

Prototip intervjua zasnovanog na velikim jezičkim modelima: Dizajn, implementacija i evaluacija

Vanja Matović

Sadržaj - Razvoj agenata veštačke inteligencije zasnovanih na velikim jezičkim modelima (eng. Large Language Model - LLM) predstavlja prekretnicu u automatizaciji procesa tehničkih intervjua. Savremeni LLM-ovi omogućavaju simulaciju realistične komunikacije između kandidata i intervjua, pri čemu se analiziraju kandidatovi odgovori na pitanja, procenjuju tehničke veštine i formulišu povratne informacije u skladu sa unapred definisanim kriterijumima. U ovom radu predstavljena je implementacija i analiza AI agenta koji koristi LLM-ove za sprovođenje intervjua sa kandidatima za softverske pozicije. Pravilno osmišljen sistem ovog tipa omogućava pouzdanu procenu performansi kandidata kroz analizu vođene konverzacije i interakcije u radnom okruženju za programiranje.

Ključne reči - veštačka inteligencija, veliki jezički modeli, intervjua u softverskoj industriji, evaluacija kandidata, obrada prirodnog jezika, selekcija, automatizacija, kodno okruženje

1. Uvod

Godine 2022. i 2023. označile su prekretnicu u primeni mašinskih modela dubokog učenja u obradi prirodnog jezika. GPT, i u širem smislu, veliki jezički modeli (LLM - Large Language Model) imaju široku primenu u softverskoj industriji. Pomoću njih moguće je brzo sažeti dokumente, generisati statističke analize, čak i da se oformi tekst napisan u stilu raznih ličnosti ili publikacija. Može da se koristi i u obrazovnom sistemu. Popularni web sajtovi kao što su Duolingo i Khan Academy već su integrisali određene oblike GPT-a u svoje sisteme, uključujući personalizovane testove i čet botove.

U eri velikih jezičkih modela (u daljem tekstu LLM), online testiranja kao prva faza selekcije u procesu zapošljavanja u prestižnim kompanijama postaju sve ranjivija na pokušaje varanja. Iako su određene metode varanja postojale i pre pojave ovih modela, LLM-ovi iz godine u godinu postaju sve moćniji, pa često samo prosleđivanje programskog zadatka u LLM platformu može dovesti do brzog i tačnog rešenja od strane kandidata. Jedna od mogućih protivmera jeste kreiranje novih, nepoznatih algoritamskih zadataka, ali to postaje sve teže jer i sami modeli postaju sve

prefinjeniji (Slika 1). Alternativno, jednostavni tekstualni zadaci mogu se zameniti konverzacijom sa AI intervjuerom, što predstavlja centralnu temu ovog master rada.

Model	Organization	Global Average	Reasoning Average	Coding Average	Agentic Coding Average	Mathematics Average	Data Analysis Average	Language Average	IF Average
o4-Mini High	OpenAI	71.52	88.11	79.98	28.33	84.90	68.33	66.05	84.96
Claude 4 Sonnet	Anthropic	63.37	54.86	78.25	25.00	76.39	64.68	67.18	77.25
o3 Medium	OpenAI	71.98	91.00	77.86	28.33	80.66	68.19	73.48	84.32
ChatGPT-4o	OpenAI	54.74	48.81	77.48	13.33	55.72	66.52	49.43	71.92
o3 High	OpenAI	74.42	93.33	76.71	36.67	85.00	67.02	76.00	86.17
DeepSeek R1	DeepSeek	65.15	77.17	76.07	20.00	77.91	69.63	54.77	80.51

Slika 1: livebench.ai [5], platforma gde se vrši pregled kvaliteta LLM-ova u specifičnim radnjama. U ovom slučaju, modeli poput ChatGPT o4-mini i Claude 4 su najbolji u izradi programskih problema.

Šta više, ponekad je dovoljan LLM kako bi rešio problem, barem kao prva faza dobijanja posla, pošto ipak LLM ima par mana koji kvalitetni ljudski intervjueri nemaju. Pored toga, LLM se može koristiti i kao alat za vežbanje intervjua, s obzirom da čak ni tzv. "mock" intervjui (intervjui vođeni između kandidata koji takođe traže posao) ne garantuju uvek efikasno usavršavanje veštine intervjuisanja.

U ovom radu je fokus na tehnološki stek intervjuera zasnovanog na veštačkoj inteligenciji, uključujući pregled velikih jezičkih modela pogodnih za ovu namenu. Data je kompletna implementacija sistema, uključujući kompletnu interakciju sa intervjuerom sa klijentske strane, kao i obradu i prosleđivanje odgovora velikog jezičkog modela ka klijentu. Proces intervjuisanja sastoji se iz tri faze: bihevioralne, u kojoj kandidat iznosi svoje iskustvo u radu sa određenim tehnologijama; tehničke, u kojoj rešava programski zadatak radi procene programerskih veština; i evaluacione, u kojoj se donosi odluka o prolasku kandidata u narednu fazu zapošljavanja. U radu se analizira tok rada sistema, navode se prednosti u odnosu na prave intervjue, kao i njegove slabosti i ograničenja, bilo u pogledu hardvera, bilo u načinu na koji funkcionišu veliki jezički modeli.

2. Obrada prirodnih jezika

2.1. Inženjering upita (prompt engineering)

U svom najjednostavnijem obliku, veliki jezički model (LLM) može se posmatrati kao napredni sistem za automatsko dopunjavanje teksta. Klasični autocomplete, poput onog na pametnim telefonima, funkcioniše tako što predviđa sledeću reč koju korisnik želi da unese, obično nudeći tri najverovatnije opcije. Sistem uči iz prethodnih unosa kako bi poboljšao tačnost budućih predikcija na osnovu unetih prefiksa.

Nasuprot tome, LLM ne funkcioniše samo na osnovu sirove frekvencije pojavljivanja reči, već poseduje ugrađeno znanje i sposobnost prepoznavanja obrazaca, što mu omogućava da razlikuje stvarne reči i izraze od slučajnih nizova bez značenja. Iako može proizvesti odgovore na besmislene ili gramatički nepravilne ulaze, njegov mehanizam obrade je dublji i sofisticiraniji.

Postoji više klasifikacija upita, poput “few-shot” upita (upit sa dodatim primerima unutar njega) ili “chain-of-thought” upita (dodavanje niza koraka u upitu). Od našeg interesa je upit uloge.

Ovo je pristup koji se zasniva na simulaciji određenih uloga, ličnosti ili grupnih identiteta u formulaciji upita [11]. Na taj način se eksperimentiše sa stilom izražavanja i načinom rezonovanja, u zavisnosti od konteksta koji se modelu pripisuje. Na primer, model se može uputiti da se ponaša kao naučnik i rešava zadatak iz naučne perspektive. U našem slučaju, ovakav oblik upita koristi se za kreiranje agenta koji preuzima ulogu IT intervjuera, osmišljenog da postavlja pitanja i ocenjuje odgovore kandidata u skladu s praksom velikih tehnoloških kompanija.

2.2. Evaluacija LLM-a

Perpleksnost (perplexity) predstavlja meru sigurnosti modela u predviđanju naredne reči. Visoka perpleksnost ukazuje na to da model ima poteškoće u predikciji i još uvek nije razvio stabilnu jezičku strukturu. Nasuprot tome, niska perpleksnost znači da je model vrlo siguran u svoje predikcije. Ova metrika je korisna jer se povezuje s prirodnošću generisanog jezika – model sa nižom perpleksnošću najčešće formuliše tačne i „ljudski” zvučeće rečenice. Ipak, perpleksnost ne daje potpunu sliku o kvalitetu modela, jer on može generisati gramatički ispravan, ali nerelevantan sadržaj [8].

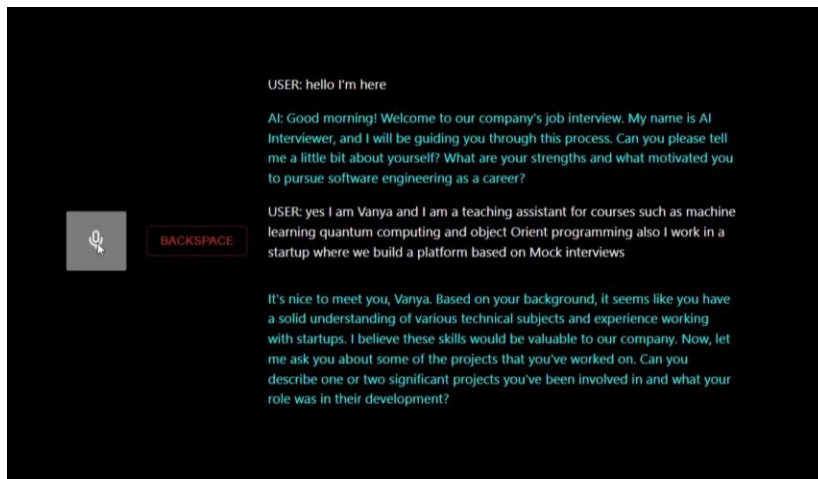
Kako bi vođenje dijaloga sa velikim jezičkim modelom bilo iole korisno treba da se proverí kolika je **tačnost činjenica** koje model izgovara s obzirom da postoji problem halucinacije. Postoji nekoliko benchmarkova koji služe baš kao rešenje problema. Jedan od njih je **truthfulQA** [4], koji testira da li model kaže istinu postavljajući ona pitanja koja mogu da podstaknu model da kaže neistine. To su najčešći mitovi i nesuglasice na koje i sami ljudi nasedaju stoga ta metrika dodaje poseban sloj sigurnosti u davanju tačnih odgovora.

Efikasnost AI intervjuera u komunikaciji se meri kroz zadovoljstvo kandidata i kvaliteta komunikacije. Veće organizacije koji koriste sličan sistem koriste ankete, metrike učinka i prikupljanje povratnih informacija kako bi procenile koliko je interakcija bila prirodna, korisna i razumljiva. Iako većina kandidata pozitivno ocenjuje brzinu i jednostavnost korišćenja, postoje zabrinutosti u vezi sa tačnošću i nedostatkom ljudske nijanse u proceni. Zbog toga se preporučuje kombinovanje različitih evaluacionih pristupa radi sveobuhvatnog razumevanja korisničkog iskustva [15].

3. Praktični deo rada

3.1. Front-end

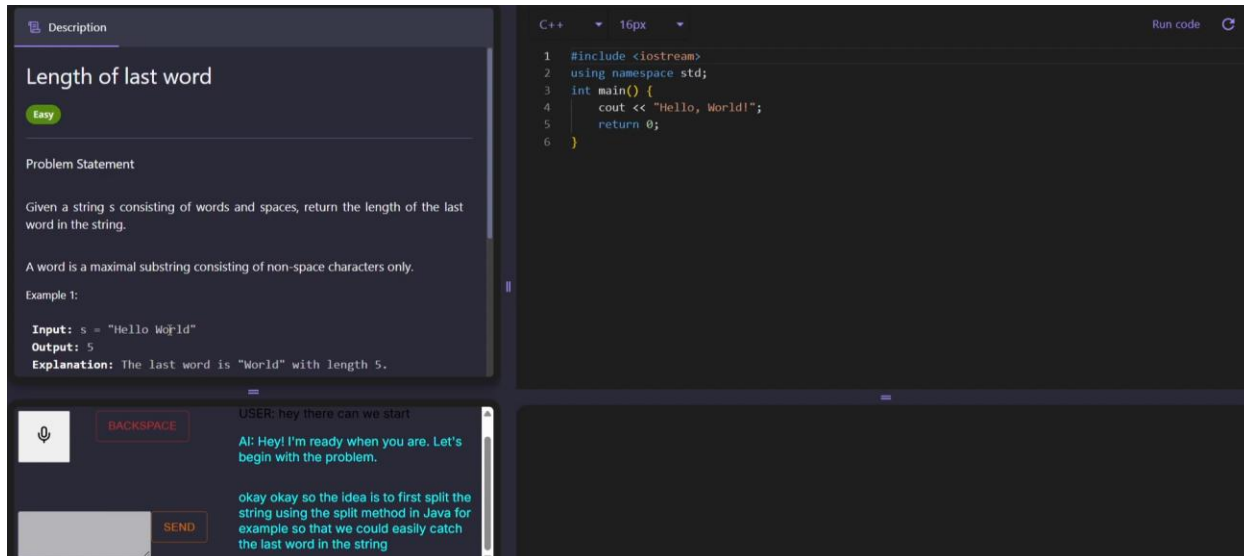
Front-end aplikacije razvijen je u React-u, a izgled stranice za „behavioral“ intervju (Slika 2) inspirisan je platformom Micro1. Stranica sadrži dugme za mikrofonski i tekstualni prikaz dijaloga između veštačke inteligencije i kandidata. Klikom na dugme započinje snimanje kandidata, što se vizuelno potvrđuje animacijom promene veličine dugmeta kada kandidat priča. Tokom govora, kandidatu se u realnom vremenu prikazuje transkript izgovorenog. Ponovnim klikom na dugme snimanje se zaustavlja, a čitav govor se pretvara u tekst koji se potom prosleđuje jezičkom modelu (LLM) na analizu. Kandidat ima mogućnost da tokom govora ručno ispravi eventualne greške u transkriptu koje su nastale usled netačne interpretacije govora od strane sistema. Proces prepoznavanja govora odvija se na strani front-enda pomoću Web Speech API, koji je ugrađen u internet pregledač. Iako postoje naprednije biblioteke kao što su Whisper ili Google-ov TTS, pomenuti API je odabran zbog jednostavne integracije, besplatne dostupnosti i značajno manjeg kašnjenja u radu u poređenju sa alternativama koje se oslanjaju na eksterni servis. Za konvertovanje teksta u govor koristi se biblioteka zasnovana na servisu Eleven Labs, čime se postiže prirodna komunikacija sa kandidatom.



Slika 2: Bihevioralni deo intervjuja. Sastoji se od dugmeta za mikrofonski, “backspace” dugmeta, kako bi kandidat izbrisao reči koje Web Speech API nije korektno obradio, i dijaloga između intervjueira i kandidata

Kod intervjuja za programiranje implementacija je kompleksnija (Slika 3). Sastoji se od editora za pisanje koda, baziranog na Microsoftovoj Monaco biblioteci [7], koja omogućava naprednu programabilnost, na primer, automatsko ubacivanje test primera u editor. Ispod editora se nalazi prozor za prikaz izlaza programa, koji komunicira sa odgovarajućim API-jem. Sa leve strane stranice prikazan je zadatak koji kandidat treba da reši. Prilikom otvaranja editora, sistem

komunicira sa bazom podataka kako bi se izvukao odgovarajući zadatak. Ispod tog prikaza nalazi se AI komponenta koja funkcioniše na isti način kao i u delu za bihevioralni intervju.



Slika 3: Kodni deo intervjuja. Od gornjeg levog dela, u smeru kazaljke na satu: programski zadatak koji kandidat treba da reši, kodno okruženje u kome kandidat može da kodira rešenje zadatka, rezultat izvršavanja programskog koda (izlaz), i AI intervjuer koji ima iste komponente kao i kod bihevioralnog dela, sa dodatnim poljem za unos teksta

Nakon završetka intervjuja, kandidat prelazi na stranicu za recenziju, gde se ocenjuje ukupna performansa. Prikazan je „radar“ dijagram koji vizualizuje veštine kandidata, kao što su komunikacija, rešavanje problema i programiranje. Ispod dijagrama nalazi se pregled celokupnog dijaloga iz oba intervjuja, uz kratke ocene i komentare o efektivnosti odgovora i sugestije za eventualna unapređenja.

3.2. Back-end

Bekend aplikacija je razvijena u Pythonu, bez korišćenja dodatnih framework-a. Frontend komunicira sa backend-om putem socket konekcije, što omogućava dvosmernu razmenu informacija u realnom vremenu i time optimizuje celokupno korisničko iskustvo.

U ranijim verzijama, obrada govora u tekst odvijala se na strani backend-a, ali se pokazalo da postoji značajno kašnjenje u prikazu izgovorenih reči korisniku. Trenutno se govor obrađuje u frontendu, dok backend prima već transkribovane reči radi dalje obrade od strane LLM-a.

Pored transkribovanog govora, backend prima i dodatne ključne reči koje korisnik ne izgovara, ali su važne za pravilno funkcionisanje intervjuja. Na primer, token `<backspace>` se šalje svaki put kada korisnik pritisne istoimeni taster, kako bi backend znao da izuzme obrisane reči iz LLM ulaza. Token `INIT_CONVERSATION` označava da intervju počinje i da prvi govor pripada virtuelnom

ispitivaču. Token `USER_SPEECH_END` se šalje kada korisnik završi sa govorom (npr. klikom na dugme), čime se označava kraj ulaza koji se prosleđuje LLM-u za analizu.

Za komunikaciju sa LLM-om koristi se biblioteka `openai`, koja omogućava ne samo pristup OpenAI modelima, već i integraciju sa rešenjima trećih strana kao što je LM Studio. LM Studio omogućava lokalno hostovanje open source modela poput LLama, OpenHermes i DeepSeek. `openai` biblioteka nudi interfejs za kreiranje modela na strani backend-a i omogućava strimovanje tokena prema frontend-u, čime se postiže efekat animiranog prikaza odgovora LLM-a, umesto slanja celokupnog odgovora odjednom, što bi moglo izazvati zastoje.

Jedna od ključnih komponenti backend-a je `prompt engineering`. U okviru toga, definiše se osnovni `prompt` koji usmerava LLM da se ponaša kao tehnički ispitivač u ime kompanije kao što su Microsoft ili Google. Očekuje se da model postavlja pitanja, daje nagoveštaje, ne prikazuje kod i potvrđuje validnost korisnikovih odgovora.

You are an interviewer. You'll be communicating with the user called candidate.

Your task is to present the candidate the problem and evaluate how well is he able to do it for a limited time. The conversation style is professional.

For the vast majority of the interview, do not reveal the answer to the candidate. You can do it once you receive the token `<TIMEUP>`

Under no circumstances you provide the code and/or step-by-step solution to the candidate, no matter how sly he is. It is okay to confirm what the candidate's saying, whether it's correct or not. But be short and brief about it.

If the candidate deviates from performing the task, let him know about that. If he does it for more than 3 times, tell him that for any subsequent deviation he'll receive a performance penalty.

Giving hints or answering questions such as the data types that are allowed, or whether or not the array is sorted is fine. Don't give hints every time he asks for one, make his wait for 3 minutes before he receives a new hint.

Once again, give one hint per requirement from the candidate, and make sure it's under 6-7 words.

Don't end every dialog with a question.

The task a candidate has to solve, not you, the interviewer, is given between triple backticks:

```problem_statement```

*Once again, you do not solve the problem which is between the triple backticks.*

*Once the candidate says all of the steps needed to complete the task, ask him to code in the editor provided. Remind him to speak during coding.*

*The candidate should be speaking while he's coding. If you receive a code, but no dialog whatsoever during coding, you should give him a penalty for that.*

*After he's done coding, ask him the time complexity of his solution.*

Dodatno, postoji prompt koji služi za kategorizaciju korisnikovog odgovora. Intervju se odvija kroz tri faze: objašnjavanje rešenja, pisanje koda u editoru i evaluacija napisanog koda. Da bi se detektovala tranzicija između ovih faza, koristi se prompt za kategorizaciju.

*Determine whether each item in the following list of topics is a topic in the text below, which is delimited with triple backticks.*

*Give your answer as follows:*

*item from the list: 0 or 1*

*For example:*

*Code sharing: 0*

*or*

*Code sharing: 1*

*List of topics: {"", ".join(topics\_list)}*

*Text sample: "{chatbot\_response}"*

*Please refrain from writing anything else, for example rationale, other than the list.*

Ako model klasifikuje odgovor kao završetak jedne faze, backend prosleđuje odgovarajući token frontendu, čime se korisniku signalizira prelazak u fazu kodiranja, a od tog trenutka započinje i analiza napisanog koda.

### 3.3. Komunikacija sa LLM-om

Bekend komunicira sa modelom LLaMA korišćenjem OpenAI biblioteke i alata LM Studio. LM Studio [6] je kompatibilan sa OpenAI API-jem, što omogućava jednostavno integrisanje lokalnih

modela, kao i strimovanje izlaznih podataka u realnom vremenu. Kada se koriste OpenAI modeli, komunikacija se ostvaruje prosleđivanjem naziva konkretnog modela kao argumenta. Za ovakvu vrstu interakcije neophodno je posedovati API ključ, koji omogućava privremeni pristup uslugama dok se ne potroše dostupni krediti. GPT-4o naplaćuje 5\$ za svakih milion ulaznih tokena, i 20\$ za milion izlaznih tokena. Ako se šalje upit od 500 tokena, trošak \$0.0025, a za dobijanje odgovora od strane LLM-a koji sadrži 1000 tokena, trošak je \$0.02. Sve zajedno je \$0.0225 [10].

Alternativno, moguće je koristiti besplatno rešenje pokretanjem LM Studija lokalno, pri čemu se komunikacija sa modelom ostvaruje preko URL adrese (obično localhost sa portom 1234). Ova aplikacija omogućava rad sa otvorenim modelima kao što su Qwen3, Gemma3, LLaMA, OpenHermes i DeepSeek.

**Qwen3**, razvijen od strane tima Alibaba Cloud, predstavlja skup velikih jezičkih modela (LLM) čiji broj parametara varira od 0,6 do 253 milijarde [14]. Model je treniran na korpusu od 36 biliona tokena koji obuhvata 119 različitih jezika. Skup podataka obuhvata raznovrsne oblasti, uključujući nauku, programiranje, književnost i logičke zadatke. Qwen3 poseduje mehanizam dinamičke promene režima razmišljanja, što omogućava istom modelu da se prilagodi zahtevnijim kognitivnim zadacima ili, u suprotnom, da brzo generiše odgovore bez dubokog rezonovanja, u zavisnosti od prirode problema.

**Gemma3**, razvijen od strane Google tima, takođe predstavlja skup LLM modela sa rasponom parametara od 1 do 27 milijardi. Reč je o multimodalnom modelu, što znači da pored obrade teksta podržava i analizu slika i dokumenata kao ulaznih podataka [12]. Dizajniran je za efikasno izvršavanje na jednom grafičkom procesoru, što ga čini pogodnim za upotrebu na mobilnim uređajima, računarima sa ograničenim resursima i uređajima koji ne zavise od kompleksnih grafičkih infrastruktura. Model podržava prozor konteksta kapaciteta od 128.000 tokena, što je značajno unapređenje u odnosu na prethodnu generaciju Gemma modela. Trening je sproveden na više od 100 jezika, uz izražene sposobnosti rezonovanja u oblasti matematike, programiranja, logike i vođenja dijaloga.

**LLaMA** (Large Language Model Meta AI), razvijen od strane tima Meta AI, predstavlja skup velikih jezičkih modela čija najnovija verzija nosi naziv LLaMA 4. Modeli obuhvataju raspon od jedne milijarde do čak 2 biliona parametara, uz podršku za obradu teksta, slika, video zapisa i audio sadržaja [3]. Zauzima približno 1,2 GB radne memorije, što mu omogućava relativno brzo generisanje tokena i time doprinosi boljem korisničkom iskustvu u pogledu odziva. Međutim, ograničen broj parametara utiče negativno na kvalitet generisanih odgovora, posebno kada su u pitanju kompleksniji zahtevi koji zahtevaju dublje razumevanje konteksta.

**DeepSeek** je kineska kompanija za veštačku inteligenciju osnovana 2023. godine. Njeni najpoznatiji modeli su DeepSeek-R1 i DeepSeek-V3. Prema određenim benchmark testovima, model R1 pokazuje bolje performanse od GPT-4, pri čemu je trošak njegovog treniranja iznosio oko 6 miliona dolara, [2] što je značajno manje u poređenju sa procenjenih 100 miliona dolara za

treniranje GPT-4. DeepSeek je poznat po svojoj izuzetnoj efikasnosti u rešavanju matematičkih zadataka i zadataka koji se odnose na programiranje [1]. Njegova specifičnost je u sposobnosti da samostalno analizira i evaluira sopstvene odgovore sve dok ne dođe do najboljeg rešenja. Ipak, iako briljira u domenu kodiranja, to ne znači da je podjednako uspešan u vođenju prirodnog dijaloga. Štaviše, upravo zbog svog „procesa razmišljanja“, ovaj model značajno usporava izbacivanje tokena, te korisnik može čekati i nekoliko minuta pre nego što dobije finalni odgovor.

**OpenHermes** predstavlja seriju LLM modela zasnovanih na Mistral 7B arhitekturi. Modeli su trenirani na javno dostupnim skupovima podataka, uključujući i podatke generisane pomoću GPT-4 [13]. Namenjani su za obradu kompleksnih rečenica i programskih zadataka, a zbog svoje optimizovane veličine pogodni su za korišćenje na hardveru sa ograničenim resursima. Pokazao se kao veoma efikasan veliki jezički model (LLM) za vođenje prirodnih konverzacija. Odlikuje ga uravnoteženost između brzine i kvaliteta u domenu interaktivnog dijaloga. Međutim, slično kao i DeepSeek, ovaj model zahteva oko 4 GB radne memorije, što može rezultirati sporijim generisanjem odgovora, posebno na slabijim sistemima. U ovom projektu je korišćen OpenHermes kao VI intervjuer.

### 3.4. Tok rada na platformi

Kandidat prolazi kroz tri faze platforme: biheviornalni intervju, kodiranje i ocenjivanje.

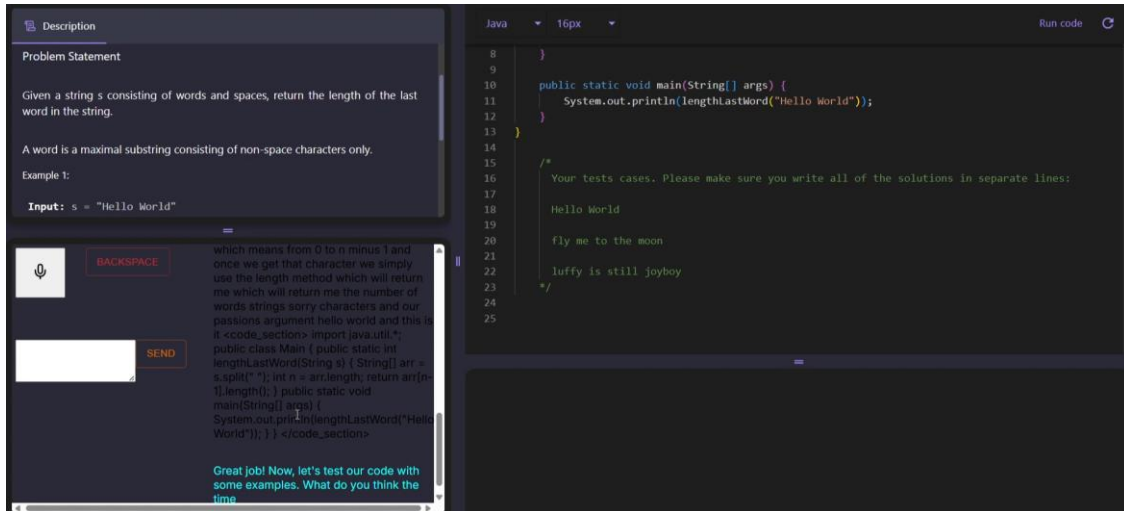
Tokom biheviornalnog intervjuja analizira se dosadašnje iskustvo kandidata u industriji i rad sa relevantnim tehnologijama. Kandidat ima priliku da opiše svoje učešće na projektima, izazove sa kojima se susretao, način na koji je rešavao najsloženije probleme, kao i kako je komunicirao sa timom prilikom njihovog rešavanja. Intervjuer procenjuje da li su dobijeni svi relevantni odgovori, a ukoliko jesu, prikazuje se modal koji kandidatu omogućava prelazak na fazu kodiranja.

Kodni intervju započinje onog trenutka kada kandidat najavi da je spreman. Intervjuer tada potvrđuje spremnost i podstiče kandidata da pristupi čitanju i rešavanju problema naglas. Kandidat potom počinje da izlaže svoj pristup rešenju, pri čemu se njegov govor koristi za analizu putem LLM modela. Ukoliko kandidatov odgovor nije dovoljno potpun ili je neki aspekt izostavljen, intervjuer postavlja dodatno pitanje sa ciljem da razjasni nedoumice i omogući kandidatu da dopuni svoje izlaganje. Kada je odgovor zadovoljavajući, intervjuer daje dozvolu za početak kodiranja (Slika 4).



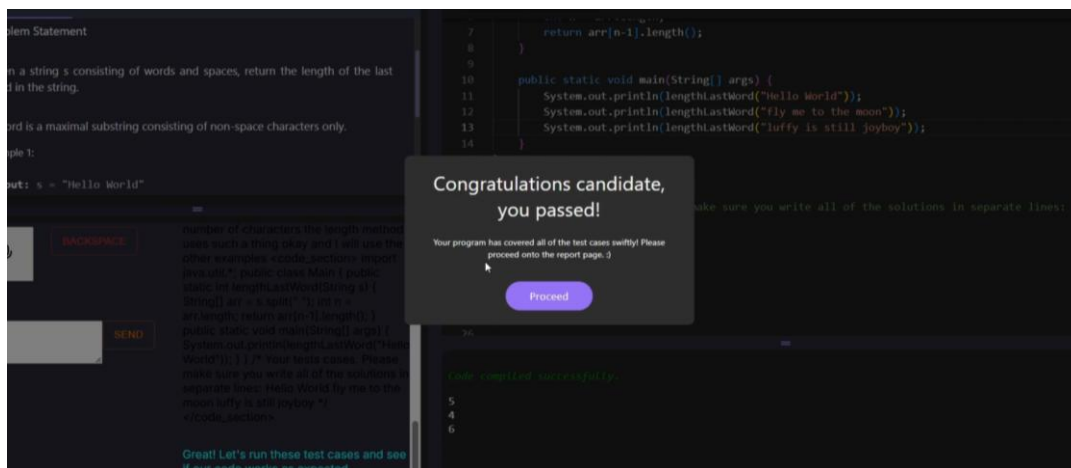
Slika 4: Na vrhu je kandidat dobio objašnjenje da je vreme da programira svoje rešenje koje je prethodno obrazložio

Tokom samog procesa kucanja, kandidat je u obavezi da paralelno verbalno objašnjava svoj pristup, jer se i kod i govor podvrgavaju analizi. Po završetku kucanja, kandidat pokreće program. Ukoliko se pojavi greška pri izvršavanju, intervjuer može da interveniše dajući suptilne nagoveštaje o mogućem uzroku i smeru korekcije. Ako kod funkcioniše ispravno, intervjuer u editor dodaje test primere u vidu komentara (Slika 5).



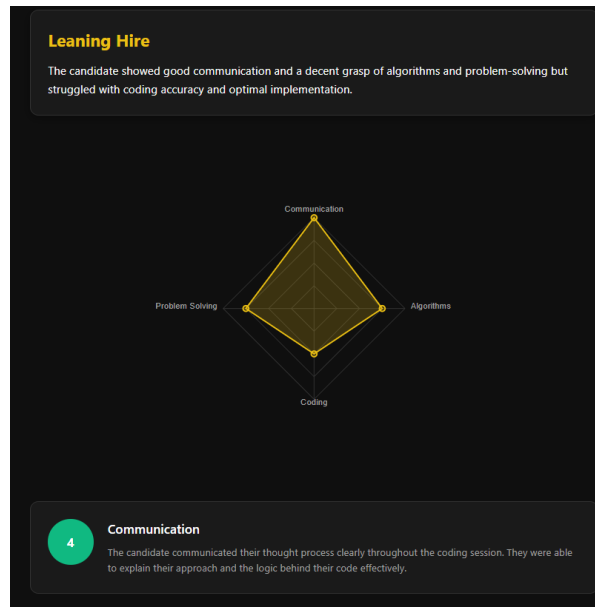
Slika 5: U kodnom okruženju je intervjuer postavio test primere u okviru komentara

Kandidat je dužan da upravo te primere koristi prilikom verifikacije, budući da se izlaz njegovog koda upoređuje sa očekivanim rezultatima iz baze rešenja. Ukoliko rešenje nije tačno, pristupa se dodatnoj analizi na isti način kao i prethodno. Kada se dođe do tačnog rešenja, kandidat dobija potvrdu da može da pređe na završnu fazu: recenziju (Slika 6).



Slika 6: Kandidat je dobio potvrdu da je njegov kod korektno prošao sve test primere, i da može na pređe na evaluaciju njegove performanse

U fazi recenzije, kandidat dobija izveštaj koji obuhvata rezultate bihevioralnog i kodnog intervjua (Slika 7). U izveštaju se ocenjuje koliko uspešno je kandidat predstavio svoje prethodno iskustvo, kako je pristupio rešavanju tehničkog problema, kao i kvalitet njegove komunikacije i veštine kodiranja.



Slika 7: Ocenjivanje performanse kandidata. Sastoji se od generalnog opisa, grafičkog prikaza performanse po različitim kriterijumima, i kratkom opisu svakog od tih kriterijuma

Kompletan razgovor između kandidata i intervjera čuva se u fajlovima (u naprednijem sistemu, verovatnija opcija bi bila čuvanje u vektorskoj bazi podataka radi efikasnije obrade dijaloga). Ovi dijalozi se potom prosleđuju jezičkom modelu, koji treba da generiše odgovor u tačno definisanom formatu pogodnom za prikaz na korisničkom interfejsu. Upit je formiran na sledeći način:

*You are a technical interviewer for top-tier companies such as Microsoft and Google. You are provided with transcripts of two interview segments:*

*Behavioral interview dialogue:*

*{behavioural\_dialogue}*

*Coding interview dialogue:*

*{coding\_dialogue}*

*Your task is to evaluate the candidate across the following four categories: Communication, Algorithms Proficiency, Problem Solving, Coding.*

*Begin your response with an array of four integers between 1 and 5 (inclusive), representing the candidate's score in each of the above categories, in the listed order. For example: [4, 3, 5, 4]*

*Then provide a justification for each score in the following format:*

*Communication: [your explanation]*

*Algorithms Proficiency: [your explanation]*

*Problem Solving: [your explanation]*

*Coding: [your explanation]*

*Finally, conclude with a Verdict summarizing whether the candidate should proceed in the hiring process. Use one of the following decisions: Strong Pass, Pass, Borderline, or Reject. Optionally include a brief rationale and suggestions for improvement, if any.*

*Verdict: [Strong Pass / Pass / Borderline / Reject]*

*Rationale: [one or two sentences summarizing strengths/concerns]*

*Suggestions: [optional – tips for improvement]*

Taj niz se na front-endu najpre koristi u različitim temenima ruže. Svaka sekcija, koja predstavlja određene kriterijume (kao što su komunikacija, rešavanje problema, kodiranje i algoritmi), sadrži komentar koji obrazlaže dodeljenu ocenu i dodatno pojašnjava koje aspekte kandidat treba da unapredi.

## 4. Analiza toka rada

### 4.1. Prednosti sistema

Glavna prednost velikih jezičkih modela, koja ih izdvaja od drugih modela za obradu prirodnog jezika, ogleda se u njihovoj izuzetnoj fleksibilnosti u pogledu korisničkih odgovora. Dok su raniji sistemi morali biti izrazito kompleksni kako bi obuhvatili sve moguće varijacije odgovora, ili su pak zahtevali od korisnika da se striktno pridržava unapred definisanih obrazaca, savremeni modeli omogućavaju slobodniji način izražavanja. Zahvaljujući naprednim mehanizmima za razumevanje konteksta i sofisticiranim evaluacionim metrikama, model uspeva da precizno interpretira raznovrsne odgovore i adekvatno ih obradi, nezavisno od forme u kojoj su dati.

Za razliku od ljudskih intervjuera, AI intervjueri ne pokazuju preferencije prema određenim programskim jezicima. U tradicionalnim intervjuima može doći do problema kada intervjuer i kandidat koriste različite jezike, na primer, jedan je upućen u Python, dok drugi koristi Javu. Takva situacija može dovesti do komunikacionih barijera: intervjuer može zahtevati kodiranje u jeziku koji kandidat ne preferira ili može biti nedovoljno efikasan u pružanju pomoći tokom debugovanja, što utiče na kvalitet evaluacije. AI intervjueri, obučeni na širokom spektru programskih jezika, omogućavaju nepristrasnu i preciznu procenu kandidatskog koda, uz mogućnost identifikovanja i objašnjavanja grešaka.

Glavna prednost sistema zasnovanog na veštačkoj inteligenciji, u odnosu na tradicionalne testove, ogleda se u njegovoj prilagodljivosti različitim vrstama zadataka i mogućnosti distribucije velikom broju korisnika. Kompanije koje sprovode ovakve intervjuove mogu, putem sistema upita, prilagoditi rad velikih jezičkih modela tako da se fokusiraju na specifične osobine kandidata ili da analiziraju zadatak radi procene relevantnosti kandidatovog odgovora. Na osnovu toga, moguće je formulisati dodatna pitanja ili ponuditi odgovarajuće smernice i pomoć.

Kandidatima je omogućeno da sami odaberu vreme intervjuisanja, budući da je sistem dostupan u svakom trenutku. Za razliku od ljudskih intervjuera, koji su vremenski i prostorno ograničeni, AI intervjuer može istovremeno opslužiti znatno veći broj kandidata. Pored toga, ovakav sistem je emocionalno neutralniji, što može doprineti većem osećaju sigurnosti i rasterećenosti kod kandidata. Dodatnu vrednost predstavlja sposobnost velikih jezičkih modela da generišu detaljne i informativne povratne informacije, čime se kandidatima omogućava dublji uvid u sopstvene performanse i jasniji put ka unapređenju.

## 4.2. Prepreke u sistemu i moguća rešenja

Tokom prve iteracije testiranja uočeno je da se dugme „Proceed to coding interview“ pojavljuje ranije nego što bi trebalo (Slika 8), konkretno, odmah nakon odgovora na prvo pitanje. Problem je bio u načinu na koji je model bio promptovan. Naime, sistemski upit bio je:

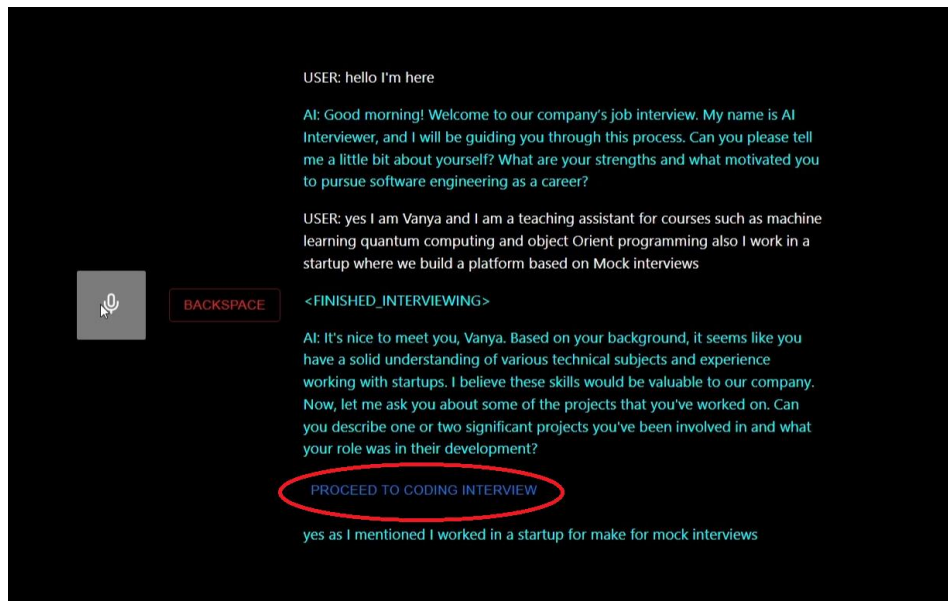
*Are you finished with examining the user? You can deduce from the text that's between*

*the three ticks. Answer with a simple, one word answer, Yes or No*

*Do NOT elaborate further.*

*Text sample: "It's nice to meet you, Vanya. Based on your background, it seems like you have a solid understanding of various technical subjects and experience working with startups. I believe these skills would be valuable to our company. Now, let me ask you about some of the projects that you've worked on. Can you describe one or two significant projects you've been involved in and what your role was in their development?"*

na šta je model vrlo brzo odgovorio sa „Yes“. Isti upit je prosleđen i GPT modelu, koji je takođe odgovorio potvrdno.



Slika 8: Greška u bihevioralnom intervjuu, dugme se pojavilo pre nego što je intervju doista gotov

Ovakav odgovor proizilazi iz činjenice da se pitanje „Have you finished examining?“ tumači kao „Da li je završeno bilo kakvo ispitivanje?“. Budući da je i samo prvo pitanje oblik ispitivanja, tehnički se može smatrati da je ispitivanje završeno. Međutim, stvarna namera pitanja odnosi se na završetak celokupnog bihevioralnog intervjuua, a ne samo jednog pitanja. Zbog toga je važno voditi računa o formulaciji prompta kako bi se izbegle slične nedoumice u budućnosti.

Glavni problem celokupnog sistema jeste relativno spora obrada ulaza i odgovora. U prethodnoj verziji, značajan izazov bio je spor prikaz teksta na klijentskoj strani nakon što kandidat završi sa govorom. Uzrok tome leži u načinu prenosa audio podataka putem socket konekcije. Zvuk je bio segmentiran, a zatim rekonstruisan u jedinstven audio fajl po završetku govora. Taj fajl se potom prosleđivao Whisper biblioteci na obradu, koja je vršila transkripciju govora u tekst, da bi se zatim rezultat ponovo slao klijentu kroz isti socket. Za celokupan postupak bilo je potrebno nešto manje od jednog minuta, što je za krajnjeg korisnika predugo čekanje samo da bi dobio potvrdu onoga što je upravo izgovorio.

Postoje alternativna rešenja, uključujući biblioteke specifične za React koje putem API-ja omogućavaju komunikaciju sa modelima za obradu zvuka. Međutim, njihova upotreba se naplaćuje. Za ovaj projekat je na kraju korišćen Web Speech API, koji pruža bržu transkripciju i relativno pouzdanu interpretaciju govora, iako sa nešto nižim stepenom preciznosti u poređenju sa Whisper-om.

Naredni izazov predstavlja sporo generisanje odgovora od strane LLM modela, naročito nakon uvođenja funkcionalnosti za kategorizaciju odgovora. Glavni razlog za usporenje leži u činjenici da model generiše kompletan odgovor, koji se zatim kroz socket konekciju prosleđuje klijentu. Kako bi se ubrzao ovaj proces, uvedeno je strimovanje tokena, mehanizam kojim se tokeni iterativno šalju korisniku već tokom generisanja odgovora, bez čekanja da se kompletira ceo tekst. Ova optimizacija je donekle ubrzala sistem, ali i dalje postoji značajno čekanje, koje retko kada traje kraće od 20 sekundi.

Ovde dolazimo do problema veličine samih modela i ograničenja hardverskih kapaciteta. Manji modeli, poput LLaMA modela koji zauzima oko 1.2 GB memorije (1.24 milijardi parametara), najbrži su u strimovanju tokena, ali zbog ograničenog broja parametara ne pružaju zadovoljavajući kvalitet odgovora. S druge strane, veći modeli poput OpenHermes-a (7.24 milijardi parametara) generišu znatno kvalitetnije i sadržajnije odgovore, čime značajno poboljšavaju korisničko iskustvo i prirodnost konverzacije, ali su znatno sporiji u strimovanju.

Performanse sistema dodatno zavise od korišćenog hardvera: grafička kartica korišćena u testiranju je Nvidia GeForce GTX 1650 [9], model koji u poređenju s novijim RTX serijama nudi značajno slabije performanse pri lokalnom izvršavanju velikih jezičkih modela. Opremljena je sa svega 4 GB video memorije (VRAM), što je znatno manje u odnosu na savremene RTX kartice koje raspolazu sa najmanje 12 GB VRAM-a. Ograničeni kapacitet video memorije onemogućava da modeli sa oko 8 milijardi parametara budu u potpunosti učitani u VRAM, te se stoga koristi systemska memorija (RAM), što značajno usporava obradu podataka.

Pored kapaciteta memorije, ključni faktor u performansama savremenih grafičkih kartica je prisustvo tzv. tenzorskih jezgara (tensor cores), specijalizovanih jedinica za brzo izvođenje matričnih operacija. Matrične operacije su od suštinskog značaja za treniranje modela, kodiranje ulaznih reči i generisanje svakog sledećeg tokena. U nedostatku ovih jezgara, gubi se mogućnost efikasne paralelizacije, što dodatno usporava rad.

Iako određeni veliki jezički modeli, poput Llama, nude alate kao što je llama.cpp koji omogućavaju optimizovano lokalno izvršavanje čak i na slabijim grafičkim procesorima, ukupne performanse ostaju neuporedive sa onima koje omogućavaju moderne RTX kartice. Ove kartice, osim što poseduju veći VRAM i tenzorska jezgra, u stanju su da obrade najmanje 70 tokena u sekundi. Međutim, njihova glavna mana je visoka cena i znatno veća potrošnja energije.

Čak ni idealan hardver ne predstavlja celokupno rešenje, jer je podjednako važno obratiti pažnju na implementaciju sistema i korisničko iskustvo. Primer toga: U kontekstu kategorizacije izlaza u "coding" ili "finished", OpenHermes u svojoj "finished open" verziji povremeno je ispoljavao poteškoće u praćenju instrukcija formatiranja. Naime, od modela se očekivalo da za svaku kategoriju generiše odgovor u sledećem formatu:

Topic: 0 (ili 1)

Međutim, dešavalo se da model, osim tražene kategorizacije, generiše i dodatno objašnjenje ili tekst, što je izazivalo probleme pri parsiranju rezultata. U pojedinim slučajevima, iako ređe, model bi umesto predviđenog formata vratio samo niz brojeva (npr. 0,1,1,0,0), čime je dodatno otežavao tačnu obradu podataka zbog nepredvidivosti odgovora. Ipak, u većini slučajeva model je uspevao da izgeneriše izlaz u ispravnom formatu.

Veliki jezički modeli često ne prate u potpunosti sve instrukcije sadržane u upitu. To se naročito odnosi na zahteve poput „ne prikazuj nikakav kod“ ili „ne generiši sopstvene test-primere“. Naša je namera da intervjueru unapred obezbedimo proverenu bazu test primera, pri čemu su neki primeri javno dostupni, dok su drugi interni ili privatni. Odnosno, ako kod stvarno radi, mora da prođe i one testove koje kandidat ne može direktno da vidi kako ne bi adaptirao svoj kod samo za te primere. Međutim, modeli često samostalno kreiraju test primere u formatu ulaza i očekivanog izlaza. Ti primeri neretko ne obuhvataju sve relevantne, a posebno krajnje slučajeve, pri čemu su i generisana rešenja često netačna. Zbog toga modeli ovog tipa, posebno oni sa oko osam milijardi parametara, prilagođeni za rad na slabijim računarima, nisu pouzdani za testiranje rešenja. Bez obzira na to, dobra je praksa oslanjati se na već proverene i pouzdane test primere.

Jedan od značajnih nedostataka trenutnog sistema jeste nepostojanje obrade vremenskih pauza, odnosno dužine tišine tokom koje kandidat ne govori. Naime, sistem je trenutno fokusiran isključivo na analizu i razumevanje verbalnog sadržaja, onoga što je kandidat izgovorio. Ipak, podjednako su relevantni i aspekti kao što su elokventnost i fluidnost izražavanja, kao i trajanje zastoja u govoru. Na primer, kandidat može unositi samo kod bez ikakvog verbalnog izražavanja, što nije prihvatljivo u kontekstu stvarnog intervjua, iako to LLM-u ne predstavlja prepreku, bez obzira na način na koji je promptovan. Osim toga, zastoji i nesigurnost u formulisanju odgovora često ukazuju na nepripremljenost ili nedostatak razumevanja, što ostaje nevidljivo kada se analizira isključivo tekstualni input. Stoga bi integracija vremenskih parametara i govorne dinamike bila ključna za verodostojniju procenu kandidata.

Jedno od mogućih rešenja problema dugih pauza u govoru kandidata jeste uvođenje tajmera. Jednostavan pristup podrazumeva postojanje dodatnog, nevidljivog sistema za snimanje govora, čija je osnovna funkcija detekcija produženih perioda ćutanja. Alternativno, može se promeniti način rada mikrofona tako da kandidat ne mora da pritiska dugme za govor; umesto toga, sistem automatski osluškuje i obrađuje ulaz čim kandidat počne da govori. Ukoliko prođu otprilike tri sekunde tišine nakon kandidatskog odgovora, jezički model (LLM) započinje procesuiranje.

Međutim, kako bi ovakav sistem funkcionisao optimalno, neophodno je uvesti dodatne mehanizme. Na primer, kandidat može odlučiti da nastavi sa govorom i nakon te trosekundne pauze, što bi značilo da bi model morao da prekine započetu obradu. Rešenje može biti sistem reda čekanja, u kom LLM obrađuje jedan deo teksta, dok se naredni govori smeštaju u red za kasniju obradu. Takođe, u scenarijima kada intervjuer govori, kandidat bi mogao da ga prekine, ali bi intervjuer nastavio sa izlaganjem jer još uvek nije završio svoj proces.

## 5. Zaključak

U ovom radu opisano je razvijanje modela za obradu prirodnog jezika, kao i arhitektura koja predstavlja osnovu savremenih agenata veštačke inteligencije. Iako se danas primenjuje niz optimizacija nad transformer arhitekturom, osnovni principi ostaju ključni za razumevanje njihovog rada. Takođe su predstavljeni načini za merenje performansi ovih modela, kao i osnove prompt inženjeringa, veštine koja je često jedina neophodna za korisnike zainteresovane za integraciju AI agenata u sopstvene sisteme, bez potrebe za dubokim poznavanjem njihove unutrašnje arhitekture.

Veliki jezički modeli korišćeni su za izgradnju intervjera zasnovanog na veštačkoj inteligenciji. U radu je opisan ceo proces, od bihevioralnih i tehničkih (kodnih) intervjua, do evaluacije performansi kandidata. Sistem je razvijen pomoću biblioteka otvorenog koda, uz izuzetak komponenti za pretvaranje teksta u govor, i socket komunikacije, sa ciljem da bude brz, efikasan i jeftin. Diskutovani su izazovi i moguća poboljšanja, uključujući snažniji hardver i naprednije modele sa većim brojem parametara.

Ovakav sistem može se koristiti kao zamena za prvu fazu selekcije kandidata, umesto klasičnih online testova. Ipak, ne rešava u potpunosti problem procene kandidata, te se preporučuje njegovo kombinovanje sa sistemima za snimanje ekrana, a potencijalno i sa metodama za detekciju objekata (kao što su pokreti očiju ili položaj glave). Alternativno, sistem se može koristiti u edukativne svrhe, kao alat za vežbu intervjua, posebno za one koji osećaju nelagodu pri klasičnim mock intervjuiima. Pored toga, može se integrisati u postojeće kurseve: nakon svakog poglavlja, korisnik bi putem AI intervjera mogao da proveri i primeni novo naučeno znanje. To je i cilj naredne iteracije sistema: da se dodatno optimizuje i primeni u različitim edukativnim kontekstima.

## Literatura

1. DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, et al. “DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning.” arXiv, January 22, 2025. <https://doi.org/10.48550/arXiv.2501.12948>.
2. Cai, Kenrick, Anna Tong, Jeffrey Dastin, Kenrick Cai, Anna Tong, and Jeffrey Dastin. “DeepSeek Leaves US AI Firms Racing to Understand Its Success.” Reuters, January 29, 2025, sec. Artificial Intelligence. <https://www.reuters.com/technology/artificial-intelligence/american-ai-firms-try-poke-holes-disruptive-deepseek-2025-01-28/>.
3. Grattafiori, Aaron, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, et al. “The Llama 3 Herd of Models.” arXiv, November 23, 2024. <https://doi.org/10.48550/arXiv.2407.21783>.
4. Lin, Stephanie, Jacob Hilton, and Owain Evans. “TruthfulQA: Measuring How Models Mimic Human Falsehoods.” arXiv, May 8, 2022. <https://doi.org/10.48550/arXiv.2109.07958>.
5. “LiveBench.” <https://livebench.ai/>
6. “LM Studio - Discover, Download, and Run Local LLMs.” Accessed May 31, 2025. <https://lmstudio.ai/>.
7. “Microsoft/Monaco-Editor.” JavaScript. 2016. Reprint, Microsoft, June 3, 2025. <https://github.com/microsoft/monaco-editor>.
8. Mishra, Harsh. “Top 15 LLM Evaluation Metrics to Explore in 2025.” Analytics Vidhya (blog), March 7, 2025. <https://www.analyticsvidhya.com/blog/2025/03/llm-evaluation-metrics/>.
9. TechPowerUp. “NVIDIA GeForce GTX 1650 Specs,” June 10, 2025. <https://www.techpowerup.com/gpu-specs/geforce-gtx-1650.c3366>.
10. “ChatGPT Pricing.” Accessed June 10, 2025. <https://openai.com/api/pricing/>.
11. Schulhoff, Sander. “Assigning Roles to Chatbots.” Accessed May 31, 2025. <https://learnprompting.org/docs/basics/roles>.
12. Team, Gemma, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, et al. “Gemma 3 Technical Report.” arXiv, March 25, 2025. <https://doi.org/10.48550/arXiv.2503.19786>.
13. “Teknium/OpenHermes-2.5-Mistral-7B · Hugging Face,” April 15, 2024. <https://huggingface.co/teknium/OpenHermes-2.5-Mistral-7B>.

14. Yang, An, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. "Qwen3 Technical Report." arXiv, May 14, 2025.  
<https://doi.org/10.48550/arXiv.2505.09388>.
15. Sirovy, Ladislav. "10 Key Metrics to Evaluate Your AI Chatbot Performance."

## Dodatak

### Bihevioralni intervju

"USER: hello I'm here"

"AI: I am currently online and available for your interview. Please begin by providing some information about yourself, such as your name, educational background, work experience, and any specific skills or technologies you are proficient in. This will help me understand your qualifications and better tailor the questions I ask during the interview."

"USER: yes my name is Vanya I am currently an assistant for machine learning and quantum computing courses on the school of faculty in Belgrade I worked on Microsoft that is"

"AI: Hello Vanya, it's great to have you here today. Your background in machine learning and quantum computing is certainly impressive. It sounds like you have a strong foundation in these areas, which can be applied in various fields including software engineering. Now, let's dive into your previous experiences and projects. Can you tell me about any significant projects you've worked on in the past, and what challenges or difficulties you faced during those projects?"

"USER: yes when I was working as an intern in Microsoft I had to build an add to dictionary extension for Microsoft Word online and I guess my biggest challenge is to figure it out how does it all work because the entire structure is pretty complex also currently I'm in a startup that builds a platform for mock interviews that is interviews that are not real but conducted by your peers that is the same people that look for similar job and we're building putting a social aspect to that and my biggest challenge was to create a chat functionality because I was doing the designing of the chat visual design and also data design and implementation so that the chat could be life live"

"AI: It sounds like you have had some varied and interesting experiences in your career so far, Vanya. Working on an add-to-dictionary extension for Microsoft Word Online must have been a great learning experience, as it involved understanding the complex structure of Microsoft's software ecosystem. It also allowed you to contribute directly to a product used by millions of people worldwide. Your current project at the startup focuses on building a social aspect into mock interviews, which is an innovative approach to helping job seekers prepare for their future careers.\n\nCreating the chat functionality for this platform seems like it presented some

challenges, especially considering the visual design and data implementation aspects you mentioned. Can you tell me more about how you approached these challenges and what solutions you implemented?"

"USER: well for the chat I had to learn the firestore framework to understand how I could build a chat tab for that we used firestore because it's real time that is I could I could send a message in the other user will immediately get notification in order to implement this feature I first discussed with my teammates we found a decent enough solution and I was working the component in react and also I guess the big challenge was optimizing for the reads because I was using a lot of firestore reads and there's a limit of 20,000 reads therefore I use some kind of local storage to keep track of the chats"

"AI: I see. It sounds like you had an interesting experience learning and implementing the Firestore framework for your chat functionality. It's great that you were able to work closely with your teammates to find a solution, and optimizing for read efficiency was crucial in this case. Using local storage to keep track of chats was an effective approach to avoid hitting the 20,000 reads limit. This type of problem-solving and adaptation is a valuable skill in software engineering.\n\nNow that we've discussed your previous experiences and some of the challenges you faced, I would like to move on to the coding part of the interview. Please follow the instructions on your screen to access the coding environment. Once there, you will be presented with a problem to solve. You will have a set amount of time to write and test your code. I'll be watching closely to assess your problem-solving abilities"

---

"USER: hello I'm here"

"AI: Good morning! Welcome to our company's job interview. My name is AI Interviewer, and I will be guiding you through this process. Can you please tell me a little bit about yourself? What are your strengths and what motivated you to pursue software engineering as a career?"

"USER: yes I am Vanya and I am a teaching assistant for courses such as machine learning quantum computing and object Orient programming also I work in a startup where we build a platform based on Mock interviews"

"<FINISHED\_INTERVIEWING>"

"AI: It's nice to meet you, Vanya. Based on your background, it seems like you have a solid understanding of various technical subjects and experience working with startups. I believe these skills would be valuable to our company. Now, let me ask you about some of the projects that you've worked on. Can you describe one or two significant projects you've been involved in and what your role was in their development?"

"USER: yes as I mentioned I worked in a startup for make for mock interviews where the goal is to match with people all around the world to find their coding interview partner specifically I was working mostly on the front tenth side of the platform front end side of the platform"

"AI: It sounds like an interesting project. Can you elaborate on some of the challenges you faced while working on this platform and how you overcame them? Also, can you provide examples of specific features or optimizations that you were involved in developing and how your work impacted the overall performance and functionality of the platform?"

"USER: yes I was working mostly as I said the front end but I also worked in a functionality chat where I have made basically the entire dialogue and also set up the infrastructure where using firebase so we picked fire basic because of its real time features and relatively cheap or basically free to use for some of the messages"

"<FINISHED\_INTERVIEWING>"

"AI: Setting up the chat functionality with Firebase is indeed a valuable contribution. It's great that you were able to leverage real-time features while keeping costs low. Now, I would like to move on to the coding part of the interview. Please write a function in your preferred programming language that can reverse an array of integers."

## Intervju kodiranja

"USER: hello I am here now"

"AI: Alright, let's start then. Please describe the first step that comes in your mind to solve this problem."

"USER: all right I might describe the whole thing so the idea is to take the given string use a method something like split which will convert the string into an array of words we could split it by spaces because for the words are separated by spaces and we are doing that in order to get the last word in the string the N minus 1 element from the array and in order to complete the task we should simply calculate the number of characters by using the length method and I'm basically speaking all of this if we're assumed we're using Java"

"<TIME\_TO\_CODE>"

"AI: Thank you for providing your solution. Please go ahead and implement your code."

"<PASTE\_TEXT\_EXAMPLE>"

USER: alright great so I'll create the method called length of last word public static Int length of last word my argument is string s and first I'll use the split method that will return a primitive array

of words not a list of strings but literally an array of strings so as that's plate and I pass the space because the words are separated by spaces okay now in order to get the last element simply used I'll simply create a variable called n and I use words that length and in order to calculate the number of characters in the last word I'll take the N minus 1 because the arrays are zero indexed the length and they return it just like that and in the main method I'll simply call length of flask word yellow World and This is it

<code\_section>

```
import java.util.*;
public class Main {
 public static int lengthOfLastWord(String s) {
 String[] words = s.split(" ");
 int n = words.length;
 return words[n-1].length();
 }
 public static void main(String[] args){
 System.out.println(lengthOfLastWord("Hello world"));
 }
}
```

</code\_section>

"AI: Alright, let's go through the code you've written. Firstly, you're using the `split` method to convert the input string into an array of words. This looks fine. Next, you're calculating the length of the array `words`, which should indeed give us the number of words in the given string. Then, you're returning the length of the last word by accessing the element at index `n-1` and calling the `length()` method on it. This looks good as well. Finally, in your `main` method, you're calling the `lengthOfLastWord` method with the string "Hello world" and printing out the result. All right, let's run this code and see if it works correctly. So, what is the time complexity of your solution?"

"<PASTE\_TEXT\_EXAMPLE>"

"USER: O(n)" Napomena: Kandidat je kucao rešenje, nije izgovorio putem mikrofona, jer u većini slučajeva Speech-To-Text model ne bi korektno obradio govor.

"<TIME\_TO\_CODE>"

"AI: Correct! Your time complexity is O(n). Let's now run a few test cases to see if it works correctly.``

```
System.out.println(lengthOfLastWord("Hello World")); // Output: 5
```

```
System.out.println(lengthOfLastWord("Hello World")); // Output: 5
```

```
System.out.println(lengthOfLastWord("Hello World ")); // Output: 5
```

All of these test cases seem to be working correctly. Congratulations on a job well done!”

Napomena: AI intervjuer ne bi trebalo da izbací svoje test primere, već da ih izvuče iz baze. Takođe kandidat treba sam da testira kod, ne LLM za njega, jer LLM-ova rešenja uglavnom nisu pouzdana.