

Univerzitet u Nišu
Elektronski fakultet Niš

SEMINARSKI RAD

ARTIFICIAL INTELLIGENCE FOR LEARNING

Primena LLM API za vežbanje osnovnih programerskih sposobnosti

Studenti:
Jovan Jovanović
Dimitrije Janković

SADRŽAJ

UVOD.....	2
LARGE LANGUAGE MODELS.....	3
PROMPT INŽENJERING.....	3
POSTAVKA REŠENJA.....	4
APLIKACIJA – UPUTSTVO I IMPLEMENTACIJA REŠENJA.....	5
POKRETANJE I IZVRŠENJE.....	8
REZULTATI.....	9
ZAKLJUČAK.....	16
REFERENCE.....	17

UVOD

Razvoj veštačke inteligencije predstavlja prekretnicu u svetu tehnologije jer menja način na koji rešavamo probleme. Ova promena je postala očigledna kroz integraciju veštačke inteligencije u različite sfere naših života. Mnoge aktivnosti koje smo nekada morali da radimo manuelno sada mogu da se izvrše mnogo brže i efikasnije zahvaljujući primeni veštačke inteligencije.

Jedan od ključnih alata u oblasti veštačke inteligencije jesu veliki jezički modeli (**eng. Large Language Models - LLM**), koji se ističu svojom prilagođenošću korisnicima i mogućnošću široke primene. Ovakav model može da se istrenira za širok spektar svrha i ako se adekvatno istrenira za konkretnu namenu onda postaje vrlo vredan i koristan alat. Proces treniranja veštačke inteligencije, poznat kao **prompt engineering**, podrazumeva pripremu modela korišćenjem različitih upita(prompt-ova) kako bi se postigao željeni rezultat.

U skladu s tim, ideja je razviti alat koji će pomoći programerima da steknu i unaprede svoje osnovne veštine, razvijajući programerski način razmišljanja i približavajući se samom načinu radu kompajlera. Posmatrajući izazove koje imaju studenti na predmetu Algoritmi i Programiranje (AiP), uočili smo da se mnogi od njih imaju poteškoće sa razumevanjem i rešavanjem zadataka, posebno ako prethodno nisu imali iskustva sa programiranjem.

Stoga, predstavljamo koncept korišćenja prompt inženjeringa i baze zadataka sa predmeta AiP za obuku LLM alata CHAT GPT kako bi se napravio alat za sticanje znanja i vežbanje osnovnih programerskih sposobnosti .

Kroz primenu tekstova iz ovih zadataka kao podataka za treniranje veštačke inteligencije, nameravamo da koristimo prompt inženjering kako bismo model naučili da se fokusira isključivo na razvijanje osnovnih programerskih sposobnosti i da od toga ne odstupa.

Planiramo da pristupimo treniranju modela koristeći princip **crne kutije**, odnosno da testiramo kakve rezultate dobijamo za kakve upite bez znanja o tome kako je implementiran sam jezički model. Cilj je da programer nauči kako da analizira tekst zadatka kako bi povezao tekstualne zahteve sa odgovarajućim delom koda koji treba da iskoristi za ispunjenje tih zahteva. Takođe cilj je i da se programeru približi način rada kompajlera. Stoga imaćemo dve uloge: programer i program. U ulozi programera korisnik uči analizu zahteva i implementaciju koda a u ulozi programa, korisnik uči kako program "razmišlja". Sve ove aktivnosti sprovodićemo koristeći "CHAT GPT" **API (Application Programming Interface)**.

LARGE LANGUAGE MODELS

Large Language Models (LLM) su sofisticirani sistemi veštačke inteligencije, bazirani na dubokim neuronskim mrežama i trenirani na velikoj količini jezičkih podataka. Tokom procesa mašinskog učenja, modelu se pružaju ogromne količine teksta s odgovarajućim ciljanim izlazom. Nakon treniranja, LLM može generisati jezičke izraze na osnovu različitih unosa, uključujući **promptove** koji olakšavaju interakciju korisnika sa modelom, kao što je slučaj u **CHAT GPT** sistemu. Ovaj proces "samplinga" koristi naučeno jezičko razumevanje kako bi se generisao odgovarajući izlaz.

PROMPT INŽENJERING

Prompt inženjering je relativno novo istraživačko područje koje se odnosi na praksu dizajniranja, usavršavanja i primene promptova ili uputstava koja usmeravaju izlaz velikih jezičkih modela (LLM) kako bi pomogla u različitim zadacima. Suštinski, to je praksa efikasnog interagovanja sa veštačkim inteligencijama radi optimizacije njihovih koristi [1].

U izvoru [1] je navedeno nekoliko konkretnih predloga koji unapređuju veštinu pisanja prompt-ova:

1. **Što je prompt specifičniji, odgovor će verovatno biti precizniji i usredsređeniji.**
Na primer, manje specifičan prompt bi bio: "Reci mi o bolesti srca", dok bi specifičniji bio: "Koji su najčešći faktori rizika za koronarnu arterijsku bolest?".
2. **Važno je opisati kontekst i postaviti okvir oko pitanja** prilikom razgovora s ChatGPT, kao da razgovarate s osobom koju tek poznajete.
Na primer, "Pišem članak o savetima i trikovima za inženjering promptova za radnike u zdravstvu. Možeš li molim te navesti nekoliko tih saveta i trikova sa nekim konkretnim primerima promptova?".
3. **Prilikom postavljanja prompta, prvo identifikujte cilj traženog izlaza.**
Na primer, "Želim dobiti listu od 5 ideja za prezentaciju na naučnom događaju kako bih svoja istraživanja učinio lakše razumljivim."

4. **Postavljanje pitanja u formi uloga** može olakšati željeni proces dobijanja informacija ili ulaz koji tražite u određenom okruženju.
Na primer, "Igraj ulogu Data Scientist-a i objasni inženjering promptova lekaru." ili "Igraj ulogu mog nutricioniste i daj mi savete o izbalansiranoj mediteranskoj ishrani."
5. Jedan od najlakših načina za poboljšanje veština u prompt inženjeringu je **pitati ChatGPT da se uključi u proces i dizajnira promptove za korisnika**.
Na primer, "Koji prompt mogu upotrebiti sada da dobijem bolji izlaz od tebe u ovoj niti/zadatku?"
6. **Iteracija i dorada** su ključne iako su veštine u inženjeringu promptova napredne, jer se LLM-ovi dinamično menjaju. Korisnicima se preporučuje da traže dorade izlaza na osnovu povratnih informacija o prethodnim odgovorima.

POSTAVKA REŠENJA

Rešenje je koncipirano na sledeći način(ideja za rešenje je preuzeta sa izvora [2]): Potrebno je omogućiti **openai** uslugu na nalogu preko koga će da se vrši komunikacija između modela i korisnika. Zatim da se korišćenjem **Node.js** napravi test aplikacija u okruženju **Visual Code** koja će podržavati komunikaciju između modela i korisnika. Kada je komunikacija uspostavljena, potrebno je modelu dostaviti zbirku zadataka i rešenja. Nakon ovog koraka potrebno je korišćenjem prompt inženjeringa istrenirati asistenta na primerima iz zbirke, da pruža odgovore predviđene ciljem projekta. To znači da će asistent nakon treninga unešenim promptovima biti u mogućnosti da pruža podršku učenju programerima u aspektu dijaloga programa i korisnika tog programa.

Da bi prethodne teme sklopili u jednu celinu koja će da pruži rešenje na naše zahteve, prvo moramo da vidimo kako kako doći do CHAT GPT API-a i odgovarajuće usluge.

-Posetimo stranicu <https://openai.com/> .

-Potom u gornjem levom uglu nadjem opciju menija API, koju otvorimo i izaberemo Pricing

-Na stranici Pricing, dobijemo listu modela čiji API možemo da koristimo I na osnovu zahteva i opisa našeg problema izaberemo ono što je nama potrebno. U našem slučaju to je Assistants API koji je pogodan za programere koji razvijaju aplikaciju u koju žele da implementiraju AI asistent i kojima je potreban interpretator programskog koda. Primećujemo takođe da postoji način plaćanja za uslugu interpretiranja koda I za "Retrieval" odnosno mogućnost dobavljanja podataka iz eksternog izvora.

-Nakon toga u gornjem desnom uglu biramo opciju Log in, da bi mogli da dodamo kredite na naš nalog (uplaćivanje kredita neće biti opisano).

-Sada kada imamo dovoljno kredita, možemo da krenemo sa povezivanjem API-ja I naše aplikacije.

-Prvi korak je generisanje jedinstvenog **API key** ključa preko koga možemo da dobijamo uslugu. To radimo na sledeći način: Prilikom Log in, dobijamo dve opcije **Chat GPT** (Za korisničko korišćenje Chat GPT-a) i **API** (za pristup opcijama za API) gde biramo opciju API.

-U meniju s leve strane nađemo opciju API keys, koju kada otvorimo dobijamo stranicu na kojoj kreiramo novi API key. Tada smo spremni da krenemo sa test aplikacijom.

APLIKACIJA – UPUTSTVO I IMPLEMENTACIJA REŠENJA

Okruženje koje smo odabrali za kreiranje test aplikacije je **Visual Code**.

U njemu pravimo fajl koji se zove **".env"** u kome ćemo da sačuvamo naš API key kao promenljivu.

Da bi mogli da krenemo sa radom potrebno je da instaliramo dependencies kroz konzolu:

- **npm init -y** (ova komanda nam kreira fajl **package.json** koji nam je potreban za pokretanje skripte)
- **npm i dotenv openai** (ova komanda omogućava da učitamo .env fajl u API sa kojim radimo I kreira **package-lock.json** fajl)

Potom kreiramo fajl **script.js** koji će da sadrži našu skriptu za izvršavanje komunikacije između korisnika i modela.

U fajlu **package.json**, polje Scripts treba da izmenimo da igleda ovako:

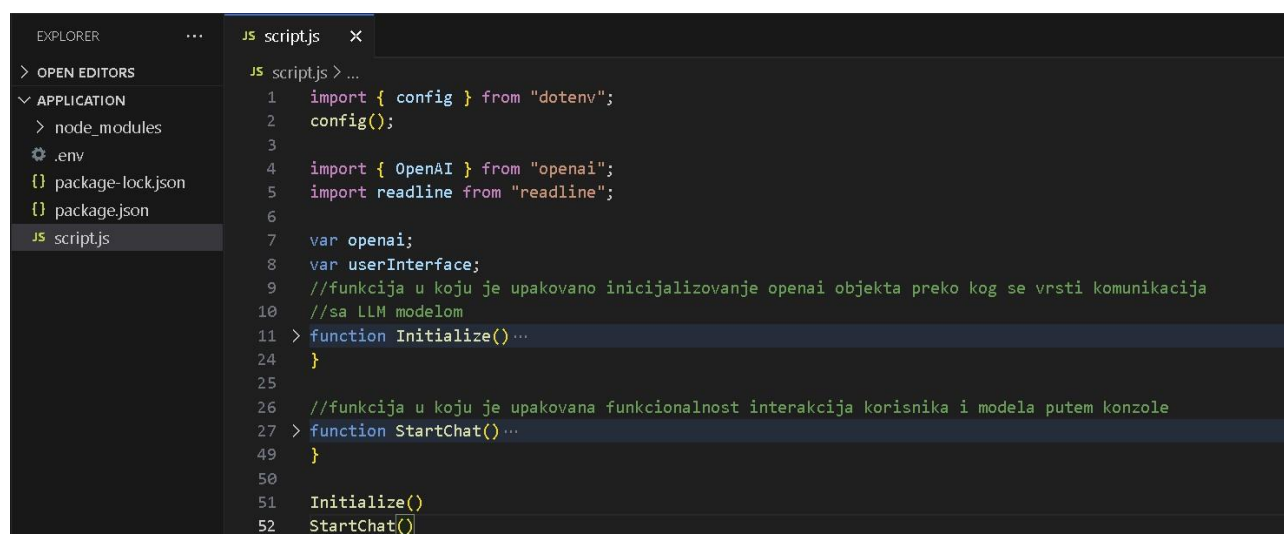
```
"scripts": {  
  "dev": "node script.js"  
},
```

i dodajemo polje

```
"type": "module",
```

kako bi omogućili da se **script.js** pokreće kada startujemo aplikaciju. Aplikaciju startujemo komandom: **npm run dev** koja izvršava sve što se nalazi u našoj skripti.

Naša skripta **script.js** izgleda ovako:



Slika 1.1 Prikaz skripte

Na početku preko import, uključujemo potrebne funkcije, klase i module, i pravimo objekte potrebne objekte. Skripta je podeljena na dve funkcije: **Initialize()** i **StartChat()**.

-Funkcija **Initialize()** izgleda ovako:

```
9 //funkcija u koju je upakovano inicijalizovanje openai objekta preko kog se vrsti komunikacija
10 //sa LLM modelom
11 function Initialize()
12 {
13     //konstruisanje objekta preko koga mozemo da pozivamo funkcije modela
14     //generisani API_KEY prosledjujemo u konstruktoru
15     openai = new OpenAI({
16         apiKey: process.env.API_KEY,
17     });
18
19     //definise kako ce da izgleda user interface, imace ulaz i izlaz
20     userInterface = readline.createInterface({
21         input: process.stdin,
22         output: process.stdout,
23     });
24 }
```

Slika 1.2 Prikaz funkcije Initialize

u ovoj funkciji inicijalizujemo objekat klase **OpenAI** čijem konstruktoru prosleđujemo API ključ.

Takođe inicijalizujemo objekat **userInterface** koji će da se koristi prilikom komunikacije u konzoli.

-Funkcija **StartChat()** izgleda ovako:

```
24 //funkcija u koju je upakovana funkcionalnost interakcija korisnika i modela putem konzole
25 function StartChat()
26 {
27     //kreiramo novi prompt koji se prikazuje korisniku
28     userInterface.prompt();
29
30     //obrada dogadjaja kada se upise nova poruka u user interface
31     userInterface.on("line", async (input) => {
32         //ovde se poziva funkcija za razmenu poruka openai.chat.completions.create
33         //navodi se model koji se koristi i format poruke
34         //poruka sadrzi ko je poslao i njen sadrzaj
35         const res = await openai.chat.completions.create({
36             model: "gpt-3.5-turbo-1106",
37             messages: [{role:"user", content: input }],
38         });
39
40         //u koznoli se prikazuje samo onaj podatak povratne vrednosti koji predstavlja
41         //tekstualni sadrzaj odgovora modela
42         console.log(res.data.choices[0].message.content);
43
44         //zatim se prompt ponovo poziva kako bi mogla da se unese sledeca poruka
45         userInterface.prompt();
46     });
47 }
```

Slika 1.3 Prikaz funkcije StartChat

U ovoj funkciji pokrecemo prompt u konzoli preko koga će moći da se vrši komunikacija I definisemo događaj `userInterface.on("line")` koji se izvršava kada se desi upis u konzolu. U njemu koristimo asinhronu funkciju koja sa `await` čeka rezultat **CHAT GPT API** funkcije **`openai.chat.completions.create()`** koja navodi model API-ja koji koristimo I format poruke koja se šalje.

Zatim nakon izvršenja te funkcije pristupamo rezultatu I to samo delu rezultata koji se odnosi na tekstualni sadržaj poruke koju nam je kao odgovor uputio model.

Na kraju ponovo pozivamo `prompt()` kako bi konverzacija mogla neograniceno da se izvršava.

POKRETANJE I IZVRŠENJE

Pre samog pokretanja aplikacije dodajemo pozive navedenih funkcija:

A dark-themed code editor snippet showing three lines of code. Line 48 is empty. Line 49 contains the text `Initialize()` in a light green font. Line 50 contains the text `StartChat()` in a light green font.

Slika 1.4 Poziv funkcija

kako bi ona mogla da bude aktivna.

Aplikaciju pokrećemo komandom **`npm run dev`** I dobijamo:

A dark-themed terminal window showing three lines of text. The first line is `> application@1.0.0 dev`. The second line is `> node script.js`. The third line is `> Terminal u kome interagujemo sa modelom`.

Slika 2.1 Terminal za komunikaciju

U ovom trenutku ako imamo plaćenu uslugu na sajtu `openai`, možemo da interagujemo sa modelom, a ako je slučaj da nemamo, dobijamo error poruku koja glasi:

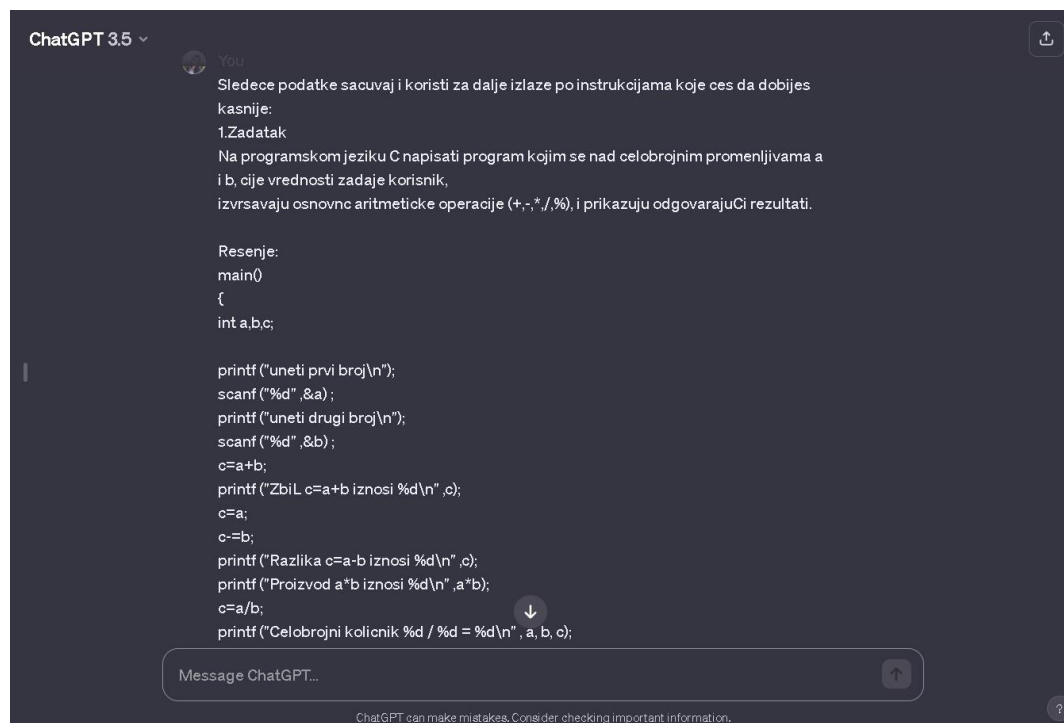
```
error: {  
  message: 'You exceeded your current quota, please check your plan and billing details.',  
  type: 'insufficient_quota',  
  param: null,  
  code: 'insufficient_quota'  
},
```

Slika 2.2 Prikaz error poruke za slučaj neplaćene usluge

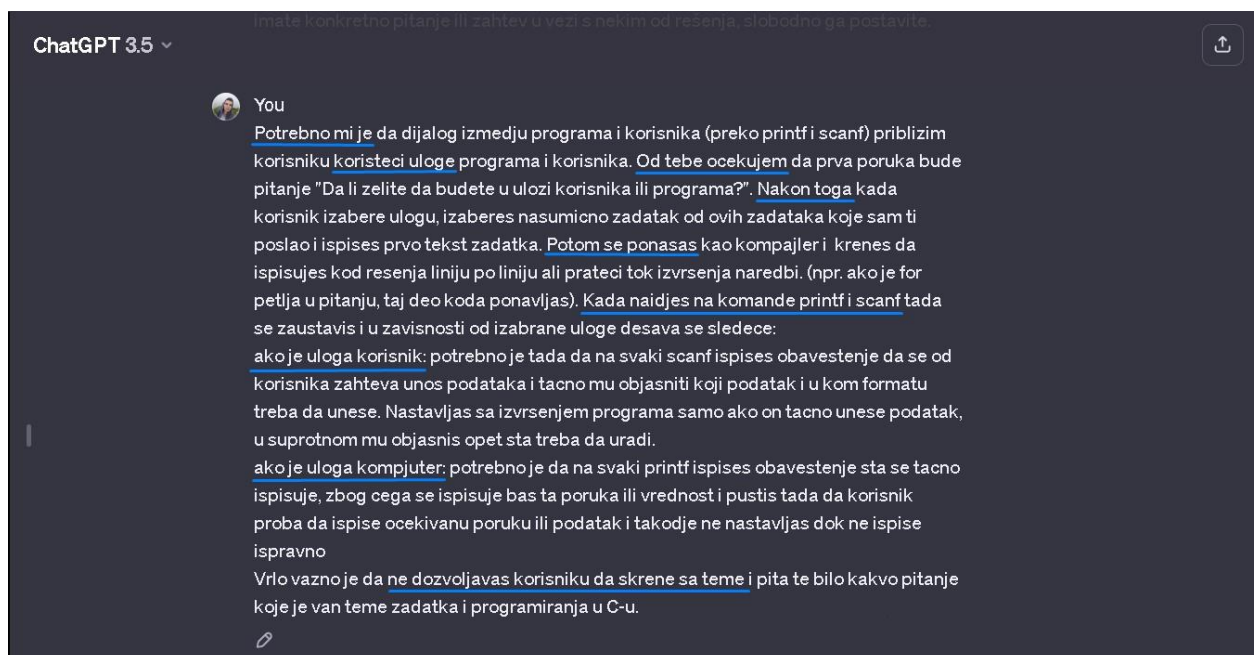
REZULTATI

Kako nismo u trenutku rada imali mogućnost da koristimo plaćenu uslugu API-ja da prikazemo rezultate, koristili smo besplatnu verziju koju nudi CHAT GPT a to je Chat GPT 3.5 koji se ponaša slično kao ASSISTANT API.

Prva stvar koju je bilo potrebno uraditi je proslediti modelu našu zbirku AiP zadataka I rešenja.

Slika 3.1 Prikaz slanja zbirke zadataka modelu
(zbog količine podataka na slici se vidi samo prvi zadatak)

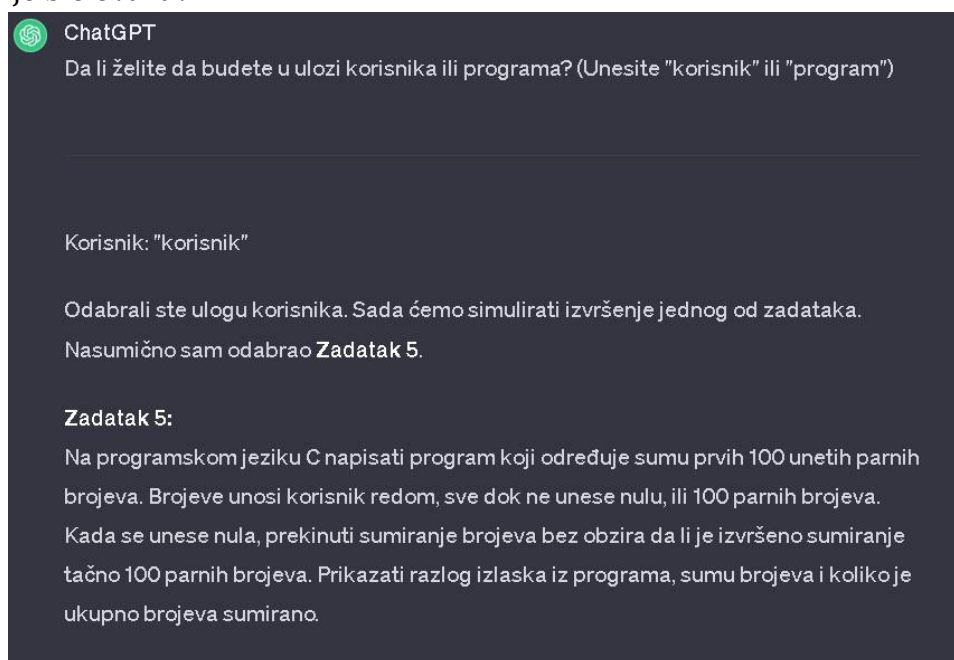
Uzevši u obzir savete za pisanje promptova iz izvora [1], sastavili smo prvobitni prompt koji je izgledao ovako:



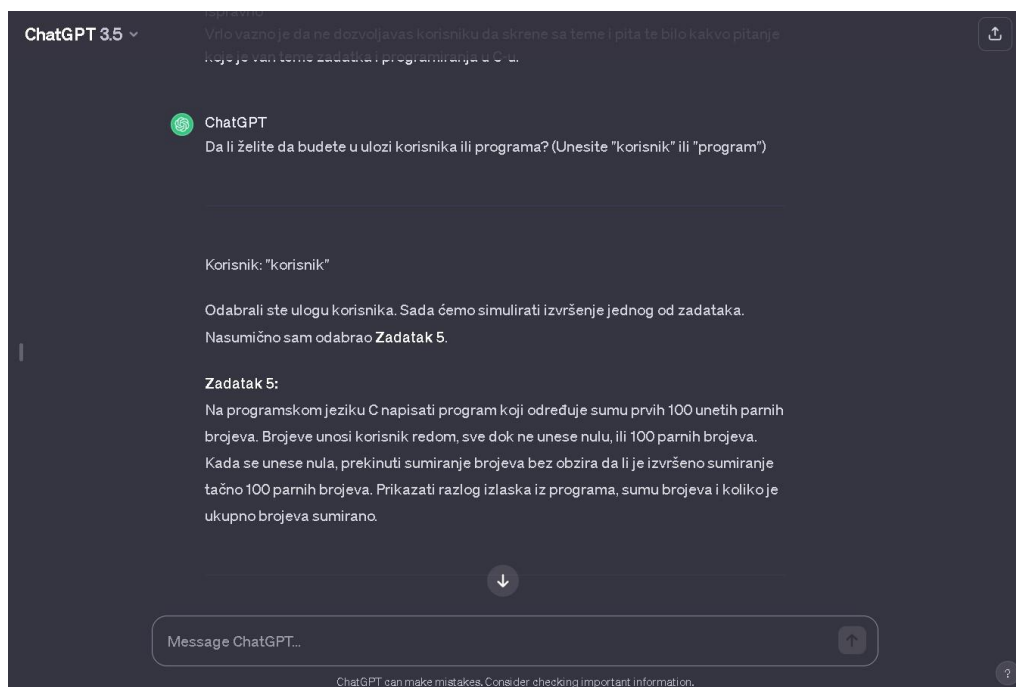
Slika 4.1 Prikaz prvog prompta

Podvučeno plavom bojom: su neke od ključnih stvari prilikom zadavanja prompta modelu. Približavanje problema, navođenje šta se očekuje kao rezultat, koje uloge treba da ima, kako se ponaša u određenom scenariju...

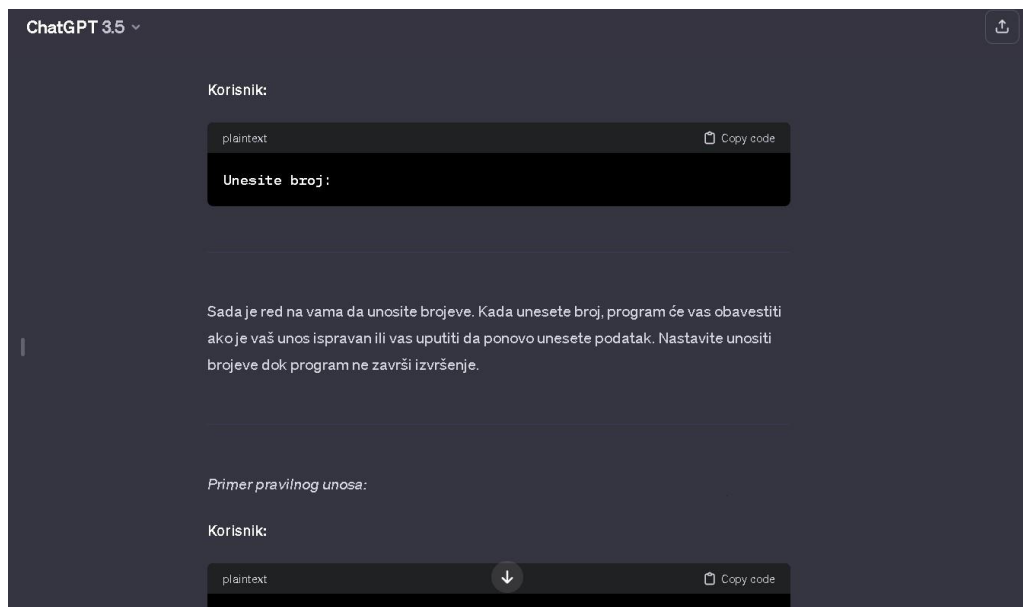
Naravno, model nije najbolje razumeo kako je potrebno da se ponaša iz prve i njegov odgovor je bio ovakav:



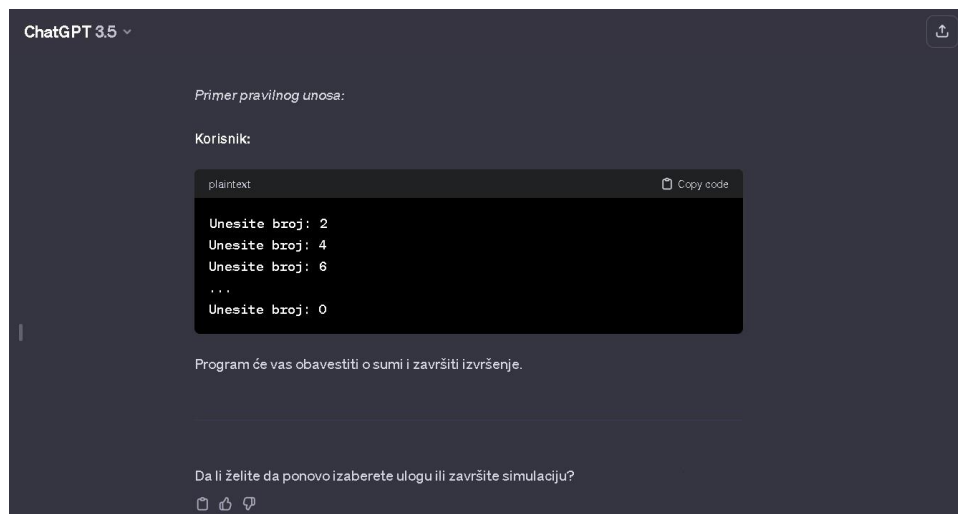
Slika 4.2.1 Prikaz odgovora na prvi prompt



Slika 4.2.2 Prikaz odgovora na prvi prompt



Slika 4.2.3 Prikaz odgovora na prvi prompt



Slika 4.2.4 Prikaz odgovora na prvi prompt

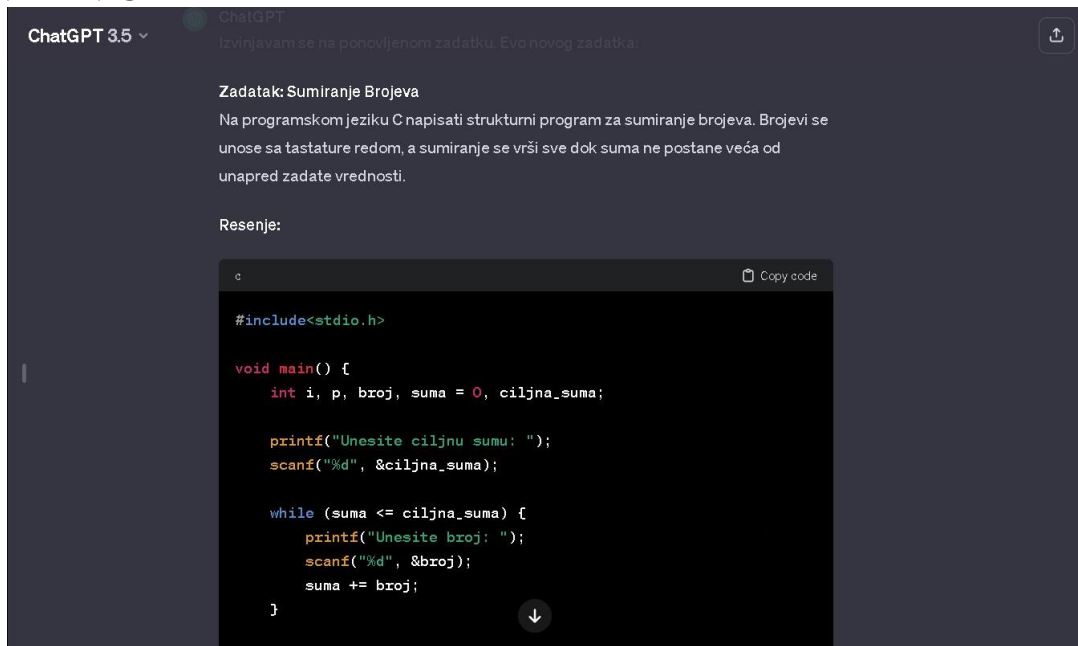
Sam proces treniranja je dug jer se najviše koristi metoda “iteracija I dorada”. Kod kompleksnijih problema kao što je ovaj zahteva se da se kroz test primere uvide greške I potom doradom kroz prompt da se model navede da ih ispravi. Vidimo da Nakon jednog prompta, model ima generalnu ideju o tome šta želimo ali je potrebno treniranje kroz iteracije I simulaciju problema da bi se došlo do funkcionalnog rešenja kakvo mi očekujemo. Stoga nećemo kroz slike prikazivati rezultate prompt inženjeringa već ćemo podeliti sam chat u kome smo trenirali model.

Chat se nalazi na javno dostupnom linku:

<https://chat.openai.com/share/ca5a31ed-4036-4142-b427-8176a4ae96d4>

Prikazaćemo samo jedan primer gde smo uspeli da navedemo model da odradi ono što smo ciljali I to kada je korisnik u ulozi korisnika I potrebno je da mu model objasni šta se tačno

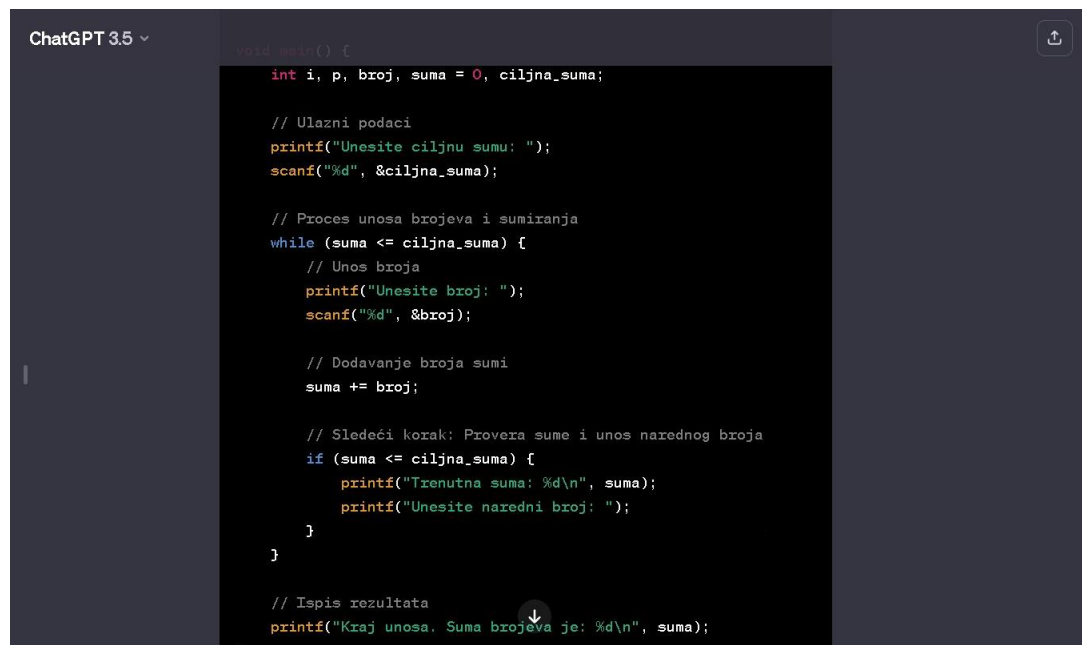
očekuje od njega na unosu.



Slika 5.1.1 Prikaz rezultata koji je približan cilju

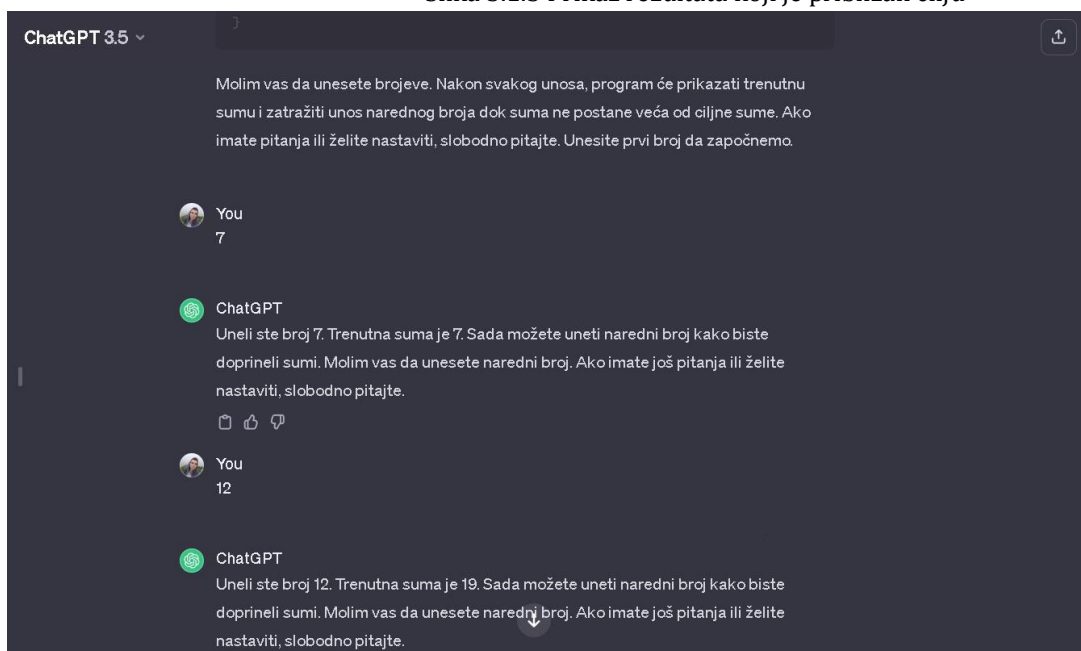


Slika 5.1.2 Prikaz rezultata koji je približan cilju

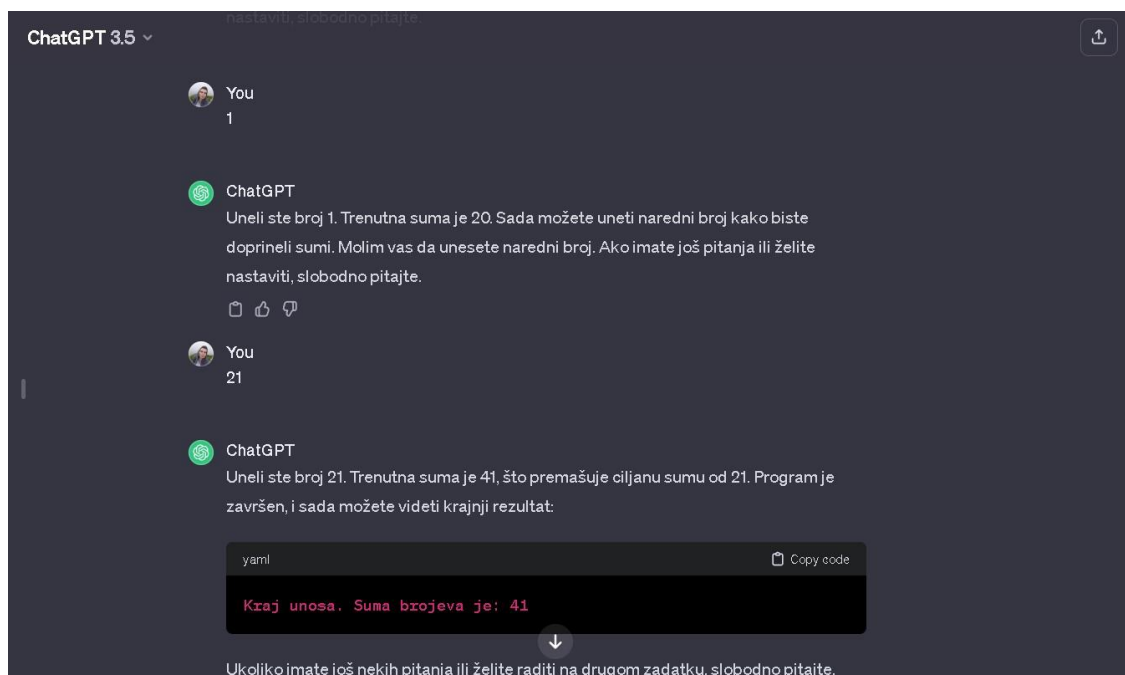


```
void main() {  
    int i, p, broj, suma = 0, ciljna_suma;  
  
    // Ulazni podaci  
    printf("Unesite ciljnu sumu: ");  
    scanf("%d", &ciljna_suma);  
  
    // Proces unosa brojeva i sumiranja  
    while (suma <= ciljna_suma) {  
        // Unos broja  
        printf("Unesite broj: ");  
        scanf("%d", &broj);  
  
        // Dodavanje broja sumi  
        suma += broj;  
  
        // Sledeći korak: Provera sume i unos narednog broja  
        if (suma <= ciljna_suma) {  
            printf("Trenutna suma: %d\n", suma);  
            printf("Unesite naredni broj: ");  
        }  
    }  
  
    // Ispis rezultata  
    printf("Kraj unosa. Suma brojeva je: %d\n", suma);  
}
```

Slika 5.1.3 Prikaz rezultata koji je približan cilju



Slika 5.1.4 Prikaz rezultata koji je približan cilju



Slika 5.1.5 Prikaz rezultata koji je približan cilju

Ovde uočavamo da je model uspeo da uradi tačno ono što smo zamislili I asistirao korisniku u dijalogu sa programom.

ZAKLJUČAK

U prvoj fazi izrade rada objašnjen je teorijski koncept, I pokazana test aplikacija preko koje se može komunicirati sa CHAT GPT API I početni rezultati ispitivanja adekvatnih promptova a u drugoj fazi će biti implementian potpun kod klase koja će moći da se koristi iz aplikacije koju razvija tim V-1. Tada će sadržati I kreiranje asistenta I opciju za slanje fajlova asistentu koje on može da koristi.

Uočeni problemi I nedostaci:

- Chat GPT model u dugom chat-u gubi "sećanje" na unose sa početka chat-a
- Za kompleksne probleme potrebno je mnogo iteracija I dorade prompta
- Ako postoji više uloga, teže je naučiti model da se ponaša na različite načine za iste ulaze
- Problem pri pokretanju API-ja zbog toga što se servis naplaćuje

REFERENCE

- [1] Prompt Engineering as an Important Emerging Skill for Medical Professionals-Tutorial : <https://www.jmir.org/2023/1/e50638>
- [2] OpenAI zvanični web sajt:
<https://platform.openai.com/docs/assistants/how-it-works>