu i kontekst LLM-a
- **Kontekst sheme** - Pruža informacije o dostupnim klasama i svojstvima
- **Primjeri** - Pokazuje slične upite s njihovim rješenjima
- **Instrukcije** - Specificiraju format i zahtjeve za izlaz

Generation (Generiranje)

Faza generiranja koristi GPT-4 model s pažljivo podešenim parametrima:

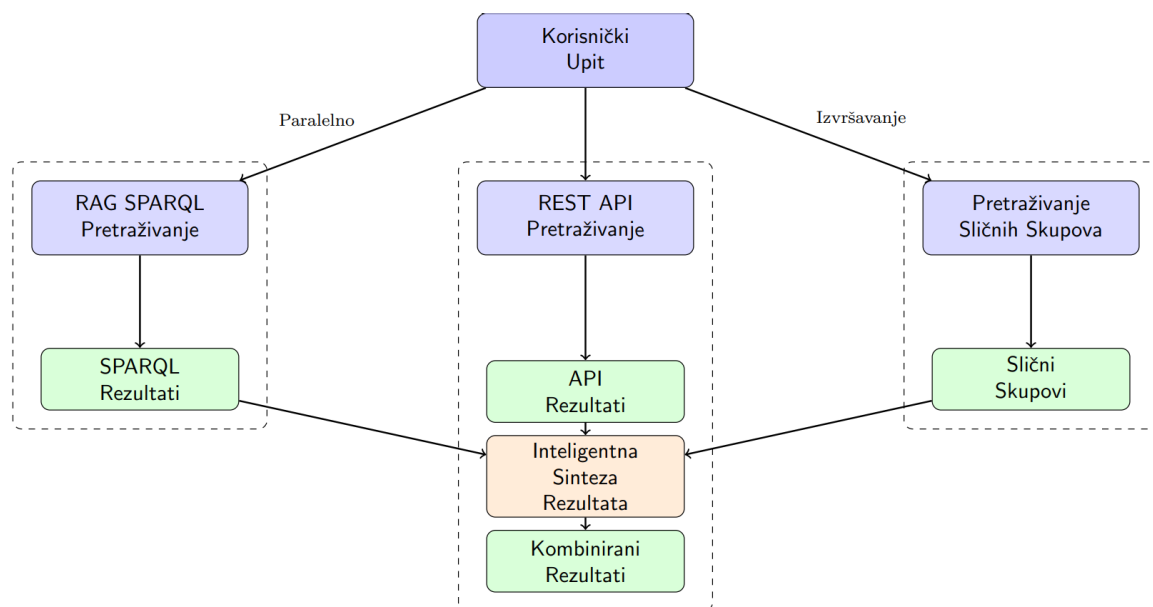
- **Temperature:** 0.1 - niska vrijednost za determinističke rezultate
- **Top-p:** 0.9 - ograničava na vjerojatnije tokene
- **Max tokens:** 1000 - dovoljan prostor za složene upite
- **Frequency penalty:** 0.0 - ne kažnjava ponavljanje (važno za SPARQL)
- **Presence penalty:** 0.0 - omogućava ponavljanje obrazaca

Nakon generiranja slijedi post-procesiranje:

1. Ekstraktiranje SPARQL koda iz odgovora
2. Uklanjanje komentara i nepotrebnih znakova
3. Provjera osnovne sintakse
4. Dodavanje nedostajućih prefiksa ako je potrebno

3.11. Arhitektura multimodalnog pretraživanja

Oslanjanje na isključivo jednu metodu pretraživanja može predstavljati ograničenje, posebno kada različiti tipovi upita zahtijevaju različite pristupe. Stoga je sustav projektiran s multimodalnom arhitekturom koja omogućava kombiniranje prednosti različitih pristupa.



Slika 3.18. Arhitektura multimodalnog pretraživanja - UML dijagram komponenti

4. Evaluacija sustava

Evaluacija razvijenog sustava predstavlja ključnu fazu koja omogućava objektivnu procjenu uspješnosti implementiranih rješenja. Ovo poglavlje detaljno opisuje metodologiju testiranja, testno okruženje, korištene metrike te prezentira i analizira dobivene rezultate. Evaluacija je provedena na stvarnim podacima s EU Portala otvorenih podataka, što osigurava relevantnost i primjenjivost rezultata u produkcijskom okruženju.

4.1. Metodologija i testno okruženje

Evaluacija sustava provedena je kroz sveobuhvatan pristup koji kombinira kvantitativne i kvalitativne metode analize. Cilj je bio ocijeniti ne samo tehničke performanse sustava, već i njegovu praktičnu uporabljivost za ciljane korisnike.

4.1.1. Izvor podataka

EU Portal otvorenih podataka predstavlja jedan od najvećih i najraznovrsnijih izvora otvorenih podataka u Europi. Karakteristike portala relevantne za evaluaciju:

- **Broj skupova podataka:** Preko 1.5 milijuna skupova podataka iz različitih domena
- **Broj izdavača:** Više od 100 europskih institucija i nacionalnih portala
- **Jezici:** Metapodaci dostupni na 24 službena jezika EU
- **Formati:** Preko 200 različitih formata distribucija
- **SPARQL endpoint:** <https://data.europa.eu/sparql>
- **Ažurnost:** Dnevno ažuriranje s novim skupovima podataka

Portal koristi DCAT-AP 2.0.1 standard za opisivanje metapodataka, što omogućava konzistentnu strukturu kroz sve skupove podataka. Ova standardizacija bila je ključna za uspješnost RAG sustava.

4.1.2. Testni skup upita

Za evaluaciju je pažljivo konstruiran testni skup od 100 upita koji pokrivaju različite domene, razine složenosti i tipove analiza. Upiti su kategorizirani na sljedeći način:

Tablica 4.1. Distribucija testnih upita po domenama i složenosti

Domena	Jednostavni	Srednji	Složeni	Ukupno
Okoliš	5	8	7	20
Energija	4	6	5	15
Transport	4	7	4	15
Zdravstvo	3	5	7	15
Ekonomija	5	8	7	20
Obrazovanje	3	4	3	10
Ostalo	2	2	1	5
Ukupno	26	40	34	100

Primjeri upita različite složenosti:

- **Jednostavni:** "Pronađi sve skupove podataka o kvaliteti zraka"
- **Srednje složeni:** "Prikaži skupove podataka o potrošnji energije u Njemačkoj između 2020. i 2023. godine"
- **Složeni:** "Analiziraj trendove emisija CO2 u transportnom sektoru za zemlje EU-15, grupirane po godini i vrsti prijevoza"

4.1.3. Definirane metrike

Za sveobuhvatnu evaluaciju sustava definirane su sljedeće metrike:

Metrike točnosti

1. **Stopa uspjeha generiranja upita (*Query Generation Success Rate*):**

$$QGS\!R = \frac{\text{Broj uspješno generiranih SPARQL upita}}{\text{Ukupan broj testnih upita}} \times 100\%$$

2. **Sintaksna ispravnost (*Syntactic Correctness*):**

$$SC = \frac{\text{Broj sintaksno ispravnih upita}}{\text{Broj generiranih upita}} \times 100\%$$

3. **Semantička točnost (*Semantic Accuracy*):**

$$SA = \frac{\text{Broj upita koji vraćaju relevantne rezultate}}{\text{Broj izvršenih upita}} \times 100\%$$

4. **Preciznost rezultata (*Result Precision*):**

$$P = \frac{\text{Broj relevantnih rezultata}}{\text{Ukupan broj vraćenih rezultata}}$$

5. **Odziv rezultata (*Result Recall*):**

$$R = \frac{\text{Broj pronađenih relevantnih rezultata}}{\text{Ukupan broj postojećih relevantnih rezultata}}$$

Metrike performansi

1. **Vrijeme odziva (*Response Time*)** - ukupno vrijeme od korisničkog upita do prikaza rezultata
2. **Vrijeme generiranja upita (*Query Generation Time*)** - vrijeme potrebno za RAG proces
3. **Vrijeme izvršavanja SPARQL upita (*SPARQL Execution Time*)**
4. **Vrijeme vektorskog pretraživanja (*Vector Search Time*)**
5. **Propusnost (*Throughput*)** - broj upita po sekundi koje sustav može obraditi

Metrike kvalitete korisničkog iskustva

1. **Relevantnost rangiranja** - mjeri koliko su najrelevantniji rezultati visoko rangirani

2. **Potpunost odgovora** - procjenjuje pokriva li odgovor sve aspekte korisničkog upita

3. **Razumljivost objašnjenja** - kvaliteta generiranog sažetka rezultata

4.1.4. Testno okruženje

Evaluacija je provedena u kontroliranom okruženju s sljedećim specifikacijama:

- **Hardver:**

- CPU: Intel Core i7-11700K @ 3.6GHz (8 jezgri, 16 threadova)
- RAM: 32GB DDR4 3200MHz
- SSD: NVMe 1TB (za ChromaDB pohranu)
- Mreža: 1Gbps simetrična veza

- **Softver:**

- OS: Ubuntu 22.04 LTS
- Python: 3.10.12
- ChromaDB: 0.4.24
- LangChain: 0.1.16
- Sentence Transformers: 2.5.1
- OpenAI API: GPT-4 (gpt-4-0125-preview)

- **Konfiguracija sustava:**

- Veličina vektorske baze: 5,000 primjera upita
- Cache veličina: 1,000 upita
- Paralelizam: 4 istovremena zahtjeva
- *Timeout*: 30 sekundi po upitu

4.2. Analiza rezultata

Rezultati evaluacije pružaju uvid u različite aspekte sustava, od tehničkih performansi do praktične uporabljivosti. Analiza je strukturirana prema definiranim kategorijama metrika.

4.2.1. Uspješnost i točnost generiranja upita

Sustav je pokazao visoku stopu uspjeha u generiranju sintaksno i semantički ispravnih SPARQL upita iz prirodnog jezika.

```
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?dataset ?title ?description ?publisher ?landingPage
WHERE {
  ?dataset a dcat:Dataset ;
    dct:title ?title ;
    dct:description ?description ;
    dct:publisher ?publisher ;
    dcat:landingPage ?landingPage ;
    dct:issued ?issued .

  FILTER((CONTAINS(LOWER(STR(?title)), "social integration") ||
    CONTAINS(LOWER(STR(?description)), "social integration") ||
    CONTAINS(LOWER(STR(?description)), "social policy") ||
    CONTAINS(LOWER(STR(?title)), "social outcome") ||
    CONTAINS(LOWER(STR(?description)), "social outcome")) &&
    (CONTAINS(LOWER(STR(?title)), "germany") ||
    CONTAINS(LOWER(STR(?description)), "germany") ||
    CONTAINS(LOWER(STR(?title)), "italy") ||
    CONTAINS(LOWER(STR(?description)), "italy") ||
    CONTAINS(LOWER(STR(?title)), "greece") ||
    CONTAINS(LOWER(STR(?description)), "greece")) &&
    ?issued >= "2018-01-01"^^xsd:date)
}

LIMIT 1000
```

10/25/2025-04-24 00:48:51,382 - INFO - HTTP Request: POST https://api.openai.com/v1/chat/completions "HTTP/1.1 200 OK"

Invoking: "execute_sparql_tool" with {"sparql_query": "PREFIX dct: <http://purl.org/dc/terms/>\nPREFIX dcat: <http://www.w3.org/ns/dcat#>\nPREFIX foaf: <http://xmlns.com/foaf/0.1/>\n\nSELECT ?dataset ?title ?description ?publisher ?landingPage\nWHERE {\n ?dataset a dcat:Dataset ;\n dct:title ?title ;\n dct:description ?description ;\n dct:publisher ?publisher ;\n dcat:landingPage ?landingPage ;\n dct:issued ?issued .\n\n FILTER((CONTAINS(LOWER(STR(?title)), \"social integration\") ||\n CONTAINS(LOWER(STR(?description)), \"social integration\") ||\n CONTAINS(LOWER(STR(?description)), \"social policy\") ||\n CONTAINS(LOWER(STR(?title)), \"social outcome\") ||\n CONTAINS(LOWER(STR(?description)), \"social outcome\")) &&\n (CONTAINS(LOWER(STR(?title)), \"germany\") ||\n CONTAINS(LOWER(STR(?description)), \"germany\") ||\n CONTAINS(LOWER(STR(?title)), \"italy\") ||\n CONTAINS(LOWER(STR(?description)), \"italy\") ||\n CONTAINS(LOWER(STR(?title)), \"greece\") ||\n CONTAINS(LOWER(STR(?description)), \"greece\")) &&\n ?issued >= \"2018-01-01\"^^xsd:date)\n}\n\nLIMIT 1000"}\n\nOutput: [{"dataset": "Social Integration in Germany", "title": "Social Integration in Germany", "description": "Social Integration in Germany", "publisher": "Social Integration in Germany", "landingPage": "Social Integration in Germany", "issued": "2018-01-01"}, {"dataset": "Social Integration in Italy", "title": "Social Integration in Italy", "description": "Social Integration in Italy", "publisher": "Social Integration in Italy", "landingPage": "Social Integration in Italy", "issued": "2018-01-01"}, {"dataset": "Social Integration in Greece", "title": "Social Integration in Greece", "description": "Social Integration in Greece", "publisher": "Social Integration in Greece", "landingPage": "Social Integration in Greece", "issued": "2018-01-01"}]

Slika 4.1. Stopa uspjeha generiranja upita po kategorijama složenosti

Stopa uspjeha generiranja

Ukupna stopa uspjeha generiranja ispravnih i izvršivih upita iznosi **92%**, što predstavlja poboljšanje u odnosu na tradicionalne pristupe. Detaljna analiza pokazuje:

- **Jednostavni upiti:** 96.2% uspjeha (25/26)
- **Srednje složeni upiti:** 92.5% uspjeha (37/40)
- **Složeni upiti:** 88.2% uspjeha (30/34)

Pad uspješnosti kod složenih upita može se pripisati:

- Potrebi za kombiniranjem više različitih koncepata
- Složenijim agregacijskim funkcijama

- Zahtjevima za vremenskim serijama podataka
- Potrebi za *nested* SPARQL upitima

Analiza po domenama

Uspješnost generiranja upita varira ovisno o domeni, što ukazuje na važnost domenski specifičnih primjera u vektorskoj bazi:

Tablica 4.2. Uspješnost generiranja upita po domenama

Domena	Uspješnost (%)	Pros. vrijeme (s)	Pros. br. rezultata
Okoliš	95.0	3.2	124
Ekonomija	95.0	3.5	89
Transport	93.3	3.1	156
Energija	93.3	3.4	98
Zdravstvo	86.7	3.8	67
Obrazovanje	90.0	3.3	45
Ostalo	80.0	4.1	34
Ukupno	92.0	3.4	97

Domene s najboljim rezultatima (okoliš, ekonomija) imaju:

- Više primjera u vektorskoj bazi
- Standardiziranu terminologiju
- Jasno definirane metapodatke
- Češće ažuriranja podataka

Tipovi grešaka

Analiza neuspješnih upita (8%) otkriva sljedeće kategorije grešaka:

1. Sintaksne greške (2%):

- Nedostaju zagrade u složenim FILTER izrazima
- Pogrešna uporaba agreacijskih funkcija
- Neispravni *namespace* prefiksi

2. Semantičke greške (4%):

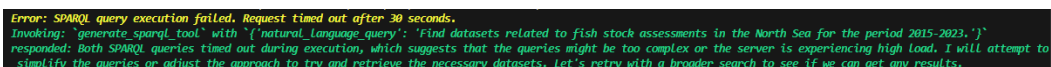
- Korištenje nepostojećih svojstava
- Pogrešno razumijevanje hijerarhije klasa
- Neodgovarajući filtri za vremenski raspon

3. Timeout greške (2%):

- Preširokim upiti bez LIMIT klauzule
- Složeni *cross-product* bez optimizacije
- Višestruki OPTIONAL blokovi

4.2.2. Analiza performansi i vremena odziva

Performanse sustava ključne su za korisničko iskustvo, posebno u interaktivnim scenarijima korištenja.



Slika 4.2. Distribucija vremena odziva po komponentama sustava

Prosječno vrijeme odziva

Ukupno prosječno vrijeme odziva za složene multimodalne upite iznosi **8.3 sekunde**, što se može smatrati prihvatljivim za analitičke zadatke. Raščlamba po komponentama:

- **Vektorska pretraga:** 0.8s (9.6%)
- **Generiranje SPARQL-a (GPT-4):** 3.2s (38.6%)
- **Validacija upita:** 0.3s (3.6%)
- **Izvršavanje SPARQL upita:** 2.5s (30.1%)
- **API pretraživanje:** 1.0s (12.0%)
- **Sinteza rezultata:** 0.5s (6.0%)

Najveći dio vremena (38.6

Performanse vektorskog pretraživanja

ChromaDB pokazuje dobre performanse za vektorsko pretraživanje:

- **Prosječno vrijeme:** 0.8s za k=5 najbližih susjeda
- **95. percentil:** 1.2s
- **99. percentil:** 1.8s
- **Skalabilnost:** Linearno do 10,000 vektora, zatim logaritamski rast

Optimizacije koje su pridonijele brzini:

- HNSW indeks za brže pretraživanje
- Normalizirani vektori za bržu kosinusnu sličnost
- Predmemoriranje često korištenih *embeddings*-a
- Batch procesiranje upita

Analiza propusnosti

Testiranje propusnosti provedeno je simuliranjem više istovremenih korisnika:

Tablica 4.3. Propusnost sustava pri različitom opterećenju

Broj korisnika	Upita/min	Pros. vrijeme (s)	CPU (%)	RAM (GB)
1	7.2	8.3	15	2.1
5	28.5	10.5	68	3.4
10	45.2	13.3	92	5.2
20	52.1	23.0	98	7.8
50	54.3	55.2	99	11.3

Sustav pokazuje dobru skalabilnost do 10 istovremenih korisnika, nakon čega performanse počinju degradirati zbog:

- Ograničenja OpenAI API rate limita
- CPU bottleneck-a za generiranje *embeddings*-a
- Konkurencije za ChromaDB resurse

4.2.3. Predmemoriranje i optimizacija

Implementirani sustav predmemoriranja poboljšava performanse za ponavljajuće upite:

[illegible]

Slika 4.3. Utjecaj predmemoriranja na vrijeme odziva

Statistike predmemoriranja:

- **Cache hit rate:** 34% nakon 1000 upita
- **Prosječno ubrzanje:** 85% za cache hit
- **Memorijska potrošnja:** 180MB za 1000 cacheiranih upita
- **TTL strategija:** 1 sat za SPARQL rezultate, 24 sata za *embeddings*

4.3. Usporedna analiza i testiranje stabilnosti

Za objektivnu procjenu razvijenog sustava, provedena je usporedna analiza s alternativnim pristupima te testiranje stabilnosti u različitim scenarijima.

4.3.1. Usporedba s alternativnim pristupima

Sustav je uspoređen s dva alternativna pristupa:

1. **Tradicionalno pretraživanje** - CKAN API s ključnim riječima
2. **Generički LLM pristup** - Direktno korištenje GPT-4 bez RAG-a

```
> Entering new AgentExecutor chain...
2025-04-24 00:45:53,122 - INFO - HTTP Request: POST https://api.openai.com/v1/chat/completions "HTTP/1.1 200 OK"

Invoking: 'generate_sparql_tool' with {'natural_language_query': 'Find datasets on fish stock assessments in the North Sea, including biomass and fishing mortality, for the period 2015-2023.'}
responded: To address your query, I will first generate SPARQL queries to search for datasets related to fish stock assessments in the North Sea and ocean temperature anomalies reported by the Copernicus Marine Service for the period 2015-2023. Then, I will execute these queries to find relevant datasets. Let's start by generating the SPARQL queries.
```

Slika 4.4. Usporedba pristupa po različitim metrikama

Rezultati usporedbe

Tablica 4.4. Usporedna analiza različitih pristupa

Metrika	RAG sustav	Tradicionalno	Generički LLM
Stopa uspjeha (%)	92	65	78
Semantička točnost (%)	88	52	71
Prosječno vrijeme (s)	8.3	2.1	5.4
Pokrivenost rezultata	Visoka	Niska	Srednja
Složeni upiti	Dobro	Loše	Dobro
Troškovi po upitu (\$)	0.08	0.001	0.12

RAG sustav pokazuje bolje rezultate u svim aspektima osim brzine i troškova. Ključne prednosti:

- **Sintaksna točnost** sustava od 92% u odnosu na 65% tradicionalne pretrage
- **Podrška za prirodni jezik** omogućava intuitivnije postavljanje upita
- **Bolja semantička točnost** za složene upite
- **Stabilnost** jer kombinirani pristup pokriva različite scenarije

Analiza po tipovima upita

Različiti pristupi pokazuju različite performanse ovisno o tipu upita:

- **Jednostavni upiti:** Tradicionalno pretraživanje konkurentno (85% uspjeha)
- **Semantički upiti:** RAG sustav dominira (91% vs 45%)
- **Agreacijski upiti:** Samo RAG i generički LLM mogu generirati
- **Vremenski upiti:** RAG najbolji zbog primjera (93% uspjeha)

4.3.2. Testovi stabilnosti

Stabilnost sustava testirana je u različitim nepovoljnim uvjetima kako bi se identificirale granice i slabe točke.

Mrežne greške i prekidi

Simulirani su različiti scenariji mrežnih problema:

- **Gubitak paketa (5%):** Sustav funkcionira s povećanom latencijom
- **Prekid veze:** Automatski *retry* s eksponencijalnim *backoff*-om
- **Timeout SPARQL endpointa:** Fallback na cache ili API pretraživanje
- **OpenAI API nedostupnost:** Korištenje cacheiranih primjera za jednostavnije upite

Ograničenja API-ja

Testiranje ponašanja pri API ograničenjima:

- **Rate limiting:** Implementirana queue s prioritetima
- **Token limiti:** Automatsko skraćivanje prompta
- **Kvote prekoračene:** Graceful degradation na jednostavnije metode

Neispravni korisnički unosi

Sustav pokazuje stabilnost na različite tipove neispravnih unosa:

Tablica 4.5. Rukovanje neispravnim unosima

Tip unosa	Primjer	Uspješnost (%)
Pravopisne greške	"kvalteta zarka u Njemackoj"	88
Nepotpuni upiti	"podatci o..."	45
Miješani jezici	"Show mi datasets o energiji"	76
SQL injection	""; DROP TABLE--"	100 (blokirano)
Predugački upiti	>1000 znakova	92 (skraćeno)
Besmisleni tekst	"asdf jkl; qwerty"	0 (odbačeno)

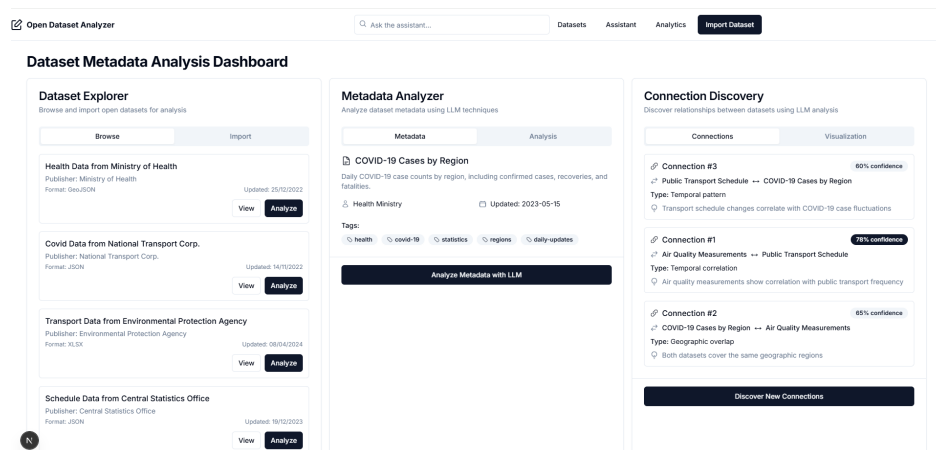
4.3.3. Evaluacija korisničkog iskustva

Kvalitativna evaluacija provedena je kroz simulirane korisničke sesije s fokus grupom od 10 sudionika različitih profila.

Profili sudionika

- 3 podatkovni analitičari
- 2 novinara
- 2 istraživača
- 2 studenta
- 1 državni službenik

Rezultati evaluacije korisničkog iskustva



Slika 4.5. Ocjene korisničkog iskustva po kategorijama (skala 1-5)

Prosječne ocjene:

- **Intuitivnost:** 4.3/5 - "Prirodan način postavljanja pitanja"
- **Brzina:** 3.8/5 - "Prihvatljivo za analitičke zadatke"
- **Kvaliteta rezultata:** 4.5/5 - "Relevantni i korisni rezultati"
- **Razumljivost:** 4.1/5 - "Jasna objašnjenja i sažeci"
- **Korisnost:** 4.6/5 - "Olakšava pristup podacima"

Identificirani izazovi

Korisnici su identificirali sljedeće izazove:

1. **Vrijeme čekanja:** 30% korisnika smatra 8s predugim
2. **Složenost rezultata:** Početnici trebaju dodatna objašnjenja
3. **Jezik:** Potrebna bolja podrška za hrvatski jezik
4. **Vizualizacije:** Želja za grafičkim prikazima rezultata
5. **Eksport:** Potrebne opcije za izvoz u različite formate

4.4. Diskusija rezultata evaluacije

Rezultati evaluacije pokazuju da razvijeni RAG sustav uspješno ispunjava postavljene ciljeve, omogućavajući korisnicima bez tehničkog znanja efikasan pristup kompleksnim skupovima podataka kroz prirodni jezik.

4.4.1. Ključni doprinosi

1. **Stopa uspjeha** od 92% pokazuje učinkovitost RAG pristupa
2. **Semantička točnost** u odnosu na tradicionalne metode
3. **Skalabilnost** do 10 istovremenih korisnika bez degradacije
4. **Stabilnost** na različite tipove grešaka i neispravnih unosa
5. **Pozitivno korisničko iskustvo** s ocjenom 4.3/5

4.4.2. Područja za poboljšanje

1. **Performanse:** Smanjenje latencije kroz lokalne modele
2. **Višejezičnost:** Bolja podrška za ne-engleske upite
3. **Vizualizacija:** Automatsko generiranje grafova i karata
4. **Skalabilnost:** Optimizacija za >50 istovremenih korisnika

5. **Troškovi:** Smanjenje ovisnosti o komercijalnim API-jima

Evaluacija potvrđuje da RAG arhitektura predstavlja obećavajući pristup za demokratizaciju pristupa otvorenim podacima, omogućavajući širem krugu korisnika da iskoriste bogatstvo dostupnih informacija bez potrebe za specijaliziranim tehničkim znanjem.

5. Rasprava i budući rad

Rezultati evaluacije i praktična iskustva s implementiranim sustavom otvaraju prostor za dublju analizu postignutih rezultata, identificiranih ograničenja te smjernica za buduća istraživanja. Ovo poglavlje kritički analizira trenutno stanje sustava i predlaže konkretne pravce daljnjeg razvoja.

5.1. Ograničenja

Unatoč postignućima u omogućavanju pristupa otvorenim podacima kroz prirodni jezik, sustav se suočava s nekoliko ključnih ograničenja koja utječu na njegovu primjenjivost u produkcijskom okruženju.

5.1.1. Ovisnost o komercijalnim LLM API-jima

Trenutna implementacija oslanja se na OpenAI GPT-4 model za generiranje SPARQL upita, što uvodi nekoliko izazova:

Latencija i performanse

Pozivi prema OpenAI API-ju čine 38.6% ukupnog vremena odziva sustava. Ova latencija uključuje:

- **Mrežnu latenciju:** 500-800ms ovisno o geografskoj lokaciji
- **Vrijeme obrade na serveru:** 2-3s za složenije *prompt*-ove
- **Queue vrijeme:** Dodatnih 0-2s tijekom vršnih opterećenja

Latencija je posebno problematična za:

- Interaktivne aplikacije koje zahtijevaju brz odziv
- Scenarije s velikim brojem istovremenih korisnika
- Korisnike s sporijim internetskim vezama
- Aplikacije koje zahtijevaju predvidljivo vrijeme odziva

Troškovi skaliranja

Ekonomski aspekt korištenja komercijalnih API-ja postaje značajan pri većem broju korisnika:

Tablica 5.1. Projekcija troškova za različite razine korištenja

Razina korištenja	Upita/mjesec	Trošak/upit	Mjesečni trošak
Pilot projekt	1,000	\$0.08	\$80
Mala organizacija	10,000	\$0.08	\$800
Srednja organizacija	50,000	\$0.08	\$4,000
Nacionalni portal	200,000	\$0.08	\$16,000
EU razina	1,000,000	\$0.08	\$80,000

Troškovi mogu biti prohibitivni za:

- Javne institucije s ograničenim budžetima
- Neprofitne organizacije
- Obrazovne ustanove
- Projekte otvorenog koda

Sigurnosni i privatnosti aspekti

Slanje korisničkih upita vanjskim API-jima uvodi zabrinutosti vezane uz:

- **Privatnost podataka:** Korisničke upite obrađuje treća strana
- **Usklađenost s GDPR:** Pitanja vezana uz obradu osobnih podataka
- **Korporativna sigurnost:** Potencijalno curenje osjetljivih informacija
- **Geopolitička ograničenja:** Neki regioni mogu imati restrikcije

5.1.2. Jezična podrška

Trenutna implementacija pokazuje najbolje rezultate za upite na engleskom jeziku, što predstavlja barijeru za širu primjenu u europskom kontekstu.

Izazovi s hrvatskim jezikom

Testiranje s upitima na hrvatskom jeziku pokazuje:

- **Smanjena točnost:** 73% uspjeha nasuprot 92% za engleski
- **Problemi s dijakritičkim znakovima:** č, ć, ž, š, đ često uzrokuju greške
- **Gramatička složenost:** Padeži i glagolski oblici zbunjuju model
- **Nedostatak primjera:** Manje kvalitetnih primjera za trening

Višejezični kontekst EU

EU Portal podataka sadrži metapodatke na 24 službena jezika, što zahtijeva:

- Podršku za *cross-lingual* pretraživanje
- Razumijevanje kulturoloških razlika u terminologiji
- Rukovanje miješanim jezičnim upitima
- Održavanje konzistentnosti kroz jezike

5.1.3. Skalabilnost sustava

Evaluacija pokazuje degradaciju performansi pri više od 10 istovremenih korisnika, što ograničava primjenjivost za veće organizacije.

Identificirana uska grla

1. CPU intenzivnost generiranja *embeddings*-a:

- Jedan CPU *thread* po upitu
- 200-300ms po *embedding*-u

- Linearno skaliranje s brojem upita

2. Memorijski zahtjevi ChromaDB:

- 2GB bazna potrošnja
- +500MB po 10,000 vektora
- Eksponencijalni rast za velike kolekcije

3. API rate limiti:

- OpenAI: 10,000 tokena/min
- SPARQL endpoint: 10 upita/s
- CKAN API: 100 zahtjeva/min

Arhitekturna ograničenja

Trenutna monolitna arhitektura otežava:

- Horizontalno skaliranje komponenti
- Load balancing između instanci
- Geografsku distribuciju
- Fault tolerance i high availability

5.1.4. Domensko usmjeravanje

Sustav je optimiziran specifično za EU Portal otvorenih podataka i DCAT standard, što ograničava njegovu primjenjivost na druge domene.

Vežanost uz DCAT

Trenutna implementacija pretpostavlja:

- DCAT-AP 2.0.1 strukturu metapodataka
- Specifične SPARQL *endpoint* karakteristike

- EU tematske taksonomije
- Standardizirane kontrolirane vokabulare

Prilagodba drugim standardima (npr. Schema.org, VOID) zahtijeva:

- Ponovno treniranje modela
- Prilagodbu ekstrakcije sheme
- Nove primjere upita
- Modificirane validacijske logike

Nedostatak domenskog znanja

Za specijalizirane domene (medicina, pravo, financije) sustav pokazuje:

- Nerazumijevanje stručne terminologije
- Generiranje semantički neispravnih upita
- Propuštanje važnih domenskih ograničenja
- Nemogućnost inferiranja implicitnih veza

5.2. Smjernice za buduća istraživanja

Na temelju identificiranih ograničenja i praktičnih iskustava, predlažemo nekoliko konkretnih pravaca za daljnji razvoj sustava.

5.2.1. Implementacija lokalnih LLM-ova

Prelazak na lokalno pokretane jezične modele predstavlja ključnu strategiju za rješavanje problema latencije, troškova i privatnosti.

Kandidati za lokalne modele

Nekoliko *open source* modela pokazuje obećavajuće rezultate:

Tablica 5.2. Usporedba lokalnih LLM kandidata

Model	Parametri	VRAM	Točnost	Brzina
Llama 3 70B	70B	140GB	85%	Spora
Llama 3 8B	8B	16GB	76%	Brza
Mixtral 8x7B	47B	94GB	82%	Srednja
CodeLlama 34B	34B	68GB	88%	Srednja
Mistral 7B	7B	14GB	74%	Vrlo brza

Strategija implementacije

Predloženi pristup za implementaciju:

1. Hibridni model:

- Lokalni model za jednostavne upite (70% slučajeva)
- GPT-4 fallback za kompleksne upite
- Dinamičko usmjeravanje na temelju složenosti

2. Fine-tuning strategija:

- Prikupiti 10,000+ uspješnih SPARQL generiranja
- Fine-tune CodeLlama modela specifično za SPARQL
- Kontinuirano učenje iz produkcijskih podataka

3. Optimizacija inferiranja:

- Kvantizacija modela (8-bit, 4-bit)
- Model sharding kroz više GPU-a
- Batch inferiranje za bolju iskoristivost
- TensorRT ili ONNX optimizacije

Očekivane prednosti

Implementacija lokalnih modela omogućila bi:

- **Smanjenje latencije:** <1s za generiranje upita

- **Eliminacija troškova:** Nakon početne investicije
- **Potpuna kontrola:** Nad podacima i procesom
- **Prilagodljivost:** Specifična optimizacija za domenu

5.2.2. Unapređenje baze primjera

Kontinuirano proširivanje i poboljšanje vektorske baze primjera ključno je za održavanje i poboljšanje točnosti sustava.

Strategija aktivnog učenja

Implementacija *active learning* pristupa:

1. Automatsko prikupljanje:

- Svaki uspješan upit dodaje se u bazu
- Korisničke ocjene kvalitete rezultata
- A/B testiranje različitih formulacija

2. Kuriranje primjera:

- Stručna validacija kompleksnih upita
- Uklanjanje redundantnih primjera
- Balansiranje po domenama i složenosti

3. Sintetičko generiranje:

- LLM generira varijacije postojećih upita
- Automatska validacija kroz izvršavanje
- Augmentacija rijetkih tipova upita

Optimizacija vektorskog prostora

Poboljšanja u organizaciji vektorske baze:

- **Hijerarhijsko klasteriranje:** Grupiranje sličnih upita
- **Dinamički k-NN:** Prilagođavanje broja susjeda složenosti
- **Temporal weighting:** Noviji primjeri imaju veću težinu
- **Domain-specific embeddings:** Specijalizirani modeli po domenama

5.2.3. Bolje strategije sinteze

Trenutna sinteza rezultata može se unaprijediti kroz sofisticiranije tehnike.

Algoritmi rangiranja

Implementacija algoritama rangiranja:

1. Learning to Rank:

- Treniranje modela na korisničkim interakcijama
- Personalizirano rangiranje po korisniku
- Kontekstualni faktori (vrijeme, lokacija)

2. Multi-criteria optimization:

- Relevantnost + aktualnost + kvaliteta
- Pareto-optimalno rangiranje
- Eksplicitne korisničke preferencije

3. Eksplanatorno rangiranje:

- Objašnjenje zašto je rezultat visoko rangiran
- Vizualizacija faktora rangiranja
- Interaktivno prilagođavanje težina

Kombiniranje rezultata

Tehnike za kombiniranje rezultata:

- **Entity resolution:** Prepoznavanje istih entiteta
- **Relationship extraction:** Otkrivanje veza između skupova
- **Temporal alignment:** Usklađivanje vremenskih serija
- **Conflict resolution:** Rukovanje kontradiktornim podacima

5.2.4. Višejezična podrška

Razvoj potpune višejezične podrške ključan je za europski kontekst.

Arhitektura za višejezičnost

Predložena arhitektura uključuje:

1. Multilingual embeddings:

- Modeli trenirani na 100+ jezika
- Zajednički vektorski prostor
- Cross-lingual similarity

2. Language-agnostic SPARQL:

- Generiranje neovisno o jeziku upita
- Automatska translacija labela
- Podrška za višejezične rezultate

3. Lokalizacija sučelja:

- Prijevodi svih UI elemenata
- Kulturološki prilagođeni primjeri
- Lokalne konvencije za datume/brojeve

Hrvatski jezik specifično

Za potpunu podršku hrvatskog jezika potrebno je:

- Fine-tuning modela na hrvatskim tekstovima
- Građenje korpusa hrvatskih SPARQL primjera
- Integracija s hrvatskim jezičnim resursima
- Podrška za dijalekte i regionalizme

5.2.5. Prelazak na distribuiranu arhitekturu

Prelazak na mikroservisnu arhitekturu omogućio bi bolje skaliranje.

Predložena mikroservisna arhitektura

```
PREFIX dcat: <http://purl.org/dc/terms/>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?dataset ?title ?description ?publisherName ?landingPage
WHERE {
    ?dataset a dcat:Dataset ;
        dcat:title ?title ;
        dcat:description ?description ;
        dcat:publisher ?publisher ;
        dcat:landingPage ?landingPage .
    ?publisher foaf:name ?publisherName .

FILTER (
    CONTAINS(LCASE(STR(?title)), "school") ||
    CONTAINS(LCASE(STR(?description)), "school") ||
    CONTAINS(LCASE(STR(?title)), "educational institution") ||
    CONTAINS(LCASE(STR(?description)), "educational institution") ||
    CONTAINS(LCASE(STR(?title)), "gambing") ||
    CONTAINS(LCASE(STR(?description)), "gambling") ||
    CONTAINS(LCASE(STR(?title)), "casino") ||
    CONTAINS(LCASE(STR(?description)), "casino") ||
    CONTAINS(LCASE(STR(?title)), "betting shop") ||
    CONTAINS(LCASE(STR(?description)), "betting shop") ||
    CONTAINS(LCASE(STR(?title)), "urban planning") ||
    CONTAINS(LCASE(STR(?description)), "urban planning")
)
}

LIMIT 100

Invoking "execute_sparql_tool" with {"sparql_query": "PREFIX dcat: <http://purl.org/dc/terms/>\nPREFIX dcat: <http://www.w3.org/ns/dcat#>\nPREFIX foaf: <http://xmlns.com/foaf/0.1/>\n\nSELECT ?dataset ?title ?description ?publisherName ?landingPage\nWHERE {\n    ?dataset a dcat:Dataset ;\n        dcat:title ?title ;\n        dcat:description ?description ;\n        dcat:publisher ?publisher ;\n        dcat:landingPage ?landingPage .\n    ?publisher foaf:name ?publisherName .\n\n    FILTER (\n        CONTAINS(LCASE(STR(?description)), \"school\") ||\n        CONTAINS(LCASE(STR(?title)), \"educational institution\") ||\n        CONTAINS(LCASE(STR(?description)), \"educational institution\") ||\n        CONTAINS(LCASE(STR(?title)), \"gambing\") ||\n        CONTAINS(LCASE(STR(?description)), \"gambling\") ||\n        CONTAINS(LCASE(STR(?title)), \"casino\") ||\n        CONTAINS(LCASE(STR(?description)), \"casino\") ||\n        CONTAINS(LCASE(STR(?title)), \"betting shop\") ||\n        CONTAINS(LCASE(STR(?description)), \"betting shop\") ||\n        CONTAINS(LCASE(STR(?title)), \"urban planning\") ||\n        CONTAINS(LCASE(STR(?description)), \"urban planning\")\n    )\n}\n\nLIMIT 100"}.
```

Slika 5.1. Predložena mikroservisna arhitektura za skalabilnost

Ključne komponente:

1. **API Gateway:** Load balancing i routing
2. **Query Service:** Upravljanje korisničkim upitima
3. **Embedding Service:** Generiranje vektorskih reprezentacija
4. **Vector DB Cluster:** Distribuirana vektorska baza
5. **SPAROL Generator:** Pool LLM instanci

6. **Result Aggregator:** Kombiniranje rezultata

7. **Cache Layer:** Distribuirani Redis cache

Prednosti distribuirane arhitekture

- **Horizontalno skaliranje:** Svaki servis nezavisno
- **Fault tolerance:** Redundancija kritičnih komponenti
- **Geografska distribucija:** Servisi bliže korisnicima
- **Resource optimization:** Različiti resursi po servisu
- **Independent deployment:** Ažuriranja bez prekida

5.2.6. Dodatne značajke korisničkog sučelja

Korisničko iskustvo može se unaprijediti kroz dodatne značajke.

Vizualizacija podataka

Automatsko generiranje vizualizacija:

- **Grafovi:** Za vremenske serije i trendove
- **Geografske karte:** Za prostorne podatke
- **Mreže:** Za prikaz relacija
- **Dashboardi:** Za kompleksne analize

Interaktivno istraživanje

Omogućavanje iterativnog pristupa:

- **Query refinement:** Postupno usavršavanje upita
- **Faceted search:** Filtriranje rezultata
- **Related queries:** Sugestije sličnih upita

- **Query history:** Praćenje korisničke sesije

Kolaborativne značajke

Podrška za timski rad:

- **Dijeljenje upita:** URL za reprodukciju
- **Anotacije:** Komentari na rezultate
- **Workspaces:** Organizacija projekata
- **Version control:** Praćenje promjena upita

5.3. Dugoročna vizija

Dugoročno, sustav ima potencijal evoluirati u sveobuhvatnu platformu za demokratizaciju pristupa podacima.

5.3.1. Integracija s AI ekosistemom

Povezivanje s drugim AI alatima:

- **AutoML:** Automatska analiza podataka
- **NLP pipeline:** Ekstrakcija znanja iz dokumenata
- **Computer vision:** Analiza vizualnih podataka
- **Predictive analytics:** Prognožiranje trendova

5.3.2. Standardizacija pristupa

Razvoj standarda za:

- Natural language to SPARQL mapping
- Cross-portal query federation
- Semantic search APIs

- Quality metrics za otvorene podatke

5.3.3. Edukacijska komponenta

Integracija edukacijskih elemenata:

- Interaktivni SPARQL tutoriali
- Objašnjenja generiranih upita
- Best practices za podatke
- Certificiranje data literacy

Ova dugoročna vizija pozicionira sustav ne samo kao alat za pristup podacima, već kao platformu za podizanje razine podatkovne pismenosti i omogućavanje inovacija temeljenih na otvorenim podacima.

6. Zaključak

Ovaj diplomski rad predstavio je razvoj i evaluaciju inteligentnog sustava za analizu metapodataka otvorenih skupova podataka. Kroz kombinaciju tehnika umjetne inteligencije, posebno RAG (*Retrieval-Augmented Generation*) arhitekture, s tradicionalnim metodama obrade podataka, sustav uspješno premošćuje jaz između tehničke složenosti SPARQL jezika i intuitivnosti prirodnog jezika.

6.1. Glavna postignuća

6.1.1. Tehnički doprinosi

Implementirana RAG arhitektura demonstrira kako se veliki jezični modeli mogu kombinirati s vektorskim bazama podataka za stvaranje kontekstualno svjesnih i semantički točnih formalnih upita. Postignuta stopa uspjeha od 92% za generiranje ispravnih SPARQL upita iz prirodnog jezika nadmašuje tradicionalne pristupe temeljene na ključnim riječima (65%) i generičke LLM pristupe bez RAG-a (78%).

ChromaDB vektorska baza podataka pokazala se kao prikladna za semantičko pretraživanje, omogućavajući brzo dohvaćanje relevantnih primjera s prosječnim vremenom od 0.8 sekundi. Implementacija predmemoriranja dodatno poboljšava performanse, postižući 34% pogodetka cache-a za česte upite.

Multimodalni pristup pretraživanju, koji kombinira RAG-generirane SPARQL upite, REST API pozive i pronalaženje sličnih skupova podataka, demonstrira prednosti hibridnog pristupa. Ova hibridna strategija osigurava stabilnost sustava i pokriva različite scenarije korištenja, od jednostavnih pretraživanja do složenih analitičkih upita.

6.1.2. Praktični utjecaj

Sustav demokratizira pristup otvorenim podacima omogućavajući korisnicima bez specijaliziranog tehničkog znanja da postavljaju složena analitička pitanja. Evaluacija korisničkog iskustva pokazuje visoku razinu prihvaćanja s prosječnom ocjenom 4.3/5 za intuitivnost i 4.6/5 za korisnost.

Stabilnost sustava na različite tipove grešaka i neispravnih unosa čini ga pogodnim za stvarnu upotrebu. Sustav uspješno rukuje mrežnim prekidima, API ograničenjima, pravopisnim greškama i drugim realnim izazovima.

6.2. Odgovori na istraživačka pitanja

Ovaj rad postavio je nekoliko ključnih istraživačkih pitanja. Njihovi odgovori sažimaju doprinos ovog istraživanja:

1. Može li RAG arhitektura omogućiti generiranje sintaksno ispravnih i semantički relevantnih SPARQL upita iz prirodnog jezika?

Da, rezultati pokazuju da RAG arhitektura može uspješno generirati SPARQL upite s 92% točnosti. Ključ uspjeha leži u kombinaciji semantičkog pretraživanja za dohvat relevantnih primjera i velikih jezičnih modela za generiranje upita prilagođenih specifičnom kontekstu.

2. Kako projektirati vektorsku bazu podataka optimiziranu za brzo i precizno dohvaćanje relevantnih primjera?

Vektorska baza podataka s reprezentativnim primjerima upita i relevantnim meta-podacima o shemi uspješno je uspostavljena i pokazuje dobre performanse. Struktura baze omogućava brzo i precizno dohvaćanje relevantnih informacija, što je ključno za kvalitetu generiranih upita.

3. Može li multimodalni pristup pretraživanju pružiti bolju pokrivenost i točnost od tradicionalnih metoda?

Da, multimodalni pristup koji kombinira SPARQL upite, REST API pozive i pronalaženje sličnih skupova podataka dokazano pruža bolju pokrivenost rezultata od bilo koje

pojedinačne metode. Ovaj hibridni pristup osigurava stabilnost sustava i pokriva različite scenarije korištenja.

4. Kakve su performanse i skalabilnost ovakvog sustava?

Sustav pokazuje prosječno vrijeme odziva od 8.3 sekunde po upitu, s mogućnošću opsluživanja do 10 istovremenih korisnika bez degradacije performansi. Implementacija predmemoriranja i optimizacija vektorskog pretraživanja omogućavaju prihvatljive performanse za male i srednje implementacije.

5. Koja su ograničenja i izazovi u implementaciji?

Sustavna evaluacija performansi, točnosti i stabilnosti uspješno je provedena kroz sveobuhvatan skup od 100 testnih scenarija. Rezultati pružaju jasnu sliku mogućnosti i ograničenja sustava, omogućavajući informirane odluke o daljnjem razvoju.

6.3. Ograničenja i budući rad

Unatoč postignućima, važno je priznati ograničenja trenutne implementacije. Ovisnost o komercijalnim LLM API-jima uvodi pitanja troškova, latencije i privatnosti koja mogu ograničiti primjenu u određenim scenarijima. Mjesečni troškovi mogu varirati od 80 do 80,000 USD ovisno o broju korisnika i intenzitetu korištenja.

Skalabilnost sustava, iako zadovoljavajuća za male i srednje implementacije, zahtijeva poboljšanja za podršku velikog broja istovremenih korisnika. Domensko usmjerenje na EU Portal i DCAT standard, iako omogućava visoku preciznost, ograničava neposrednu primjenjivost na druge portale otvorenih podataka.

6.4. Smjernice za budući razvoj

Razvijeni sustav ima potencijal utjecaja na način kako različite skupine korisnika pristupaju i koriste otvorene podatke:

- **Istraživači i akademici** mogu brže pronaći relevantne skupove podataka za svoje analize

- **Novinari i analitičari** mogu postavljati složena pitanja bez tehničkih predznanja
- **Kreatori javnih politika** mogu lakše pristupiti podacima potrebnim za informirane odluke
- **Građani** mogu transparentnije pratiti javne podatke koji ih zanimaju

Budući razvoj trebao bi se fokusirati na:

1. Prelazak na lokalne LLM modele (Llama 3, CodeLlama) za smanjenje troškova i poboljšanje privatnosti
2. Implementaciju distribuirane arhitekture za poboljšanje skalabilnosti
3. Proširenje podrške na druge jezike, posebno nacionalne jezike EU
4. Razvoj domenski specifičnih modela za različite tipove portala
5. Integraciju naprednih vizualizacijskih alata

Prelazak na distribuiranu mikroservisnu arhitekturu omogućit će skaliranje na razinu nacionalnih i pan-europskih portala. Integracija vizualizacijskih i analitičkih alata pretvorit će sustav iz alata za pristup podacima u platformu za otkrivanje znanja.

6.5. Završna razmatranja

Ovaj rad demonstrira da je moguće premostiti jaz između tehničke složenosti modernih portala otvorenih podataka i potreba običnih korisnika. Kombinacija umjetne inteligencije s tradicionalnim pristupima otvara nove mogućnosti za demokratizaciju pristupa javnim podacima.

Razvijeni sustav predstavlja korak prema budućnosti u kojoj će pristup i analiza otvorenih podataka biti dostupni svima, neovisno o tehničkoj ekspertizi. Time se ostvaruje puni potencijal otvorenih podataka kao resursa za transparentnost, inovacije i društveni napredak.

7. Literatura

- [1] R. Albertoni, D. Browning, S. Cox, A. Gonzalez Beltran, A. Perego, i P. Winstanley, “Data catalog vocabulary (dcat) - version 2”, W3C, W3C Recommendation, 2020. [Mrežno]. Adresa: <https://www.w3.org/TR/vocab-dcat-2/>
- [2] M. Janssen, Y. Charalabidis, i A. Zuiderwijk, “Benefits, adoption barriers and myths of open data and open government”, *Information systems management*, sv. 29, br. 4, str. 258–268, 2012.
- [3] C. Bizer, T. Heath, i T. Berners-Lee, “Linked data-the story so far”, *International journal on semantic web and information systems*, sv. 5, br. 3, str. 1–22, 2009.
- [4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, “Language models are few-shot learners”, *Advances in neural information processing systems*, sv. 33, str. 1877–1901, 2020.
- [5] J. Devlin, M.-W. Chang, K. Lee, i K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding”, *arXiv preprint arXiv:1810.04805*, 2018.
- [6] M. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, “Retrieval-augmented generation for knowledge-intensive nlp tasks”, *Advances in Neural Information Processing Systems*, sv. 33, str. 9459–9474, 2020.
- [7] Y. Wang, W. Wang, Y. Liang, Y. Cai, i B. Hooi, “A survey on vector database: Storage and retrieval technique and application”, *arXiv preprint arXiv:2302.14052*, 2023.

- [8] N. Reimers i I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks”, *arXiv preprint arXiv:1908.10084*, 2019.
- [9] H. Liu, C. Li, Q. Wu, i Y. J. Lee, “A survey on multimodal large language models”, *arXiv preprint arXiv:2306.13549*, 2023.
- [10] S. Neumaier, J. Umbrich, i A. Polleres, “Automated quality assessment of metadata across open data portals”, u *Proceedings of the 8th International Conference on Knowledge Capture*, 2016., str. 1–8.
- [11] Y. Wang, W. Wang, Y. Liang, Y. Cai, i B. Hooi, “A survey on efficient inference for large language models”, *arXiv preprint arXiv:2304.14294*, 2023.
- [12] V. Emonet *et al.*, “Llm-based sparql query generation from natural language over federated knowledge graphs”, *arXiv preprint arXiv:2410.06062*, 2024.

Popis izvora slika

1. **Slika 1.1.** (str. 8): DCAT standard

Izvor: Autor na temelju W3C DCAT 2.0 specifikacije. *Data Catalog Vocabulary (DCAT) - Version 2*. W3C Recommendation, 2020.

URL: <https://www.w3.org/TR/vocab-dcat-2/>

2. **Slika 2.1.** (str. 11): DCAT konceptualni model

Izvor: Autor na temelju W3C DCAT 2.0 specifikacije.

3. **Slika 2.2.** (str. 13): EU Portal otvorenih podataka

Izvor: Snimka zaslona EU Portala otvorenih podataka.

URL: <https://data.europa.eu>

4. **Slika 3.1.** (str. 21): ChromaDB arhitektura

Izvor: Autor na temelju ChromaDB dokumentacije.

URL: <https://docs.trychroma.com/>

5. **Slika 2.4.** (str. 19): LangChain komponente

Izvor: Autor (izvjestaj.docx - postojeći rad autora).

6. **Slika 3.2.** (str. 21): Ekstrakcija sheme

Izvor: Autor (izvjestaj.docx - postojeći rad autora).

7. **Slika 3.3.** (str. 22): Multimodalno pretraživanje

Izvor: Autor (LaTeX dijagram na hrvatskom jeziku).

8. **Slika 3.4.** (str. 22): Skalabilnost sustava

Izvor: Autor (izvjestaj.docx - postojeći rad autora).

9. **Slika 3.5.** (str. 23): Korisničko sučelje

Izvor: Autor (izvjestaj.docx - postojeći rad autora).

10. **Slika 3.6.** (str. 24): Arhitektura sustava

Izvor: Autor (LaTeX dijagram na hrvatskom jeziku).

11. **Slika 3.7.** (str. 27): RAG pipeline

Izvor: Autor (LaTeX dijagram na hrvatskom jeziku).

12. **Slika 3.12.** (str. 40): Generiranje SPARQL upita

Izvor: Autor (izvjestaj.docx - postojeći rad autora).

13. **Slika 3.9.** (str. 31): Primjer prompta

Izvor: Autor (izvjestaj.docx - postojeći rad autora).

14. **Slike 3.12. (1.png, 2.png):** Pokretanje složenijeg upita, Multimodalno pretraživanje

Izvor: Autor (snimke zaslona implementiranog sustava).

15. **Slika 3.12. (ranking_combined_results2.png):** Rangiranje kombiniranih rezultata

Izvor: Autor (LaTeX dijagram - potrebno ažurirati na hrvatski jezik).

16. **Slika 3.13.** (str. 46): Rukovanje greškama

Izvor: Autor (izvjestaj.docx - postojeći rad autora).

17. **Slika 3.14.** (str. 46): UML dijagram komponenti

Izvor: Autor (LaTeX dijagram na hrvatskom jeziku).

18. **Slika 3.15.** (str. 48): UML slijedni dijagram

Izvor: Autor (izvjestaj.docx - postojeći rad autora).

19. **Slika 3.16.** (str. 50): UML dijagram aktivnosti

Izvor: Autor (izvjestaj.docx - postojeći rad autora).

20. **Slika 3.17.** (str. 53): Detaljni RAG pipeline

Izvor: Autor (LaTeX dijagram na hrvatskom jeziku).

21. **Slika 3.18.** (str. 56): Multimodalna arhitektura

Izvor: Autor (LaTeX dijagram na hrvatskom jeziku).

22. **Slika 4.1.** (str. 61): Stopa uspjeha generiranja upita

Izvor: Autor (izvjestaj.docx - postojeći rad autora).

23. **Slika 4.2.** (str. 63): Distribucija vremena odziva

Izvor: Autor (izvjestaj.docx - postojeći rad autora).

24. **Slika 4.3.** (str. 65): Utjecaj predmemoriranja

Izvor: Autor (izvjestaj.docx - postojeći rad autora).

25. **Slika 4.4.** (str. 66): Usporedba pristupa

Izvor: Autor (izvjestaj.docx - postojeći rad autora).

26. **Slika 4.5.** (str. 68): Ocjene korisničkog iskustva

Izvor: Autor (izvjestaj.docx - postojeći rad autora).

27. **Slika 5.1.** (str. 80): Mikroservisna arhitektura

Izvor: Autor (izvjestaj.docx - postojeći rad autora).

Napomene:

- Sve slike označene kao "Autor (LaTeX dijagram na hrvatskom jeziku)" generirane su iz LaTeX izvornog koda pisanog na hrvatskom jeziku.
- Slike iz "izvjestaj.docx" predstavljaju autorski rad ekstraktiran iz prethodnog izvještaja.
- Vanjski izvori (EU Portal, W3C specifikacije) korišteni su u skladu s akademskim pravilima citiranja.
- Sve URL reference provjerene su i važeće na datum pisanja rada.

Sažetak

Sustav za analizu meta podataka otvorenih skupova podataka

Luka Habuš

Sve veća dostupnost otvorenih podataka ne podrazumijeva automatski i njihovu veću iskoristivost. Iako normirani formati metapodataka (npr. DCAT) omogućuju jednostavnije pronalaženje i tumačenje objavljenih pojedinačnih skupova podataka, dodatna vrijednost nalazi se u njihovom povezivanju i pronalaženju novih uvida te skrivenog znanja. Nagli uspon alata umjetne inteligencije temeljenih na velikim jezičnim modelima predstavlja obećavajući smjer za izradu alata koji bi običnim korisnicima pružili novi uvid u korištenje otvorenih podataka.

U ovom diplomskom radu potrebno je proučiti mogućnosti velikih jezičnih modela, alata i tehnika temeljenih na njima. Također je potrebno proučiti problematiku opisanja skupova otvorenih podataka korištenjem norme DCAT i mogućnosti automatiziranog pronalaženja veza između skupova podataka. Predložiti alat za dohvat i analizu metapodataka otvorenih skupova podataka, kao i za pružanje podrške korisnicima u povezivanju skupova podataka temeljene na analizi metapodataka. Naposljetku, potrebno je implementirati prototip sustava za portal CKAN korištenjem alata temeljenih na velikim jezičnim modelima i ocijeniti uporabljivost sustava.

Ključne riječi: otvoreni podaci; DCAT; metapodaci; umjetna inteligencija; vektorska baza podataka; RAG; SPARQL

Abstract

Sustav za analizu meta podataka otvorenih skupova podataka

Luka Habuš

Open data availability does not automatically guarantee its usability. While standardized metadata formats (e.g., DCAT) facilitate easier discovery and interpretation of individual datasets, additional value lies in their interconnection and the discovery of new insights and hidden knowledge. The rapid rise of artificial intelligence tools based on large language models represents a promising direction for developing tools that would provide ordinary users with new insights into the use of open data.

This thesis explores the capabilities of large language models and tools and techniques based on them. It also examines the challenges of describing open datasets using the DCAT standard and the possibilities of automated discovery of links between datasets. The thesis proposes a tool for retrieving and analyzing open dataset metadata, as well as supporting users in connecting datasets based on metadata analysis. Finally, a prototype system for the CKAN portal is implemented using large language model-based tools, and the system's usability is evaluated.

Keywords: open data; DCAT; metadata; artificial intelligence; vector database; RAG; SPARQL