Kako ukrotiti velik jezikovni model nad lokalnim korpusom

Vili Podgorelec, Tadej Lahovnik, Grega Vrbančič

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko,
Maribor, Slovenija
vili.podgorelec@um.si, tadej.lahovnik1@um.si, grega.vrbancic@um.si

V prostrani, neukročeni divjini umetne inteligence se je pojavila in postavila v ospredje nova generativna vrsta: veliki jezikovni modeli. Ti orjaki s svojimi milijardami parametrov tavajo po podatkovni pokrajini, lačni vzorcev in vpogledov v obilje besedil. Toda izkoriščanje njihove moči ni enostaven podvig. Lahko so nepredvidljivi, nagnjeni k halucinacijam in se pogosto težko držijo teme. Predvsem pa jih ni preprosto udomačiti. V članku predstavljamo pristop, s katerim lažje ukrotimo te velikane z uporabo pomenskega povezovanja z lokalnim korpusom besedil. Pristop združuje surovo moč velikih jezikovnih modelov s specifičnostjo in ustreznostjo lokalnih podatkov. Raziskali bomo, kako tak pristop omogoči ne le učinkovito, temveč tudi vsebinsko specifično generiranje odgovorov, pri čemer so zagotovljene natančne in podrobne informacije o vašem specifičnem podjetju ali panogi. V članku poskušamo na preprost način predstaviti zapletenost tovrstnega procesa usposabljanja, vključno s strateško uporabo generiranja z razširjenim iskanjem, ki našemu modelu omogoča učinkovit dostop do lokalnih virov znanja. Če bomo v svojem namenu uspeli, vas prispevek ne bo opremil le s kakšnim novim spoznanjem, ampak boste tudi pripravljeni, da se podate na lastno pustolovščino z generativno umetno inteligenco. Na tej pustolovščini se vam rade volje pridružimo, da skupaj ukrotimo velikega dobrodušnega velikana. Naj se torej krotenje začne!

Ključne besede:
generativna umetna inteligenca
veliki jezikovni modeli
semantično povezovanje
vektorske vložitve
prilagoditev modela

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.1

1 Uvod

Generativna umetna inteligenca (GAI) in veliki jezikovni modeli (LLM), kot je npr. ChatGPT, so v zadnjih letih dosegli izjemen napredek in prepoznavnost. Ti modeli so usposobljeni na ogromnih korpusih besedil, kar jim omogoča izjemno natančno modeliranje in posledično sposobnost generiranja besedil v naravnem jeziku. Zaradi obsežnega učenja nad ogromno količino besedil so ti modeli pridobili široko in poglobljeno razumevanje jezika ter znanje s številnih področij, kar bi lahko primerjali z neko vrsto splošne izobrazbe. Njihova uporabnost se kaže na različnih področjih, predvsem pa so se izkazali pri aplikacijah za odgovarjanje na vprašanja v naravnem jeziku. Sistemi za odgovarjanje na vprašanja, ki temeljijo na LLM, omogočajo uporabnikom, da pridobijo hitre, natančne in kontekstno ustrezne odgovore na kompleksna vprašanja. Takšni sistemi so izjemno koristni na različnih področjih, vključno s tehnično podporo, izobraževanjem, raziskovanjem ali celo v zdravstveni oskrbi. Moč teh sistemov je v njihovi sposobnosti, da razumejo naravni jezik na visoki ravni in se prilagodijo specifičnim potrebam uporabnikov.

Kljub njihovi široki uporabnosti se pogosto pojavi potreba po tem, da se LLM osredotočijo na specifično področje znanja, ki je lahko predstavljeno v lokalnih dokumentih, do katerih model ni imel dostopa med svojim učenjem. Tradicionalni pristop bi zahteval ponovno učenje ali fino uglaševanje modela na novih podatkih, kar pa je proces, ki zahteva izjemne računske vire in je časovno zelo potraten. Ponovno učenje modela na specifičnih podatkih zahteva tudi poglobljeno tehnično znanje in dostop do zmogljive strojne opreme, kar ni vedno izvedljivo.

Da bi se izognili tem izzivom, lahko namesto tega uporabimo pristop, ki modelu omogoča dostop do lokalnih dokumentov brez ponovnega učenja. Tak pristop vključuje uporabo orodij, kot so vektorske baze podatkov in tehnike iskanja podobnosti, ki omogočajo modelu hitro pridobivanje relevantnih informacij iz lokalnih virov. Z vektorskimi bazami podatkov lahko dokumente pretvorimo v vektorske predstavitve, ki omogočajo hitro iskanje in pridobivanje najbolj relevantnih informacij na podlagi vprašanj uporabnikov. Ko LLM pridobi dostop do teh lokalnih virov, lahko informacije iz teh dokumentov spretno vključi v svoje odgovore. S tem pristopom lahko razvijalci informacijskih rešitev izkoristijo moč generativne AI, ne da bi pri tem potrebovali ogromne računske vire za ponovno učenje modelov. To omogoča učinkovitejše, lažje prilagodljive in predvsem bolj ekonomične rešitve za specifične potrebe uporabnikov. Poleg tega tak pristop omogoča hitrejše prilagajanje modela novim podatkom. Ker ni potrebe po dolgotrajnem in zahtevnem procesu ponovnega učenja, se lahko razvijalci hitro odzivajo na spremembe in posodobitve v lokalnih dokumentih. To je še posebej pomembno v dinamičnih okoljih, kjer se informacije pogosto spreminjajo in je hitra prilagodljivost ključna za zagotavljanje relevantnih odgovorov. Uporaba generativne umetne inteligence na ta način prinaša tudi dodatne prednosti, kot so izboljšana natančnost odgovorov in boljša uporabniška izkušnja. Model, ki lahko dostopa do specifičnih informacij v realnem času, je sposoben zagotavljati kontekstno ustreznejše in bolj specifične odgovore, kar povečuje zadovoljstvo uporabnikov in njihovo zaupanje v sistem.

Seveda pa je potrebno tovrsten pristop primerno zasnovati in ga ustrezno realizirati. V trenutni praksi namreč ostaja še veliko izzivov, kako navedene komponente povezati v zanesljiv in dobro delujoč sistem, pri tem pa se izogniti pastem, ki prežijo na razvijalce. V članku predstavljamo rešitev mISLec, ki smo jo zasnovali prav v duhu učinkovite uporabe velikih jezikovnih modelov nad specifičnimi zbirkami lokalnih dokumentov, pri čemer je rešitev karseda neodvisna od uporabljenega pred-naučenega LLM, uporabimo pa lahko tako obstoječe storitve preko klicev API, kot tudi lokalno nameščen veliki jezikovni model. S pravilno uporabo tehnologij in integracijo lokalnih virov informacij lahko ustvarimo zmogljive, prilagodljive in učinkovite sisteme za odgovarjanje na vprašanja v naravnem jeziku, ki ustrezajo specifičnim potrebam uporabnikov brez potrebe po obsežnem ponovnem učenju modelov.

2 Generativna umetna inteligenca in veliki jezikovni modeli

Ljudje uporabljamo jezik kot komunikacijski sistem za medsebojno posredovanje idej, čustev in informacij. Kot dojenčki zaznavamo smiselne pomene besed in z odraščanjem postanemo bolj spretni pri prilagajanju svojega govora. Jezik nam pomaga razumeti sebe in svet okoli nas. Edinstvenost človeka pri učenju jezika je sposobnost posploševanja in posledično učenja iz omejene izpostavljenosti jeziku, še posebej pri otrocih. Kar je praktično neprimerljivo s sodobnimi modeli umetne inteligence. Potrebna so bila desetletja raziskovanja in razvoja sistemov, ki bi lahko ustvarili človeku podobne odzive za naloge obdelave naravnega jezika (angl. Natural Language Processing, NLP), kot so pogovor, samo-dokončanje besedila in prevajanje jezika [1].

Generativna umetna inteligenca (angl. Generative Artificial Intelligence, GAI) in veliki jezikovni modeli (angl. Large Language Models, LLM) predstavljajo enega pomembnejših in odmevnejših napredkov na področju umetne inteligence, saj ponujajo zmogljivosti pri ustvarjalnih oz. generativnih nalogah, kot je na primer ustvarjanje vsebin, ter pri razumevanju naravnega jezika.

V zadnjih letih je generativna UI doživela bliskovit razvoj, ki ga poganjajo napredki v arhitekturah globokega učenja in razpoložljivost velikih podatkovnih zbirk ter računskih virov. Ta razvoj je privedel do pojava več-modalnih modelov, ki lahko hkrati obravnavajo besedilo in slike, kar dodatno širi obseg uporabe generativnih modelov UI. Poleg tega integracija nenadzorovanih in delno nadzorovanih algoritmov inteligentnim sistemom omogoča, da se samodejno odzivajo na ukaze in samostojno ustvarjajo vsebino, pri čemer vsebino črpajo iz predhodno ustvarjenih zbirk podatkov [2].

2.1. Generativna umetna inteligenca

Generativna umetna inteligenca je eno najhitreje razvijajočih se področij v sodobni digitalni pokrajini, ki izkazuje izjemno zmogljivost pri ustvarjanju visokokakovostne, kontekstualno ustrezne vsebine, ki je skorajda ne ločimo od tiste, ki jo ustvari človek. V obdobju, ko aplikacije, kot je ChatGPT, postavljajo rekorde za najhitreje rastočo uporabniško bazo z izkazovanjem znanja neodvisnega od domen, se GAI uveljavlja kot pomemben generator vsebin v digitalnem prostoru. Napredki na področju strojnega učenja (angl. Machine Learning, ML) in globokega učenja (angl. Deep Learning, DL) so omogočili razširitev tradicionalnih nalog umetne inteligence kot so regresija, klasifikacija in priporočila, na ustvarjanje edinstvene, realistične in kreativne vsebine [3].

GAI omogoča inovacije na različnih področjih, vključno s poslovnimi modeli in storitvami. Na primer, sistemi za podporo strankam lahko zdaj uporabljajo GAI za predlaganje ustreznih odzivov na pogovore, kar povečuje učinkovitost in kakovost storitev. Prav tako omogoča avtomatizacijo in optimizacijo procesov v podjetjih, kar lahko vodi do boljšega odločanja in izboljšanja operativnih učinkovitosti.

Najprepoznavnejše rešitve v okviru GAI vključujejo generativne modele, kot so GAN (angl. Generative Adversarial Networks), ki se uporabljajo za ustvarjanje realističnih slik (npr. rešitvi DALL-E [4] in Midjourney [5]) in videoposnetkov, ter transformerje, ki so temelj modelov, kot je ChatGPT [4], in se uporabljajo za ustvarjanje besedil. GAN modeli so pokazali izjemne rezultate na področju umetnosti, zabave in oblikovanja, kjer lahko ustvarjajo izjemno realistične in kreativne vizualne vsebine. Transformerji pa so revolucionirali področje obdelave naravnega jezika, kjer se uporabljajo za različne naloge, kot so prevajanje, povzemanje besedil in ustvarjanje pogovorov [2].

Raziskave in razvoj na področju GAI so osredotočeni na izboljšanje hitrosti, zmogljivosti in učinkovitosti teh modelov. Kljub temu pa ostajajo številna vprašanja o temeljnih načelih, aplikacijah in družbeno-ekonomskem vplivu GAI še vedno neraziskana. To predstavlja izziv, saj jasna podoba in razumevanje generativne umetne inteligence še nista dokončno oblikovana. GAI v splošnem prinaša številne priložnosti za inovacije in izboljšave v različnih industrijah, vendar je hkrati pomembno, da se zavedamo tudi izzivov, ki jih prinaša, ter si prizadevamo za odgovorno in etično uporabo te tehnologije.

2.2. Veliki jezikovni modeli

Veliki jezikovni modeli (angl. Large Language Models, LLM) so skupina pristopov temelječih na metodah in tehnikah globokega učenja, usposobljeni za razumevanje, generiranje in obdelavo besedil v naravnem jeziku. Med najbolj znanimi in uporabljenimi LLM je serija generativnih prednaučenih transformerjev (angl. Generative Pre-trained Transformer, GPT), ki vključuje različice, kot je na primer GPT-4, razvite s strani OpenAI. LLM so učeni na ogromnih količinah besedilnih podatkov, kar jim omogoča prepoznavanje kompleksnih vzorcev v jeziku ter generiranje koherentnih in kontekstualno ustreznih besedil. Osnovni princip delovanja velikih jezikovnih modelov temelji na uporabi transformerjev, napredne arhitekture nevronskih mrež, ki je bila prvič predstavljena v članku »Attention is All You Need« [6]. Tranformerji uporabljajo mehanizem pozornosti (angl. attention mechanism), ki omogoča modelu, da teži k pomembnim delom vhodnega besedila in jih poveže z ustreznimi izhodi. To bistveno izboljša zmogljivost modelov pri obdelavi zaporednih podatkov v primerjavi s predhodnimi pristopi, kot so rekurentne nevronske mreže (angl. Recurrent Neural Network, RNN) [7] in mreže z dolgim kratkoročnim spominom (angl. Long Short-Term Memory, LSTM) [8].

Učenje velikih jezikovnih modelov navadno poteka v dveh korakih: pred-učenje (angl. pretraining) in uglaševanje (angl. fine-tuning). V procesu predučenja je model izpostavljen ogromnim količinam besedil iz različnih virov, kot so knjige, spletni članki in druge oblike pisne komunikacije. V tem koraku se model uči jezikovne strukture, slovnice, konteksta in pomena besed ter fraz. Po pred-učenju je model sposoben generirati besedilo, ki je jezikovno pravilno in kontekstualno ustrezno, vendar morda ni specifično prilagojeno za reševanje določenih nalog. V procesu uglaševanja se model dodatno uči na manjših, bolj specializiranih sklopih podatkov, ki so prilagojeni specifičnim nalogam, kot so prevajanje, povzemanje besedil, odgovarjanje na vprašanja ali analiza sentimenta. To omogoča modelu, da izboljša svojo uspešnost in uporabnost pri reševanju specifičnih problemov.

Osnovni gradniki velikih jezikovnih modelov vključujejo več ključnih komponent. Med najpomembnejše sodita vektorizacija in vložitev (angl. embedding). Vektorizacija je proces pretvorbe besedilnih podatkov v številčne predstavitve, ki jih lahko obdelujejo nevronske mreže. Vložitev pa je tehnika, ki omogoča pretvorbo besed v vektorje z nižjo dimenzionalnostjo, kjer so besede s podobnimi pomeni bližje druga drugi v vektorskem prostoru. Te vektorske predstavitve omogočajo modelu razumevanje semantičnih odnosov med besedami in frazami, kar je ključnega pomena za generiranje kontekstualno ustreznega besedila. Uporaba vektorizacije in vložitve omogoča modelom, da bolje razumejo in obdelujejo kompleksne jezikovne strukture. Vektorizacija prav tako zagotavlja, da so vsi podatki predstavljeni v obliki, ki je primerna za matematično obdelavo, medtem ko vložitev omogoča učinkovitejše in natančnejše razumevanje pomena in konteksta besed. Na primer, model lahko prepozna, da sta besedi "kralj" in "kraljica" semantično povezani ter da imata podobne kontekste uporabe, kar omogoča generiranje bolj naravnih in koherentnih besedil.

Veliki jezikovni modeli imajo široko paleto aplikacij, ki segajo od avtomatskega prevajanja in generiranja besedil do analize sentimenta in pogovornih robotov. Njihova sposobnost razumevanja in generiranja naravnega jezika odpira številne možnosti za izboljšanje komunikacije med ljudmi in stroji. Vendar pa uporaba LLM prinaša tudi izzive, kot so vprašanja o pristranskosti, etičnih vidikih in vplivu na zasebnost. Kljub temu ostajajo veliki jezikovni modeli ena najbolj obetavnih tehnologij v sodobnem razvoju umetne inteligence, ki že zdaj pomembno vpliva na številna področja znanosti in industrije.

2.3. Vektorizacija, žetoni in vložitev

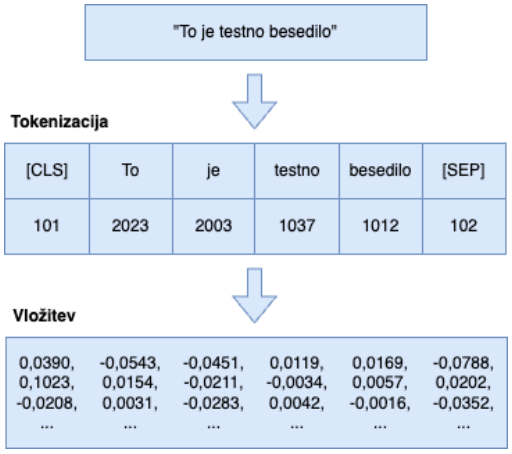
Vektorji igrajo ključno vlogo pri delovanju LLM in GAI. Da bi razumeli njihov pomen, je bistveno razumeti, kaj so vektorji in kako se ustvarjajo ter uporabljajo v LLM. V matematiki in fiziki je vektor objekt, ki ima velikost ter smer in se tipično uporabljajo za predstavitev količin, ki jih ni mogoče v celoti opisati z eno samo številko kot so sila, hitrost ali premik, saj imajo tako velikost kot tudi smer. Na področju LLM se vektorji uporabljajo za predstavitev besedila ali podatkov v numerični obliki, ki jo lahko model razume in obdela. Ta predstavitev je znana kot vložitev (angl. embedding). Vložitve so visokodimenzionalni vektorji, ki zajemajo semantični pomen besed,

stavkov, lahko tudi celotnih dokumentov. Postopek pretvorbe besedila v vložitev omogoča LLM-jem, da izvajajo različne naloge obdelave naravnega jezika, kot so ustvarjanje besedila, analiza občutkov in povzemanje besedil. Ker stroji razumejo samo številke, se podatki, kot so besedilo in slike, pretvorijo v številske vektorje [9].

Izvedba različnih operacij nad takšnimi vektorji, kot je na primer izračun kosinusne podobnosti, lahko uporabimo za namen iskanja oz. ugotavljanja podobnosti med vektorji oz. v primeru procesiranja naravnega besedila ugotavljanja podobnosti med besedami ali besedili. Vektorizacija in vložitev omogočata modelom, da razumejo in obdelujejo kompleksne jezikovne strukture. Prav tako pa zagotavlja, da so vsi podatki predstavljeni v obliki, ki je primerna za matematično obdelavo, medtem ko vložitev omogoča učinkovitejše in natančnejše razumevanje pomena in konteksta besed.

Žetoni (tokens) so osnovne enote podatkov, ki jih obdelujejo LLM-ji. V kontekstu besedila je žeton lahko beseda, del besede (koren besede) ali celo znak, odvisno od pristopa oz. načina tokenizacije. Ko je besedilo posredovano skozi tokenizator, se ustvarjeni žetoni zakodirajo na podlagi specifične sheme kodiranja v vektorje, ki jih LLM lahko razume. Z uporabo tokenizatorja vhodno besedilo razdelimo na besede, dele besed (npr. koren) ali celo posamezne znake, odvisno od uporabljenega tokenizerja. Ko gredo žetoni skozi dekoder, jih je mogoče enostavno znova prevesti v besedilo. Običajno se dolžina konteksta LLM-jev nanaša na enega ključnih dejavnikov razlikovanja med različicami LLM. Tehnično gledano gre za sposobnost LLM, da sprejme določeno število žetonov kot vhod in ustvari drug niz žetonov kot izhod. Tokenizator je prav tako odgovoren za kodiranje poziva LLM (vnosa) v žetone in odgovora LLM (izhoda) nazaj v besedilo [10].

Z vložitvijo dosežejo LLM globoko razumevanje jezika, kar omogoča različne naloge, kot so analiza sentimenta, povzemanje besedila in odgovarjanje na vprašanja z različnimi nivoji razumevanja in zmožnostmi ustvarjanja. So vstopna točka v LLM, vendar se uporabljajo tudi zunaj LLM za pretvorbo besedila v vektorje ob ohranjanju semantičnega konteksta. Na Slika 1 je prikazan postopek tokenizacije in vložitve ene povedi.



Slika 1: Primer tokenizacije in vložitve.

Najpogostejši pristopi k vektorizaciji in vložitvam vključujejo različne tehnike. Ena izmed najpogostejših tehnik je uporaba Word2Vec, ki pretvori besede v vektorje tako, da upošteva njihov kontekst v besedilu. Metoda GloVe (angl. Global Vectors for Word Representation) je še ena priljubljena tehnika, ki temelji na statističnih podatkih o sočasnem pojavljanju besed v besedilu. BERT (angl. Bidirectional Encoder Representations from Transformers) je sodobnejša tehnika, ki uporablja dvostransko kodiranje za razumevanje pomena besed z upoštevanjem konteksta na levi in desni strani besede [10].

Vektorizacija, žetoni in vložitve so tako temeljni gradniki, ki omogočajo delovanje velikih jezikovnih modelov in so ključni za razumevanje ter generiranje naravnega jezika.

2.4. Pomensko povezovanje besedil (RAG)

Retrieval-augmented generation (RAG) predstavlja inovativni pristop k izboljšanju delovanja LLM z namenom, da bi ti lahko odgovarjali na specifična vprašanja v spremenljivem kontekstu. Temelj vseh temeljnih modelov, vključno z LLM-ji, je arhitektura transformatorjev, ki pretvarja ogromne količine surovih podatkov v stisnjene reprezentacije njihove osnovne strukture. Ta osnovna reprezentacija omogoča prilagoditev modela različnim nalogam z dodatnim finim uglaševanjem na označenem, domensko specifičnem znanju.

Pred uvedbo LLM-jev so namreč digitalni pogovorni agenti sledili ročno določenemu toku dialoga, kjer so potrdili namen sogovornika, pridobili zahtevane informacije in dostavili odgovor v obliki univerzalne šablone. Za enostavna vprašanja je ta metoda delovala dobro, vendar je imela svoje omejitve. Pričakovanje in pisanje odgovorov na vsako možno vprašanje, ki bi ga stranka lahko zastavila, je bilo zamudno. Če scenarij za neko vprašanje ni obstajal, pogovorni robot ni imel funkcionalnosti improvizacije [11].

Kljub uporabi LLM pa fino uglaševanje modelov redko zagotavlja popolno pokritost znanja, potrebnega za odgovarjanje na zelo specifična vprašanja v dinamičnem okolju. Leta 2020 je Meta (takrat znana kot Facebook) predstavila ogrodje, imenovano retrieval-augmented generation (RAG) [12], da bi LLM-jem omogočila dostop do informacij, ki presegajo njihove učne podatke. RAG omogoča LLM-jem, da se opirajo na specializirano znanje in tako odgovorijo na vprašanja na bolj natančen način. V RAG sistemu model odgovarja na vprašanja z iskanjem in brskanjem po vsebini v knjigi, namesto da bi se zanašal zgolj na svoje spominske zmožnosti.

RAG vključuje dve fazi: pridobivanje in generiranje vsebine. V fazi pridobivanja algoritmi iščejo in pridobivajo delčke informacij, ki so relevantni za uporabnikov poziv ali vprašanje. V odprtem okolju lahko te informacije prihajajo iz indeksiranih dokumentov na spletu ali iz drugih virov, v zaprtem podjetniškem okolju pa se običajno uporablja ožji nabor internih virov zaradi večje varnosti in zanesljivosti. To zunanjo znanje se nato »priloži« uporabnikovemu pozivu in posreduje jezikovnemu modelu. V fazi generiranja LLM črpa iz obogatenega poziva in svoje notranje reprezentacije učnih podatkov, da ustvari odgovor, ki je prilagojen uporabniku v tistem trenutku. Odgovor se nato posreduje klepetalnemu botu skupaj s povezavami do virov [11]. Poenostavljen koncept RAG je predstavljen na sliki Slika 2.

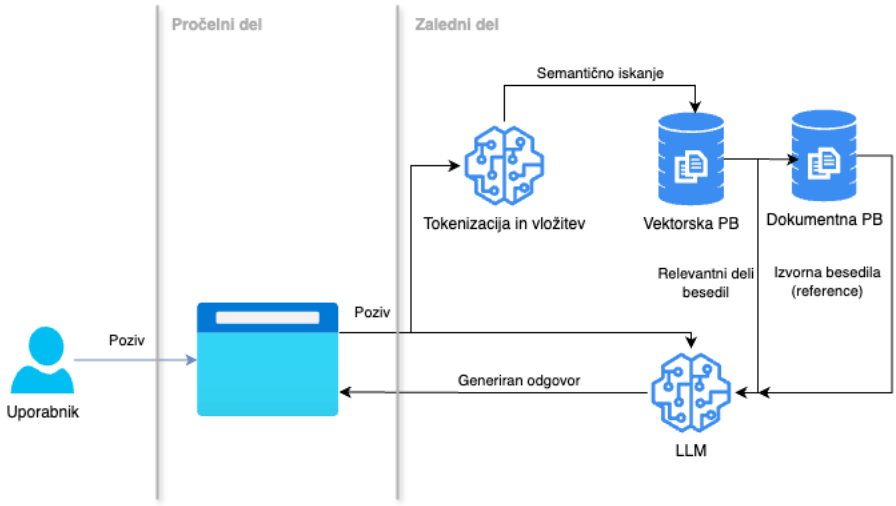
Slika 2: Poenostavljen koncept delovanja RAG.

RAG je trenutno najbolj znano orodje za povezovanje LLM-jev z najnovejšimi, preverljivimi informacijami in zmanjševanje stroškov nenehnega ponovnega učenja in posodabljanja modelov. RAG se zanaša na sposobnost obogatiti uporabniške pozive z relevantnimi informacijami, ki so shranjene v vektorjih, matematičnih predstavitvah podatkov. Vektorske baze podatkov lahko učinkovito indeksirajo, shranjujejo in pridobivajo informacije za stvari, kot so priporočilni sistemi in klepetalni boti. Vendar pa RAG ni popoln in še vedno ostajajo številni izzivi pri pravilnem izvajanju RAG. Eden ključnih je zagotovo na kak način učinkovito in efektivno pridobiti relevantne dele zunanjih podatkov oz. kako izbrati, katere informacije priložiti uporabniškemu pozivu.

3 Zasnova rešitve

Rešitev smo zasnovali z mislimi na čim enostavnejšo integracijo različnih lastnih domenskih tekstovnih korpusov, pri čemer je cilj, da je razvita rešitev neodvisna od uporabljanega velikega jezikovnega modela. Ključ pri vpeljavi lastnih domenskh korpusov je v uporabi ogradja RAG in inteligentni integraciji rezultatov tega z uporabniškim pozivom. Glede na podane želje oziroma zahteve smo zasnovali konceptualno zasnovo naše rešitve, kot je prikazano na sliki Slika 3. Po podanem uporabniškem pozivu se ta preko tokenizacije in vložitve pretvori v vektorje, ki se uporabijo za namen semantičnega iskanja podobnosti z obstoječimi vloženimi vektorji v vektorski podatkovni bazi. Vsak vgrajen vektor poseduje tudi referenco na izvoren dokument hranjen v dokumentni podatkovni bazi. Z rezultati semantičnega iskanja obogaten poziv se nato posreduje uporabljenemu LLM, ki lahko deluje lokalno ali pa v obliki REST API. Odgovor, generiran s strani LLM, posredujemo uporabniku.

Predstavljena konceptualna zasnova rešitve se navezuje na ključno funkcionalnost – pogovornega robota, obogatenega s specifičnim domenskim znanjem v obliki poljubno sestavljenih besedilnih korpusov, pri čemer ni potrebno LLM dodatno uglaševati ali učiti. V ta namen rešitev poleg osnovnje predstavljene funkcionalnosti vključuje tudi funkcionalnosti ustvarjanja različnih korpusov, tokenizacije in vgrajevanja podanih besedil v vektorsko podatkovno bazo, itd.



Slika 3: Konceptualna zasnova rešitve.

4 Rešitev mISLec

V tem poglavju bomo podrobneje predstavili razvito rešitev. Pročelje (angl. *front-end*) rešitve mISLec temelji na ogrodju React, ki omogoča razvoj ponovno uporabljivih komponent. Zaledje (angl. *back-end*) je implementirano v programskem jeziku Python in temelji na ogrodju FastAPI. Podatki in naloženi dokumenti se hranijo v NoSQL podatkovni bazi MongoDB, ki omogoča učinkovito hrambo tako strukturiranih kot tudi nestrukturiranih podatkov.

4.1. Funkcionalnosti in delovanje rešitve

Razvito rešitev sestavlja več komponent, ki omogočajo interakcijo z velikimi jezikovnimi modeli in delo z lokalnimi korpusi. Glavne komponente rešitve so:

- **navigacijski meni**, ki omogoča dostop do različnih delov rešitve,
- **upravljanje z dokumenti**, ki uporabnikom omogoča nalaganje dokumentov, pregled vseh naloženih dokumentov in urejanje vsebine (odstranjevanje odvečnih informacij, kot je npr. vsebina v glavi in nogi dokumenta),
- **vektorizacija dokumentov**, ki velikim jezikovnim modelom omogoča učinkovito obdelavo in razumevanje vsebine dokumentov,
- **upravljanje z lokalnimi korpusi**, ki uporabnikom omogoča ustvarjanje lokalnih korpusov, dodajanje dokumentov v lokalni korpus in vektorizacijo vseh dokumentov v lokalnem korpusu,
- **upravljanje z uporabniki**, ki je na voljo administratorjem in jim omogoča pregled vseh uporabnikov ter spreminjanje njihovih vlog v sklopu rešitve mISLec, in
- **interakcija z velikimi jezikovnimi modeli**, ki uporabnikom omogoča uporabo lokalnega korpusa in komunikacijo z izbranim modelom.

Vektorizacija dokumentov se izvede nad tekstovno vsebino naloženega dokumenta. Ob samem nalaganju dokumenta se njegova vsebina razdeli na manjše enote, tj. posamezne stavke, ki predstavljajo vhod za operacijo vektorizacije. Ta je implementirana s pomočjo knjižnic *transformers* in *torch*. Knjižnica *transformers* posamezne stavke tokenizira, knjižnica *torch* pa ustvarjene žetone (angl. *token*) pretvori v vektorsko obliko, nato pa vektorje še normalizira. Ustvarjeni vektorji se nato shranijo v vektorsko bazo Qdrant, ki omogoča visoko zmogljivo vektorsko iskanje v velikem obsegu. Slika 4 prikazuje implementacijo vektorizacije dokumentov.

```python
def embed_sentences(document: Document) -> Document:
    encoded_input = tokenizer(
        document['raw_sentences_text'], padding=True, truncation=True, return_tensors='pt')

    with torch.no_grad():
        model_output = model(**encoded_input)
        sentence_embeddings = model_output[0][:, 0]
    sentence_embeddings = torch.nn.functional.normalize(
        sentence_embeddings, p=2, dim=1)

    document['embeddings'] = {
        'sentence_embeddings': sentence_embeddings
    }

    operation_info = qdrant_client.upsert(
        collection_name=COLLECTION_NAME,
        wait=True,
        points=models.Batch(
            ids=[str(uuid.uuid4()) for _ in document['raw_sentences_text']],
            payloads=[
                {
                    'text': sentence,
                    'document_id': str(document['_id']),
                    'user_id': str(document['user']),
                }
                for sentence in document['raw_sentences_text']
            ],
            vectors=[embedding for embedding in sentence_embeddings],
        ),
    )

    return document
```

Slika 4: Vektorizacija dokumentov.

Po vektorizaciji se lahko dokumenti uporabijo pri interakciji z velikimi jezikovnimi modeli. Ob uporabi lokalnih korpusov se na zaledje ob pozivu (angl. *prompt*) posreduje tudi seznam vključenih dokumentov. Poziv se nato vektorizira in normalizira ter uporabi za iskanje *top_k* (število, ki ga uporabniki definirajo v nastavitvah) podobnih stavkov iz dokumentov v lokalnem korpusu, kot je prikazano na sliki Slika 5.

```python
def get_similar_sentences(sentences: list[str], user_id: str, corpus: list[str], top_k: int = 5) -> dict:
    encoded_input = tokenizer(sentences, padding=True,
                              truncation=True, return_tensors='pt')

    with torch.no_grad():
        model_output = model(**encoded_input)
        sentence_embedding = model_output[0][:, 0]

    sentence_embeddings = torch.nn.functional.normalize(
        sentence_embedding, p=2, dim=1)

    sentence_embeddings_list = sentence_embeddings.tolist()

    filter_query = models.Filter(
        must=[
            models.FieldCondition(
                key='document_id',
                match=models.MatchAny(any=corpus)
            )
        ]
    )

    search_queries = [models.SearchRequest(
        vector=embedding, filter=filter_query, limit=top_k, with_payload=True) for embedding in sentence_embeddings_list]

    search_results = qdrant_client.search_batch(
        collection_name=COLLECTION_NAME,
        requests=search_queries,
    )

    flattened_search_results = [
        item for sublist in search_results for item in sublist]

    filtered_search_results = sorted(
        flattened_search_results, key=lambda x: x.score)

    return filtered_search_results
```

Slika 5: Iskanje podobnih stavkov.

Pridobljeni podobni stavki se nato uporabijo pri inženiringu poziva (angl. *prompt engineering*), kot prikazuje slika Slika 6. Najprej se definirajo navodila za generiranje odziva. Če uporabnik le-teh ni spremenil, se uporabijo privzeta navodila. Nato se navodilom pripne uporabnikova poizvedba in navodila, ki se navezujejo na pričakovan format odgovora. Na koncu se pripnejo še pridobljeni podobni stavki, iz katerih veliki jezikovni model črpa znanje.

```python
def build_prompt(question: str, references: list, custom_prompt: str):
    default_prompt = f"""
    You are a machine learning researcher.
    Your answers should be exact with given explanation in such way
    that non-experts can understand you. Keep your answer concise and to the point.
    """

    if custom_prompt:
        instructions = custom_prompt
    else:
        instructions = default_prompt

    prompt = f"""
    {instructions}
    Respond to the following prompt:
    {question}
    """

    prompt += """
    Format your response using Markdown syntax. Do not use headings.
    """

    if len(references) > 0:
        prompt += """
        You've selected the most relevant passages from your writings to use
        as source for your answer. Omit citations and references from your answer.
        """

    for _, reference in enumerate(references, start=1):
        text = reference.payload['text']
        prompt += f"- {text}\n"

    document_ids = list(set([reference.payload['document_id']
                             for reference in references]))

    return prompt, document_ids
```

Slika 6: Inženiring poziva.

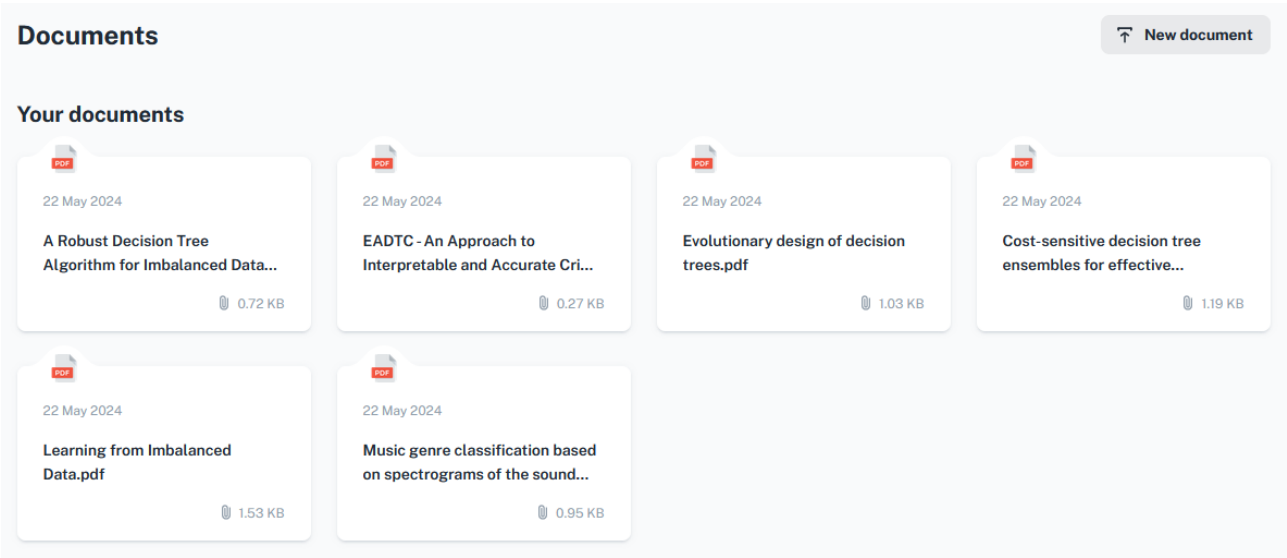
Slika 7 prikazuje celoten postopek obdelave poslanega sporočila, ki vključuje vektorizacijo poziva, iskanje podobnih stavkov, inženiring poziva in pripravo podatkov za komunikacijo z LLM (prikazan je primer uporabe Cohere API; na podoben način lahko uporabimo klice drugih API-jev ali lokalno naložen model LLM), ki vrne odgovor na uporabnikovo vprašanje.

```python
async def send_message(chat, corpus, token, custom_prompt, collection, database):
    co = cohere.Client(COHERE_API_KEY)

    messages = [{'role': message.role, 'text': message.text}
                for message in chat.messages]

    references = []
    if (len(corpus) != 0):
        last_message = messages[-1]['text']
        last_message_sentences = re.split('[.!?]', last_message)
        similar_sentences = get_similar_sentences(
            last_message_sentences, str(token['user']['_id']), corpus, top_k=token['user']['similarity'])
        prompt, references = build_prompt(
            last_message, similar_sentences, custom_prompt)
    else:
        prompt, _ = build_prompt(chat.messages[-1].text, [], custom_prompt)

    response = co.chat(
        message=prompt,
        chat_history=messages[:-1]
    )
```

Slika 7: Pošiljanje sporočila.

4.2. Primeri delovanja

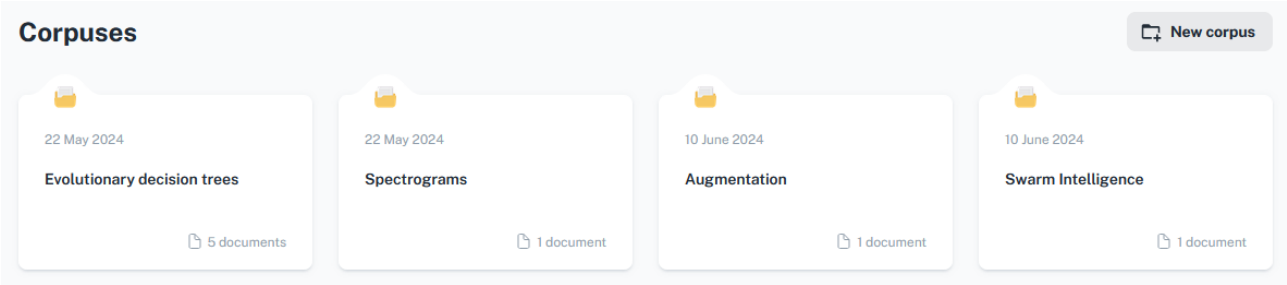
Pred začetkom uporabe lokalnih korpusov v pogovorih z velikimi jezikovnimi modeli je treba izvesti več korakov za pripravo teh korpusov. Najprej je treba naložiti dokumente, ki vsebujejo relevantne informacije. Po uspešnem nalaganju se bodo dokumenti prikazali na nadzorni plošči, kot je prikazano na sliki Slika 8. Ko so dokumenti naloženi, se dokumenti vektorizirajo. Ta korak je ključnega pomena, saj modelom omogoča učinkovito obdelavo in razumevanje vsebine dokumentov.



Slika 8: Pregled naloženih dokumentov.

Po uspešni vektorizaciji sledi združevanje dokumentov v lokalne korpuse. Uporabniki lahko ustvarijo več različnih korpusov, ki pokrivajo različna domenska področja. Po uspešnem ustvarjanju se korpusi prikažejo na nadzorni

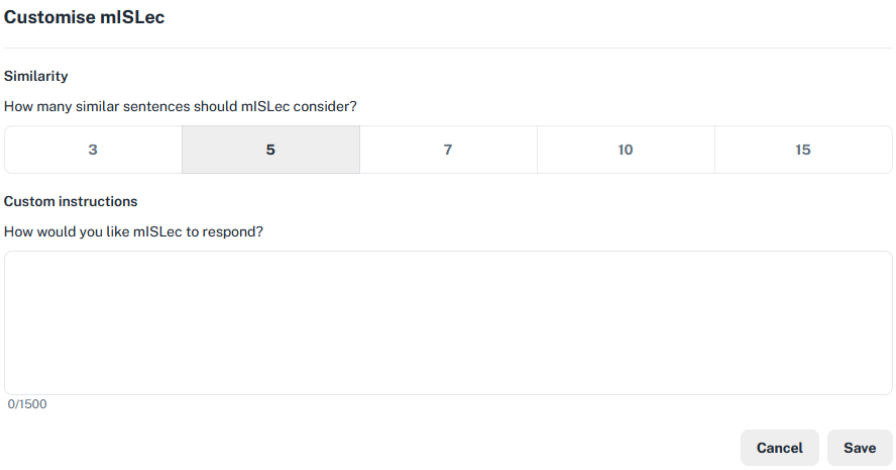
plošči, kot je prikazano na sliki Slika 9. V vsakem pogovoru lahko izberejo natanko en lokalni korpus, iz katerega bo veliki jezikovni model črpal znanje.



Slika 9: Pregled ustvarjenih korpusov.

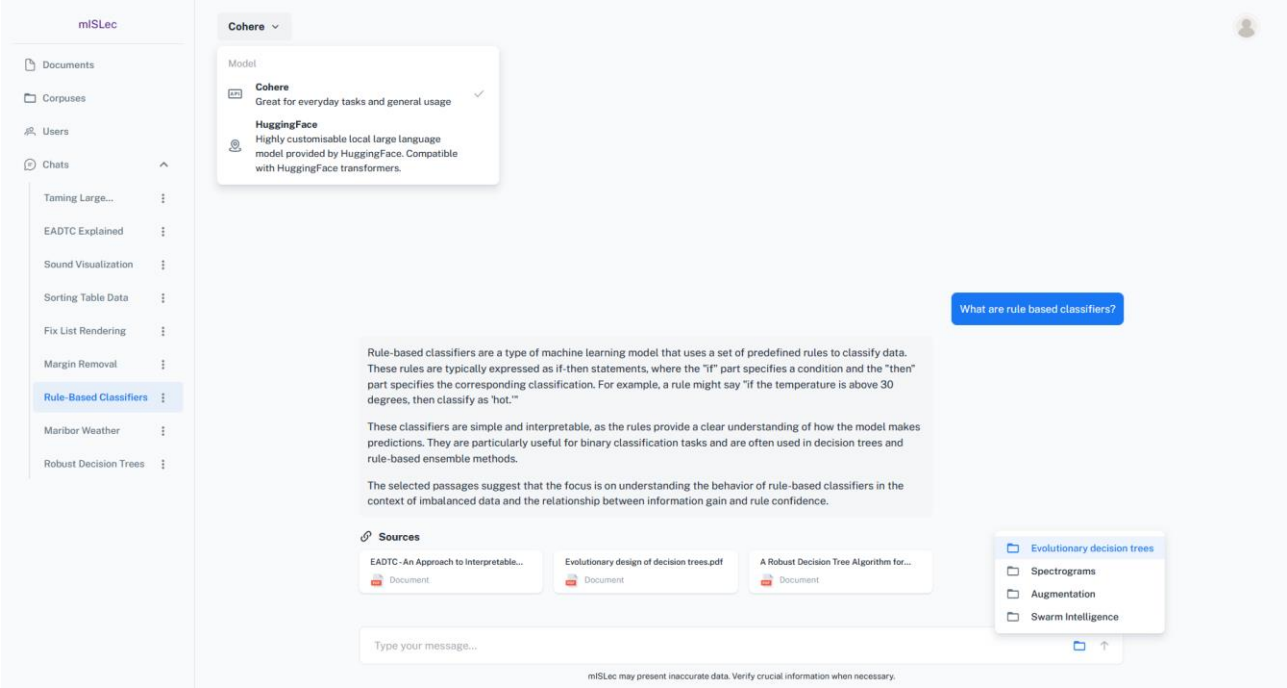
Rešitev mISLec je prilagodljiva in uporabnikom ponuja personalizacijo. Slika 10 prikazuje nastavitve, ki jih uporabniki lahko spreminjajo in tako optimizirajo svoje izkušnje z rešitvijo glede na specifične zahteve in želje. Možne nastavitve vključujejo:

- število stavkov, ki jih bo rešitev analizirala pri iskanju odgovorov v lokalnih korpusih, kar omogoča natančnejšo prilagoditev globine in širine iskanja, in
- prilagojena navodila za generiranje odziva, kot so dodelitev vlog (npr. svetovalec, učitelj, tehnična podpora itd.) ali osebnostnih karakteristik (npr. formalen, prijazen, strokoven itd.), kar omogoča ustvarjanje odgovorov, ki so bolj v skladu z željenim tonom in slogom komunikacije.



Slika 10: Personalizacija rešitve mISLec.

Slika 11 prikazuje primer interakcije med uporabnikom in rešitvijo mISLec. V levem zgornjem kotu lahko opazimo vse velike jezikovne modele, ki so uporabniku na voljo, kar omogoča preglednost in enostavno izbiro želenega modela. V osrčju slike je prikaz tekstovnih sporočil, izmenjanih med uporabnikom in velikim jezikovnim modelom, ki vizualno ponazarjajo dialog in potek komunikacije. Sporočila so jasno razporejena, kar omogoča enostavno sledenje pogovoru. Poleg tega je v desnem spodnjem kotu slike prikazana možnost izbire lokalnega korpusa. Če uporabnik ob pošiljanju sporočila izbere lokalni korpus, bo v odgovoru, ki ga prejme, vključen tudi seznam virov, iz katerih je veliki jezikovni model črpal znanje. Ta funkcionalnost je uporabna za preverjanje zanesljivosti in izvora informacij, saj uporabniku omogoča dostop do natančnih referenc. Celoten prikaz interakcije poudarja uporabniku prijazno zasnovo rešitve mISLec, ki omogoča učinkovito in pregledno komunikacijo z jezikovnimi modeli ter dodatno zagotavlja transparentnost z vključitvijo virov informacij.



Slika 11: Primer interakcije med uporabnikom in rešitvijo mISLec.

5 Zaključek

V članku smo predstavili pristop krotitve velikih jezikovnih modelov nad lokalnim korpusom besedil z uporabo ogrodja za pomensko povezovanje besedil RAG, s katerim smo omogočili splošno naučenemu LLM dostop do novih domensko specifičnih informacij, pri čemer se izognemo kakršnemukoli dodatnemu učenju ali uglaševanju LLM. Očitna prednost takšnega pristopa je torej uporaba splošnih LLM modelov, brez potrebe po dodatnem učenju, po drugi strani pa je ključen izziv takšnega pristopa, na kak način poiskati katere informacije in koliko le-teh priložiti uporabniškemu pozivu, da bo generiran odziv LLM čim bolj zadovoljiv.

Literatura

[1] F. Barreto, L. Moharkar, M. Shirodkar, V. Sarode, S. Gonsalves, and A. Johns, "Generative Artificial Intelligence: Opportunities and Challenges of Large Language Models," *Lecture Notes in Networks and Systems*, vol. 699 LNNS, pp. 545–553, 2023, doi: 10.1007/978-981-99-3177-4_41.

[2] A. A. Linkon *et al.*, "Advancements and applications of generative artificial intelligence and large language models on business management: A comprehensive review," *Journal of Computer Science and Technology Studies*, vol. 6, no. 1, pp. 225–232, 2024.

[3] L. Banh and G. Strobel, "Generative artificial intelligence," *Electronic Markets*, vol. 33, no. 1, pp. 1–17, 2023, doi: 10.1007/s12525-023-00680-1.

[4] OpenAI, "ChatGPT." Accessed: Jun. 17, 2024. [Online]. Available: https://chatgpt.com/

[5] I. Midjourney, "Midjourney." Accessed: Jun. 17, 2024. [Online]. Available: https://www.midjourney.com/home

[6] A. Vaswani *et al.*, "Attention Is All You Need," *Adv Neural Inf Process Syst*, vol. 2017-December, pp. 5999–6009, Jun. 2017, Accessed: Jul. 18, 2024. [Online]. Available: https://arxiv.org/abs/1706.03762v7

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "(1986) D. E. Rumelhart, G. E. Hinton, and R. J. Williams, 'Learning internal representations by error propagation,' Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. I, D. E. Rumelhart and J. L. McClelland (Eds.) Cambridge, MA: MIT Press, pp. 318-362," *Neurocomputing, Volume 1*, pp. 675–695, Jan. 2024, doi: 10.7551/MITPRESS/4943.003.0128.

[8] S. Hochreiter and J. ¨ Urgen Schmidhuber, "Long Short-Term Memory".

[9] "The Building Blocks of LLMs: Vectors, Tokens and Embeddings - The New Stack." Accessed: Jul. 18, 2024. [Online]. Available: https://thenewstack.io/the-building-blocks-of-llms-vectors-tokens-and-embeddings/

[10] "Tokenization in Machine Learning Explained." Accessed: Jul. 18, 2024. [Online]. Available: https://vaclavkosar.com/ml/Tokenization-in-Machine-Learning-Explained

[11] "What is retrieval-augmented generation (RAG)? - IBM Research." Accessed: Jul. 24, 2024. [Online]. Available: https://research.ibm.com/blog/retrieval-augmented-generation-RAG

[12] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *Adv Neural Inf Process Syst*, vol. 2020-December, May 2020, Accessed: Jul. 24, 2024. [Online]. Available: https://arxiv.org/abs/2005.11401v4