Izkušnje z uporabo lokalnih modelov LLM v sistemu COBISS

Rene Svenšek, Andrej Krajnc, Bojan Štok

IZUM – Institut informacijskih znanosti, Maribor, Slovenija
rene.svensek@izum.si, andrej.krajnc@izum.si, bojan.stok@izum.si

V preteklih letih smo v sistemu COBISS na podlagi velikih jezikovnih modelov in pristopa RAG razvili pametnega asistenta. Začetna implementacija je temeljila na uporabi zunanjih storitev (OpenAI), nato pa smo z uvedbo lokalnih modelov in knjižnice LangChain4j poenostavili razvoj in izboljšali razširljivost rešitev. Pozitivne izkušnje so potrdile našo strateško usmeritev k uporabi lokalno nameščenih odprtokodnih modelov, kar omogoča večji nadzor nad podatki, večjo zasebnost in nižje stroške. Umetno inteligenco smo razširili tudi na druga področja sistema COBISS. V aplikaciji COBISS Lib smo nadgradili obstoječe asistente in dodatno razvili več novih rešitev. Mikrostoritev CAT-AI omogoča avtomatizirano ekstrakcijo bibliografskih metapodatkov iz dokumentov in slik. Z umetno inteligenco podprta pomoč uporabnikom (angl. *help desk*) omogoča semantično iskanje po interni bazi znanja. Implementirali smo iskanje v naravnem jeziku, ki uporabniške poizvedbe pretvori v strukturirane iskalne zahteve za iskanje po bibliografskih oziroma raziskovalnih virih. Rešitve temeljijo na javanskem ekosistemu (Quarkus, LangChain4j) in arhitekturi mikrostoritev. V prispevku so predstavljene praktične izkušnje pri uvajanju lokalnih modelov in integraciji umetne inteligence v sistem COBISS.

Ključne besede:
COBISS
Quarkus
LangChain4j
lokalni modeli
veliki jezikovni modeli

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-188-3
DOI https://doi.org/10.18690/um.feri.7.2026.2

1 Uvod

Umetna inteligenca v zadnjih letih vse izraziteje prodira v različna področja družbe in postaja pomemben gradnik sodobnih informacijskih sistemov. Posebej velik vpliv imajo veliki jezikovni modeli, ki omogočajo razumevanje in tvorjenje naravnega jezika, odgovarjanje na vprašanja, semantično iskanje informacij in avtomatizacijo številnih opravil, ki so bila doslej v veliki meri odvisna od človeškega dela. Zaradi teh zmožnosti se umetna inteligenca vse pogosteje uporablja tudi za izboljšanje uporabniške izkušnje in optimizacijo poslovnih procesov.

Tudi knjižnično okolje ponuja številne primere uporabe, pri katerih lahko umetna inteligenca pomembno prispeva k učinkovitejšemu delu in lažjemu dostopu do informacij. V sistemu COBISS se je njen razvoj začel z uporabo zunanjih storitev velikih jezikovnih modelov, ki so omogočile hitro preverjanje uporabnosti tovrstnih tehnologij v praksi. Nadaljnji razvoj odprtokodnih modelov in lokalne infrastrukture je omogočil prehod na arhitekturo, ki zagotavlja večji nadzor nad podatki, boljšo razširljivost in ponovno uporabo skupnih komponent pri razvoju različnih AI-storitev.

2 Od zunanjih storitev do lokalnih jezikovnih modelov

Razvoj rešitev umetne inteligence v sistemu COBISS je potekal postopno, skladno z razvojem velikih jezikovnih modelov in spremljajočih tehnologij. Prve implementacije so temeljile na uporabi zunanjih storitev, saj so omogočale hiter razvoj in preverjanje uporabnosti umetne inteligence v knjižničnem okolju [1]. Z nadaljnjim razvojem odprtokodnih modelov in lokalne infrastrukture se je pokazala možnost prehoda na arhitekturo, ki omogoča večji nadzor nad podatki, boljšo razširljivost in učinkovitejšo integracijo umetne inteligence v različne storitve sistema COBISS [2].

2.1. Prva generacija AI-rešitev

Prve rešitve umetne inteligence v sistemu COBISS so temeljile na velikih jezikovnih modelih, dostopnih prek storitve OpenAI. Takšen pristop je omogočil hitro preverjanje uporabnosti velikih jezikovnih modelov pri podpori knjižničnim storitvam in razvoj prve generacije pametnega asistenta v aplikaciji COBISS Lib. Asistent je uporabnikom pomagal pri odgovarjanju na vprašanja o uporabi sistema in pri hitrejšem iskanju informacij v obstoječi dokumentaciji.

Ker veliki jezikovni modeli sami po sebi nimajo dostopa do internih podatkov sistema COBISS, je bila rešitev zasnovana na pristopu Retrieval-Augmented Generation (RAG). Uporabniška poizvedba se najprej pretvori v vektorsko predstavitev, nato pa se v vektorski bazi poiščejo vsebinsko najbolj podobni dokumenti. Pridobljeni izseki se vključijo v poziv (angl. *prompt*) jezikovnemu modelu, ki pripravi odgovor na podlagi posredovanega konteksta. Takšen pristop zmanjšuje pojav halucinacij in omogoča uporabo ažurnega domenskega znanja brez dodatnega učenja modela.

Začetna implementacija je pokazala, da je kombinacija velikih jezikovnih modelov in pristopa RAG primerna za različne primere uporabe v sistemu COBISS. Pri uporabi zunanjih storitev pa so se hkrati pokazale tudi omejitve, povezane z odvisnostjo od zunanjega ponudnika, stroški uporabe, zakasnitvami pri komunikaciji z oddaljenimi storitvami ter vprašanji varovanja podatkov in zasebnosti. Čeprav se v oblak niso pošiljali občutljivi podatki, je bila možnost njihove obdelave zunaj lastne infrastrukture pomemben dejavnik pri načrtovanju nadaljnjega razvoja.

2.2. Evolucija k lokalnim modelom

Z razvojem odprtokodnih velikih jezikovnih modelov in zmogljivejše strojne opreme je postala izvedljiva tudi uporaba lokalno nameščenih modelov. Zato smo postopno prešli na arhitekturo, ki omogoča uporabo lokalnih modelov kot privzete izbire. S tem smo pridobili popoln nadzor nad izvajanjem modelov, zmanjšali stroške uporabe in poenostavili vključevanje umetne inteligence v druge storitve sistema COBISS. Prehod ni pomenil zgolj

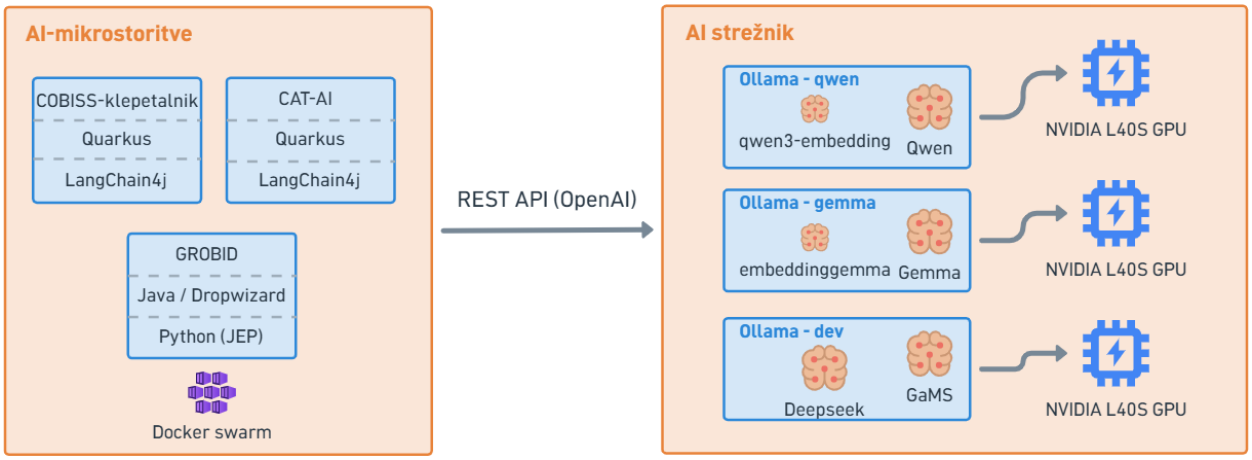
zamenjave ponudnika jezikovnega modela, temveč tudi priložnost za poenotenje razvoja AI-rešitev, saj so skupni gradniki, kot so RAG, vektorske baze in enotni programski vmesniki, omogočili ponovno uporabo komponent pri razvoju različnih aplikacij. Posledično je bilo mogoče umetno inteligenco razširiti tudi na nova področja, kot so katalogizacija, pomoč uporabnikom in iskanje v naravnem jeziku.

Kot kažejo izkušnje pri razvoju sistema COBISS, so sodobni odprtokodni modeli za številne domensko omejene naloge že dovolj zmogljivi, da predstavljajo alternativo komercialnim storitvam. Zunanji modeli praviloma še vedno zagotavljajo višjo kakovost odgovorov in hitrejši dostop do najnovejših modelov, lokalni modeli pa omogočajo večji nadzor nad podatki, bolj predvidljive stroške in lažjo integracijo v interno okolje. Prav zato je bila odločitev za lokalno izvajanje modelov predvsem odgovor na potrebe po večji neodvisnosti, boljši obvladljivosti in tesnejši integraciji umetne inteligence v obstoječi ekosistem COBISS.

Prehod na lokalne modele je zahteval tudi prilagoditev programske arhitekture. V nadaljevanju je predstavljena arhitektura AI-rešitev, ki omogoča uporabo različnih jezikovnih modelov in ponovno uporabo skupnih komponent pri razvoju storitev v sistemu COBISS.

3 Arhitektura AI-rešitev v sistemu COBISS

Po prehodu na lokalne velike jezikovne modele je bilo treba zagotoviti arhitekturo, ki omogoča njihovo uporabo v različnih delih sistema COBISS. Namesto razvoja ločenih, izoliranih rešitev za posamezne primere uporabe je bil zasnovan poenoten razvojni pristop. Ta temelji na mikrostoritveni arhitekturi, enotnem razvojnem ogrodju in skupni infrastrukturi za izvajanje jezikovnih modelov. Takšna zasnova omogoča enostavnejši razvoj novih AI-rešitev, njihovo neodvisno nameščanje in ponovno uporabo preverjenih razvojnih vzorcev.



Slika 1: Infrastruktura AI-rešitev v sistemu COBISS

3.1. Infrastruktura za izvajanje modelov

Lokalni veliki jezikovni modeli se izvajajo na namenskem strežniku, opremljenem s tremi grafičnimi procesorji NVIDIA L40S, od katerih ima vsak 48 GB grafičnega pomnilnika (skupaj 144 GB VRAM-a). Takšna konfiguracija omogoča hkratno izvajanje več različnih modelov in zagotavlja zadostno zmogljivost za podporo zahtevnejšim storitvam umetne inteligence.

Za strežbo in upravljanje modelov se uporablja platforma Ollama; ta deluje kot vmesna plast (angl. *inference engine*) in zagotavlja enoten vmesnik REST API, ki je popolnoma združljiv s standardom OpenAI (angl. *OpenAI-compatible API*). Ta združljivost je ključna, saj omogoča preprosto integracijo z obstoječimi orodji, ogrodji (npr. LangChain, LlamaIndex) in knjižnicami brez potrebe po prilagajanju izvorne kode aplikacij — zamenja se le ciljni naslov (angl. *base URL*) strežnika.

Ollama v tej arhitekturi skrbi za:

- Dinamično upravljanje pomnilnika. Samodejno nalaga in prazni modele iz grafičnega pomnilnika glede na trenutne zahteve AI-storitev.
- Porazdelitev bremena (angl. load balancing). Optimizira izrabo vseh treh razpoložljivih grafičnih procesorjev, kar omogoča vzporedno obdelavo več zahtev hkrati.
- Fleksibilnost storitev. Na strežniku je lahko hkrati nameščenih več različnih modelov (npr. manjši modeli za hitro generiranje besedila in večji za kompleksnejše analize), posamezna AI-storitev pa glede na svoje specifične zahteve dinamično izbere najprimernejšega.

Takšno centralizirano izvajanje modelov bistveno poenostavlja njihovo upravljanje, posodabljanje, dodajanje novih različic in natančno spremljanje delovanja celotnega sistema.

3.2. Mikrostoritvena arhitektura AI-rešitev

Vse AI-rešitve v sistemu COBISS so implementirane kot samostojne mikrostoritve. Takšen modularni pristop omogoča neodvisen razvoj, testiranje in nameščanje posameznih komponent, hkrati pa bistveno poenostavlja njihovo integracijo v obstoječi ekosistem COBISS.

Za razvoj mikrostoritev se uporablja napredno ogrodje Quarkus, ki je optimizirano za okolji Kubernetes in Java v oblaku [3]. Izbira tega ogrodja prinaša več ključnih prednosti. To so:

- Učinkovitost in odzivnost. Quarkus omogoča izdelavo lahkih aplikacij s hitrim časom zagona (angl. fast startup time) in nizko porabo pomnilnika, kar je ključno pri izvajanju v kontejnerskih okoljih.
- Možnost nativnega prevajanja. Podpora za GraalVM omogoča prevajanje aplikacij neposredno v nativno strojno kodo, s čimer se še dodatno zmanjša pomnilniški odtis (angl. memory footprint) posamezne mikrostoritve.
- Ohlapna povezanost (angl. loose coupling). Posamezne AI-storitve z drugimi komponentami sistema komunicirajo prek standardiziranih REST-vmesnikov. Zaradi tega so storitve med seboj neodvisne, kar zmanjšuje medsebojne odvisnosti in preprečuje, da bi morebitna napaka v eni storitvi vplivala na delovanje preostalega sistema.

Takšna arhitekturna zasnova zagotavlja visoko stopnjo razširljivosti (angl. *scalability*), saj je mogoče najbolj obremenjene AI-storitve poljubno in neodvisno skalirati glede na trenutne potrebe uporabnikov sistema COBISS.

3.3. Integracija velikih jezikovnih modelov

Za integracijo velikih jezikovnih modelov v razvojno okolje Java se uporablja knjižnica LangChain4j, ki predstavlja enoten abstrakcijski sloj za komunikacijo z modeli [4]. Posamezna AI-mikrostoritev prek ogrodja LangChain4j definira parametre za izbrani model – naj gre za klasični jezikovni model (LLM), model vlaganja (angl. *embedding model*) ali multimodalni veliki jezikovni model (angl. Multimodal-Large-Language Model – MLLM) za analizo multimodalnih podatkov. Vsi zahtevki se nato prek REST-vmesnika posredujejo centraliziranemu strežniku Ollama.

Takšen pristop standardizira razvoj AI-rešitev, saj razvijalcem omogoča uporabo enakega programskega modela ne glede na tip in različico izbranega modela v ozadju. Zamenjava ali posodobitev uporabljenega modela praviloma zahteva le spremembo konfiguracijskih nastavitev aplikacije, medtem ko jedro poslovne logike posamezne storitve ostane nespremenjeno.

Poleg osnovne komunikacije z modeli lahko posamezne storitve glede na specifične potrebe vključujejo tudi kompleksnejše arhitekturne gradnike. Mednje spadajo sistemi za iskanje s podporo znanja: RAG, ustvarjanje besedilnih vektorjev in integracija z vektorskimi bazami podatkov. Povezava med ogrodjema Quarkus in LangChain4j tako tvori tehnično osnovo za uvajanje funkcionalnosti umetne inteligence v sistem COBISS.

Opisana arhitektura predstavlja skupno osnovo za razvoj vseh AI-rešitev v sistemu COBISS. Poenoten razvojni pristop, centralizirano izvajanje jezikovnih modelov in mikrostoritvena zasnova omogočajo hitre razvojne iteracije in enostavno vključevanje novih funkcionalnosti.

4 Implementirane rešitve in primeri uporabe

Na podlagi predstavljene arhitekture je bilo v sistemu COBISS razvitih več rešitev, ki uporabljajo velike jezikovne modele za podporo različnim poslovnim procesom. Čeprav so te rešitve namenjene različnim profilom uporabnikov in rešujejo posebne težave, vse temeljijo na enotnem, poenotenem razvojnem pristopu, opisanem v prejšnjem poglavju. V nadaljevanju so predstavljene ključne implementirane rešitve, njihove specifične arhitekturne značilnosti in konkretni primeri uporabe v praksi.

4.1. COBISS-klepetalnik za knjižničarje in uporabnike knjižnic

Prva rešitev, razvita na predstavljeni arhitekturi, je COBISS-klepetalnik. Gre za na umetni inteligenci temelječo storitev, ki nadomešča klasične klepetalne kanale, v katerih odgovore podajajo operaterji.

Sprva je bil uveden v aplikaciji COBISS Lib kot pametni asistent za knjižničarje, v katerem podpira vsakodnevno delo z zagotavljanjem hitrih odgovorov na vprašanja o funkcionalnostih sistema in uporabi programske opreme.

Na spletnem portalu COBISS Plus bo klepetalnik kmalu na voljo tudi slovenskim uporabnikom knjižnic. V tem primeru je namenjen predvsem pridobivanju informacij o storitvah sistema COBISS in uporabi kataloga ter odgovarjanju na splošna vprašanja uporabnikov.

Obe rešitvi temeljita na skupni zaledni mikrostoritvi. Ključna razlika med njima je v uporabniškem vmesniku in v naboru dokumentov, ki sestavljajo bazo znanja, na podlagi katere se ustvarjajo odgovori. Medtem ko klepetalnik za knjižničarje temelji predvsem na priročnikih in strokovnih navodilih, klepetalnik za uporabnike knjižnic izhaja predvsem iz zbirk najpogosteje zastavljenih vprašanj.

Arhitektura je zasnovana z vidika ponovne uporabe in razširljivosti. Namesto razvoja ločenih klepetalnikov za posamezne aplikacije je implementiran enoten zaledni sistem, ki omogoča:

- indeksiranje različnih zbirk dokumentacije,
- izbiro ustreznega vira znanja glede na kontekst uporabe,
- generiranje odgovorov, prilagojenih posamezni skupini uporabnikov.

Takšen pristop omogoča uvajanje klepetalnika v druge dele sistema COBISS ob minimalnih dodatnih razvojnih stroških. Dodajanje novih klepetalnikov v ekosistem COBISS ne zahteva posegov v zaledno programsko kodo, temveč zgolj pripravo in indeksiranje ustrezne dokumentacije.

Vse zaledne izboljšave – bodisi fino nastavljanje iskalnih algoritmov, optimizacija priprave konteksta ali nadgradnje komunikacijskega sloja z jezikovnim modelom – so tako samodejno na voljo vsem aplikacijam, ki uporabljajo skupno zaledno storitev.

4.1.1. Postopek indeksiranja in priprava baze znanja za COBISS-klepetalnik

Implementacija RAG temelji na uporabi lokalnega namenskega modela Qwen3-Embedding, ki se uporablja tako pri pripravi baze znanja kot tudi pri obdelavi uporabniških poizvedb.

Postopek gradnje baze znanja poteka v naslednjih korakih:

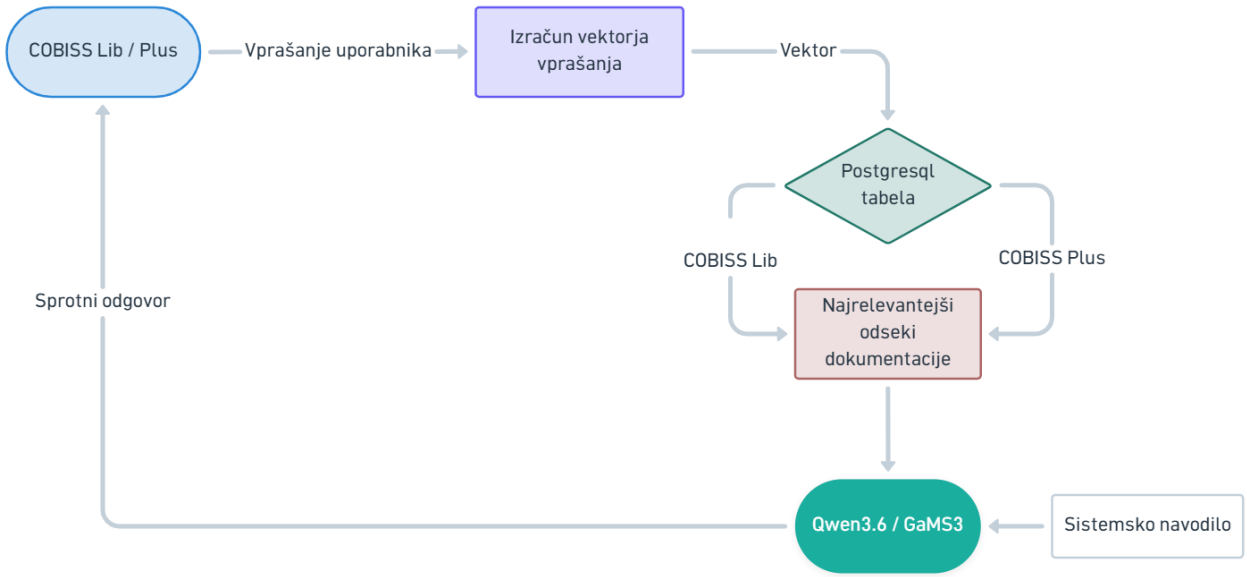
1. Razrez dokumentov (angl. *chunking*). Izvorna dokumentacija posamezne storitve se razdeli na manjše, logično zaokrožene vsebinske odseke, s čimer se zmanjša šum pri kasnejšem iskanju.
2. Vektorizacija z modelom Qwen3-Embedding. Za vsak besedilni odsek se na podlagi modela vlaganja (angl. *embedding model*) izračuna njegova vektorska predstavitev, ki matematično popiše semantični pomen besedila.
3. Vnos v namenske tabele. Vektorji se skupaj z izvornim besedilom in pripadajočimi metapodatki shranijo v ločeno podatkovno tabelo v PostgreSQL, ki je dodeljena posamezni AI-storitvi.

4.1.2. Obdelava poizvedb v COBISS-klepetalniku

Ko zaledna mikrostoritev prejme uporabniško vprašanje, se celoten postopek obdelave izvede dinamično glede na izvor poizvedbe, in sicer kot sledi:

1. Pretvorba v vektor. Uporabniško vprašanje se najprej pretvori v vektorsko predstavitev z enakim modelom Qwen3-Embedding.
2. Semantično iskanje in usmerjanje. Sistem glede na izvorno aplikacijo (COBISS Lib ali portal COBISS Plus) izbere pripadajočo tabelo v bazi podatkov in iz nje izlušči semantično najpomembnejše odseke dokumentacije.
3. Kontekstualizacija poziva. Pridobljeni odseki se v obliki konteksta priložijo uporabniškemu vprašanju in sistemskim navodilom (angl. *system prompt*), pri čemer COBISS-klepetalnik za knjižničarje uporablja lokalni model Qwen3.6 (boljša večjezična podpora), COBISS-klepetalnik za slovenske uporabnike knjižnic pa uporablja lokalni model GaMS3 (boljša podpora za slovenski jezik).

Celoten potek obdelave poizvedbe je vizualno prikazan na sliki 2.



Slika 2: Potek obdelave poizvedbe v COBISS-klepetalniku

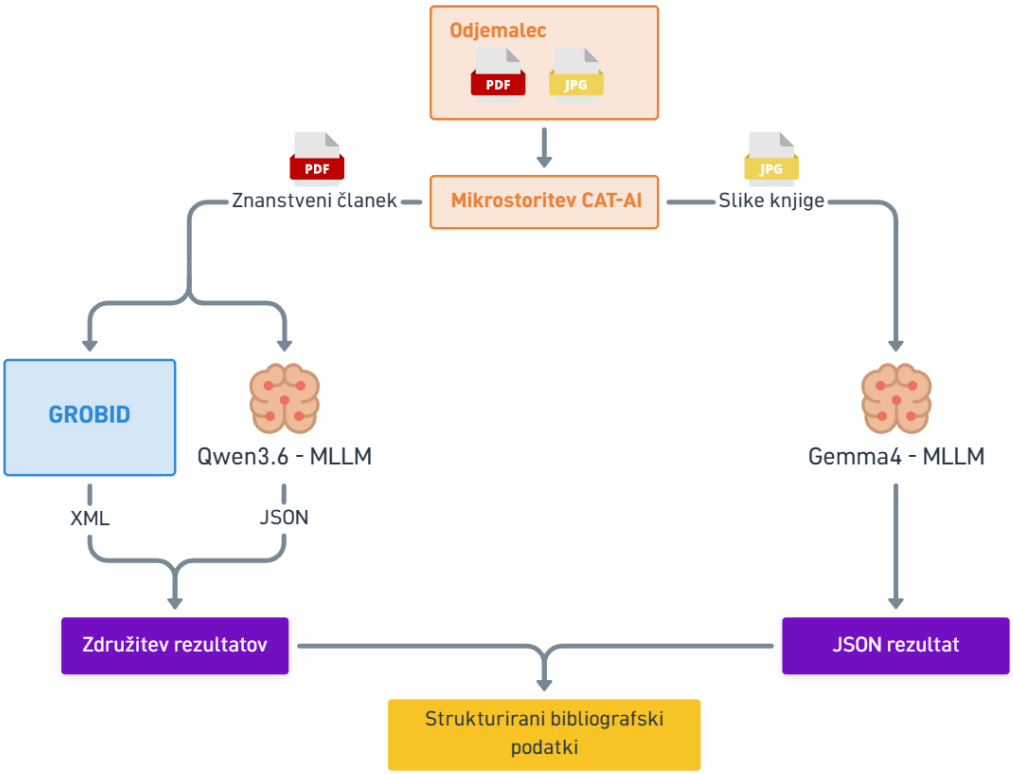
4.2. CAT-AI

CAT-AI je namenska mikrostoritev, razvita za avtomatizirano ekstrakcijo bibliografskih metapodatkov iz različnih vrst gradiva. Njena osrednja naloga je priprava strukturiranih metapodatkov, ki predstavljajo temelj za nadaljnjo gradnjo bibliografskih zapisov, s čimer se zmanjša obseg ročnega vnosa podatkov pri katalogizaciji.

Mikrostoritev je zasnovana modularno, zato lahko glede na vrsto vhodnega dokumenta dinamično izbere najprimernejši cevovod obdelave. Trenutno podpira dva ključna scenarija:

- ekstrakcija metapodatkov iz znanstvenih člankov in
- ekstrakcija metapodatkov iz naslovne in zadnje strani knjige.

Čeprav oba postopka sledita istemu cilju, tj. pripravi kakovostnih bibliografskih metapodatkov, uporabljata različne tehnološke pristope, prilagojene značilnostim posamezne vrste vhodnega gradiva.



Slika 3: Potek obdelave znotraj mikrostoritve CAT-AI

Potek obdelave znotraj mikrostoritve CAT-AI se začne tako, da po prejemu vhodnega dokumenta sistem najprej določi njegov tip in nato izbere ustrezno procesno vejo. Pri znanstvenih člankih se izvede vzporedna obdelava z orodjem GROBID za ekstrakcijo bibliografskih podatkov in jezikovnim modelom Qwen3.6, ki dodatno analizira vsebino dokumenta. Pri obdelavi naslovne in zadnje strani knjige pa ekstrakcijo metapodatkov neposredno iz slikovnih vhodov izvede vizualni jezikovni model Gemma4.

Ne glede na izbrano procesno pot je rezultat vedno poenoten nabor strukturiranih bibliografskih metapodatkov v standardizirani obliki. Tako pripravljeni podatki so neposredno uporabni v nadaljnjih postopkih katalogizacije in se lahko brez dodatnih prilagoditev vključijo v osrednji katalogizacijski proces sistema COBISS.

4.2.1. Ekstrakcija metapodatkov iz znanstvenih člankov

Pri znanstvenih člankih gre praviloma za dokumente PDF, iz katerih lahko pridobimo besedilno vsebino. Pri obdelavi znanstvenih člankov v besedilni obliki mikrostoritev vzporedno sproži dva neodvisna postopka obdelave. To sta:

- Strukturna analiza (GROBID). Ta postopek temelji na uporabi zunanje storitve GROBID, ki je specializirana za strojno učenje in analizo strukture znanstvenih člankov ter iz njih lušči strukturirane bibliografske informacije.
- Semantična analiza (Qwen3.6). Ta postopek uporablja lokalni veliki jezikovni model Qwen3.6. Modelu se kot kontekst posredujejo uvodne strani dokumenta PDF, saj je na njih praviloma večina ključnih podatkov, kot so naslov članka, avtorji, povzetek in podatki o publikaciji.

Oba postopka se izvajata sočasno, njuni delni rezultati pa se po zaključeni obdelavi v zaledju deterministično združijo v enoten, skladen nabor metapodatkov. Takšen hibridni pristop uspešno združuje prednosti obeh metod: GROBID zagotavlja stabilno ekstrakcijo strogo strukturiranih podatkov iz znanstvenih publikacij, medtem ko lokalni jezikovni model dopolni oziroma preveri informacije, ki jih GROBID zaradi posebnih postavitev strani morda ni prepoznal ali jih ni mogel enoznačno določiti. Kombinacija obeh metod se je v praksi izkazala za znatno zanesljivejšo od samostojne uporabe katerega koli izmed navedenih orodij.

4.2.2. Ekstrakcija metapodatkov iz naslovnic knjig

Naslovna in zadnja stran knjige sta slikovno gradivo, pri katerem nimamo na voljo besedilne vsebine, temveč le grafiko. Pri razvoju komponente za ekstrakcijo metapodatkov iz slikovnega gradiva sta bila preizkušena dva različna pristopa, ki nazorno prikazujeta prednost uvedbe sodobnih vizualnih modelov.

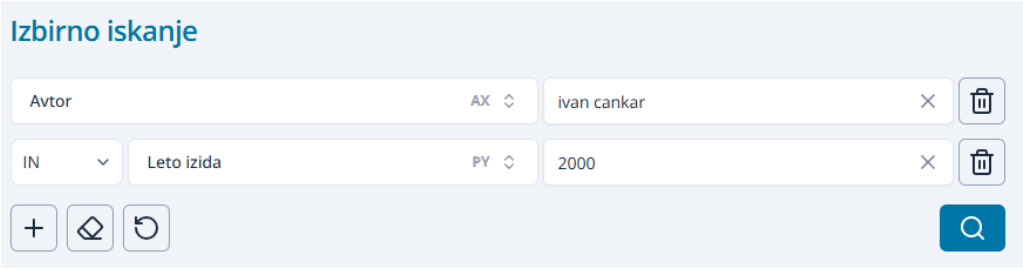
Prvotni pristop je temeljil na lastni storitvi OCR (optična prepoznava znakov), ki je iz slik najprej izluščila golo besedilo, nato pa je aplikacija s programsko logiko poskušala prepoznati posamezne bibliografske elemente (naslov, avtor, založba). Ta pristop ni dosegal zadovoljive stopnje natančnosti, saj je bila uspešnost močno odvisna od kakovosti fotografije, kompleksnosti postavitve besedila in grafične zasnove naslovnice.

Zaradi omenjenih omejitev je bil uveden sodobnejši pristop, ki v celoti temelji na uporabi multimodalnega jezikovnega modela (angl. Multimodal Large Language Model – MLLM) Gemma4. Mikrostoritev CAT-AI lahko v okviru ene zahteve sočasno obdela več slik iste publikacije (npr. fotografije naslovne strani, hrbtne strani in kolofona). Model Gemma4 nato na podlagi celotnega nabora vizualnih vhodov pripravi strukturiran odgovor v obliki zapisa JSON z vsemi izluščenimi bibliografskimi metapodatki. Ta multimodalni pristop se je izkazal za bistveno uspešnejšega, saj model pri interpretaciji ne bere le golega besedila, temveč upošteva celotno vizualno hierarhijo, velikost in razmerja med posameznimi grafičnimi elementi na slikah, kar mu omogoča natančno razlikovanje med naslovom knjige, podnaslovom in imenom avtorja.

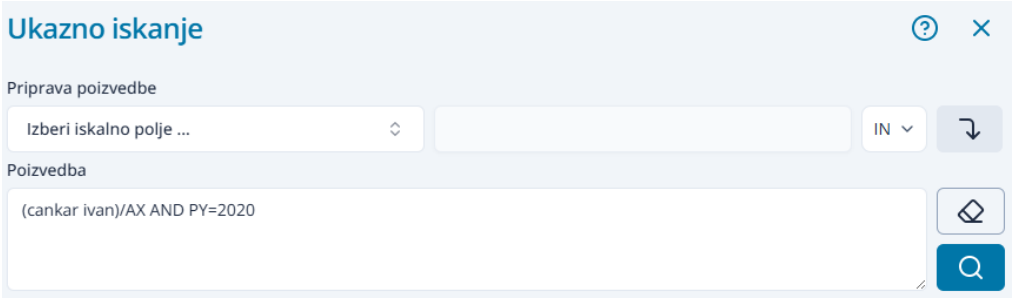
4.3. Iskanje v naravnem jeziku

Tretja implementirana rešitev omogoča iskanje po bibliografskih zapisih z uporabo naravnega jezika. Namenjena je poenostavitvi uporabniške izkušnje, predvsem za tiste uporabnike, ki ne poznajo zapletene sintakse bibliografskih poizvedb ali razpoložljivih iskalnih polj v sistemu COBISS. Namesto ročnega sestavljanja strukturiranih iskalnih izrazov lahko uporabnik svojo zahtevo opiše v prostem, naravnem jeziku, sistem na zaledni ravni pa jo samodejno pretvori v pravilno oblikovano bibliografsko poizvedbo, ki jo nato izvede obstoječi iskalni mehanizem COBISS.

Če želimo poiskati knjige Ivana Cankarja, izdane leta 2000, moramo zdaj v aplikaciji COBISS Plus uporabiti izbirno ali ukazno iskanje.

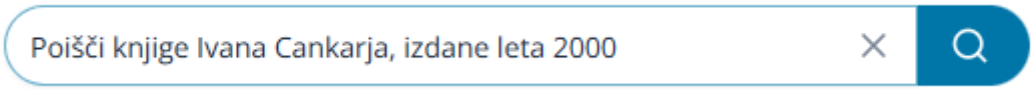


Slika 4: Izbirno iskanje v aplikaciji COBISS Plus



Slika 5: Ukazno iskanje v aplikaciji COBISS Plus

Poleg tega smo uvedli iskanje v naravnem jeziku, na primer: »Poišči knjige Ivana Cankarja, izdane leta 2000.«



Slika 6: Iskanje v naravnem jeziku

Rešitev temelji na uporabi lokalnega jezikovnega modela, katerega primarna naloga ni ustvarjanje neposrednega besedilnega odgovora uporabniku, temveč napredna semantična analiza uporabnikove zahteve in njena pretvorba v strukturiran iskalni niz (angl. *Text-to-Query*). Med tem procesom model iz uporabnikovega besedila prepozna posamezne bibliografske entitete in atribute ter jih preslika v ustrezna iskalna polja. To so:

- Identifikacija entitet. Prepoznavanje avtorja, naslova, založnika, jezika ali ključnih besed.
- Logično povezovanje. Če uporabnik znotraj ene zahteve poda več pogojev, model natančno ohrani njihove logične odvisnosti in jih poveže z ustreznimi Boolovimi operatorji (AND, OR, NOT).
- Normalizacija vrednosti. Model prepoznane vrednosti standardizira — osebna imena pretvori v osnovno obliko (imenovalnik), prepozna in pravilno zapiše časovne intervale (npr. »izdano leta 2000« pretvori v ustrezen operator enakosti) ter izraze ustrezno razvrsti.

Pomemben del implementacije predstavlja strogo usmerjanje jezikovnega modela prek sistemskih navodil (angl. *system prompt*). Z natančno definiranimi pravili, kontekstualnimi omejitvami in natančnim opisom dovoljenih operacij je model usmerjen tako, da iz uporabniške zahteve izlušči izključno podatke, ki jih je uporabnik dejansko navedel. Modelu je prek sistemskega poziva izrecno prepovedano kakršno koli sklepanje ali dopolnjevanje manjkajočih informacij, rezultat njegove obdelave pa je izključno čista, izvorna iskalna poizvedba (angl. *native query*) brez dodatnega pojasnjevalnega besedila ali klepetalnih razlag.

Tako pripravljena in strukturirana poizvedba se nato posreduje obstoječemu, uveljavljenemu iskalnemu jedru sistema COBISS. Ta pristop prinaša ključno arhitekturno prednost: umetna inteligenca se uporablja izključno kot vmesnik za razumevanje uporabnikove namere (angl. *intent recognition*), medtem ko iskanje, filtriranje in vračanje zadetkov še vedno izvaja preverjen in zanesljiv iskalni sistem. S tem je zagotovljena 100-odstotna združljivost z

obstoječimi indeksi in poslovno logiko COBISS-a, popolna odsotnost tveganja za halucinacije modela glede obstoja gradiva in hkrati izrazito poenostavljena uporabniška izkušnja.

Slika 7: Potek obdelave poizvedbe v naravnem jeziku

Proces transformacije nazorno ponazarja naslednji scenarij:

1. Uporabnikov vnos: »Poišči knjige Ivana Cankarja, izdane leta 2000.«
2. Obdelava z lokalnim LLM-jem. Model analizira vnos na podlagi pravil v sistemskem pozivu, prepozna avtorja ter časovni pogoj in generira čisto bibliografsko poizvedbo: (cankar ivan)/AX AND PY=2020
3. Izvedba. Nastala poizvedba se v ozadju neposredno izvede nad bibliografsko zbirko, sistem pa uporabniku vrne natančne zapise, ki ustrezajo vsem podanim kriterijem.

5 Pridobljene izkušnje in praktične ugotovitve

Predstavljene implementacije jasno potrjujejo, da izbrana skupna arhitektura omogoča razširljiv razvoj raznolikih AI-storitev za podporo procesom v sistemu COBISS. Kljub specifičnim zahtevam posameznih primerov uporabe se je pri razvoju izkazalo, da si vse rešitve delijo sorodne tehnične izzive. Ti so povezani predvsem z integracijo lokalnih jezikovnih modelov, učinkovitim upravljanjem strojnih virov, inženiringom pozivov (angl. *prompt engineering*) ter zagotavljanjem zanesljivosti in determinističnosti rezultatov v produkcijskem okolju.

5.1. Izbira in kakovost lokalnih modelov

Ena ključnih ugotovitev pri razvoju rešitev je, da izbira ustreznega jezikovnega modela bistveno vpliva na kakovost končnih rezultatov. Med razvojem smo preizkusili več odprtokodnih modelov (družine Qwen, Llama in Mistral, modela Gemma in GaMS …). Ker so se modeli razlikovali po zmogljivosti in primernosti za različne naloge, smo izbiro vedno prilagodili konkretni funkcionalnosti.

Večji modeli z več parametri na splošno zagotavljajo kakovostnejše odgovore in boljše razumevanje kompleksnih nalog, čeprav velikost sama po sebi ne zadošča za doseganje zadovoljivih rezultatov. Enako pomembna sta se izkazala ustrezno zasnovano sistemsko navodilo in kakovost vhodnega konteksta. Pri tem je bilo treba posebno pozornost nameniti omejitvi kontekstnega okna, saj preseganje največjega dovoljenega števila vhodnih žetonov (angl. *tokens*) onemogoči pravilno obdelavo zahteve.

Pri izbiri modelov so pomembno vlogo igrale njihove specifične lastnosti:

- Večjezičnost. Za večjezične funkcionalnosti, kot je AI-klepetalnik v aplikaciji COBISS Lib, se je družina modelov Qwen izkazala za primernejšo od drugih, saj je zagotavljala kakovostnejše odgovore v različnih jezikih.
- Način razmišljanja (angl. thinking mode). Pri zahtevnejših nalogah lahko ta način bistveno izboljša kakovost rezultatov, čeprav podaljša čas generiranja odgovorov.
- Optimizacija virov. Največji razpoložljivi model ni vedno najboljša izbira. Če manjši model dosega primerljivo kakovost, pogosto predstavlja primernejšo rešitev za produkcijsko okolje, in sicer zaradi nižje porabe pomnilnika, hitrejšega izvajanja in večje prepustnosti sistema.

Univerzalnega jezikovnega modela, ki bi bil optimalen za vse primere uporabe, ni. Učinkovita uporaba lokalnih modelov zahteva izbiro glede na zahteve posamezne naloge, pri čemer je poleg zmogljivosti modela treba upoštevati tudi kakovost navodil (angl. *prompt*), pripravo konteksta in razpoložljive strojne vire. Uspešna uvedba lokalnih modelov je rezultat usklajenega delovanja modela, ustrezno zasnovanih navodil in premišljene sistemske arhitekture.

5.2. Tehnični izzivi pri razvoju in uvajanju rešitev

Poleg izbire ustreznih modelov so pomemben del razvoja predstavljali tehnični izzivi, povezani z njihovim učinkovitim izvajanjem, stabilnostjo infrastrukture in zagotavljanjem strukturiranih izhodov.

Prvi večji izziv je bil povezan z upravljanjem več modelov na isti strojni infrastrukturi. Privzeto okolje Ollama modele dinamično nalaga v pomnilnik in jih po določenem času neaktivnosti odstrani. Če je bilo zaradi omejene količine grafičnega pomnilnika (VRAM) hkrati uporabljenih več različnih modelov, so se pojavljale situacije, ko je sistem že naloženi model odstranil in ga zamenjal z drugim, ki ga je zahtevala nova poizvedba. To je povzročalo nepredvidljivo obnašanje in občutno daljše odzivne čase pri sočasni uporabi različnih funkcionalnosti.

Za odpravo teh težav smo infrastrukturo prilagodili tako, da posamezne storitve Ollama izvajajo vnaprej določene modele na točno določenih grafičnih procesorjih (GPU), kot je prikazano na sliki 1. S to segmentacijo smo dosegli predvidljivo razporeditev virov, preprečili nepotrebno ponovno nalaganje modelov in izboljšali splošno odzivnost.

Dodatno težavo je predstavljalo generiranje strukturiranih odgovorov v obliki JSON. Kljub ustrezno oblikovanim sistemskim navodilom modeli niso vedno vrnili veljavne sintakse, zato smo implementirali mehanizem za dodatno validacijo odgovorov. Če sistem zazna neveljaven odgovor, samodejno izvede ponovno poizvedbo z dodatnimi navodili, ki model usmerijo k popravku strukture. Takšen večstopenjski pristop se je izkazal za bistveno zanesljivejšega od zanašanja na enkratno generiranje.

Pri razvoju smo se soočali tudi s hitrimi spremembami v AI-ekosistemu. Posodobitve okolja Ollama in ustreznih knjižnic so občasno spremenile privzeto delovanje sistema ali prepisale obstoječe konfiguracije. Izrazit primer je bila sprememba obnašanja ogrodja Vulkan, zaradi katere se modeli niso več nalagali na vnaprej določeno kartico GPU, temveč so se razporedili čez vse razpoložljive naprave. To je zahtevalo takojšnje prilagajanje konfiguracijskih datotek in uvedbo protokola za ponovno preverjanje celotne infrastrukture po vsaki večji posodobitvi knjižnic.

6 Zaključek

Izkušnje pri razvoju in uvajanju rešitev na podlagi lokalnih velikih jezikovnih modelov v sistemu COBISS potrjujejo, da so odprtokodni modeli dovolj zmogljivi za uporabo v produkcijskih okoljih. Pri tem so se kot ključni dejavniki uspeha izkazali premišljena izbira modelov glede na posamezen primer uporabe, kakovostno načrtovanje sistemskih navodil ter zanesljiva in razširljiva infrastruktura.

Razvoj predstavljenih rešitev je prinesel tudi številne tehnične izzive, povezane z upravljanjem strojnih virov, konfiguracijo izvajalnega okolja in zagotavljanjem zanesljivih izhodov. Njihovo reševanje je omogočilo oblikovanje enotne arhitekture, ki predstavlja trdno osnovo za nadaljnji razvoj in širšo uporabo umetne inteligence v sistemu COBISS.

Nadaljnji razvoj bo usmerjen v posodobitev infrastrukture z uvedbo ogrodij vLLM in LiteLLM ter uporabo nove strežniške opreme z zmogljivejšimi grafičnimi procesorji. vLLM je ogrodje, optimizirano za učinkovito izvajanje velikih jezikovnih modelov, ki omogoča hitrejše generiranje odgovorov in boljšo izrabo pomnilnika grafičnih procesorjev [5]. LiteLLM pa predstavlja enoten vmesnik za povezovanje in upravljanje različnih jezikovnih modelov ter ponudnikov LLM-storitev, kar poenostavlja integracijo, preklapljanje med modeli in njihovo centralizirano upravljanje [6].

S tem pričakujemo učinkovitejše izvajanje modelov, boljšo izrabo strojnih virov in podporo večjim jezikovnim modelom, kar bo omogočilo razvoj novih in zahtevnejših funkcionalnosti. Tako bomo lahko razvijali naprednejše rešitve, ki bodo še učinkoviteje podpirale delo knjižničarjev in uporabnikov sistema COBISS.

Literatura

[1] AMBROŽIČ, Vojko, KRAJNC, Andrej, ŠTOK, Bojan. Primer razvoja pametnega asistenta. V: PAVLIČ, Luka (ur.), BERANIČ, Tina (ur.), HERIČKO, Marjan (ur.). OTS 2024 : sodobne informacijske tehnologije in storitve : zbornik 27. konference : [Maribor, 4. in 5. september 2024]. 1. izd. Maribor: Univerza v Mariboru, Univerzitetna založba, 2024. Str. 15-24, ilustr. ISBN 978-961-286-893-2. https://press.um.si/index.php/ump/catalog/book/903/chapter/68, DOI: 10.18690/um.feri.4.2024.2

[2] KRAJNC, Andrej, AMBROŽIČ, Vojko, ŠTOK, Bojan. Uporaba javanske knjižnice za velike jezikovne modele. V: PAVLIČ, Luka (ur.), BERANIČ, Tina (ur.), HERIČKO, Marjan (ur.). OTS 2025 : sodobne informacijske tehnologije in storitve : zbornik 28. konference : [Maribor, 3. in 4. september 2025]. 1. izd. Maribor: Univerza v Mariboru, Univerzitetna založba, 2025. Str. 175-186, ilustr. ISBN 978-961-299-036-7. https://press.um.si/index.php/ump/en/catalog/book/1000/chapter/788, DOI: 10.18690/um.feri.7.2025

[3] Quarkus, https://quarkus.io/, obiskano 29. 7. 2026

[4] LangChain4j, https://github.com/langchain4j/langchain4j/, obiskano 29. 7. 2026

[5] vLLM, https://vllm.ai/, obiskano 29. 7. 2026

[6] LiteLLM, https://www.litellm.ai/, obiskano 29. 7. 2026