

Analiza mogućnosti primjene prilagođenih GPT rješenja u procesu razvoja softvera

Čivčija, Jakov

Undergraduate thesis / Završni rad

2024

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

Permanent link / Trajna poveznica: <https://urn.nsk.hr/urn:nbn:hr:211:520569>

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

Download date / Datum preuzimanja: **2025-03-01**



Repository / Repozitorij:

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



SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Jakov Čivčija

ANALIZA MOGUĆNOSTI PRIMJENE
PRILAGOĐENIH GPT RJEŠENJA U
PROCESU RAZVOJA SOFTVERA

ZAVRŠNI RAD

Varaždin, 2024.

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

Jakov Čivčija

Matični broj: 0016152574

Studij: Informacijski Poslovni sustavi

**ANALIZA MOGUĆNOSTI PRIMJENE PRILAGOĐENIH GPT
RJEŠENJA U PROCESU RAZVOJA SOFTVERA**

ZAVRŠNI RAD

Mentor/Mentorica:

Doc. dr. sc. Marko Mijač

Varaždin, rujan 2024.

Jakov Čivčija

Izjava o izvornosti

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

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovom radu ću objasniti povijest GPT modela, njegovu arhitekturu i uporabu u različitim industrijama. Tako da će biti spomenuti GPT-1, GPT-2, GPT-3, GPT-4 sa svim svojim zanimljivostima i mogućnostima. Upoznat će se arhitektura sa svojim komponentama poput tokenizatora i dekodera. Dotaknut ću se prilagođenih GPT rješenja, te objasniti različite tehnike koje se koriste prilikom prilagođavanja. Prikazat ću alate kojima se može odraditi prilagodba.

Za kraj, uzet ću jedan od alata i različitim tehnikama pokušati riješiti proces razvoja softvera. Napraviti ću usporedbu između rezultata i zaključiti je li prilagodba korisna.

Ključne riječi: GPT, PrivateGPT, RAG, few-shot promptin

Sadržaj

Sadržaj	iii
1. Uvod	1
2. Veliki jezični modeli.....	2
2.1. Povijest i evolucija GPT modela	2
2.1.1. GPT-1	2
2.1.2. GPT-2	2
2.1.3. GPT-3	3
2.1.4. GPT-4	6
2.2. Arhitektura GPT modela	7
2.3. Primjena GPT modela u raznim industrijama	8
3. Prilagodba GPT rješenja.....	10
3.1. Tehnike prilagodbe GPT rješenja	11
3.2. Pregled alata za izradu prilagođenih GPT rješenja	13
3.2.1. PrivateGPT	13
3.2.2. Otvoreni modeli.....	15
3.2.3. Prilagođeni GPTjevi (ChatGPT).....	16
3.2.4. Ostali alati	18
4. Primjena PrivateGPT-ja u aktivnostima softverskog procesa.....	19
4.1. Izbor alata za praktični dio rada.....	19
4.1.1. Instalacija PrivateGPT-a	19
4.2. Analiza i specifikacija zahtjeva	21
4.3. Primjena GPT modela u fazi dizajna i arhitekture.....	27
4.4. Korištenje GPT modela za automatizaciju implementacije	32
4.5. GPT modeli u testiranju softvera	33
4.6. Održavanje softvera uz podršku GPT modela.....	37
4.7. Osvrt	40
5. Zaključak.....	42

Popis literature	43
Popis slika	45
Popis tablica	46

1. Uvod

Kroz dugi razvoj informacijskih tehnologija, dolazilo je puno novih revolucionarnih rješenja koja su potpuno promijenila neke industrije. To je počelo sa Alan Turingom tijekom 40-ih i 50-ih godina prošlog stoljeća. Nastavilo se sa novim prvim programima i neuronskim mrežama koje su mogle igrati šah. MYCIN i DENDRAL sustavi su postali poznati zbog svoje specifične primjene. Zahvaljujući strojnom učenju i napretku algoritama, zainteresiranost za AI se povećala i trenutno je još na visokoj razini.

Početakom 2020-ih dogodio se veliki skok u domeni umjetne inteligencije. OpenAI sa svojim ChatGPT-om je ubrzo postao viralan i ono što je se prije smatralo nemogućim preko noći je postalo stvarnost. Ubrzo su ljudi počeli koristiti tu tehnologiju u svakodnevnicima kako bi im pomogla u raznim poslovima.

GPT tehnologija je sigurno jako korisna, što pokazuje činjenica da sve više industrija pokušava implementirati GPT rješenja u svoje proizvode. Neki ljudi su skeptični i protive se tome, jer postoji opasnost od narušavanja privatnosti, izmišljanja informacije i velike greške koja može napraviti problem.

Kako bi što bolje i efektivnije koristili GPT modele, iste se može prilagoditi. Postoje različite tehnike uz pomoć kojih se mogu prilagoditi GPT modeli. Neke od tehnika su fino podešavanje, RAG i poticanje modela uz nekoliko primjera. Tehnike poput finog podešavanja koriste jako velike količine resursa, posebno utrošeno vrijeme i računalna snaga. Postoje različiti alati i pristupi koji omogućuju prilagodbu GPT-a za posebne potrebe pojedinca ili neke organizacije. Primjerice alat PrivateGPT omogućuje korisnicima da pristupi i koristi modele lokalno, što im pruža veću razinu kontrole i privatnosti.

Svake godine izlaze noviji unaprijeđeniji modeli GPT-a. Trenutno se izrađuje GPT-5 koji je sigurno dosta bolji od trenutne verzije GPT-4 koja može činiti čuda. Problemi s kojim se ova industrija može susresti su resursi poput računalne snage i struje jer svaki zahtjev treba računalnu snagu koja troši struju, a kompleksni izračuni traže puno snage. Ovo se najviše vidi ukoliko se na privatnom računalu vrti instanca LLM-a.

2. Veliki jezični modeli

Veliki jezični modeli su ključna komponenta iza generiranja teksta. U osnovi sastoje se od velikih obučanih transformacijskih modela koji su obučeni za predviđanje sljedeće riječi (ili, preciznije tokena) na temelju unesenog teksta.[1] Trenirani su na velikim količinama podataka kroz takozvano strojno učenje. Veliki jezični modeli se danas uglavnom temelje na arhitekturi transformera, kao što je GPT (engl. Generative Pretrained Transformer). Arhitekture koje su se koristile ranije su konvolucijske neuronske mreže (CNN) i rekurentne neuronske mreže (RNN). Neki od poznatih LLM-ova su Google BERT (Bidirectional Encoder Representations from Transformers), Google Gemini, Google PaLM, Meta LLaMA, Claude i ChatGPT. Arhitektura GPT je jedna od obitelji LLM-a koja se najviše koristi danas, a razvila ga je tvrtka OpenAI. Postala je jako popularna široj populaciji u jako kratkom vremenu.

2.1. Povijest i evolucija GPT modela

2.1.1. GPT-1

GPT-1 je prvi GPT model kojeg je OpenAI razvio 2018 godine. Treniranje prvog GPT modela sastojalo se od dvije faze. Prva faza se sastojala od nenadziranog treniranja unaprijed na skupu podataka BookCorpus koji je sadržavao oko sedam tisuća knjiga. Razlog uporabe ovog skupa podataka proizlazi iz toga što sadrži jako duge odlomke teksta i time pomaže modelu da dugoročno bolje upije informacije o jeziku. Druga faza se sastojala od nadziranog finog podešavanja (engl. fine tuning) gdje su podaci bili označeni kako bi model naučio razlikovati i diskriminirati podatke. Neki od tih označenih podataka su bili dokumenti sa pitanjima i odgovorima. Što se tiče specifikacija, model GPT-1 ima broj parametara od čak 120 milijuna. Ovaj model nije bio dostupan široj javnosti, nego je napisan znanstveni rad o njemu.[2]

2.1.2. GPT-2

GPT-2 model je varijanta modela GPT-1 koja je puno unaprijeđena u odnosu na prijašnju verziju. Za razliku od GPT-1, ovaj model je izašao u javnost, no izlazio je postepeno. Također je bio treniran na internetu, za razliku od GPT-1. Prva verzija modela izašla je u veljači 2019 godine. Sadržavala je 124 milijuna parametara. Druga verzija izašla je u svibnju iste godine.

Sadržavala je 355 milijuna parametara i skup podataka sa izlazima od sva četiri modela (124, 355, 744 i 1500 milijuna parametara). Skup podataka pomaže ljudima i klasifikatorima u procjeni sintetičkog teksta i pristranosti ugrađenih u GPT-2 generirane izlaze. Sljedeća verzija je izašla u kolovozu iste godine, te je sadržavala 744 miliona parametara. Također uz tu verziju je izašao prvi izvještaj i dodatna dokumentacija na githubu od strane developera.

Zadnja verzija GPT-2 ugledala je svjetlo dana petog studenog 2019 godine. Sadržavala je čak 1,5 milijardi parametara (1500 milijuna). Uz zadnju verziju izvještaj i dokumentacija su dodatno ažurirani. Prilikom izlaska prve verzije GPT-2, OpenAI je napravio odluku da izvorni kod ne bude dostupan javnosti. Razlozi zbog kojih je donesena takva odluka su ti da bi se GPT-2 mogao maliciozno upotrijebiti. Najpoznatiji primjer je taj da je generirani tekst od modela dosta svjež, te bi ga spammeri mogli koristiti i zaobilaziti sigurnosne filtere kako bi napravili ogromnu štetu žrtvama. Prilikom treniranja, istražitelji su uočili da model može generirati vulgaran i rasistički sadržaj, što je isto jedan od razloga odluke. No ograničeni pristup su dobile određene novinarske agencije. Partneri koji su pomagali u istraživanju su Cornell University, The Middlebury Institute of International Studies, The University of Oregon i The University of Texas at Austin.

Neka od otkrića istraživača ukazuju na to da ljudi smatraju generirani sadržaj GPT-2 uvjerljivim. To istraživanje je obavio Cornell University sa anketom u kojoj su ispitanici odgovarali za modele od 355, 744 i 1500 milijuna parametara. Ispitanici su davali odgovore u rasponu od 1 do 10. Najmanju ocjenu dobio je 355 milijunski model, sa ocjenom od 6,07. Sljedeći je bio 774 model sa ocjenom od 6,72, a najbolje ocjenjen model je model sa 1,5 milijardi parametara, sa ocjenom od 6,91. Ovo istraživanje je potaklo OpenAI da izbaci najveći model u javnost, jer je razlika u ocjenama dovoljno velika.

OpenAI je razvio model, kojemu je zadatak otkrivati generirani tekst od GPT-2, sa preciznošću od oko 95%. [3]

2.1.3. GPT-3

GPT-3 model je sljedeća generacija GPT-a koja je izašla u javnost 2020 godine. Ovaj model je ubrzo postao viralan preko cijelog svijeta. Korisnici društvenih mreža su bili oduševljeni sa njegovim mogućnostima. To je bio razlog zašto su ga počeli koristiti u široj

javnosti, čak i oni koji nisu entuzijastični za tehnologiju. Model sadrži dosta iteracija, što se vidi iz priložene tablice. [4]

Ime modela	n parametara	n slojeva	d modela	n glava	d glava	Veličina serije	Brzina učenja
GPT-3 Small	125M	12	768	12	64	0.5M	$6.0 \cdot 10^{-4}$
GPT-3 Medium	350M	24	1024	16	64	0.5M	$3.0 \cdot 10^{-4}$
GPT-3 Large	760M	24	1536	16	96	0.5M	$2.5 \cdot 10^{-4}$
GPT-3 XL	1.3B	24	2048	24	128	1M	$2.0 \cdot 10^{-4}$
GPT-3 2.7B	2.7B	32	2560	32	80	1M	$1.6 \cdot 10^{-4}$
GPT-3 6.7 B	6.7B	32	4096	32	128	2M	$1.2 \cdot 10^{-4}$
GPT-3 13B	13.0B	40	5140	40	128	2M	$1.0 \cdot 10^{-4}$
GPT-3 175B or „GPT-3“	175.0B	96	12288	96	128	3.2M	$0.6 \cdot 10^{-4}$

Tablica 1: Prikaz GPT-3 modela u fazi treniranja.

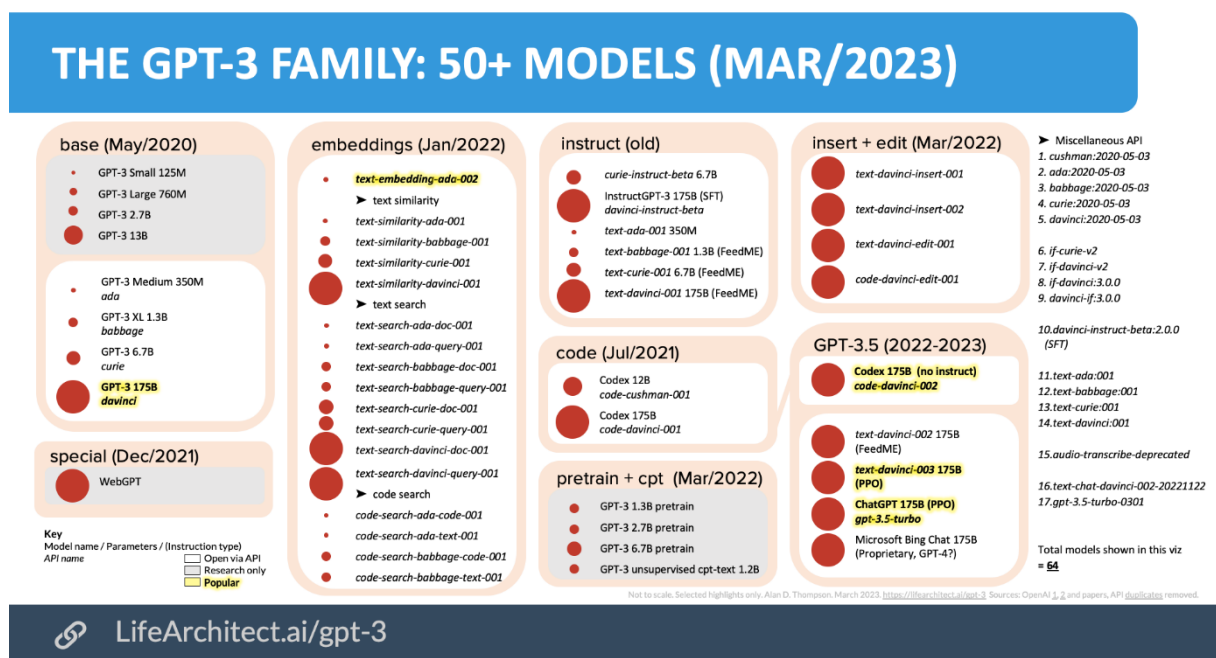
Iz priložene tablice se vidi da je prvo postojalo čak 8 različitih modela GPT-3, koji su bili u rasponu od 125 milijuna do 175 milijardi parametara. Kasnije je temeljni GPT-3 dobio više nasljednika, neki od njih su babbage-002, davinci-002, GPT-3.5-turbo. [4]

Skup podataka	Kvantitet tokena	Postotak u treningu	Epohe potrebne za treniranje 300B tokena
Common Crawl (filtered)	410 milijardi	60%	0.44
WebText2	19 milijardi	22%	2.9
Books1	12 milijardi	8%	1.9
Books2	55 milijardi	8%	0.43
Wikipedia	3 milijarde	3%	3.4

Tablica 2: Prikaz skupova podataka za treniranje GPT-3

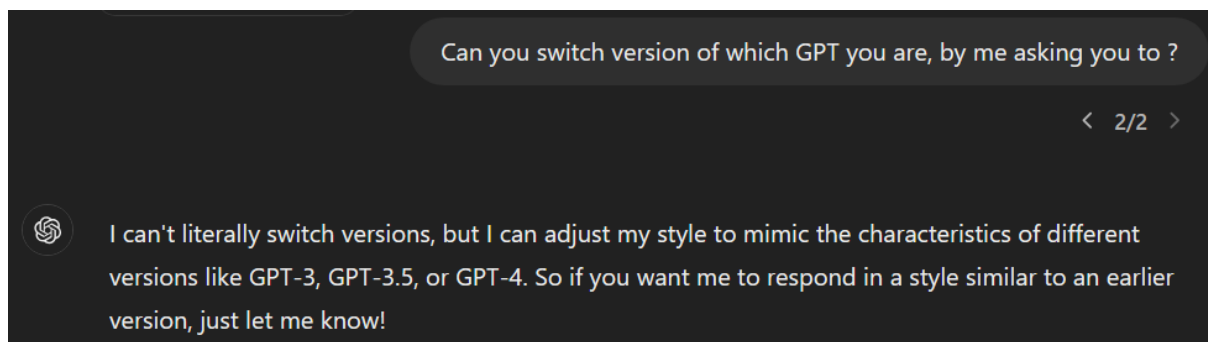
Iz tablice 2 se vidi da su uporabljeni različiti skupovi podataka za treniranje GPT-3. Dosta velik udio je imao Common Crawl koji je zapravo baza podataka koja je nastala skeniranjem interneta. Prilikom inicijalnog prikupljanja podataka od 2016 do 2019, podaci su bili veličine 45TB kompresiranog teksta. Nakon filtriranja veličina podataka je spala na 570GB. To je velika razlika u odnosu na prijašnju generaciju koja je imala skup podataka od oko 40GB. Prilikom treniranja kvalitetniji skup podataka je više puta sampliran, kao što su Books 1, dok su CommonCrawl i Books2 samplirani manje od jedan put. GPT-3 je bolji od GPT-2 u gotovo svim

zadacima. To uključuje zadatke vezane za jezike, naime novija generacija razumije jezike dosta bolje, što omogućava korisnicima da pišu na svom jeziku i dobiju odgovore na tom jeziku. Fino podešavanje u GPT-3 zahtijeva dosta manji skup podataka za razliku od prijašnje generacije kako bi se postigao isti uspjeh. Generirani tekst više nalikuje onome što bi napisao čovjek, a ne stroj. Također bolje razumije kontekst i koristi znatno širi vokabular u usporedbi sa svojim prethodnikom. No to sve ne dolazi besplatno. Zbog veličine i kompleksnosti GPT-3, brzina kojom obavlja zadatke je sporija u odnosu na GPT-2. Isti razlozi su upravo krivi i za cijenu, jer je potrebno više računalnih resursa kako bi se izvršili zadaci, tako da je skuplji od prethodnika. Bilo je moguće koristiti model kroz OpenAI API, no od svibnja 2024 godine samo unaprijeđene verzije od GPT 3 (davinci-002, babbage-002, GPT-3.5-turbo) su još dostupne javnosti kroz API. Potpora za davinci-002 i babbage-002 istječe 26 listopada 2024 godine. [4] [5]



Slika 1: Prikaz svih modela iz generacije GPT-3 (izvor: [6])

Iz navedene slike se vidi preko 50 modela koji su iz treće generacije GPT-a. Danas (rujan 2024.) se više ne koriste modeli treće generacije prilikom komuniciranja, iako je moguće pitati model da se ponaša kao stariji, ipak neće promijeniti verziju, nego će pokušati replicirati odgovore kakve bi prijašnja verzija dala. [7]



Slika 2: Upit mogućnosti promjene verzije (izvor: slika autora)

2.1.4. GPT-4

GPT-4 je najnovija generacija GPT modela. Izašla je u ožujku 2023 godine. Velik je skok u odnosu na GPT-3. Isprva nije bila dostupna besplatno svim korisnicima, no trenutno je dostupna. Svaki korisnik koji ne plaća može koristiti verziju GPT-4o, no nakon nekoliko upita, izgubi pravo koristiti tu verziju do određenog vremenskog razdoblja, te razgovor počne voditi u obliku verzije GPT-4o mini. Ukoliko korisnik ima „plus“ plan, onda ima pristup GPT-4o, GPT-4o mini i GPT-4. GPT-4 također ima mogućnost prepoznavanja slika, naprednih analiza podataka, uporabe uploadanih datoteka od strane korisnika, te pretrage interneta. Simulirane ispite prolazi jako dobro i često pri vrhu. Od matematike, biologije, kemije, povijesti i programiranja do drugih. Poznavanje različitih jezika je unaprijeđeno i točnost je u većini slučajeva iznad 70%, što je bolje nego bilo koji AI do tad. To ne znači da je četvrta generacija nepogrešiva, nekada izmisli ili dadne potpuno krivu informaciju. Veliki problem developerima je etičnost samog AI-a, nikako nebi bilo poželjno da netko napravi bombu uz pomoć GPT-a, te je potrebno treniranje kako se nebi tako nešto dogodilo. Rana verzija GPT-4 je korisniku davala informacije o tome kako napraviti bombu, dok usavršena zadnja verzija ne daje takve informacije korisnicima. Jedan od drugih problema je privatnost korisnika, što znači da AI ne pamti, to jest ne uči iz razgovora sa korisnicima, jer bi tako ugrozio privatnost istih. Privatni podaci prikupljeni sa interneta se filtriraju kako ne bi završili u modelu. Broj parametara GPT-4 je 1.76 bilijuna, što je dosta velik skok u odnosu na prijašnju generaciju sa 175 milijardi. Broj slojeva je 120, broj tokena oko 13 bilijuna. Cijena treniranja je oko \$63 milijuna. Postoji teorija da je GPT-4 zapravo sastavljen od 8 manjih modela od kojih svaki ima 220 milijardi parametara, koji su povezani u jednu cjelinu uz pomoć Mixture of Experts (MoE) arhitekture.

[8]

2.2. Arhitektura GPT modela

Kako bi shvatili kako GPT modeli funkcioniraju, prvo moramo razumjeti da je cilj LLM-a da pogađa riječi. Riječ je zapravo reprezentirana jednim ili više tokena. Token je komad teksta, i većina riječi sadrži samo jedan token, no ukoliko je riječ ogromna, može biti sastavljena od više tokena. Također ukoliko se piše na drugom jeziku za razliku od engleskog, riječi mogu biti sastavljene od više ili manje tokena.

Tokens	Characters
6	26
I am writing final thesis.	

Tokens	Characters
9	18
Pišem završni rad.	

Slika 3: Tokenizacija engleskog jezika (izvor: slika autora) Slika 4: Tokenizacija hrvatskog jezika (izvor: slika autora)

Iz navedenog primjera koji je djelo OpenAI tokenizera može se primijetiti da pet engleskih riječi: „I am writing final thesis.“, koje se sastoje od 26 znakova ima 6 tokena, dok prijevod na hrvatskom od tri riječi: „Pišem završni rad.“, sadrži čak 9 tokena, a ima samo 18 karaktera. Cijela zamisao GPT arhitekture je da model sam prepozna (pogodi) koji bi tokeni trebali ići nakon ovih tokena kako bi vratio odgovor. Ukoliko napišemo: „nebo je“, model bi pretpostavio da mora napisati plavo kako bi dao odgovor. Svaki model može različito interpretirati tokene i generirati iste. Također postoje granice koliko tokena može primiti, na primjer GPT 3.5 turbo ima granicu od 4096 tokena, dok nasljednik GPT-4 ima granicu od 32768 tokena, što znači da je moguće daleko veći upit dati GPT-4 nego njegovom prethodniku. Iako je logično pomisliti da transformer treba primiti string vrijednost, zapravo se ti stringovi moraju tokenizirati i pretvoriti u brojeve (tokene) kako bi dalje mogao funkcionirati.

Jedna od tih komponenata je sloj umetanja (engl. embedding layer). Sloj umetanja je zaslužan za pretvaranje tokena u vektor koji onda dobije vrijednost d_{model} što označava njegovu dužinu, a dužina je parametar koji specificira dimenzionalnost modela. Svaki token ima svoje mjesto u vektorskom prostoru i može se uočiti da su razlike udaljenosti između riječi muško i žensko gotovo jednake razlikama udaljenosti riječima kralj i kraljica, jer AI prepozna da su to dosta slični pojmovi koji se koriste u sličnom kontekstu pa ih ocjenjuje dosta slično. Ovaj dio arhitekture je jako bitan kako bi došli do konačnog odgovora.

Druga od komponenata koja je dosta slična sloju umetanja je pozicijski enkoder. Zadatak pozicijskog enkodera je da prepozna kojim slijedom tokeni (riječi) u odgovoru trebaju biti poredani. Nije isto napisati: „Mačka je lovila miša“ i „Miš je lovio mačku“, iako su vrijednosti tokena iste, njihov redoslijed nije i to može potpuno promijeniti narav odgovora što bi bilo jako

loše. Pozicijski enkoder također sadrži veličinu vektora d_{model} koji koristi prilikom izračuna. Račun se dobije uz pomoć funkcija sinus i kosinus. Bez ove funkcije dobili bi točan odgovor svih tokena, no njihov redoslijed ne bi bio dobro poredan, te bi imali samo beskorisnu hrpu riječi.

Sljedeća komponenta na redu je dekodera. Funkcija dekodera je da uzme informacije od prijašnje dvije komponente, znači od sloja umetanja i pozicijskog enkodera, te da uz pomoć tih informacija generira token koji bi trebao biti odgovor. Prilikom generiranja tokena, prolazi se kroz puno slojeva dekodera, a jedan od bitnih slojeva je pažnja više glava (multi-headed attention). Multi headed attention je jako bitan sloj, jer on uspijeva prepoznati šta točno upit želi, to jest kojim riječima u upitu bi trebao dati najviše pažnje. Zove se multi-headed (više glava) iz razloga što u isto vrijeme paralelno računa vrijednost bitnosti svake riječi u upitu. Također se koristi tehnologija maske koja prikriva tokene koji dolaze poslije, jer prilikom procesa prolazi kroz svaki token od početka, te kada obrađuje token broj 2, ne zna koji je sljedeći kako nebi pokvario izračun. Kada završi sa svim tokenima, onda uz pomoć svih informacija, što uključuje sloj umetanja i pozicijski enkoder, može donijeti odgovor u obliku sljedećeg tokena.

Generator je posljednji korak u procesu arhitekture transformera. Uzima sve vrijednosti dobivene prilikom prijašnjih dijelova arhitekture i mapira ih u vokabularni prostor. Čime se dobije distribucija vjerojatnosti za svaku poziciju u slijedu, što se koristi kako bi se dobili završni tokeni. Ovo osigurava da je izlazni odgovor sastavljen od ulaznog upita i generiranih tokena.[9]

2.3. Primjena GPT modela u raznim industrijama

GPT model se može primjenjivati u mnogim industrijama. Jedna od tih industrija je informatička (IT) industrija. U IT industriji GPT ima jako široku primjenu. Primjer toga bi bila pomoć u procesu razvoja softvera, što će ovaj rad pobliže istražiti, to uključuje donošenje samih odluka o specifikacijama softvera pa čak i do izgradnje tog softvera, te kasnijeg održavanja istog. Osim što pomaže u takvom razvoju, također se može koristiti u različitim igricama i modifikacijama za njih. [10]

U industriji pravljenja sadržaja se značajno osjetio utjecaj AI-a, a udio u tome ima i GPT model. Može se naći dosta videa koji su generirani od strane kombinacije više različitih AI-a. GPT posebno pomaže influencerima u davanju kreativnih ideja za videozapise. U budućim promjenama, GPT-4 će imati mogućnost generiranja različitih glasova, koji će biti prilagodljivi s obzirom na zahtjev korisnika. Ovo će biti jako korisno kreatorima sadržaja, jer će vrlo lako imati na raspolaganju jako različite glasove, što će dodatno olakšat posao. [11]

U industriji prevođenja GPT je jako koristan. Njegovi prijevodi se poboljšavaju sa svakom novijom iteracijom, što se može primijetiti u usporedbama GPT-3 i GPT-4. Najpopularnija aplikacija za učenje drugih jezika Duolingo ima pretplatničku verziju max koja funkcionira uz pomoć GPT-4. Ova verzija ima dvije mogućnosti. Jedna od tih mogućnosti je objašnjenje odgovora, ukoliko je došlo do greške, korisnici mogu upitati aplikaciju gdje su točno pogriješili, te dobiti odgovor uz pomoć GPT-4. Druga mogućnost se naziva roleplay. U roleplayu, GPT-4 generira razgovore koji su približni stvarnim razgovorima, tako korisnici mogu okušati svoje znanje u prirodnijim razgovorima. [12], [13]

U zdravstvenoj industriji GPT još nije previše iskorišten. Istraživanja vezana za GPT-4 pokazuju da uporaba alata može jako poboljšati ovu industriju. Može prepoznati različite dijagnoze, čak one koje se dogode u jednom od 100000 slučajeva, što bi samo stručni doktori sa puno iskustva mogli prepoznati. Problem zbog kojeg se još ne koristi je taj što postoji mogućnost haluciniranja i davanja lažnog odgovora. Također mašina nema etični kompas kao čovjek, te neće dodatno paziti da ne napravi kobnu grešku. No potencijal postoji, te bi eventualno moglo efikasnije poboljšati zdravstveni proces.[14]

U automobilnoj industriji ChatGPT ima svoje koristi. Različite poznate automobilske gigantske firme poput Mercedes-a, BMW-a, Toyote i Forda su napravile ili razvijaju integraciju GPT-a u svojim automobilima. Najveća uporaba je na prostoru zabavnog i navigacijskog sustava. Naime vozači mogu komunicirati zvukovno sa GPT-om i time mijenjati postavke u autu. Mercedes je razvio mogućnost da vozači mogu razgovarati o raznim stvarima, dok BMW i Toyota pružaju uslugu pravo vremenskog obavještanja o prometu, te savjetima o zanimljivim mjestima što je prilagođeno korisniku. [15]

U pravnoj industriji postoji mogućnost uporabe umjetne inteligencije sa transformerskom arhitekturom. Može pomoći odvjetnicima da sroči zvukovni zapis, također može pomoći u istraživanju zakona i sudskih odluka koji mogu biti presudni. Zanimljiv slučaj se dogodio u New Yorku 2023 godine, kada su zaposleni odvjetnici iz firme Levidow, Levidow & Oberman P.C bili sankcionirani od strane sudca jer su citirali slučaj koji je ChatGPT izmislio. [16]

GPT ima dosta upotreba u puno različitih industrija. Svijet je tek počeo koristiti dio potencijala koji tehnologije poput umjetne inteligencije i samog GPT-a mogu ponuditi. Veliki problemi dolaze u tome što je još dosta velika mogućnost pogreške koja u nekim slučajevima može biti kobna po ljudski život. Ostali problemi su privatnost i etičnost zbog kojih je primjena ograničena.

3. Prilagodba GPT rješenja

U prijašnjem dijelu obrazloženo je gdje se točno mogu koristiti GPT rješenja. U ovom poglavlju bit će obrazloženo zašto je prilagodba GPT rješenja korisna. Najbitniji dio posla koji GPT model obavlja je povećanje efikasnosti i ubrzanje procesa. Postoje različite grupe kojima bi prilagođena GPT rješenja pomogla. To su ogromne korporacije, ali postoje i manje tvrtke, kao i mali tim ljudi ili jedan čovjek. Potrebe koje korporacije imaju su jako velike, te ih je zato teže zadovoljiti, to je jedan od razloga zašto je dosta skupo uputiti se u prilagodbu GPT rješenja velikim korporacijama. Cijene treniranja mogu dosegnuti visoke šesteroznamenkaste iznose, pa čak i nekoliko milijuna u određenim slučajevima. No ako mogu optimizirati ogromne procese uz pomoć alata, onda imaju tu mogućnost za razliku od manjih grupa i to im se dugoročno može jako puno isplatiti. Smatra se da će za buduće treniranje biti potrebno dosta više novaca, što će potencijalno dosegnuti cifre od milijarde i više. Razlog tome je kompleksnost, skupa infrastruktura i cijena električne energije. Manje firme ne mogu izdvojiti toliko puno novaca i resursa za tako nešto, no opet postoje rješenja koja su zadovoljavajuća. Pojedinaac može iskoristiti prilagođeno GPT rješenje za dosta stvari. Veliki razlog prilagodbe GPT rješenja običnom korisniku je taj što može dovoljno prilagoditi da AI ima poseban stil izražavanja kakav bi korisnik htio. To znači da može rješavati odgovore na mailove, pisati poruke i to će izgledati uvjerljivo kao da je došlo od same osobe. Također može postavljati sadržaj na društvene mreže, te pomoći u poslovnom svijetu iako to možda nije preporučljivo ako se očekuje od osobe da to sama učini. Ukoliko se osoba bavi pisanjem blogova, neki dio može potpuno generirati i uvrstiti u članak, a ukoliko je dobro napravljen posao prilagodbe, neće se moći lako prepoznati da je to od strane transformerske arhitekture napisano. Ukoliko postoji nekakav skup podataka koji je korisnik pratio dugo vremena, taj se dokument može ubaciti kako bi se pravile raznorazne statističke analize, te izvlačili zaključci. Iako je jako velik skup podataka kojim se trenirao GPT-3 i GPT-4, opet na jako specifičnim stvarima im nedostaje tehničko, produbljeno znanje. Jedna od mogućnosti bi bila uzeti skup podataka koji odgovara rupi znanja osnovnog modela i time učiniti GPT sposobnim na tu temu. Također postoji dosta knjiga i materijala koji uopće nisu prisutni na internetu i možda su specifični za malu regiju ili neko mjesto, netko bi imao razloga istrenirati AI na tim informacijama. [17]

Tehnika prilagodbe	Korporacije	Srednje tvrtke	Male tvrtke	Pojedinci
RAG	Da	Da	Da	Da
Analiza podataka	Da	Da	Može	Može
Specifično znanje iz nišnih izvora	Da	Može	Ne	Ne
Upravljanje velikim podacima	Da	Može	Ne	Ne
Treniranje modela	Da	Može	Ne	Ne
Fino podešavanje	Da	Može	Ne	Ne
Optimizacija velikih procesa	Da	Ne	Ne	Ne

Tablica 3: Prikaz mogućnosti prilagodbi za različite korisnike

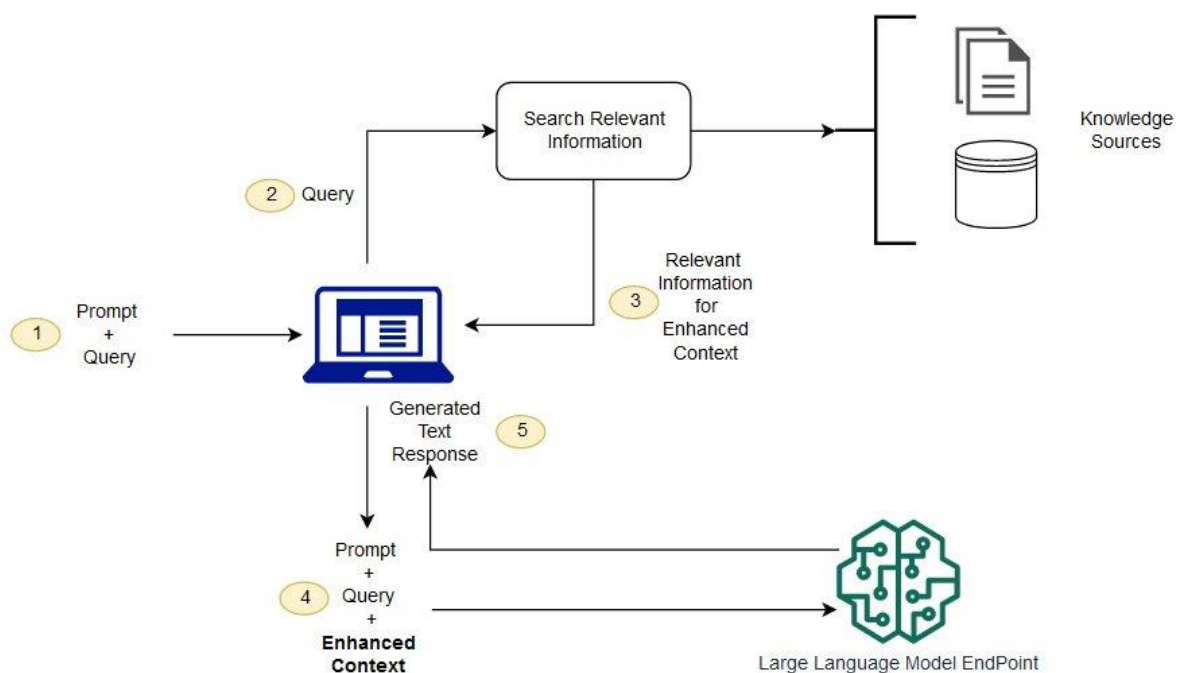
3.1. Tehnike prilagodbe GPT rješenja

Postoji dosta različitih tehnika koja omogućavaju prilagodbu GPT rješenja. Tehnike se razlikuju po složenosti, količini podataka, vremenskom trajanju, cilju i vrsti. Spomenut ću tehnike poput finog podešavanja, RAG, poticanja modela sa nekoliko i sa jednim primjerom.

Fino podešavanje je tehnika u kojoj se model koji je već treniran dodatno prilagođava potrebama. Uključuje nadgledanu vrstu učenja, ali je moguće imati samo nadgledanu ili polu nadgledanu verziju učenja. Moguće je prilagoditi model koji prepoznaje automobile da prepoznaje više vrsta i klasifikacija automobila kroz moć finog podešavanja. Iako izvorna verzija nekog modela nije sposobna programirati, moguće je kroz skup podataka istrenirati taj model da obavlja programerske dužnosti. Polu nadgledana verzija uključuje podatke koji su označeni i podatke koji nisu označeni. Takva vrsta treniranja pokazala je obećavajuće rezultate u treniranju NLP i vizualnih podataka, što olakšava problem manjka označenih podataka. Postoji puno različitih metoda za fino podešavanje, a jedna od njih je parametar efikasno fino podešavanje (PEFT) koja pomaže da model bude efikasniji i da ne dođe do katastrofalnog zaboravljanja. Katastrofalno zaboravljanje je pojava prilikom procesa finog podešavanja gdje model zaboravlja prijašnje znanje. Najjednostavniji koncept je potpuno fino podešavanje u kojem se cijela neuralna mreža ažurira. Moguće je da dođe do katastrofalnog zaboravljanja, tako da se mora paziti na određene postavke prilikom potpunog finog podešavanja. Za razliku od potpunog finog podešavanja, već spomenuti PEFT je u nekim slučajevima bolja opcija. Prilikom PEFT-a, manje se cijeli model ažurira, nego samo pojedinačni dijelovi, što donosi manju cijenu i uloženo vrijeme, dok su rizici izbjegnuti. Djelomično fino podešavanje ažurira samo vanjske slojeve kako bi se umanjio broj potrebnih resursa za izvršavanje zahtjeva. Aditivno fino podešavanje je tehnika u kojoj se za razliku od djelomičnog tuniranja dodaju novi parametri, što znači da će vrijeme treniranja biti duže, no memorijski zahtjevi će biti smanjeni. Fino podešavanje velikih LLM-ova je moguće, iako oni dosta dobro obavljaju poslove, opet

postoji mjesta za poboljšanje. Jedna od tehnika koja se u takvim slučajevima koristi je treniranje uz pomoć povratne informacije čovjeka (RLHF). Neki od zahtjeva mogu biti dosta kompleksni, te čovjek mora davati povratnu informaciju na kvalitetu odgovora. Ovo treniranje se također može koristiti da se model prilagodi određenoj publici ili skupini ljudi. Neki od razloga uporabe finog podešavanja su prilagođavanje stila, specijalizacija, rješavanje rubnih slučajeva i umetanje svojstvenih podataka. Ova vrsta prilagodbe je dosta resursno i vremenski intenzivna, te ju nije moguće praktično odraditi za svrhe ovog rada.[18]

Retrieval-Augmented Generation (RAG) je tehnika prilagodbe koja se široko koristi. To je zapravo framework koji koristi tehnologiju vektorskih baza. Ovo omogućuje da se AI brzo prilagodi novim informacijama uz uporabu jakih algoritama za pretraživanje, bilo to pretraživanje korisnikovih datoteka, web stranica ili cijelih baza podataka. Prilikom procesiranja podataka, iste filtrira, tokenizira i dodatno rafinira za uporabu. Nakon što završi sa tim, lako može dati nove odgovore koji su poduprti novim informacijama.[19]

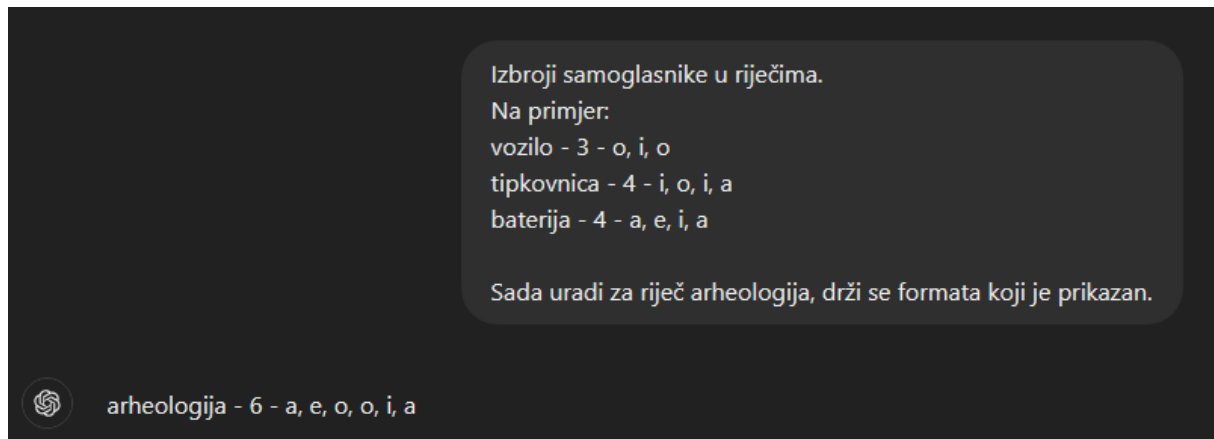


Slika 5: Pregled procesa generiranja odgovora uz uporabu RAG (izvor: [20])

Iz priložene slike se vidi kako proces generiranja izgleda nakon što je tehnika RAG upotrijebljena. Korisnik napravi zahtjev sa svojim upitom, taj upit se preusmjeri na izvore informacija koji su spremljeni u vektorsku bazu podataka zbog procesiranja sa RAG-om. Ukoliko se pronađu informacije koje odgovaraju upitu u tim bazama, onda taj dio podataka ide zajedno sa upitom dalje do LLM-a koji onda ima više informacija i tako može dati kvalitetniji odgovor nego da nema tih informacija. RAG će dalje biti demonstriran prilikom praktičnog dijela

ovog rada. Razlog tome je što ne zahtijeva previše intenzivnih resursa i potrebno mu je malo vremena da izvrši upite.

Poticanje modela sa nekoliko primjera (engl. few shot prompting) je jedna od tehnika koja ne zahtijeva materijale i dokumente i obavlja se prilikom formiranja upita. Naime, LLM-u nekad treba dodatan kontekst kako bi mogao odrađivati određeni zadatak. Kako bi izvršili tehniku few shot promptinga, prilikom postavljanja upita moramo dati više primjera kako obaviti određeni zadatak. Few shot prompting se može jako dobro iskoristiti ako se ispravno odradi.[21]



Slika 6: Primjer few-shot promptinga (izvor: slika autora)

Iz navedenog primjera kroz tehniku few shot prompting, prilagodio sam GPT da odgovara na način koji želim. Ideja je bila da izbroji samoglasnike (a, e, i, o, u), ispiše broj istih, te da redoslijedom prikazuje svaki od samoglasnika. Treba obraćati pozornost na pisanje upita, moj prijašnji upit nije bio dovoljno specifičan, pa se nije pridržavao formata, no ipak je ispravno odradio zadatak. Taj problem sam riješio dodavanjem zadnjeg dijela rečenice kada sam specifikirao da se drži formata koji je prikazan. Ova tehnika nije savršena, jer vrlo lako može napraviti grešku ako je zadatak dovoljno kompleksan. Sa few shot promptingom mogu svi korisnici eksperimentirati i velik je bonus da je dosta jednostavno za izvesti.

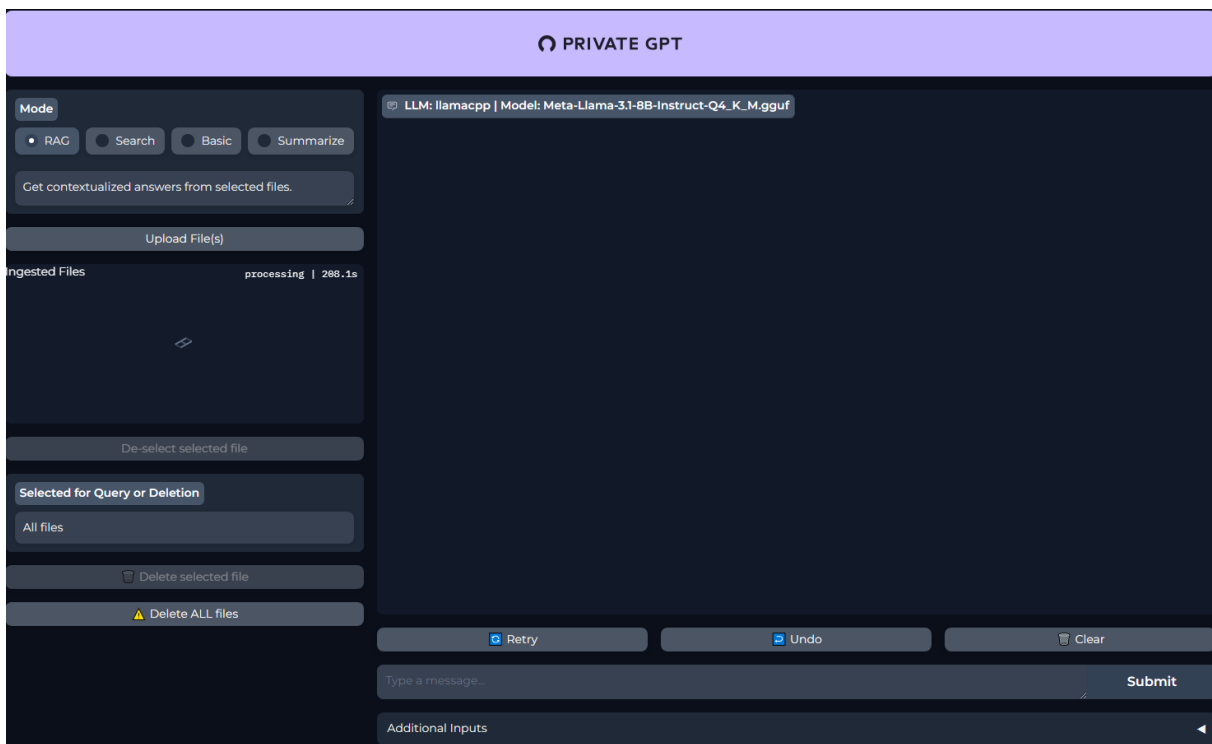
Poticanje modela uz jedan primjer (engl. one shot prompting) je dosta sličan prijašnjoj tehnici i lako se uoči razlika. Razlika između njih je ta da one shot ima samo jedan primjer, dok few shot ima više primjera što sigurno pomaže.

3.2. Pregled alata za izradu prilagođenih GPT rješenja

3.2.1. PrivateGPT

PrivateGPT je jedan od najpoznatijih alata za upotrebu prilagođenih GPT rješenja. Dobio je na popularnosti iz razloga što ga svatko može instalirati na osobno računalo. U svojoj osnovi, PrivateGPT nije samo korisničko sučelje za "konzumaciju" jezičnih modela, već platforma koja

omogućuje korisnicima potpunu kontrolu nad obradom prirodnog jezika, obradom upita i radom s vektorskim bazama podataka. Potpuno je privatna što znači da ne postoji ugroženost od curenja osobnih podataka. Iako to sve dobro zvuči, ukoliko je postavljen potpuno lokalno, onda se i LLM, umetanje i vektorske baze sve vrte lokalno, što znači da mašina treba biti jaka kako bi sve to brzo funkcioniralo. PrivateGPT čak ima svoje vizualno sučelje koje je intuitivno. Moguće je promijeniti puno postavki, tako da korisnik ima veliku kontrolu i slobodu. LLM koji se izvodi se isto tako može promijeniti i postaviti gotovo bilo koji. [22]



Slika 7: Pregled PrivateGPT vizualnog sučelja (izvor: slika autora)

Prilikom uporabe, PrivateGPT možemo konfigurirati da radi u sljedećim načinima uporabe. Prvi, upit dokumenata (engl. Query docs) je zapravo RAG tehnika koja je objašnjena u prošlom poglavlju. Kako bi iskoristili tu tehniku, prvo moramo uploadati datoteku uz pomoć gumba za uploadanje, te pričekati dok se to obradi. Nakon obrade može se koristiti query docs nad selektiranom datotekom. Drugi način je pretraga u dokumentima. Uvjet za ovaj način je isti kao i prije, mora postojati uploadana datoteka koja je procesirana. Ovaj način pretražuje datoteke i vraća odlomke tekstova gdje je pronađen odgovor. Treći način je LLM chat (basic) u kojem dolazi do obične komunikacije sa instaliranim LLM-om. Ovom tehnikom se ne koristi nikakva prilagodba, te nema potrebe koristiti ju ako korisnik ima pristup internetu, jer će dosta brže dobiti odgovore od ChatGPT-a. Četvrti način je summarize, koja ukratko može objasniti neki uploadani dokument. Za prosječan query je potrebno minuta do dvije, dok procesiranje datoteke može trajati od jedne minute, pa i dosta duže, naravno sve ovisi o veličini datoteka i specifikacijama lokalne mašine. Ukoliko se posjeduje jaka grafička, moguće je prilagoditi

postavke i konfiguraciju da se sve vrti uz pomoć resursa sa grafičke kartice, što jako puno ubrza svaku operaciju u PrivateGPT-u.

3.2.2. Otvoreni modeli

Postoje različite opcije za LLM koji se izvodi u PrivateGPT-u. Korisnici se mogu odlučiti za bilo koju opciju, ovisno o potrebama.

Opcija	Opis	Napomena
ollama	Potporna za Ollama LLM, lokalno	llms-ollama
llama-cpp	Potporna za LlamaCPP, lokalno	llms-llama-cpp
sagemaker	Potporna za Amazon Sagemaker LLM, zahtijeva završne točke	llms-sagemaker
openai	Potporna za OpenAI LLM, zahtijeva OpenAI API ključ	llms-openai
openailike	Potporna za LLM-ove treće strane, kompatibilna sa OpenAI API ključem	llms-openai-like
azopenai	Potporna za Azure OpenAI LLM, zahtijeva Azure završne točke	llms-azopenai
gemini	Potporna za Gemini LLM, zahtijeva Gemini API ključ	llms-gemini

Tablica 4: Prikaz različitih opcija modela PrivateGPT-a.

Ovo su opcije za uporabu posebnih LLM-ova u PrivateGPT-u. Može se primjetiti da su samo prva dva lokalna, a ostali funkcioniraju uz pomoć API ključeva i završnih točki.

Ollama i llama-cpp su dosta slične opcije, jer se izvode potpuno lokalno bez povezanosti internetu, no opet postoje razlike. Razlika koja je jako bitna je brzina performansi na lokalnom računaru.

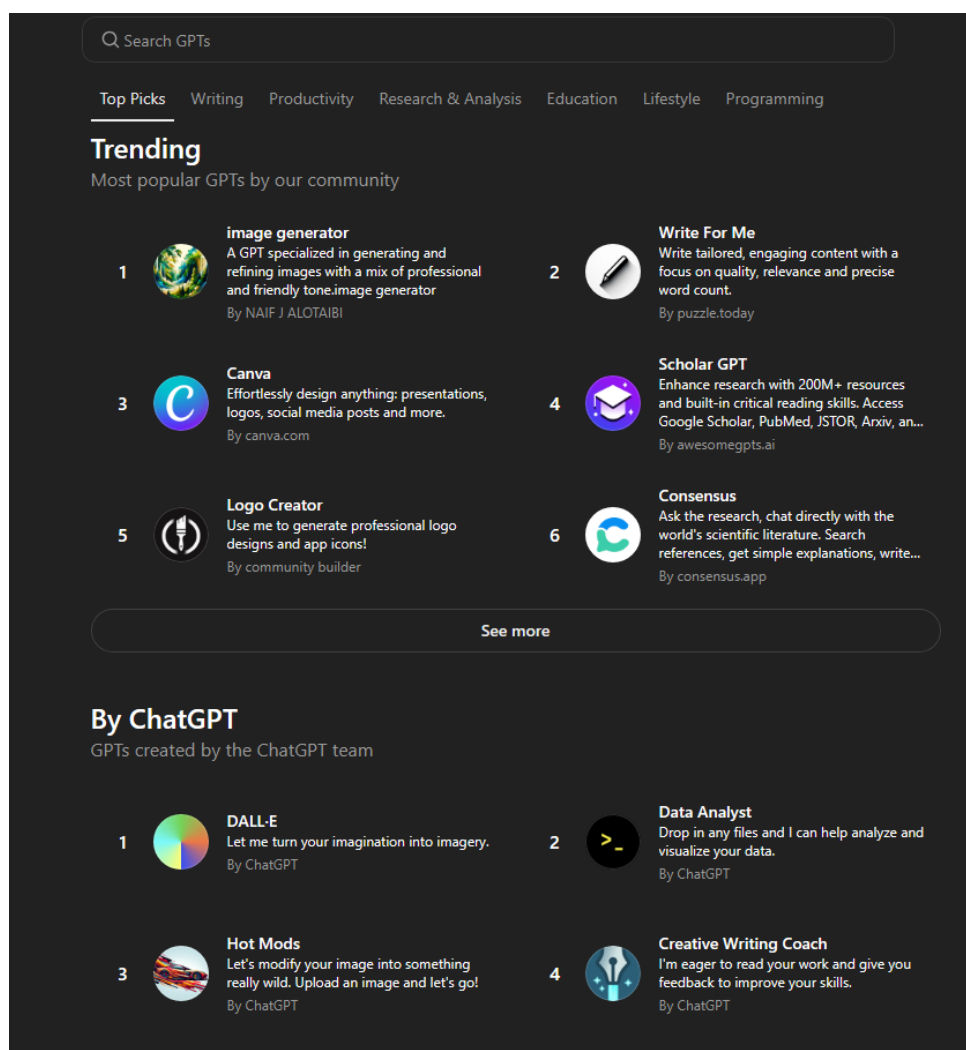
Ostale opcije su sagemaker, openai, openailike, azopenai i gemini. Za razliku od prve dvije opcije, ove nisu potpuno neovisne i potrebna je povezanost na internet. Također je potreban API ključ koji se mora dobiti uz pomoć registracije i prijave na web stranice. Sagemaker je razvijen od strane Amazona i treba dobiti pristup njegovom endpointu. Može se koristiti za postavljanje različitih modela. Openai opcija omogućuje pristup OpenAI modelima, ali isto tako zahtijeva API ključ. Openailike je zanimljiva opcija koja omogućuje pristup modelima koji su kompatibilni sa OpenAI, tako da je potreban OpenAI API ključ. Azopenai daje pristup OpenAI

modelima poput GPT-4o, GPT-4o mini, GPT-3.5-Turbo. Gemini je google tehnologija za LLM rješenje, te izabirom te opcije se pristupa Gemini modelima, također zahtijeva API ključ.

Opredjelio sam se za llama-cpp opciju iz dva razloga. Prvi razlog je potpuno lokalno pokretanje. Kada je okruženje potpuno lokalno, nikakve promjene ne mogu utjecati na izvršavanje zadatka. Nestanak interneta, promjena uvjeta suradnje i privatnost su dosta bitni faktori pri odlučivanju. Drugi razlog su performanse. Llama-cpp brže odrađuje upite za razliku od ollama.

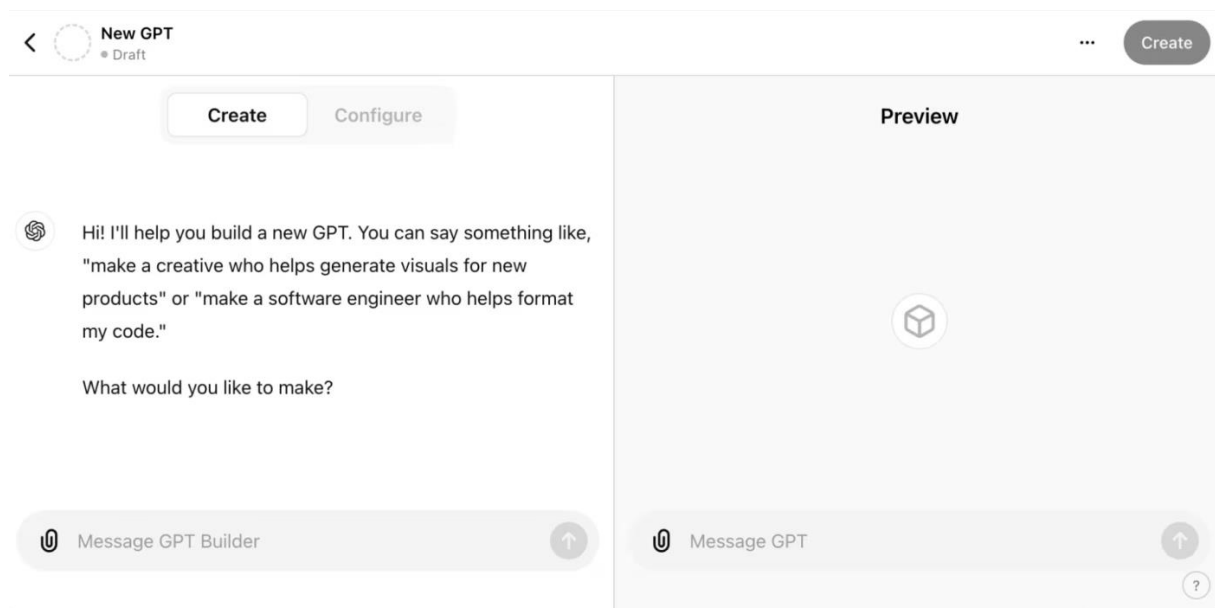
3.2.3. Prilagođeni GPTjevi (ChatGPT)

Drugi alati koji služe za prilagodbu GPT je sam GPT-4 uz svoju premium verziju. Naime moguće je praviti dosta različitih prilagođenih GPT-a i postavljati ih na Internet. Razvijena je trgovina u kojoj se nalaze prilagođene verzije GPT-a od strane korisnika što se vidi u slici ispod [8].



Slika 8: Pregled GPT trgovine (izvor: slika autora)

Kao što se vidi, postoje razne prilagodbe za potpuno različite uporabe. Od generiranja slika, potpore u dizajniranju, do pisanja i naprednog razumijevanja programskih jezika i zadataka. Ovo je dosta zanimljiv aspekt, jer se napravljeni prilagođeni GPT model može podijeliti sa svojim prijateljima i širem svijetu. Svaki GPT na trgovini ima svoju ocjenu, koja se kreće od 1 do 5, te broj konverzija. Konverzije označavaju broj uspješnih uporaba određenog modela. No nije svaki model besplatno dostupan čak i ako korisnik ima pretplatu na ChatGPT. Neki od modela su dosta kompleksni i skupi za istrenirati, tako da može postojati dodatna cijena na korištenje nekih od njih. [23]



Slika 9: Prikaz vizualnog sučelja GPT buildera (izvor: [24])

Službena podrška od strane OpenAI olakšava posao prilagođavanja rješenja, no potrebno je imati plaćenu verziju kako bi mogli pristupiti tom djelu. Mnogo opcija omogućava konfigurabilnost, što znači da se može značajno prilagoditi prema osobnim potrebama. Moguće je postaviti sliku na taj prilagođeni model, no osim što se može bilo koja slika uploadati, također sam GPT može generirati i postaviti sliku ovisno o korisnikovim uputama. Postoji prompt koji služi za izgradnju i prilagodbu, dok drugi dio ekrana zauzima prompt gdje se može testirati rješenje. Već su spomenute tehnike few shot promptinga i RAG-a koje se ovdje mogu koristiti. Uz pomoć few shot promptinga mogu se zadati posebne karakteristike tom specifičnom modelu. Na primjer može se njegov stil pisanja podesiti dovoljno da bude jedinstven i da koristi jako pozitivne riječi. Datoteke se mogu ubacivati za uporabu kroz spomenuti RAG. Ukoliko netko želi imati privatnog pomoćnika u kuhinji, može ubaciti datoteku

u kojoj se nalazi puno recepata neke specifične kuhinje i onda može koristiti taj prilagođeni model prilikom kuhanja kako bi bio siguran da nije neki korak ili začim zaboravio. Neke od posebnih opcija koje se mogu uključiti su pretraga interneta, DALL-E generator slika i interpreter koda. Opcija pretrage interneta omogućava pristup internetu kako bi se došlo do novih informacija koje nisu bile dostupne prilikom treniranja modela. Opcija DALL-E generator slika omogućava uporabu posebnog generatora koji može generirati slike. Zadnja opcija interpreter koda daje posebnu mogućnost naprednog koda koji se vrti i koji se može upotrijebiti za prilagođeni model koji ima veze sa programiranjem, matematičkim, fizičkim i statističkim računanjima. On zapravo napiše svoj kod i onda uz pomoć tog koda donosi izračune. Dodatna jako bitna opcija za prilagođeni model GPT-a je ta da je moguće konfigurirati API i pozivati krajnje točke (engl. endpoint). Ovo je jako moćna funkcionalnost i može se iskoristiti za veće potrebe. Na primjer firma može prilagoditi GPT, te napraviti API koji pruža potporu korisnicima ukoliko imaju nekakav problem. Nakon što se potpuno završi sa procesom prilagodbe, korisnik može odlučiti s kim želi podijeliti model. Ukoliko želi da model bude privatn i za to postoji opcija. Ostale dvije opcije su opcija pristupa uz pomoć linka, što znači da svatko tko ima link može pristupiti, a druga opcija je opcija objavljivanja na trgovinu, što znači da svatko može javno pristupiti uz pomoć trgovine.

3.2.4. Ostali alati

Neki od ostalih alata su Huggings Face Transformer biblioteka. Ona omogućava treniranje uz pomoć tehnike finog podešavanja na svom API-u za treniranje. Moguće je postaviti svoja rješenja na model hub što je jako slično trgovini od OpenAI-a.[25]

AdapterHub je biblioteka izrađena na tehnologiji transformer modela GPT-a. Dopušta različite tehnike finog podešavanja koje su jako brze i efikasne, jer ne podešava cijeli model, nego samo određene parametre.[26]

EleutherAI je također jedna od mogućnosti za prilagodbu modela. Otvorenog je koda i svi mogu pristupiti izrađenim modelima GPT-Neo i GPT-J koji su alternative GPT-3. Nemaju restrikcije zbog licenci, a jako su slični OpenAI GPT modelu.[27]

Ostali od alata su NVIDIA Megatron-LM, Googleov T5 i Reformer. Oni nisu lako dostupni korisnicima sa malim resursima.

4. Primjena PrivateGPT-ja u aktivnostima softverskog procesa

4.1. Izbor alata za praktični dio rada

Nakon određenog razmatranja između opcija, odlučio sam u ovom radu koristiti PrivateGPT. Razlog toj odluci je taj što je PrivateGPT potpuno lokalan, te se može više konfigurirati, iako treba više vremena da procesira datoteke i obradi korisničke zahtjeve.

4.1.1. Instalacija PrivateGPT-a

Da bismo koristili PrivateGPT, bilo ga je potrebno instalirati na lokalno računalo. Vodio sam se instalacijskim vodičem objavljenim na službenim stranicama PrivateGPT-a Slijedio sam proces instalacije sa službenih PrivateGPT izvora i sa jednog članka.[28], te jednim od dostupnih tutorijala na internetu.[29] Prvo sam klonirao službeni github repozitorij od PrivateGPT-a na svoje računalo. Nakon toga ušao sam u klonirani direktorij PrivateGPT. Prije sljedećih koraka prvo se mora instalirati python, jer gotovo sve skripte se pokreću u python jeziku. Python sam na windows instalirao uz pomoć miniconde, koja je mala verzija anaconde koja sadrži samo conda, python i ostale pakete potrebne za njihov rad. Prilikom instaliranja miniconde, potrebno je izabrati opciju da se varijable sačuvaju u PATH. Ta opcija je jako bitna kako bi cmd (command prompt) mogao prepoznati naredbe conda alata. Ukoliko se slučajno nije dodao PATH prilikom instalacije, moguće je ponovno pokrenuti instalaciju i to popraviti ili ručno urediti varijable sustava kroz windowsov program (Edit the system environment variables). Ukoliko je se sve instaliralo kako treba, potrebno je restartirati cmd kako bi imao osvežene putanje varijabli. Kada se nalazimo u kloniranom repozitoriju sa githuba, treba pokrenuti sljedeću komadnu: „*conda activate privateGPT*“. Ta komanda otvara novi condin okoliš za razvoj u kojem se mogu postaviti posebni paketi i ovisnosti za izvršavanje zadatka. Okoliš ima naziv PrivateGPT i u njemu se mogu instalirati sve što je potrebno kako bi se pokrenuo privateGPT. Jedan od paketa koji je dosta bitan i s kojim je dosta lakše upravljati pythonom i ovisnostima se naziva *poetry*. Kako bi se instalirao poetry potrebno je izvršiti sljedeće komande u konzoli.

```
conda install -c conda-forge pipx
```

```
pipx install poetry
```

```
set "PATH=%USERPROFILE%\local\pipx\venvs\poetry\Scripts;%PATH%"
```

Ovaj niz komandi instalira pipx koji onda instalira poetry. Zadnja komanda postavlja putanju poetryu kako bi se dalje poetry mogao koristiti. Nakon što je uspješno poetry instaliran i nakon što može obavljati svoje dužnosti, potrebno je pokrenuti jednu komandu uz njegovu pomoć.

```
poetry install --extras „ui llms-llama-cpp vector-stores-qdrant embeddings-huggingface“
```

Ova komanda instalira sve ovisnosti trenutnog okruženja (PrivateGPT-a), te također instalira dodatne ovisnosti koje su potrebne za ispravan rad. Te ovisnosti su ui, llms-llama-cpp, vector-stores-qdrant i embeddings-huggingface. Napomena, ove ovisnosti mogu biti drugačije i tu korisnik može odabrati vrstu LLM-a, umetanja i vektorskih baza. Nakon instalacije koja može potrajati jer se dohvaća i instalira dosta različitih paketa, potrebno je pokrenuti skripte koje su generirane. Prvi korak u tome bi bio ulazak u područje gdje se skripte nalaze i preimenovanje datoteke jer nije naziva koji je potreban. Nakon toga izlazi se iz tog foldera i pokreću skripte. Nakon izvršavanja skripti potrebno je postaviti posebnu varijablu okruženja koja se naziva PGPT_PROFILES i varijablu PYTHONPATH. Sljedeće komande odrađuju objašnjeni postupak.

```
cd scripts
ren setup setup.py
cd ..
poetry run python scripts/setup.py
set PGPT_PROFILES=local
set PYTHONPATH=.
```

Ovime je završeno inicijalno postavljanje i može se pokušati pokrenuti PrivateGPT uz pomoć komande.

```
poetry run python -m uvicorn private_gpt.main:app --reload --port 8001
```

```
C:\Users\jakov\Desktop\Faks\6 Semestar\Završni Rad\privateGPT>cd privateGPT
C:\Users\jakov\Desktop\Faks\6 Semestar\Završni Rad\privateGPT\privateGPT>conda activate privateGPT
(privateGPT) C:\Users\jakov\Desktop\Faks\6 Semestar\Završni Rad\privateGPT\privateGPT>poetry run python -m uvicorn private_gpt.main:app --reload --port 8001
INFO: Will watch for changes in these directories: ['C:\Users\jakov\Desktop\Faks\6 Semestar\Završni Rad\privateGPT\privateGPT']
INFO: Uvicorn running on http://127.0.0.1:8001 (Press CTRL+C to quit)
INFO: Started reloader process [91972] using WatchFiles
14:10:35.198 [INFO] private_gpt.settings.settings_loader - Starting application with profiles=['default']
14:10:47.781 [INFO] private_gpt.components.llm.llm_component - Initializing the LLM in mode=llamacpp
llama_model_loader: loaded meta data with 33 key-value pairs and 292 tensors from C:\Users\jakov\Desktop\Faks\6 Semestar\Završni Rad\privateGPT\privateGPT\models\Meta-Llama-3.1-8B-Instruct-Q4_K_M.gguf (version GGUF V3 (latest))
llama_model_loader: Dumping metadata keys/values. Note: KV overrides do not apply in this output.
llama_model_loader: - kv 0:          general.architecture str           = llama
llama_model_loader: - kv 1:          general.type str                = model
llama_model_loader: - kv 2:          general.name str                = Meta Llama 3.1 8B Instruct
llama_model_loader: - kv 3:          general.finetune str            = Instruct
llama_model_loader: - kv 4:          general.basename str            = Meta-Llama-3.1
llama_model_loader: - kv 5:          general.size_label str          = 8B
llama_model_loader: - kv 6:          general.license str             = llama3.1
llama_model_loader: - kv 7:          general.tags arr[str,6]         = ["facebook", "meta", "pytorch", "llam...
llama_model_loader: - kv 8:          general.languages arr[str,8]     = ["en", "de", "fr", "it", "pt", "hi", ...
llama_model_loader: - kv 9:          llama.block_count u32           = 32
llama_model_loader: - kv 10:         llama.context_length u32        = 131072
llama_model_loader: - kv 11:         llama.embedding_length u32      = 4096
llama_model_loader: - kv 12:         llama.feed_forward_length u32   = 14336
llama_model_loader: - kv 13:         llama.attention.head_count u32  = 32
llama_model_loader: - kv 14:         llama.attention.head_count_kv u32 = 8
llama_model_loader: - kv 15:         llama.ropе.freq_base f32       = 500000.000000
llama_model_loader: - kv 16:         llama.attention.layer_norm_rms_epsilon f32 = 0.000010
llama_model_loader: - kv 17:         general.file_type u32           = 15
llama_model_loader: - kv 18:         llama.vocab_size u32           = 128256
llama_model_loader: - kv 19:         llama.ropе.dimension_count u32  = 128
llama_model_loader: - kv 20:         tokenizer.ggml.model str        = gpt2
llama_model_loader: - kv 21:         tokenizer.ggml.pre str          = llama-bpe
```

Slika 10: Prikaz konzole pokrenutog PrivateGPT-a (izvor: slika autora)

Ukoliko je sve instalirano kako treba, ovako bi konzola trebala izgledati na uspješnom pokretanju. Vidi se i poveznica pristupa vizualnom sučelju u web pregledniku. Vrti se na ip adresi 127.0.0.1:8001 što je uobičajena IP adresa za lokalno pokrenute servere, zapravo je isto kao i da se upiše localhost:8001. Praktični dio rada će se odvijati uz pomoć vizualnog sučelja, iako je moguće slati zahtjeve na API.

4.2. Analiza i specifikacija zahtjeva

Analiza zahtjeva je jedna od prvih aktivnosti u procesu razvoja softvera. Jako je važna, jer se u njoj postavljaju temelji za sve što će kasnije biti izrađeno. Ova faza ne zahtijeva nikakvo programiranje, nego postavlja očekivanja gotovog proizvoda u točno izrečenim zahtjevima.

Nakon prvog upita običnom ne prilagođenom modelu, isti je postavio dosta pitanja kako bi dao dosta bolji odgovor. Ispravno je prepoznao kako prvi upit nije imao dovoljno informacija, te nakon konzultacije sa klijentom postavio sam upit sa dodatnim informacijama.

Pitanje: *Imamo klijenta koji želi imati aplikaciju u kojoj drži evidenciju potrošenih novčanih sredstava. Želi aplikaciju imati na stolnom računalu, a mi smo baš specijalizirani u izradi C# tehnologije. Nije potrebno koristiti aplikaciju na više stolnih računala. Možeš li mi pomoći u analizi zahtjeva za ovaj problem ?*

Odgovor: *Želite li da aplikacija bude dostupna samo na jednom stolnom računalu ili će biti moguće pristupiti aplikaciji s bilo kojeg stolnog računala u mreži?*

Koji su potrebni dijelovi aplikacije? Na primjer, treba li omogućiti:

- *Unos potrošenih novčanih sredstava?*
- *Prikaz evidencije potrošenih novčanih sredstava?*
- *Računanje ukupnog potrošenog iznosa?*
- *Prikaz statistike potrošenih novčanih sredstava (npr. najveći potrošak, najveći broj potrošenih novčanih sredstava u jednom mjesecu)?*
- *Drugi funkcionalnosti?*

Koji su potrebni korisnički interfejsi? Na primjer, treba li aplikaciju koristiti kao:

- *Grafički korisnički interfejs (GUI)?*
- *Komandna linija (CLI)?*
- *API za integraciju s drugim aplikacijama?*

Koji su potrebni zahtjevi za sigurnost? Na primjer, treba li omogućiti:

- *Korištenje korisničkog imena i lozinke za pristup aplikaciji?*
- *Korištenje autentifikacije (npr. OAuth)?*
- *Korištenje šifriranja podataka?*

Koji su potrebni zahtjevi za skalabilnost? Na primjer, treba li omogućiti:

- *Skaliranje aplikacije za više korisnika?*
- *Skaliranje aplikacije za više stolnih računala?*

Koji su potrebni zahtjevi za podršku? Na primjer, treba li omogućiti:

- *Korištenje podrške za korisnike (npr. FAQ, dokumentacija)?*
- *Korištenje podrške za tehničke probleme (npr. GitHub, issue tracker)?*

Nakon što je model dobio dodatne informacije, dao je rješenje koje je imalo svoje dobre, ali i loše stvari. Loše stvari su ne definiranost zahtjeva i točnije specifikacije.

Pitanje: *Ponovno sam upitao klijenta i dao mi je sljedeće odgovore: Klijent zahtijeva mogućnost unosa novčanih sredstava, prikaz evidencije potrošenih novčanih sredstava i računanje ukupnog potrošenog iznosa. Sučelje treba biti grafičko. Nije potrebna nikakva sigurnost, aplikaciju koristi samo jedan korisnik i nije mu potrebna podrška. Generiraj mi sve zahtjeve po kojima mogu točno nastaviti sa izradom softverskog rješenja.*

Odgovor:

1. *Baza podataka: za spremanje evidencije potrošenih novčanih sredstava. Moguće je koristiti lokalnu bazu podataka (npr. SQLite) jer aplikacija koristi samo jedan korisnik.*
2. *Grafički korisnički interfejs (GUI): za omogućavanje korisniku unosa novčanih sredstava, prikaza evidencije i računanja ukupnog potrošenog iznosa. Moguće je koristiti biblioteku poput Windows Forms ili WPF za stvaranje GUI-a.*

3. *Logika aplikacije: za upravljanje evidencijom i računanje ukupnog potrošenog iznosa. Moguće je koristiti C# za implementaciju logike aplikacije.*

Komponente aplikacije moguće je razvrstati u sljedeće kategorije:

- *Korisnički interfejs: GUI koji omogućava korisniku unos novčanih sredstava, prikaz evidencije i računanje ukupnog potrošenog iznosa.*
- *Baza podataka: lokalna baza podataka koja sprema evidenciju potrošenih novčanih sredstava.*
- *Logika aplikacije: C# kod koji upravlja evidencijom i računa ukupni potrošeni iznos.*

Planiranje sljedećih koraka za implementaciju aplikacije:

1. *Kreiranje lokalne baze podataka za spremanje evidencije potrošenih novčanih sredstava.*
2. *Implementacija GUI-a koji omogućava korisniku unos novčanih sredstava, prikaz evidencije i računanje ukupnog potrošenog iznosa.*
3. *Implementacija logike aplikacije koja upravlja evidencijom i računa ukupni potrošeni iznos.*

Jedna od spomenutih tehnika koju sa svojim ograničenjima mogu koristiti je few shot prompting, tako da sam za ovaj proces napravio upit isti kao i gore, te dodao primjere za odgovore.

Upit: *"Klijent zahtijeva aplikaciju koja omogućava praćenje kućnog budžeta. Potrebna je mogućnost unosa prihoda i rashoda, prikaz bilance i mogućnost prikaza grafičkog izvještaja. Sučelje treba biti jednostavno i pristupačno. Možeš li analizirati zahtjeve i dati pregled ključnih funkcionalnosti?"*

Odgovor:

Funkcionalni zahtjevi:

Unos prihoda/rashoda:

Aplikacija treba omogućiti korisnicima unos prihoda i rashoda s relevantnim detaljima kao što su iznos, datum i kategorija. Uneseni podaci trebaju biti spremljeni u bazi podataka i dostupni dza daljnju obradu i prikaz.

Prikaz bilance:

Aplikacija treba izračunavati ukupne prihode i rashode i prikazivati trenutnu bilancu korisniku. Prikaz bilance treba biti ažuriran u stvarnom vremenu na temelju unesenih podataka.

Grafički izvještaj:

Korisnici trebaju imati mogućnost generiranja grafičkog izvještaja koji vizualizira trendove prihoda i rashoda kroz vrijeme. Grafički izvještaj treba biti interaktivan i omogućiti filtriranje

podataka prema različitim kriterijima, poput datuma ili kategorije.

Ne-funkcionalni zahtjevi:

Jednostavno sučelje:

Sučelje aplikacije treba biti intuitivno i lako razumljivo, s minimalnim brojem koraka potrebnih za unos podataka i pregled bilance. Dizajn sučelja treba biti čist i organiziran, s jasnim oznakama i uputama za korisnike.

Performanse:

Aplikacija treba biti brza u odgovorima na korisničke akcije, poput unosa podataka i generiranja izvještaja. Treba osigurati da aplikacija može obraditi veće količine podataka bez značajnog usporavanja.

Pristupačnost:

Aplikacija treba biti pristupačna svim korisnicima, uključujući one s invaliditetom, kroz primjenu principa dizajna pristupačnosti (npr. čitljivost, kontrast).

Primjer 2:

Upit: "Klijent želi aplikaciju za praćenje troškova putovanja. Potrebna je mogućnost unosa informacija o putovanjima (datum, destinacija, trošak), kreiranje izvještaja o ukupnim troškovima, i unos proračuna za svako putovanje. Možeš li analizirati ovaj zahtjev i definirati glavne funkcionalnosti?"

Odgovor:

Funkcionalni zahtjevi:

Unos troškova putovanja:

Aplikacija treba omogućiti korisnicima unos informacija o svakom putovanju, uključujući datum, destinaciju i trošak. Detalji o putovanjima trebaju biti pohranjeni u bazi podataka i biti dostupni za pregled i analizu. Izračun ukupnih troškova: Aplikacija treba automatski izračunavati i prikazivati ukupne troškove svih putovanja. Korisnici trebaju imati mogućnost pregleda detaljnih troškova po putovanju ili ukupnih troškova za određeni vremenski period.

Praćenje proračuna:

Korisnici trebaju moći postaviti proračun za svako putovanje i pratiti koliko je proračun potrošen ili preostao. Aplikacija treba pružiti vizualizaciju razlike između stvarnog troška i proračuna.

Ne-funkcionalni zahtjevi:

Jednostavno sučelje:

Sučelje aplikacije treba biti jednostavno za navigaciju i unos podataka, s jasnim opcijama za dodavanje i pregled putovanja. Treba omogućiti brzi pristup ključnim informacijama poput ukupnih troškova i proračuna bez potrebe za kompliciranim postupcima.

Performanse:

Aplikacija treba brzo odgovarati na korisničke akcije, posebno pri unosu novih troškova i generiranju izvještaja. Treba osigurati da aplikacija učinkovito obrađuje velike količine podataka bez usporavanja.

Pristupačnost:

Dizajn aplikacije treba biti u skladu s principima pristupačnosti, osiguravajući da svi korisnici, uključujući one s invaliditetom, mogu lako koristiti aplikaciju. Treba implementirati funkcionalnosti poput podrške za čitatelje ekrana i prilagodljive veličine teksta.

Odgovor koji sam dobio je bio jako kvalitetan i jasan za pročitati.

Odgovor: Na temelju zahtjeva klijenta, moguće je identificirati sljedeće ključne funkcionalnosti za aplikaciju za praćenje potrošenih novčanih sredstava:

Funkcionalni zahtjevi:

1. **Unos novčanih sredstava:** Korisnici trebaju moći uneti informacije o novčanim sredstvima, uključujući datum, iznos i kategoriju.
2. **Prikaz evidencije:** Aplikacija treba prikazivati evidenciju potrošenih novčanih sredstava, uključujući ukupne iznose i detalje o pojedinim unosima.
3. **Računanje ukupnog potrošenog iznosa:** Aplikacija treba automatski izračunavati i prikazivati ukupni potrošeni iznos.
4. **Grafički izvještaj:** Korisnici trebaju imati mogućnost generiranja grafičkog izvještaja koji vizualizira trendove potrošenih novčanih sredstava kroz vrijeme.

Ne-funkcionalni zahtjevi:

1. **Jednostavno sučelje:** Sučelje aplikacije treba biti intuitivno i lako razumljivo, s minimalnim brojem koraka potrebnih za unos podataka i pregled evidencije.
2. **Performanse:** Aplikacija treba biti brza u odgovorima na korisničke akcije, posebno pri unosu novih podataka i generiranju izvještaja.
3. **Pristupačnost:** Aplikacija treba biti pristupačna svim korisnicima, uključujući one s invaliditetom, kroz primjenu principa dizajna pristupačnosti.

Dodatni zahtjevi:

1. **Baza podataka:** Aplikacija treba imati bazu podataka koja će spremiti informacije o novčanim sredstvima i evidenciji.
2. **Korisnički interfejs:** Aplikacija treba imati korisnički interfejs koji će omogućiti korisnicima unos podataka i pregled evidencije.
3. **Grafički izvještaj:** Aplikacija treba imati mogućnost generiranja grafičkog izvještaja koji će vizualizirati trendove potrošenih novčanih sredstava kroz vrijeme.

Ovi zahtjevi će biti osnova za izradu aplikacije za praćenje potrošenih novčanih sredstava.

Upladao sam datoteku koja sadrži informacije o analizi i specifikaciji zahtjeva. Datoteka je sadržavala prezentacije sa kolegija Programsko Inženjstvo i Razvoj Programskih Proizvoda, te relevantne lekcije sa interneta spojene u jedan PDF, zajedno sa transkriptom razgovora s klijentom. Postavio sam pitanje modelu, da provjeri datoteku i da odgovori. Odgovorio je i nije tražio dodatno pojašnjenje za svoju analizu što je jako loše jer bez velike preciznosti možemo izraditi softver kojeg klijent nije želio. Nakon dodatnog pojašnjenja na kojem sam ja inzistirao, novi odgovor je bio jako dobar, te je ponudio rješenje koje je bilo dosta dobro, te bolje u završnici od rješenja koje je obični model ponudio. Napomena, rješenje koje je izgenerirano RAG metodom će se koristiti u upitima za sljedeći proces.

Odgovor: *Na temelju klijentovih zahtjeva, moguće je izdvojiti sljedeće funkcionalne i nefunkcionalne zahtjeve za softversko rješenje:*

Funkcionalni zahtjevi:

1. **Unos novčanih sredstava:**

- * Korisnik može unijeti novčane sredstva u aplikaciju.
- * Aplikacija treba spremiti unosene podatke u bazi podataka.

2. **Prikaz evidencije potrošenih novčanih sredstava:**

- * Aplikacija treba prikazati evidenciju potrošenih novčanih sredstava korisniku.
- * Evidencija treba sadržavati informacije o svakom unosu novčanih sredstava (npr. datum, iznos, tip novčane sredstva).

3. **Računanje ukupnog potrošenog iznosa:**

- * Aplikacija treba računati ukupni potrošeni iznos novčanih sredstava.
- * Aplikacija treba prikazati ukupni potrošeni iznos korisniku.

Nefunkcionalni zahtjevi:

1. **Sučelje:**

- * Aplikacija treba imati grafičko sučelje.

2. **Korisnici:**

- * Aplikacija će koristiti samo jedan korisnik.
- * Nije potrebna nikakva sigurnost.

3. **Podrška:**

- * Nije potrebna podrška za korisnika.

4. **Tehnologija:**

- * Aplikacija će biti izrađena u C# tehnologiji.

5. **Platforma:**

- * Aplikacija će biti izrađena za stolno računalo.

6. **Baza podataka:**

** Aplikacija će koristiti bazu podataka za spremanje podataka.*

Dodatne informacije:

Nije potrebno više informacija za nastavak rada na ovom projektu.

Zadovoljan sam sa odgovorima primjenjenih tehnika prilagodbe modela, jer mislim da jasno ukazuju na funkcionalne i nefunkcionalne zahtjeve.

4.3. Primjena GPT modela u fazi dizajna i arhitekture

Sljedeća aktivnost je proces dizajna i arhitekture. U ovom procesu mora se detaljno razraditi svaki dio budućeg softvera. Moraju se donijet odluke koje će zadovoljit funkcionalne, ali i nefunkcionalne zahtjeve. Dakle pruža potpuni prikaz arhitekture, različitih slojeva i odnosa između tih slojeva. Želim dobiti odgovor koji sadrži bitne elemente u arhitekturi. Neki od elemenata su slojevitost aplikacije, te podatkovni modeli i baza podataka koja će se koristiti.

Prvi upit bio je na osnovnoj razini modela. Upit je izgledao ovako:

Na temelju definiranog funkcionalnog zahtijeva, definiraj mi osnovnu arhitekturu rješenja te podatkovni model.

Funkcionalni zahtjev:

Unos novčanih sredstava:

Korisnik može unijeti novčane sredstva u aplikaciju.

Aplikacija treba spremiti unosene podatke u bazi podataka.

Upisao sam definirane funkcionalne i ne funkcionalne zahtjeve iz prijašnjeg zadatka i zatražio da mi da osnovnu arhitekturu rješenja te podatkovni model za prvi funkcionalni zahtjev (unos novčanih sredstava) kako bi mogao nastaviti sa sljedećim procesima.

Odgovor: *Na temelju definiranog funkcionalnog zahtijeva, osnovna arhitektura rješenja bi bila sljedeća:*

Arhitektura rješenja:

- 1. Korisnik koristi aplikaciju za unos novčanih sredstava.*
- 2. Aplikacija prikuplja unosene podatke (npr. iznos, tip novčane sredstva, datum itd.).*
- 3. Aplikacija sprema unosene podatke u bazi podataka.*

Podatkovni model:

1. **Tabela "Novčani sredstva":**

* *id (primarni ključ): jedinstveni identifikator za svaki unos novčanih sredstava.*

* *iznos: iznos unesenih novčanih sredstava.*

* *tip: tip novčane sredstva (npr. kreditna kartica, novčići itd.).*

* *datum: datum unosa novčanih sredstava.*

* *korisnik_id: referenca na korisnika koji je unio novčane sredstva.*

Baza podataka:

- *Relacijska baza podataka (MS SQL server) će biti korištena za spremanje podataka.*

Entity Framework:

- *Entity Framework će biti korišteno za apstrahiranje relacijske perspektive i pristup podacima u objektno-orijentiranom stilu.*

Tehnikom few shot prompting napravio sam upit i priložio više primjera. Jedan od primjera izgleda ovako:

Dokumentacija za Dizajn i Arhitekturu Aplikacije Praćenja Kućnog Budžeta

1. Sloj korisničkog sučelja (UI):

- **Tehnologija:** *Implementiran koristeći Windows Presentation Foundation (WPF) za bogato korisničko sučelje i XAML za dizajn.*
- **Funkcionalnosti:**
 - *Unos prihoda i rashoda: Korisnici mogu unositi podatke putem intuitivnog obrasca s poljima za iznos, datum, i kategoriju.*
 - *Prikaz bilance: Trenutna bilanca se prikazuje u obliku sažetka na početnom ekranu s mogućnošću detaljnog pregleda.*
 - *Grafički izvještaji: Vizualizacija podataka pomoću grafičkih prikaza kao što su trakasti i kružni grafikoni koji prikazuju trendove prihoda i rashoda.*
- **Posebna pažnja:** *Osigurati da sučelje bude responsivno i prilagodljivo različitim veličinama ekrana i rezolucijama.*

2. Sloj poslovne logike:

- **Funkcionalnosti:**
 - *Kalkulacije: Izračunavanje ukupnog prihoda, rashoda i trenutne bilance. Implementacija pravila za validaciju unesenih podataka.*
 - *Upravljanje podacima: Komunikacija sa slojem podataka za dohvaćanje i pohranu informacija.*

- **Posebna pažnja:** Osigurati da poslovna logika bude modularna i lako testirana. Implementirati asinkroni pristup za dugotrajne operacije kako bi se poboljšala korisnička iskustva.

3. Sloj podataka:

- **Tehnologija:** Korištenje SQLite baze podataka za lokalnu pohranu podataka. SQLite je odabran zbog svoje jednostavnosti i lakog upravljanja lokalnim podacima.
- **Funkcionalnosti:**
 - **Pohrana podataka:** Svi podaci o transakcijama, korisničkim postavkama i povijesti treba biti pohranjeni u SQLite bazi podataka.
- **Posebna pažnja:** Osigurati integritet podataka i implementirati odgovarajuće mehanizme za zaštitu podataka od mogućih oštećenja ili gubitka.

4. Sigurnost i pristupačnost:

- **Sigurnost:** Budući da aplikaciju koristi samo jedan korisnik, sigurnosni zahtjevi su minimalni. Međutim, osigurati da se podaci ne mogu lako izgubiti ili oštetiti.
- **Pristupačnost:** Sučelje treba biti dizajnirano s obzirom na principe pristupačnosti, uključujući podršku za čitatelje ekrana i odgovarajuće kontraste boja.

Na osnovu toga sam dobio sljedeći odgovor.

Odgovor: Dokumentacija za Dizajn i Arhitekturu Aplikacije za Upravljanje Novčanim Sredstvima

Sloj korisničkog sučelja (UI):

Tehnologija: Implementiran koristeći Windows Presentation Foundation (WPF) za bogato korisničko sučelje i XAML za dizajn.

Funkcionalnosti:

Unos novčanih sredstava: Korisnik može unijeti novčane sredstva u aplikaciju putem intuitivnog obrasca s poljima za iznos, datum i kategoriju.

Prikaz evidencije potrošenih novčanih sredstava: Trenutna evidencija potrošenih novčanih sredstava se prikazuje u obliku sažetka na početnom ekranu s mogućnošću detaljnog pregleda.

Grafički izvještaji: Vizualizacija podataka pomoću grafičkih prikaza kao što su trakasti i kružni grafikoni koji prikazuju trendove potrošenih novčanih sredstava.

Računanje ukupnog potrošenog iznosa: Aplikacija računa ukupni potrošeni iznos novčanih sredstava i prikazuje rezultat u grafičkom sučelju.

Posebna pažnja: Osigurati da sučelje bude responsivno i prilagodljivo različitim veličinama ekrana i rezolucijama.

Sloj poslovne logike:

Funkcionalnosti:

Upravljanje podacima: Komunikacija sa slojem podataka za dohvaćanje i pohranu informacija o novčanim sredstvima.

Kalkulacije: *Izračunavanje ukupnog potrošenog iznosa novčanih sredstava.*

Validacija podataka: Provjera ispravnosti unesenih podataka (npr. ispravni formati datuma, pozitivni iznosi).

Posebna pažnja: *Osigurati da poslovna logika bude modularna i lako testirana. Implementirati asinkroni pristup za dugotrajne operacije kako bi se poboljšala korisnička iskustva.*

Sloj podataka:

Tehnologija: *Korištenje SQLite baze podataka za lokalnu pohranu podataka. SQLite je odabran zbog svoje jednostavnosti i lakog upravljanja lokalnim podacima.*

Funkcionalnosti:

Pohrana podataka: *Svi podaci o novčanim sredstvima, potrošenim iznosima i evidenciji treba biti pohranjeni u SQLite bazi podataka.*

Posebna pažnja: *Osigurati integritet podataka i implementirati odgovarajuće mehanizme za zaštitu podataka od mogućih oštećenja ili gubitka.*

Sigurnost i pristupačnost:

Sigurnost: *Budući da aplikacija koristi samo jedan korisnik, sigurnosni zahtjevi su minimalni. Međutim, osigurati da se podaci ne mogu lako izgubiti ili oštetiti.*

Pristupačnost: *Sučelje treba biti dizajnirano s obzirom na principe pristupačnosti, uključujući podršku za čitatelje ekrana i odgovarajuće kontraste boja.*

Ovaj odgovor je jako kvalitetan i sadrži dosta opširan opis. Problem koji sam uočio je taj da u originalnim uputama klijenta, postoji samo jedan klijent koji koristi aplikaciju na samo jednom uređaju. Primjer u few shot promptingu je imao pristupačnost za ljude sa poteškoćama gledanja, tako da je taj primjer imao jako velik utjecaj na ovaj tu odgovor koji nema smisla, jer bi klijent sam spomenuo da ima problema sa vidom i da bi se nešto tu trebalo napraviti.

GPT je prilagođen sa dokumentima koji objašnjavaju šta je točno dizajn i kakve su sve arhitekture moguće, bilo je potrebno neko vrijeme da obradi datoteku. Prompt je napravljen, te je jako dugo trebalo da vrati odgovor (oko 7 minuta). Odgovor koji sam dobio je bio sljedeći:

Dokumentacija arhitekture i dizajna je važan korak u procesu razvoja softvera. Ovdje je potpuna dokumentacija za zahtjeve koje smo imali:

Dokumentacija arhitekture i dizajna

Naziv projekta: Aplikacija za evidenciju potrošenih novčanih sredstava

Opis projekta: Aplikacija će omogućiti korisniku da unese novčane sredstva, prikaže evidenciju potrošenih novčanih sredstava i računa ukupni potrošeni iznos.

Arhitektura projekta:

Sloj korisničkog sučelja (WPF): Aplikacija će imati grafičko sučelje koje će omogućiti korisniku da unese novčane sredstva, prikaže evidenciju potrošenih novčanih sredstava i računa ukupni potrošeni iznos.

Sloj logike aplikacije: Aplikacija će imati sloj logike koji će upravljati unosom novčanih sredstava, prikazom evidencije potrošenih novčanih sredstava i računanjem ukupnog potrošenog iznosa.

Sloj baze podataka: Aplikacija će koristiti bazu podataka za spremanje unosenih podataka.

Dizajn aplikacije:

Unos novčanih sredstava:

+ Korisnik će unijeti novčane sredstva u aplikaciju kroz formu za unos.

+ Aplikacija će spremiti unosene podatke u bazu podataka.

Prikaz evidencije potrošenih novčanih sredstava:

+ Korisnik će vidjeti evidenciju potrošenih novčanih sredstava u aplikaciji kroz formu za prikaz.

+ Aplikacija će prikazati evidenciju u grafičkom sučelju.

Računanje ukupnog potrošenog iznosa:

+ Aplikacija će računati ukupni potrošeni iznos novčanih sredstava.

+ Aplikacija će prikazati rezultat računanja u grafičkom sučelju.

Kriteriji zahtjeva:

Lakosti uporabe: Aplikacija treba biti laka za uporabu i korisniku treba biti jednostavno upoznat s njom.

Brzina rada: Aplikacija treba raditi brzo i bez zamora.

Nedostatak sigurnosti: Aplikacija nije potrebna sigurnost, jer koristi samo jedan korisnik.

Nedostatak podrške: Aplikacija nije potrebna podrška, jer koristi samo jedan korisnik.

Rješenje problema:

Unos novčanih sredstava: Aplikacija će koristiti formu za unos za unos novčanih sredstava.

Prikaz evidencije potrošenih novčanih sredstava: Aplikacija će koristiti formu za prikaz za prikaz evidencije potrošenih novčanih sredstava.

Računanje ukupnog potrošenog iznosa: Aplikacija će koristiti algoritam za računanje ukupnog potrošenog iznosa.

Ova dokumentacija arhitekture i dizajna će biti temelj za nastavak razvoja aplikacije.

Odgovor koji sam dobio imao je dobre elemente, ali nije potpuno zadovoljavajući. Sa trenutnim alatima na raspolaganju, nije bilo moguće izgenerirati različite dijagrame, tako da taj dio neće biti obrađen.

4.4. Korištenje GPT modela za automatizaciju implementacije

Nakon što smo razmislili i upotpunili dokumentaciju o arhitekturi i dizajnu, možemo nastaviti sa sljedećim korakom, a to je implementacija programskog koda. Implementaciju ću pokušat automatizirati sa GPT modelom. Očekujem kvalitetan kod iz ove aktivnosti.

Prilagođenom GPT modelu sam dao PDF koji se sastoji od puno različitih C# izvora. Tu su najbolje prakse C# i najbolje prakse prilikom izrade Windows Forms aplikacija. Upitom sam zatražio da mi generira kod za sloj poslovne logike, s obzirom na dokumentaciju o dizajnu i arhitekturi koju sam dobio od prijašnjeg procesa. Odgovor sam dobio u obliku koda.

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace BusinessLogicLayer
{
    public class EvidencijaPotrošenihNovčanihSredstava
    {
        private readonly DbContext _dbContext;

        public EvidencijaPotrošenihNovčanihSredstava(DbContext dbContext)
        {
            _dbContext = dbContext;
        }

        public void UnosNovčanihSredstava(NovčaniSredstva novčaniSredstva)
        {
            _dbContext.NovčaniSredstva.Add(novčaniSredstva);
            _dbContext.SaveChanges();
        }
    }
}
```

```

public void PrikazEvidencijePotrošenihNovčanihSredstava ()
{
    var evidencija = _dbContext.NovčaniSredstva.ToList();
    // Prikaz evidencije u grafičkom sučelju
}

public void RačunanjeUkupnogPotrošenogIznosa ()
{
    var    ukupniIznos    =    _dbContext.NovčaniSredstva.Sum(ns    =>
ns.Iznos);
    // Prikaz rezultata računanja u grafičkom sučelju
}
}

public class NovčaniSredstva
{
    public int Id { get; set; }
    public decimal Iznos { get; set; }
}

public class DbContext
{
    public DbSet<NovčaniSredstva> NovčaniSredstva { get; set; }
}
}

```

Kod nije najkvalitetniji iz par razloga. Imena nekih funkcija su preduga i teška za čitati. Bilo bi poželjno da neke od funkcija vraćaju specifične podatke i da se postavi ispravan povratni tip.

4.5. GPT modeli u testiranju softvera

Kako bi se uvjerali da određeni kod ispravno funkcionira, treba ga rigorozno testirati. To se obavlja u fazi testiranja softvera. Očekujem kvalitetan kod uz pomoć kojeg se mogu vršiti testiranja.

Postavio sam upit da izgenerira jedinične testove za kod generiran RAG metodom.

```

using Xunit;
using Moq;
using System.Collections.Generic;
using System.Linq;

namespace BusinessLogicLayer.Tests
{
    public class EvidencijaPotrošenihNovčanihSredstavaTests
    {
        [Fact]
        public void UnosNovčanihSredstava_DodajeNovčaniSredstvaUBazu ()
        {
            // Aranžiranje
            var dbContextMock = new Mock<DbContext>();

```



```

        var novčaniSredstva = new NovčaniSredstva { Iznos = 10.0m };
        var novčaniSredstvaList = new List<NovčaniSredstva> {
novčaniSredstva };

        dbContextMock.Setup(db =>
db.NovčaniSredstva).Returns(novčaniSredstvaList);
        var evidencijaPotrošenihNovčanihSredstava = new
EvidencijaPotrošenihNovčanihSredstava(dbContextMock.Object);

        // Aktiviranje

evidencijaPotrošenihNovčanihSredstava.UnosNovčanihSredstava(novčaniSredstva
);

        // Preverjanje
        dbContextMock.Verify(db =>
db.NovčaniSredstva.Add(It.IsAny<NovčaniSredstva>()), Times.Once);
        dbContextMock.Verify(db => db.SaveChanges(), Times.Once);
    }

    [Fact]
    public void PrikazEvidencijePotrošenihNovčanihSredstava_VracaEvidenciju()
    {
        // Aranžiranje
        var dbContextMock = new Mock<DbContext>();
        var novčaniSredstvaList = new List<NovčaniSredstva> { new
NovčaniSredstva { Iznos = 10.0m } };

        dbContextMock.Setup(db =>
db.NovčaniSredstva).Returns(novčaniSredstvaList);
        var evidencijaPotrošenihNovčanihSredstava = new
EvidencijaPotrošenihNovčanihSredstava(dbContextMock.Object);

        // Aktiviranje
        var evidencija =
evidencijaPotrošenihNovčanihSredstava.PrikazEvidencijePotrošenihNovčanihSre
dstava();

        // Preverjanje
        Assert.IsType<List<NovčaniSredstva>>(evidencija);
        Assert.Equal(1, evidencija.Count);
    }

    [Fact]
    public void RačunanjeUkupnogPotrošenogIznosa_VracaUkupniIznos()
    {
        // Aranžiranje
        var dbContextMock = new Mock<DbContext>();
        var novčaniSredstvaList = new List<NovčaniSredstva> { new
NovčaniSredstva { Iznos = 10.0m } };

        dbContextMock.Setup(db =>
db.NovčaniSredstva).Returns(novčaniSredstvaList);
        var evidencijaPotrošenihNovčanihSredstava = new
EvidencijaPotrošenihNovčanihSredstava(dbContextMock.Object);

```

```

        // Aktiviranje
        var ukupniIznos = evidencija.PotrošenihNovčanihSredstava.RačunanjeUkupnogPotrošenogIznosa();

        // Preverjanje
        Assert.Equal(10.0m, ukupniIznos);
    }
}

```

Kod nije najkvalitetniji. Sadrži duga imena varijabli, te nije ispravno nazvao funkcije.

Few shot promptingom dobio sam sljedeći odgovor:

```

[TestClass]
public class EvidencijaPotrošenihNovčanihSredstavaTests
{
    [TestMethod]
    public void Test_UnosNovčanihSredstava_SredstvaIspravnoUnesena ()
    {
        var dbContext = new DbContext();
        var evidencija = new EvidencijaPotrošenihNovčanihSredstava(dbContext);
        var novčaniSredstva = new NovčaniSredstva { Iznos = 100 };
        evidencija.UnosNovčanihSredstava(novčaniSredstva);

        var sredstva = dbContext.NovčaniSredstva.ToList();

        Assert.AreEqual(1, sredstva.Count);
        Assert.AreEqual(100, sredstva.First().Iznos);
    }

    [TestMethod]
    public void Test_RačunanjeUkupnogPotrošenogIznosa_IznosIspravnoRačunat ()
    {
        var dbContext = new DbContext();
        var evidencija = new EvidencijaPotrošenihNovčanihSredstava(dbContext);
        var novčaniSredstva1 = new NovčaniSredstva { Iznos = 50 };
        var novčaniSredstva2 = new NovčaniSredstva { Iznos = 75 };

        evidencija.UnosNovčanihSredstava(novčaniSredstva1);
        evidencija.UnosNovčanihSredstava(novčaniSredstva2);

        var ukupniIznos = evidencija.RačunanjeUkupnogPotrošenogIznosa();

        Assert.AreEqual(125, ukupniIznos);
    }
}

```

Ovaj odgovor je jako dobar, jer se pridržava dosta dobre konvencije o imenovanju funkcija testiranja, gdje zadnji dio u imenu funkcije sadrži očekivani rezultat.

Koristeći metodu RAG, uploadao sam datoteku koja sadrži detaljno opisane elemente jediničnog testiranja. Dao sam kod koji je izgeneriran iz protekle RAG faze te napravio upit da mi napiše jedinične testove.

```
using Xunit;
using Moq;
using System.Collections.Generic;
using System.Linq;

namespace BusinessLogicLayerTests
{
    public class EvidencijaPotrošenihNovčanihSredstavaTests
    {
        [Fact]
        public void UnosNovčanihSredstava_DodajeNovčaniSredstavaUBazuPodataka ()
        {
            // Aranžman
            var dbContextMock = new Mock<DbContext>();
            var novčaniSredstva = new NovčaniSredstva { Id = 1, Iznos = 10 };
        };

        var evidencija = new List<NovčaniSredstva> { novčaniSredstva };

        dbContextMock.Setup(db =>
db.NovčaniSredstva).Returns(evidencija);
        var evidencijaPotrošenihNovčanihSredstava = new
EvidencijaPotrošenihNovčanihSredstava (dbContextMock.Object);

        // Akcija

evidencijaPotrošenihNovčanihSredstava.UnosNovčanihSredstava (novčaniSredstva
);

        // Preveravanje
        dbContextMock.Verify(db =>
db.NovčaniSredstva.Add(novčaniSredstva), Times.Once);
        dbContextMock.Verify(db => db.SaveChanges(), Times.Once);
    }

    [Fact]
    public void PrikazEvidencijePotrošenihNovčanihSredstava_PrikazujeEvidenciju ()
    {
        // Aranžman
        var dbContextMock = new Mock<DbContext>();
        var novčaniSredstva = new List<NovčaniSredstva> { new
NovčaniSredstva { Id = 1, Iznos = 10 } };

        dbContextMock.Setup(db =>
db.NovčaniSredstva).Returns(novčaniSredstva);
        var evidencijaPotrošenihNovčanihSredstava = new
EvidencijaPotrošenihNovčanihSredstava (dbContextMock.Object);

        // Akcija
```

```

evidencijaPotrošenihNovčanihSredstava.PrikazEvidencijePotrošenihNovčanihSredstava();

        // Preveravanje
        dbContextMock.Verify(db => db.NovčaniSredstva.ToList(),
Times.Once);
    }

    [Fact]
    public void RačunanjeUkupnogPotrošenogIznosa_RačunaUkupniIznos ()
    {
        // Aranžman
        var dbContextMock = new Mock<DbContext>();
        var novčaniSredstva = new List<NovčaniSredstva> { new
NovčaniSredstva { Id = 1, Iznos = 10 } };

        dbContextMock.Setup(db =>
db.NovčaniSredstva).Returns(novčaniSredstva);
        var evidencijaPotrošenihNovčanihSredstava = new
EvidencijaPotrošenihNovčanihSredstava(dbContextMock.Object);

        // Akcija
        var ukupniIznos =
evidencijaPotrošenihNovčanihSredstava.RačunanjeUkupnogPotrošenogIznosa();

        // Preveravanje
        dbContextMock.Verify(db => db.NovčaniSredstva.Sum(ns =>
ns.Iznos), Times.Once);
        Assert.Equal(10, ukupniIznos);
    }
}
}

```

Testovi izgledaju dobro, postavlja komentare za arrange act i assert. Nazivi funkcija bi trebali biti malo bolji. No imena funkcija su jako duga i velika.

Najzadovoljniji sam tehnikom upita poticanja modela uz nekoliko primjera.

4.6. Održavanje softvera uz podršku GPT modela

U fazi održavanja softvera, često može doći do refaktoriranja koda. Refaktoriranje može biti iz dva razloga. U kodu se može nalaziti greška, ili jednostavno funkcionalnost treba drugačije implementirati, možda poboljšati. Drugi razlog je promjena verzije programskog jezika. Kada se mijenja verzija programskog jezika, treba paziti na promjene u sintaksi zbog kojih može doći do pucanja programa.

Napravio sam upit običnom modelu i dao mu originalni kod sloja poslovne logike i zatražio sam da mi refaktorira kod za async i await. Upit je izgledao ovako:

Pitanje: Refaktoriraj kod, tako da koristi `await` i `async`. Originalni kod izgleda ovako:

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace BusinessLogicLayer
{
    public class EvidencijaPotrošenihNovčanihSredstava
    {
        private readonly DbContext _dbContext;
        ...
    }
}
```

Odgovor: Ovdje je refaktorirani kod koji koristi `await` i `async`:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BusinessLogicLayer
{
    public class EvidencijaPotrošenihNovčanihSredstava
    {
        private readonly DbContext _dbContext;

        public EvidencijaPotrošenihNovčanihSredstava(DbContext dbContext)
        {
            _dbContext = dbContext;
        }

        public async Task UnosNovčanihSredstavaAsync(NovčaniSredstva
novčaniSredstva)
        {
            _dbContext.NovčaniSredstva.Add(novčaniSredstva);
            await _dbContext.SaveChangesAsync();
        }

        public async Task PrikazEvidencijePotrošenihNovčanihSredstavaAsync()
        {
            var evidencija = await _dbContext.NovčaniSredstva.ToListAsync();
            // Prikaz evidencije u grafičkom sučelju
        }

        public async Task RačunanjeUkupnogPotrošenogIznosaAsync()
        {
            var ukupniIznos = await _dbContext.NovčaniSredstva.SumAsync(ns
=> ns.Iznos);
            // Prikaz rezultata računanja u grafičkom sučelju
        }
    }

    public class NovčaniSredstva
    {
        public int Id { get; set; }
    }
}
```

```

        public decimal Iznos { get; set; }
    }

    public class DbContext
    {
        public DbSet<NovčaniSredstva> NovčaniSredstva { get; set; }
    }
}

```

U ovom refaktoriranom kodu metode UnosNovčanihSredstava, PrikazEvidencijePotrošenihNovčanihSredstava i RačunanjeUkupnogPotrošenogIznosa su promijenjene u metode async koje koriste await za pozivanje metoda SaveChanges i ToListAsync i SumAsync.

RAG tehnikom sam dao datoteku koja sadrži relevantne informacije za asinkrono programiranje, jer se uvidilo da bi takva promjena trebala nastati. Ponovio sam upit od gore.

Odgovor:

Da bi refaktorirali kod da koristi async i await, moramo promijeniti metode koje koriste baze podataka da vrše operacije asinhrono.

Prvo, moramo promijeniti tip metode UnosNovčanihSredstava da bude async i koristiti await za čekanje završetka operacije u bazi podataka.

```

public async void UnosNovčanihSredstava(NovčaniSredstva novčaniSredstva)
{
    _dbContext.NovčaniSredstva.Add(novčaniSredstva);
    await _dbContext.SaveChangesAsync();
}

```

Slično, moramo promijeniti metode koje koriste bazu podataka da vrše operacije asinkrono.

```

public async Task PrikazEvidencijePotrošenihNovčanihSredstava()
{
    var evidencija = await _dbContext.NovčaniSredstva.ToListAsync();
    // Prikaz evidencije u grafičkom sučelju
}

public async Task RačunanjeUkupnogPotrošenogIznosa()
{
    var ukupniIznos = await _dbContext.NovčaniSredstva.SumAsync(ns =>
ns.Iznos);
    // Prikaz rezultata računanja u grafičkom sučelju
}

```

U ovom primjeru, koristimo ToListAsync i SumAsync metode koje su dostupne u Entity Framework Core-u za asinhrono izvršavanje operacija u bazi podataka.

Kod koji koristi async i await mora biti pozvan u asinhronom kontekstu, kao što je primjer u nastavku:

```
public async Task Main()
{
    var dbContext = new DbContext();
    var evidencijaPotrošenihNovčanihSredstava = new
EvidencijaPotrošenihNovčanihSredstava(dbContext);

    await evidencijaPotrošenihNovčanihSredstava.UnosNovčanihSredstava(new
NovčaniSredstva { Iznos = 100 });
    await
evidencijaPotrošenihNovčanihSredstava.PrikazEvidencijePotrošenihNovčanihSre
dstava();
    await
evidencijaPotrošenihNovčanihSredstava.RačunanjeUkupnogPotrošenogIznosa();
}
```

U ovom primjeru, koristimo Main metodu kao asinhroni kontekst za pozivanje metoda koje koriste async i await.

Kod osnovnog modela bez upotrebe RAG načina je kvalitetniji. Razlog je što metode vraćaju Task a ne void.

4.7. Osvrt

Prilikom uporabe PrivateGPT-a, prvo sam isti morao instalirati. Prvi pokušaj instalacije na Linux virtualnoj mašini je bio uspješan. Sučelje je funkcioniralo i zahtjevi su se mogli slat. Trebalo je oko dvije minute da se procesira zahtjev i da se nekakav odgovor ispiše. Također sam isprobao funkcionalnost uploadanja datoteka što je funkcioniralo. Bilo je potrebno duže vremena da se procesira datoteka, što naravno ovisi o veličini datoteke. Nakon testiranja kompliciranijih upita i dosta velikih datoteka, zahtjev bi se obrađivao po više minuta i nastala bi greška te bi cijeli proces zastao. Više puta sam pokušao, no uvijek je završavalo isto, sa crvenim slovima u ispisu konzole u pozadini i porukom o grešci na sučelju. Zaključio sam da je moguć razlog tome što virtualna mašina ne može pristupiti svim resursima računala, te zato dolazi do problema. Nakon toga sam pokušao instalirati alat na glavni operacijski sustav (Windows 10 verzija 22H2).

U aktivnostima analiza i specifikacija zahtjeva, tehnike prilagodbe RAG i few shot prompting su se pokazale dobre. U aktivnostima testiranja i održavanja tehnika prilagodbe few shot prompting se pokazala dobrom. U ostalim aktivnostima poput dizajna i arhitekture i automatizacijom implementacije tehnike prilagodbe se nisu pokazale najbolje. Može se zaključiti da tehnike prilagodbe sigurno mogu pomoć u procesu razvoja softvera.

5. Zaključak

U ovom radu odrađena je povijest GPT-a od prve do zadnje generacije. GPT 1 nije bio dostupan javnosti, GPT-2 je bio velik skok u odnosu na prvu generaciju i bio je dostupan ljudima. GPT-3 je imao preko 50 različitih modela. GPT-4 je najmoćniji i sposoban je prolaziti ispite iz različitih industrija. Dio rada se također fokusirao na samu arhitekturu tehnologije transformera u kojem je objašnjeno kako zapravo funkcionira ova tehnologija. Nadalje smo vidili kako se GPT modeli koriste u stvarnom svijetu u različitim industrijama, te je zaključeno da polako ali sigurno njegova uporaba samo može imati više utjecaja i koristi s obzirom na poboljšanja koja obećava.

Sljedeće poglavlje fokusiralo se na tehniku i alate prilagodbe GPT-a. Tu su opisane tehnike poput finog podešavanja, poticanja modela sa nekoliko primjera i RAG-a koje imaju svoje prednosti i mane. Alati koji su prikazani su customGPT builder od strane OpenAI-a i PrivateGPT koji se izvodi lokalno. Zanimljivosti i funkcionalnosti i jednog i drugog su opisane, te razlozi zašto sam se opredjelio baš za PrivateGPT. Također sam prikazao instalaciju korak po korak.

Zadnje poglavlje uključivalo je praktični rad prilagodbe GPT modela gdje sam na pokušao dobiti što kvalitetnije odgovore uz pomoć tehnika RAG i few shot promptinga. Odgovore sam dobivao i uz obični model bez ikakve prilagodbe kako bi se moglo utvrditi pomaže li prilagodba u procesu razvoja softvera.

Mogu zaključiti da metode RAG i few shot prompting sigurno mogu poboljšati razvoj softvera. Na primjer metoda RAG pomaže u definiranju zahtjeva i specifikiranju dizajna i arhitekture, dok se u procesu automatizacije implementacije nije najbolje pokazala. Smatram da je razlog tome što osnovni model već dosta dobro zna generirati kod. Few-shot prompting se također pokazao dobar u definiranju zahtjeva, dizajna i arhitekture, no opazio sam da dosta pažnje daje primjerima koji su mu pokazani i time kviri kvalitetu odgovora. U fazi testiranja se pokazao jako dobar, jer ispravan primjer sa dobrom konvencijom imenovanja pomoglo je da rezultat bude jako kvalitetan. Trenutno prilagodba može donijeti pozitivne rezultate, možda čak i više u prilagodbi za drugu vrstu zadatka, koju osnovni model ne zna najbolje obavljati.

GPT modeli će napredovati u budućnosti, te možda uopće neće biti potrebe za bilo kakvom prilagodbom. Potrebno je paziti da se ne oslanjamo previše na GPT (umjetnu inteligenciju općenito), jer uvijek postoji rizik od neispravnog funkcioniranja.

Popis literature

- [1] "HuggingFace LLM Tutorial." [Online]. Available: https://huggingface.co/docs/transformers/en/llm_tutorial
- [2] K. N. Alec Radford and I. S. Tim Salimans, "Improving Language Understanding by Generative Pre-Training." Accessed: Feb. 09, 2024. [Online]. Available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
- [3] M. B. Irene Solaiman *et al.*, "Release Strategies and the Social Impacts of Language Models." [Online]. Available: <https://arxiv.org/abs/1908.09203>
- [4] B. M. Tom B Brown *et al.*, "Language Models are Few-Shot Learners." [Online]. Available: <https://arxiv.org/abs/2005.14165>
- [5] "OpenAI API pricing." [Online]. Available: <https://openai.com/api/pricing/>
- [6] "The GPT-3 Family: 50+ Models." [Online]. Available: <https://lifecycle.ai/gpt-3/>
- [7] "ChatGPT." [Online]. Available: <https://chatgpt.com/>
- [8] "GPT-4 research." [Online]. Available: <https://openai.com/index/gpt-4-research/>
- [9] Bea Stollnitz, "The Transformer architecture of GPT models." [Online]. Available: <https://bea.stollnitz.com/blog/gpt-transformer/>
- [10] "How Chat GPT is revolutionising the IT industry." [Online]. Available: <https://mitiztechnologies.com/blog/how-chat-gpt-is-revolutionising-the-it-industry>
- [11] Nathan Thompson, "AI for Content Creation: How to Get Started (& Scale)." [Online]. Available: <https://www.copy.ai/blog/ai-content-creation#:~:text=AI%20content%20tools%20leverage%20machine,ad%20copy%2C%20and%20much%20more.>
- [12] Dmitrii Lukianov, "Is GPT-4 Better at Translating than Google Translate." [Online]. Available: <https://medium.com/akvelon/is-gpt-4-better-at-translation-than-google-translate-2fd39730af0e>
- [13] Duolingo team, "Introducing Duolingo Max, a learning experience powered by GPT-4." [Online]. Available: <https://blog.duolingo.com/duolingo-max/>
- [14] Hilary Brueck, "The newest version of ChatGPT passed the US medical licensing exam with flying colors - and diagnosed a 1 in 100,000 condition in seconds." [Online]. Available: <https://www.businessinsider.com/chatgpt-passes-medical-exam-diagnoses-rare-condition-2023-4>
- [15] AIContentfy team, "ChatGPT in the Automotive Industry: Improving Vehicle Navigation." [Online]. Available: <https://aicontentfy.com/en/blog/chatgpt-in-automotive-industry-improving-vehicle-navigation>

- [16] Purdue Global Law School, "The potential benefits and risks of ChatGPT in legal practice." [Online]. Available: <https://www.purduegloballawschool.edu/blog/news/risks-and-benefits-chat-gpt-legal-practice>
- [17] Nick Pericle, "A Custom GPT Can Be Whatever Your Company Needs It To Be." [Online]. Available: <https://www.profitoptics.com/blog/a-custom-gpt-can-be-whatever-your-company-needs-it-to-be#:~:text=Custom%20GPTs%20enhance%20productivity%20by,and%20training%20a%20new%20employee.>
- [18] Dave Bergmann, "What is fine-tuning." [Online]. Available: <https://www.ibm.com/topics/fine-tuning>
- [19] Google cloud team, "What is Retrieval-Augmented Generation (RAG)?" [Online]. Available: <https://cloud.google.com/use-cases/retrieval-augmented-generation>
- [20] AWS amazon team, "What is RAG (Retrieval-Augmented Generation)?" [Online]. Available: <https://aws.amazon.com/what-is/retrieval-augmented-generation/>
- [21] Prompting guide team, "Few-shot Prompting." [Online]. Available: <https://www.promptingguide.ai/techniques/fewshot>
- [22] PrivateGPT docs team, "PrivateGPT Main Concepts." [Online]. Available: <https://docs.privategpt.dev/installation/getting-started/main-concepts>
- [23] "GPT store." [Online]. Available: <https://chatgpt.com/gpts>
- [24] Jessica Lau, "How to create a custom GPT: A beginner's guide." [Online]. Available: <https://zapier.com/blog/custom-chatgpt/>
- [25] "Hugging Face Transformers." [Online]. Available: <https://huggingface.co/docs/transformers/en/index>
- [26] "Adapter Hub." [Online]. Available: <https://adapterhub.ml/>
- [27] "Eleuther AI." [Online]. Available: <https://www.eleuther.ai/>
- [28] "PrivateGPT Installation." [Online]. Available: <https://docs.privategpt.dev/installation/getting-started/installation>
- [29] Bedy Kharisma, "Run PrivateGPT on Windows."

Popis slika

Slika 1: Prikaz svih modela iz generacije GPT-3 (izvor: [6]).....	5
Slika 2: Upit mogućnosti promjene verzije (izvor: slika autora)	6
Slika 3: Tokenizacija engleskog jezika (izvor: slika autora)	7
Slika 4: Tokenizacija hrvatskog jezika (izvor: slika autora)	7
Slika 5: Pregled procesa generiranja odgovora uz uporabu RAG (izvor: [20]).....	12
Slika 6: Primjer few-shot promptinga (izvor: slika autora)	13
Slika 7: Pregled PrivateGPT vizualnog sučelja (izvor: slika autora)	14
Slika 8: Pregled GPT trgovine (izvor: slika autora).....	16
Slika 9: Prikaz vizualnog sučelja GPT buildera (izvor: [24])	17
Slika 10: Prikaz konzole pokrenutog PrivateGPT-a (izvor: slika autora)	21

Popis tablica

Tablica 1: Prikaz GPT-3 modela u fazi treniranja.....	4
Tablica 2: Prikaz skupova podataka za treniranje GPT-3	4
Tablica 3: Prikaz mogućnosti prilagodbi za različite korisnike.....	11
Tablica 4: Prikaz različitih opcija modela PrivateGPT-a.....	15