SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 847

VELIKI JEZIČNI MODELI ZA OPTIMIZACIJU, PLANIRANJE I OBJAŠNJIVOST U INTELIGENTNIM AGENTSKIM SUSTAVIMA

Antun Tišljar

Zagreb, srpanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 847

VELIKI JEZIČNI MODELI ZA OPTIMIZACIJU, PLANIRANJE I OBJAŠNJIVOST U INTELIGENTNIM AGENTSKIM SUSTAVIMA

Antun Tišljar

Zagreb, srpanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 3. ožujka 2025.

DIPLOMSKI ZADATAK br. 847

Pristupnik:	**Antun Tišljar (0036508971)**
Studij:	Računarstvo
Profil:	Znanost o podacima
Mentor:	izv. prof. dr. sc. Alan Jović

Zadatak: **Veliki jezični modeli za optimizaciju, planiranje i objašnjivost u inteligentnim agentskim sustavima**

Opis zadatka:

Sustavi zasnovani na inteligentnim agentima važni su za dobivanje relevantnih informacija jer povećavaju preciznost i prilagodljivost temeljnih zadataka. U ovom diplomskom radu potrebno je istražiti primjenu velikih jezičnih modela (engl. large language model, LLM) kao inteligentnih agenata u razvoju sustava, s naglaskom na optimizaciju točnosti i dugoročno planiranje kroz zaključivanje u više koraka. U radu je potrebno teoretski objasniti arhitekturu LLM-ova, njihove mehanizme zaključivanja i metode za osiguranje transparentnosti i objašnjivosti, te kako oni zadržavaju i koriste kontekst tijekom duljih interakcija. Nakon toga, treba istražiti kako LLM-ovi funkcioniraju kao autonomni inteligentni agenti koji mogu obavljati složene zadatke. Pritom treba razmotriti metode za procjenu točnosti agenata, uključujući koherentnost odgovora i dosljednost zaključivanja, s naglaskom na transparentnost u procesu donošenja odluka. Rad također treba obraditi problem pristranosti u LLM-ovima tako da se identificiraju izvori pristranosti u podacima za učenje te predložiti strategije za povećanje pravednosti i pouzdanosti. U praktičnom dijelu rada treba razviti prototip agenta temeljenog na LLM-u koji usvaja sve teorijske koncepte, demonstrirajući sposobnost obrade podataka, planiranje, donošenja informiranih odluka i pružanja odgovora s objašnjenjem.

Rok za predaju rada: 4. srpnja 2025.

Zahvaljujem svom mentoru, Izv. prof. dr. sc. Alanu Joviću, na strpljenju i podršci koje mi je pružao tijekom mentorstva.

Sadržaj

1. Uvod

Sustavi inteligentnih agenata imaju ključnu ulogu u područjima poput automatizacije, personaliziranih preporuka i napredne podrške pri donošenju odluka [1]. Kako stvarne primjene postaju sve složenije i dinamičnije, raste potreba za prilagodljivim i učinkovitijim rješenjima. Veliki jezični modeli (engl. *large language models*, LLMs) predstavljaju značajan napredak u transformaciji mogućnosti obrade prirodnog jezika (engl. *natural language processing*, NLP), zaključivanja i rješavanja problema [2, 3].

U području umjetne inteligencije, agent se definira kao autonomni entitet koji opaža, zaključuje i djeluje s ciljem ostvarivanja određenih zadataka. Za razliku od tradicionalnih sustava temeljenih na pravilima, LLM-ovi omogućuju veću fleksibilnost, prilagodbu kontekstu i sofisticirane sposobnosti zaključivanja. Ipak, integracija LLM-ova u sustave agenata donosi izazove, poput netransparentnih procesa zaključivanja te pristranosti naslijeđenih iz podataka za učenje, što otvara pitanja o pravednosti i pouzdanosti [4].

Ovaj rad istražuje integraciju LLM-ova u inteligentne agente, s naglaskom na poboljšanje točnosti, planiranja, transparentnosti i objašnjivosti. Rad objedinjuje teorijsku analizu, pregled literature i praktični razvoj. Konkretno, ciljevi ovog rada su:

1. Objasniti arhitekturu i mehanizme LLM-ova u usporedbi s tradicionalnim metodama.

2. Ispitati zaključivanje u više koraka i upravljanje kontekstom kod LLM-ova.

3. Analizirati metode koje osiguravaju transparentnost i objašnjivost.

4. Razmotriti izvore pristranosti i predložiti strategije za povećanje pravednosti.

5. Razviti i evaluirati prototip agenta koji demonstrira planiranje, optimizaciju i objaš-

njive odluke.

Struktura rada započinje temeljnim konceptima LLM-ova, nakon čega slijede detaljne rasprave o zaključivanju, transparentnosti, objašnjivosti i ublažavanju pristranosti. Sljedeća poglavlja opisuju praktične implementacije i evaluacije, dok završni dio donosi sintezu ključnih nalaza i smjernice za buduća istraživanja.

Glavni doprinos rada je istraživanje i analiza inteligentnih agenata temeljenih na LLM-ovima, s posebnim naglaskom na transparentno, interpretabilno i pouzdano oblikovanje.

2. Veliki jezični modeli: arhitektura

2.1. Predstavljanje ulaza

2.1.1. Tokenizacija

Prije nego što veliki jezični model može obraditi tekstualni ulaz, potrebno je podatke pretvoriti u numerički format koji model može interpretirati. Taj proces započinje tokenizacijom, odnosno razdvajanjem teksta na manje jedinice zvane tokeni. Tokeni mogu predstavljati riječi, dijelove riječi (engl. *subwords*) ili čak pojedinačne znakove, ovisno o korištenoj tokenizacijskoj shemi [5, 6].

Vrste tokenizacijskih shema:

- **Tokenizacija na razini riječi**: Svaka riječ je token. Ova metoda nije otporna na nepoznate (engl. *out-of-vocabulary*) riječi.
- **Tokenizacija na razini dijelova riječi**: Algoritmi poput Byte-Pair Encoding (BPE) i WordPiece razdvajaju riječi na češće korištene dijelove, što omogućuje modelu obradu rijetkih i novih riječi njihovom dekompozicijom (npr. “elven-king” postaje “elven”, “-”, “king”).
- **Tokenizacija na razini bajta ili znaka**: Najmanje jedinice; fleksibilna metoda, ali manje učinkovita u prepoznavanju semantičkog značenja.

Tokenizatori, programi koji provode tokenizaciju, često zasebno tretiraju interpunkciju, razmake i velika/mala slova. Na primjer, “sing” i “sing;” su različiti tokeni. Velika i mala slova mogu biti razdvojena ili preslikana na različite načine, ovisno o modelu.

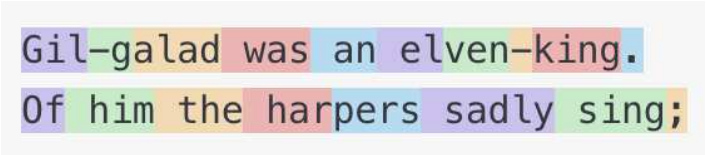
LLM-ovi rade unutar ograničenog prozora konteksta (npr. 2.048 do 128.000 tokena,

ovisno o modelu). Ako ulazni tekst premašuje ovu duljinu, dijeli se na manje, često i preklapajuće dijelove. Svaki dio obrađuje se zasebno, što može utjecati na sposobnost modela da prepozna širi kontekst.

Na primjer, razmotrimo sljedeći tekst:

Gil-galad was an elven king.
Of him the harpers sadly sing;

Gornji odlomak najprije se dijeli na tokene pomoću tokenizatora prije nego što ga model obradi. Slika 2.1. prikazuje kako se ovaj tekst razlaže na tokene koristeći OpenAI-jev tokenizator Svaki token se zatim preslikava u jedinstveni cjelobrojni identifikator, što je prikazano na slici 2.2.



Slika 2.1. Tokenizacija Tolkienovih stihova korištenjem OpenAI tokenizatora. Svaki obojeni blok predstavlja zaseban token koji model proizvodi.

```
[156974, 6559, 195559, 673, 448, 650, 1066, 12, 6962, 558, 2566, 2395,
290, 3664, 36729, 71089, 6211, 26]
```

Slika 2.2. Jedinstveni cjelobrojni identifikatori za svaki token proizveden OpenAI-jevim tokenizatorom. Ovi identifikatori čine ulaz u sloj vektorske reprezentacije tokena.

2.1.2. Sloj vektorske reprezentacije tokena

Sloj vektorske reprezentacije tokena (engl.i dalje *embedding layer*) preslikava svaki indeks tokena u kontinuirani, zbijeni vektor u prostoru visoke dimenzionalnosti. Time se modelu omogućuje prepoznavanje semantičkih i sintaktičkih sličnosti među tokenima, što rezultira učinkovitijim učenjem i generalizacijom [7].

Rad s ID-ovima tokena izravno ne bi omogućio modelu razumijevanje odnosa među riječima ili dijelovima riječi, budući da je svaki ID samo jedinstveni broj bez inherentnog značenja. *Embedding* sloj to rješava učenjem vektorskih prikaza na način da su slični tokeni (npr. “king” i “queen”) smješteni blizu jedan drugoga u *embedding* prostoru, dok su nepovezani tokeni (npr. “king” i “apple”) znatno udaljeniji. Ovo preslikavanje pruža

neuronskoj mreži bogatu, kontinuiranu strukturu ulaza koja kodira jezične odnose na način koji je koristan za daljnju obradu.

Matematički, *embedding* sloj je zapravo tablica pretraživanja, odnosno matrica *embeddinga* $E \in \mathbb{R}^{|V| \times d}$, gdje je $|V|$ veličina rječnika, a d dimenzionalnost svakog *embedding* vektora. Za svaki token t_i, njegov cjelobrojni indeks odabire odgovarajući redak u E, čime se dobiva embedding vektor x_i:

$$x_i = E_{t_i} \tag{2.1}$$

Za slijed tokena $(t_1, t_2, \ldots, t_n)$, ulazna matrica $X \in \mathbb{R}^{n \times d}$ konstruira se kao:

$$X = \begin{bmatrix} E_{t_1} \\ E_{t_2} \\ \vdots \\ E_{t_n} \end{bmatrix} \tag{2.2}$$

Ovi vektori *embeddinga* zatim se prosljeđuju sljedećim slojevima modela (kao što su transformerski blokovi), gdje služe kao temelj za svu daljnju obradu.

Učenje *embeddinga* Matrica *embeddinga* E nije statična; ona se uči tijekom učenja modela pomoću algoritma propagacije pogreške unatrag (engl. *backpropagation*). Kako model prolazi kroz sve više podataka, vektori *embeddinga* se prilagođavaju kako bi se poboljšala sposobnost predviđanja ili generiranja teksta. Ovaj proces učenja omogućuje da *embeddinzi* odražavaju širok raspon jezičnih svojstava, poput sinonimije, vrste riječi i odnosa analogije. Na primjer,

$$E_{\text{king}} - E_{\text{man}} + E_{\text{woman}} \approx E_{\text{queen}}$$

gdje E_{token} označava vektor *embeddinga* koji odgovara određenom tokenu.

Pozicijsko kodiranje Budući da transformerski modeli ne prepoznaju redoslijed tokena u nizu sami po sebi, na vektor *embeddinga* se u ovoj fazi dodaje kodiranje pozicije (engl. *positional encodings*) [8]. Ta kodiranja omogućuju modelu da prepozna i iskoristi

sekvencijalnu strukturu jezika dodavanjem informacije o poziciji svakog tokena.

Ukratko, *embedding* sloj pretvara sirove ID-ove tokena u smislene, kontinuirane vektorske prikaze koji služe kao temelj za sve daljnja neuronska računanja u velikom jezičnom modelu. Zahvaljujući ovom postupku, model može učinkovito modelirati jezične odnose, što je ključno za visoke performanse današnjih dubokih modela za obradu teksta.

2.2. Arhitektura transformera

Arhitektura transformera, koju su predstavili Vaswani i suradnici [8] 2017. godine, označava veliki iskorak u području obrade prirodnog jezika i dubokog učenja. Prije pojave transformera, sekvencijski modeli poput povratnih neuronskih mreža (engl. *Recurrent Neural Networks*, RNN), LSTM-ova (engl. *Long Short-Term Memory*) i GRU-ova (engl. *Gated Recurrent Units*) obrađivali su sekvence token po token. Takav pristup ograničavao je paralelizaciju i otežavao prepoznavanje dugoročnih ovisnosti, dijelom zbog problema poput nestajućih gradijenata.

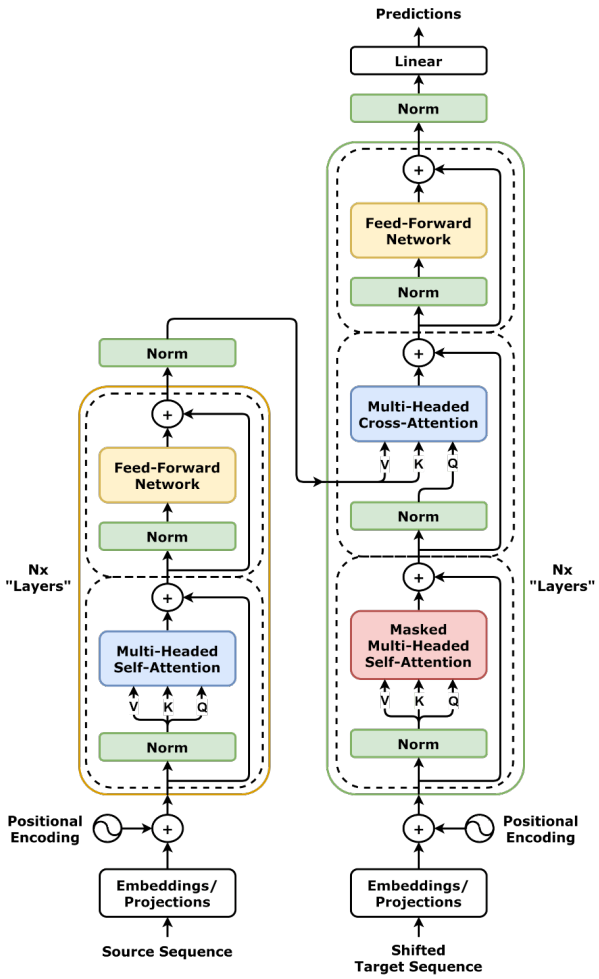
Ključna inovacija transformera je mehanizam samopozornosti (engl. *self-attention*), koji omogućuje modelu da izravno poveže sve pozicije unutar sekvence odjednom. Na taj način transformer može prepoznati kontekstne odnose između bilo koja dva tokena, neovisno o njihovoj udaljenosti, i obraditi sve tokene istovremeno, što značajno poboljšava računalnu učinkovitost i skalabilnost.

Ovaj pristup se ubrzo pokazao revolucionarnim. Modeli temeljeni na transformeru, poput BERT-a [9] i GPT-a [10], postavili su nove standarde u širokom rasponu zadataka iz obrade prirodnog jezika. Sposobnost transformera da uči bogate, kontekstne značajke podataka, u kombinaciji s prikladnošću za učenje na velikim skupovima podataka, omogućila je razvoj velikih jezičnih modela koji su nadmašili prethodne arhitekture i po fleksibilnosti i po performansama.

Osnovne komponente transformera su samopozornost, pozornost s više glava za čitanje, unaprijedne potpuno povezane mreže, pozicijsko kodiranje, rezidualne veze, normalizacija slojeva i duboko slaganje slojeva. Upravo zahvaljujući njima, suvremeni LLM-ovi postižu iznimne performance.

2.2.1. Osnove rada transformera

U svojoj osnovi, transformer se sastoji od niza identičnih slojeva, pri čemu svaki sloj sadrži dvije glavne komponente: pozornost s više glava za čitanje, i unaprijedno potpuno povezanu mrežu. Ove komponente povezane su rezidualnim vezama (engl. *skip connections*) i normalizacijom slojeva, što omogućuje izgradnju dubokih i stabilnih mreža.



Slika 2.3. Arhitektura transformer modela, prema Vaswani i sur. [8]. Slika reproducirana uz dopuštenje u edukativne svrhe.

Arhitektura prikazana na slici 2.3. uključuje i koder (engl. *encoder*) i dekoder (engl. *decoder*) iz originalnog modela transformera. Modeli koji koriste samo koder (npr. BERT) najčešće se primjenjuju za razumijevanje jezika, dok se modeli temeljeni isključivo na dekoderu (npr. GPT) koriste za generiranje jezika i čine osnovu većine suvremenih LLM agenata. U nastavku će naglasak biti na dekoderskoj arhitekturi.

Ulaz u transformer Ulaz u prvi sloj transformera dobiva se tako da se za svaki token zbroje njegov vektor *embeddinga* i pripadajuće pozicijsko kodiranje:

$$h_i^{(0)} = embedding(t_i) + \text{positional\_encoding}(i)$$

gdje je $h_i^{(0)}$ ulazni vektor za token i na najnižem sloju mreže.

Budući da mehanizam samopozornosti sam po sebi ne uzima u obzir redoslijed tokena, pozicijska kodiranja se dodaju vektorima *embeddinga* kako bi se modelu prenijele informacije o pozicijama tokena u sekvenci.

Samopozornost Mehanizam samopozornosti omogućuje svakom tokenu u ulaznoj sekvenci da "obraća pažnju" na sve ostale tokene, čime se hvataju ovisnosti bez obzira na njihovu međusobnu udaljenost. Intuitivno, svaki token u sekvenci dobiva tri različite reprezentacije, poznate kao pitanje (engl. *query*), ključ (engl. *key*) i vrijednost (engl. *value*):

- **Pitanje**: "Što tražim?"
- **Ključ**: "Što mogu ponuditi?"
- **Vrijednost**: "Koji sadržaj dijelim ako netko obrati pažnju na mene?"

Matematički, to se ostvaruje projekcijom svakog ulaznog vektora $h_i^{(l-1)}$ (skriveno stanje iz prethodnog sloja) u tri vektora—pitanje q_i, ključ k_i i vrijednost v_i—pomoću naučenih linearnih transformacija. Za sekvencu predstavljenu kao $H^{(l-1)} = [h_1^{(l-1)}, h_2^{(l-1)}, ..., h_n^{(l-1)}]$, matrice pitanja, ključeva i vrijednosti računaju se na sljedeći način:

$$Q = H^{(l-1)}W^Q, \quad K = H^{(l-1)}W^K, \quad V = H^{(l-1)}W^V$$

gdje su $W^Q, W^K, W^V \in \mathbb{R}^{d \times d_k}$ matrice naučenih težina.

Pozornost (engl. *Attention*) se zatim izračunava pomoću skaliranog skalarnog produkta između svakog pitanja i svih ključeva:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{2.3}$$

Dijeljenje s $\sqrt{d_k}$ sprječava da skalarni produkti postanu preveliki s povećanjem dimenzionalnosti, što bi moglo dovesti do toga da funkcija *softmax* daje vrlo male gradijente i time oteža učenje [8]. Softmax osigurava da zbroj težina u mehanizmu pozornosti za svaki token iznosi jedan, određujući koliko pažnje svaki ulazni token posvećuje ostalima.

Pozornost s više glava za čitanje Umjesto da se oslanjaju na samo jedan mehanizam pozornosti, transformeri koriste pozornost s više glava za čitanje (engl. *multi-head attention*). To znači da se više mehanizama pozornosti (tzv. "glava") izvodi paralelno, a svaka ima svoj skup naučenih težina. Izlazi svih glava potom se spajaju (konkateniraju) i linearno transformiraju kako bi se dobila konačna reprezentacija:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

pri čemu je svaka glava neovisan izračun samopozornosti:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Ovakva arhitektura omogućuje modelu prepoznavanje različitih odnosa i informacija unutar sekvence na različitim pozicijama i razinama. Budući da svaka glava može usmjeriti pažnju na specifične aspekte podataka, poput lokalne sintakse, dugotrajnih ovisnosti ili pojedinih semantičkih značajki, pozornost s više glava za čitanje rezultira bogatijim i izražajnijim modelom.

Unaprijedno potpuno povezana mreža Nakon mehanizma pozornosti s više glava za čitanje, reprezentacija svakog tokena prolazi kroz unaprijedno potpuno povezanu mrežu (engl. *feed-forward network*, FFN) koja se primjenjuje zasebno na svaku poziciju u sekvenci. Ova mreža radi neovisno i jednako za svaki token unutar sloja, što znači da se ista unaprijedno potpuno povezana mreža paralelno primjenjuje na *embedding* svakog tokena, dok svaki sloj transformera ima svoj vlastiti skup težina. Važno je napomenuti da se u ovoj fazi ne dijeli informacija između tokena, već kontekst dolazi isključivo iz mehanizma pozornosti.

Matematički, FFN se sastoji od dvije linearne transformacije između kojih se nalazi

nelinearna aktivacijska funkcija, najčešće ReLU:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

gdje su W_1, W_2 matrice težina, a b_1, b_2 vektori pomaka. Prva transformacija obično projicira ulaz u prostor veće dimenzionalnosti, čime se povećava kapacitet modela, zatim slijedi ReLU nelinearnost, a potom se podaci vraćaju u izvornu dimenziju. Na taj način model može primijeniti složene nelinearne transformacije na reprezentacije tokena i dodatno unaprijediti modeliranje podataka.

Rezidualne veze i sloj normalizacije Kako bi se stabiliziralo učenje i omogućila izgradnja dubljih modela, transformeri koriste rezidualne veze (engl. *residual connections*) i sloj normalizacije (engl. *layer normalization*). Za svaku podkomponentu (pozornost s više glava za čitanje i unaprijedno potpuno povezana mreža), ulaz se zbraja s izlazom podkomponente, a rezultat se zatim normalizira:

$$\text{Output} = \text{LayerNorm}(x + \text{Sublayer}(x))$$

Rezidualne veze omogućuju lakši protok informacija i gradijenata kroz mrežu, čime se ublažava problem nestajućih gradijenata i omogućuje učinkovito učenje vrlo dubokih modela. Sloj normalizacije dodatno stabilizira aktivacije normalizirajući njihovu distribuciju na svakoj poziciji, što modelu pomaže da brže konvergira i bolje generalizira.

Slaganje slojeva Slaganjem većeg broja transformerskih slojeva model postupno izgrađuje bogate, kontekstno osjetljive reprezentacije svakog tokena. Rani slojevi uglavnom se fokusiraju na lokalne ili sintaktičke informacije, dok kasniji slojevi hvataju dugotrajne ovisnosti i apstraktne relacije, što omogućuje modelu da se nosi s kompleksnošću prirodnog jezika.

Ova arhitektura čini temelj modela baziranih na transformeru, na kojoj počivaju iznimne sposobnosti suvremenih velikih jezičnih modela.

2.3. Ciljevi učenja i modeliranje jezika

Suvremeni veliki jezični modeli uče se prvenstveno koristeći autoregresivno modeliranja jezika (engl. *autoregressive language modeling*), pri čemu model uči predviđati sljedeći token u sekvenci. Iako je maskirano modeliranje jezika (engl. *masked language modeling*) također važno za modele poput BERT-a koji su usmjereni na razumijevanje jezika, u ovom se dijelu fokusiramo na autoregresivni pristup koji koriste generativni LLM-ovi.

Autoregresivno (kauzalno) modeliranje Autoregresivno modeliranje jezika osnovni je cilj učenja za generativne modele poput GPT-a. U ovom pristupu model se uči da, na temelju svih prethodnih tokena u sekvenci, predvidi sljedeći token. Ovakav model često se naziva i kauzalnim ili *left-to-right* modelom jezika, jer je svako predviđanje zasnovano samo na prethodnim tokenima u nizu.

Formalno, za sekvencu tokena $(x_1, x_2, \ldots, x_n)$, model uči vjerojatnost cijele sekvence kao produkt uvjetnih vjerojatnosti:

$$P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid x_1, x_2, \ldots, x_{i-1}) \tag{2.4}$$

Tijekom učenja koristi se posebno kauzalno maskiranje, koje osigurava da svaki token može "obraćati pažnju" samo na sebe i prethodne tokene, ali ne i na buduće pozicije. Ovo maskiranje je nužno jer, iako je tijekom učenja cijela ciljna sekvenca poznata, modelu se mora onemogućiti da "gleda unaprijed" i koristi informacije iz budućih tokena pri predviđanju. Suprotno tome, tijekom generiranja teksta (inferencije), model generira jedan po jedan token; u svakom koraku budući tokeni još ne postoje, pa maskiranje nije potrebno.

Za razliku od izvornog transformera koji koristi i koder i dekoder (vidi sliku 2.3.), GPT-ovi i slični LLM-ovi koriste samo dekoderski blok s kauzalnim maskiranjem, što ih čini prirodno prilagođenima za generativne zadatke.

Funkcija gubitka: unakrsna entropija Za učenje jezičnog modela potrebno je kvantificirati koliko se predviđena distribucija vjerojatnosti modela podudara sa stvarnom sekvencom tokena. To se postiže korištenjem funkcije gubitka unakrsne entropije (engl.

cross-entropy), koja mjeri razliku između predviđene vjerojatnosti i stvarnog ishoda.

Za sekvencu tokena $(x_1, x_2, \ldots, x_n)$, model se uči tako da maksimizira vjerojatnost stvarne sekvence. To je ekvivalentno minimiziranju negativne logaritamske vjerojatnosti, što dovodi do izraza za gubitak unakrsne entropije:

$$\mathcal{L}_i = -\log P(x_i \mid x_1, x_2, \ldots, x_{i-1}) \tag{2.5}$$

gdje $P(x_i \mid x_1, x_2, \ldots, x_{i-1})$ označava vjerojatnost pridruženu sljedećem stvarnom tokenu.

Ukupni gubitak za naučenu sekvencu obično se računa kao suma (ili srednja vrijednost) gubitka unakrsne entropije za sve tokene u sekvenci:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_i \tag{2.6}$$

Minimiziranjem ovog gubitka tijekom učenja model se potiče da ispravnim tokenima pridjeljuje veće vjerojatnosti čime se poboljšava njegova sposobnost generiranja ili predviđanja koherentnog jezika.

Optimizacija gubitka Minimizacija gubitka unakrsne entropije provodi se korištenjem optimizacijskih algoritama temeljenih na gradijentima, poput optimizatora Adam [11]. Tijekom učenja, gradijenti gubitka računaju se metodom propagacije pogreške unatrag, a parametri se postupno ažuriraju radi smanjenja ukupnog gubitka. Ovaj proces odvija se na velikim skupovima tekstnih podataka i ponavlja tijekom više epoha dok model ne postigne optimalne parametre.

2.3.1. Prethodno učenje i fino podešavanje

Veliki jezični modeli svoje iznimne sposobnosti postižu kroz proces učenja u dvije faze: prethodno učenje (engl. *pretraining*) i fino podešavanje (engl. *finetuning*).

Prethodno učenje U fazi prethodnog učenja model se izlaže ogromnoj količini nestrukturiranih tekstnih podataka kao što su knjige, članci i internetske stranice, koristeći autoregresivno modeliranje opisano ranije [10, 2]. Učenje je najčešće *self-supervised*, što znači da model uči iz sirovog teksta bez potrebe za ručno označenim podacima. Pre-

dviđanjem sljedećeg tokena na svakoj poziciji u sekvenci, model postupno stječe široko razumijevanje gramatike, činjenica, zaključivanja i općeg znanja o svijetu. Opseg prethodnog učenja, kako u veličini skupa podataka tako i u broju parametara modela, ključan je za iznimne performanse suvremenih LLM-ova.

Fino podešavanje Nakon faze prethodnog učenja, model se može dodatno prilagoditi specifičnim zadacima kroz proces finog podešavanja . U ovoj fazi, model se uči na manjem broju primjeraka prilagođenim određenom zadatku ili domeni, često uz označene primjere. Fino podešavanje omogućuje modelu da se specijalizira, primjerice za slijeđenje uputa, odgovaranje na pitanja, generiranje koda ili vođenje dijaloga. U novijim LLM-ovima koriste se i napredne tehnike kao što su podešavanje prema uputama (engl. *instruction tuning*) [12] i podržano učenje uz ljudsku povratnu informaciju (engl. *reinforcement learning from human feedback*, RLHF) [13] kako bi se uskladili rezultati modela s ljudskim preferencijama i poboljšala pouzdanost.

2.4. Kontekstni prozor kod transformerskih modela

Suvremeni veliki jezični modeli obrađuju ulazni tekst u segmentima koji se nazivaju kontekstni prozori (*context window*). Kontekstni prozor označava fiksni broj tokena na koje model može obratiti pažnju u svakom trenutku. Na primjer, GPT-2 podržava do 1.024 tokena, dok najnoviji modeli poput GPT-4o mogu obraditi čak 128.000 tokena.

Ovo ograničenje proizlazi iz same arhitekture transformerskih modela, u kojoj samopozornost izračunava međusobne odnose između svih tokena unutar prozora [8]. Za sekvencu od n tokena, mehanizam sampozornosti računa vrijednost pozornosti za svaki mogući par tokena, što rezultira matricom pozornosti dimenzija $n \times n$. Zbog toga memorijski i računalni zahtjevi rastu kvadratno s veličinom kontekstnog prozora, pa nije moguće obraditi proizvoljno dugačke tekstove.

Osnovne strategije poput skraćivanja (zadržavanja samo najnovijih tokena) ili segmentacije (dijeljenja dužeg teksta u odvojene prozore) ponekad se koriste kao inženjerska rješenja, ali takvi pristupi ne mogu u potpunosti sačuvati globalni kontekst.

2.5. Implikacije za sustave agenata

Arhitektura velikih jezičnih modela jako utječe na mogućnosti i opcije oblikovanja koje su dostupne u sustavima inteligentnih agenata. Nekoliko ključnih komponenti ima posebno važnu ulogu:

Tokenizacija i *embedding* slojevi Robusna tokenizacija i učinkoviti *embedding* slojevi omogućuju agentima fleksibilno tumačenje raznolikih i suptilnih korisničkih poruka, čime se podupire semantičko razumijevanje i komunikacija u različitim situacijama.

Samopozornost Mehanizmi samopozornosti i pozornost s više glava za čitanje u transformeru agentima omogućuju dinamično usmjeravanje pažnje na relevantne informacije unutar kontekstnog prozora. Ova sposobnost je ključna za precizno donošenje odluka, korak po korak zaključivanje te za obradu složenih upita koji zahtijevaju agregaciju informacija iz više izvora.

Kontekstni prozor Veličina i način upravljanja kontekstni prozorom izravno utječu na sposobnost agenta za dugoročno planiranje, korak po korak razmišljanje i održavanje koherentnog dijaloga. Veći kontekstni prozori poboljšavaju agentovu sposobnost upravljanja opsežnim i kontekstom bogatim zadacima.

Autoregresivno modeliranje Autoregresivna priroda učenja LLM-ova prirodno podržava sekvencijalno zaključivanje i generiranje strukturiranih planova, što agentima omogućuje stvaranje koherentnih i kontekstni svjesnih odgovora i planova.

Arhitektura velikih jezičnih modela određuje ne samo njihove osnovne sposobnosti, već i utječe na njihove prednosti i ograničenja unutar sustava inteligentnih agenata. Ova perspektiva bit će temelj za detaljniju analizu zaključivanja, transparentnosti i praktične izgradnje agenata u narednim poglavljima.

3. Zaključivanje u velikim jezičnim modelima

Napretci u LLM-ovima [2] značajno su promijenili područje obrade prirodnog jezika, omogućujući ne samo generiranje teksta, već i sve složenije oblike zaključivanja. U ovom kontekstu, zaključivanje podrazumijeva sposobnost modela da izvede zaključke, slijedi logiku u više koraka, rješava probleme i pruži objašnjenja koja nadilaze puko pamćenje ili prepoznavanje površinskih obrazaca.

Jedan od ključnih uvida nedavnih istraživanja je da napredno zaključivanje u LLM-ovima ne proizlazi samo iz poboljšanja u arhitekturi, već ovisi o veličini modela. Tek kad modeli pređu određenu veličinu, iznenada se javljaju napredne sposobnosti zaključivanja poput sposobnost rješavanja problema kroz više koraka te učenje temeljem konteksta dok su one kod manjih modela obično slabo izražene ili ih uopće nema. [14].

Za razliku od klasičnih simboličkih AI sustava koji zaključivanje ostvaruju eksplicitnom manipulacijom logičkih pravila, LLM-ovi usvajaju implicitne strategije zaključivanja kroz izlaganje velikim skupovima tekstnih podataka. Korištenjem dubokih neuronskih arhitektura i mehanizama pozornosti, mogu povezivati činjenice, izvoditi dedukcije i prilagođavati se dvosmislenim ili nedovoljno specificiranim problemima, sve u okviru ograničenja nametnutih podacima za učenje i veličinom modela.

3.1. Strategije postavljanja upita za poticanje zaključivanja u velikim jezičnim modelima

Veliki jezični modeli sposobni su povezivati činjenice i logiku kako bi rješavali zahtjevne zadatke koji zahtijevaju zaključivanje u više koraka. Ipak, te sposobnosti često neće doći do izražaja bez pažljivo osmišljenih osmišljenih korisničkih upita (engl. i dalje *prompt*)

koji jasno potiču eksplicitno zaključivanje. Sve više istraživanja pokazuje da način na koji je prompt formuliran može jako utjecati na to hoće li model uspješno razložiti zadatak, pratiti potrebne međukorake te doći do ispravnog rješenja.

Promptanje putem lanca razmišljanja Jedna od najutjecajnijih strategija je promptanje putem lanca razmišljanja (engl. *chain of thought*, CoT) [15], kojom se modelu eksplicitno nalaže da prije davanja konačnog odgovora iznese međukorake zaključivanja. Primjerice, pogledajmo kako model odgovara na zadatak s brojevima ovisno o načinu promptanja.

Prompt

Sauron forges 20 rings of power. He gives 3 to the elves, 7 to the dwarves, and 9 to men. How many rings does Sauron keep for himself?

Odgovor modela

Sauron keeps 9 rings for himself.

Slika 3.1. Standardni prompt koji ne potiče objašnjavanje često rezultira netočnim odgovorom.

Standardni prompt često dovodi do brzog, ali netočnog odgovora (vidi sliku 3.1.).

Prompt s lancem razmišljanja

Sauron forges 20 rings of power. He gives 3 to the elves, 7 to the dwarves, and 9 to men. How many rings does Sauron keep for himself? **Explain your reasoning step by step.**

Odgovor modela

Sauron starts with 20 rings. He gives 3 to the elves, so he has $20 - 3 = 17$ rings left. He then gives 7 to the dwarves, leaving him with $17 - 7 = 10$ rings. Finally, he gives 9 to men, so $10 - 9 = 1$ ring remains. Therefore, Sauron keeps 1 ring for himself.

Slika 3.2. Prompt koji koristi lanac razmišljanja potiče model na izvođenje međukoraka, što rezultira točnim odgovorom.

No, kada se modelu izričito zatraži da obrazloži svoje zaključivanje korak po korak, znatno je vjerojatnije da će doći do točnog rješenja (vidi sliku 3.2.). Ovakav pristup posebno je učinkovit kod matematičkog zaključivanja, logičkih zadataka i problema koji zahtijevaju rješavanje kroz više međukoraka.

Napredne tehnike promptanja Osim pristupa razmišljanja putem lanca misli, razvijene su i brojne napredne strategije za dodatno unapređenje zaključivanja u LLM-ovima:

- **Tehnika samokonzistentnosti** (engl. *Self-consistency*): Umjesto oslanjanja na jedan niz zaključivanja, model generira više neovisnih rješenja te odabire najčešći ili najdosljedniji odgovor. Ova tehnika povećava robusnost i pouzdanost, osobito kod zadataka s kompleksnim putem zaključivanja [16].

- **Promptanje bilježenjem** (engl. *Scratchpad*): Modelu se daje „radni prostor" za bilježenje međurezultata ili izračuna, čime se potiče strukturirano višekoračno zaključivanje, posebno u matematičkim i znanstvenim područjima [17].

- **Naizmjenično zaključivanje i djelovanje** (engl. *Reason and Act*, ReAct): Ovaj pristup izmjenjuje eksplicitne korake zaključivanja i akcije, poput pretraživanja alata ili dohvaćanja činjenica. Model se prompta da naizmjenično „misli" i „djeluje", što poboljšava rezultate na zadacima koji uključuju vanjske alate i traženje informacija [18].

- **Promptanje stablom razmišljanja** (engl. *Tree-of-thought*): Umjesto jednog linearnog niza, model paralelno istražuje više smjerova zaključivanja („grana"), procjenjujući alternative i povremeno se vraćajući na prethodne izbore. Time se omogućuje temeljitije istraživanje mogućih rješenja [19].

- **Verifikacija i refleksija:** Prompti koji modelu nalažu da pregleda, kritički procijeni ili provjeri vlastite odgovore (ili odgovore drugih modela) mogu pomoći u otkrivanju pogrešaka i povećanju pouzdanosti [20].

- **Sokratsko ili dijaloško promptanje:** Postavljanjem pitanja u dijaloškom formatu model se vodi kroz složeno zaključivanje tako da problem raščlani na manja, lakše rješiva potpitanja.

Utjecaj i primjena promptanja Različite strategije promptanja, samostalno ili zajedno, omogućuju LLM-ovima bolje zaključivanje i pouzdanije rješavanje zadataka koji zahtijevaju logiku, planiranje ili objašnjenja. U sve većoj mjeri ove tehnike pronalaze mjesto u sustavima inteligentnih agenata, gdje modeli moraju rješavati nove izazove, po-

vezivati informacije iz različitih izvora ili opravdavati svoje odluke. S razvojem područja, osmišljavanje dobrih promptova ostaje središnja metoda za iskorištavanje i tumačenje zaključivačkih mogućnosti velikih jezičnih modela.

3.2. Upravljanje kontekstom

Kao što je opisano u poglavlju 2.4., LLM-ovi temeljeni na transformerskoj arhitekturi rade s kontekstnim prozorom fiksne veličine, što ograničava pristup informacijama izvan određenog broja tokena. Iako je to dovoljno za mnoge primjene, takav pristup stvara izazove kod zadataka koji zahtijevaju zaključivanje na temelju duljih dokumenata, višestrukih uzastopnih interakcija ili integraciju činjenica iz udaljenih dijelova teksta.

Primjerice, ako ulazna sekvenca premaši veličinu kontekstnog prozora modela, ranije informacije mogu biti izostavljene, što može dovesti do toga da model „zaboravi" ključne detalje ili izgubi praćenje entiteta, logike ili toka razgovora. To otežava zaključivanje temeljeno na povezivanju udaljenih informacija, sažimanje i uspješno izvršavanje složenijih zadataka.

Kako bi se riješili ovi izazovi, razvijena su različita rješenja:

- **Segmentacija ili klizni prozor**: Tekst se dijeli na preklapajuće segmente, od kojih se svaki obrađuje neovisno. Ovakav pristup može djelomično očuvati kontekst, ali globalna koherentnost i dalje ostaje izazov.
- **Dohvatom poboljšano generiranje** (engl. *Retrieval-Augmented Models*, RAG): Modeli poput RAG-a [21] proširuju ulaz relevantnim vanjskim dokumentima ili memorijom, čime se efektivno povećava radni kontekst modela.
- **Arhitekture s poboljšanom memorijom ili mehanizmom pozornosti:** Varijante poput Transformer XL [22] i Longformer [23] omogućuju obradu znatno dužih sekvenci ponovnim korištenjem skrivenih stanja ili prilagodbom mehanizma pozornosti.

Unatoč ovim napretcima, pouzdano zaključivanje nad udaljenim informacijama i dalje ostaje otvoreni problem, što dodatno potiče istraživanja u području upravljanja kontekstom, integracije memorije i skalabilnih arhitektura.

3.3. Ograničenja i otvoreni izazovi

Unatoč impresivnom napretku, veliki jezični modeli i dalje pokazuju značajna ograničenja u području zaključivanja i pouzdanosti.

Halucinacije i nepouzdano zaključivanje LLM-ovi su skloni halucinacijama: generiranju tečnih i uvjerljivih izjava koje su zapravo netočne ili potpuno neutemeljene [24]. To može uključivati izmišljene reference, pogrešno predstavljene činjenice ili lažne logičke poveznice. Halucinacije nastaju zato što model predviđa tekst na temelju statističkih obrazaca iz podataka za učenje, bez eksplicitnih mehanizama za provjeru istinitosti.

Ograničenja u dubokom zaključivanju, matematici i zdravom razumu Iako su LLM-ovi vrlo uspješni u razumijevanju prirodnog jezika, često imaju poteškoće sa zadacima koji zahtijevaju duboko logičko zaključivanje, precizne matematičke izračune ili stabilan zdrav razum. Na primjer, model može pokušati odigrati nedozvoljeni potez u šahu, pogriješiti u višekoračnoj aritmetici ili dati nedosljedne odgovore na slična pitanja. Takvi neuspjesi ukazuju na to da se modeli više oslanjaju na prepoznavanje obrazaca nego na istinsko simboličko ili logičko zaključivanje.

Poznati primjeri neuspjeha Dodatni izazovi uključuju poteškoće u zadržavanju informacija kroz duže tekstove [25], davanje krhkih ili zavaravajućih objašnjenja te pokazivanje pretjeranog samopouzdanja čak i kad su modeli nesigurni.

Zasićenje skaliranjem Iako su mnogi napretci u zaključivanju LLM-ova rezultat povećanja veličine modela i količine podataka za učenje, novija istraživanja upućuju na to da samo povećavanje broja parametara i računalnih resursa donosi sve manje koristi, osobito u pogledu pouzdanosti i dubljeg zaključivanja [14]. To sugerira da je za daljnji napredak potrebna inovacija u arhitekturi, bolji ciljevi učenja ili kvalitetnija integracija s vanjskim alatima i mehanizmima provjere, umjesto samog povećavanja modela.

Danas se intenzivno istražuju rješenja poput dohvatom poboljšanog generiranja, poboljšanih metoda procjene pouzdanosti i novih arhitektura za rad s proširenim kontekstom. Međutim, potpuno robusno i pouzdano zaključivanje u LLM-ovima još uvijek ostaje otvoreni izazov.

4. Pozdanost velikih jezičnih modela

Kako se veliki jezični modeli sve češće primjenjuju u visokorizičnim područjima gdje su odluke ključne, njihova pozdanost postaje od presudne važnosti [26, 27]. Povjerenje u LLM-ove temelji se na nekoliko ključnih svojstava:

- **Transparentnost**: koliko je lako ispitati strukturu modela i proces donošenja odluka,
- **Objašnjivost**: sposobnost pružanja jasnih, ljudima razumljivih razloga za dobivene odgovore,
- **Pristranost**: sustavna odstupanja uzrokovana neravnotežom u podacima za učenje,
- **Pravednost**: nastojanje da se izbjegne pojačavanje ili reproduciranje društvenih nejednakosti [28],
- **Povjerenje**: dosljednost i robustnost modela, osobito u neizvjesnim ili novim situacijama (vidi 3.3.).

Ova su svojstva međusobno duboko povezana: transparentnost i objašnjivost olakšavaju otkrivanje problema s pristranošću i pouzdanošću, dok pravednost i robusne performanse ovise o razumijevanju i praćenju ponašanja modela.

4.1. Transparentnost i objašnjivost

Transparentnost označava koliko je moguće pregledati i razumjeti unutarnju strukturu, parametre i podatke za učenje nekog jezičnog modela. Objašnjivost je sposobnost modela

da pruži jasna, ljudima razumljiva obrazloženja svojih odgovora ili ponašanja [26]. Dok transparentnost "otvara crnu kutiju", objašnjivost omogućuje da se unutarnji mehanizmi modela prevedu u razloge koji imaju smisla korisnicima. Obje ove osobine ključne su za povjerenje, nadzor i odgovornu primjenu LLM-ova.

Prepreke objašnjivosti Zbog iznimne složenosti LLM-ova, koji često sadrže milijarde parametara i učeni su na ogromnim, često netransparentnim skupovima podataka, ovi modeli su po svojoj prirodi teško objašnjivi [27]. Odluke modela zapisane su u visokodimenzionalnim, distribuiranim reprezentacijama, pa rijetko postoji izravna veza između pojedinih ulaza, unutarnjih stanja i izlaza. Takav nedostatak transparentnosti može prikriti skrivene pristranosti ili potencijalne probleme u radu modela, te ograničiti korisnicima mogućnost povjerenja u rezultate ili otklanjanja pogrešaka.

Tehnike za objašnjivost Istraživači koriste različite metode kako bi poboljšali objašnjivost LLM-ova. Vizualizacija mehanizma pozornosti (engl. *Attention visualization*) prikazuje koji ulazni tokeni najviše utječu na određeni izlaz kod transformerskih modela, iako pozornost ne odražava uvijek stvarne procese zaključivanja [29]. Metode atribucije zasluga, poput integriranih gradijenata (engl. *Integrated Gradients*) dijele "zasluge" za izlaz pojedinim ulaznim značajkama, što pomaže u otkrivanju najutjecajnijih dijelova prompta [30]. Sondiranje (engl. *Probing*) uključuje učenje jednostavnih klasifikatora na aktivacijama modela radi otkrivanja koje se vrste informacija prenose kroz različite slojeve [31]. Svaka od ovih metoda pruža djelomičan uvid, no nijedna još uvijek ne može u potpunosti objasniti odluke modela na način razumljiv ljudima.

Objašnjivost u praksi U stvarnim primjenama objašnjivost se često poboljšava eksplicitnim koracima zaključivanja ili naknadnim obrazloženjima. Na primjer, promptanje putem lanca razmišljanja, opisano u poglavlju 3.1., potiče modele da generiraju međukorake zaključivanja, čime njihov postupak postaje transparentniji i lakše provjerljiv. Ipak, i dalje se raspravlja o tome predstavljaju li takva objašnjenja stvarno vjernu sliku unutarnje logike modela ili su, što se čini vjerojatnijim, tek uvjerljiva, ali površna obrazloženja. [32].

4.2. Pristranost i pravednost

Pristranost u LLM-ovima odnosi se na sustavna odstupanja u izlazima modela koja proizlaze iz neravnoteža, praznina ili iskrivljenih distribucija u podacima za učenje [28].

Izvori pristranosti Glavni izvor pristranosti u LLM-ovima su golemi i heterogeni skupovi podataka korišteni za učenje. Kada su određene domene, teme ili jezični stilovi prekomjerno zastupljeni u korpusu, model će te obrasce vjerojatnije reproducirati, dok se manje zastupljene teme mogu zanemariti ili netočno interpretirati [27]. Pristranost može nastati i zbog načina uzorkovanja podataka, pogrešaka u anotacijama ili ciljeva optimizacije koji nenamjerno favoriziraju popularan ili učestalo prisutan sadržaj.

Primjeri pristranosti u velikim jezičnim modelima LLM-ovi često pokazuju pristranost prema učestalim ili „sigurnim" odgovorima, oslanjajući se na generičke obrasce iz podataka za učenje, čak i kada su precizniji ili kontekstno primjereniji odgovori dostupni. To može dovesti do zanemarivanja rijetkih, dvosmislenih ili rubnih slučajeva. Na primjer, model koji generira SQL upite može sustavno proizvoditi opće „SELECT *" upite ako takvi oblici dominiraju u korpusu, iako bi u određenom kontekstu bili prikladniji ciljani i učinkovitiji izrazi.

Mjerenje i dijagnosticiranje pristranosti Pristranost prema sigurnim odgovorima obično se otkriva ispitivanjem modela na rijetkim temama, dvosmislenim upitima ili netipičnim situacijama. Analiza raznolikosti i specifičnosti izlaza modela otkriva favorizira li model sustavno uobičajene odgovore na račun točnijih, ali rjeđih opcija [33].

Strategije ublažavanja pristranosti Pristranost se može smanjiti upotrebom pažljivo oblikovanih promptova. Izričite upute poput „budi precizan", „razmotri rijetke slučajeve" ili „izbjegavaj generičke odgovore" mogu potaknuti model da generira konkretnije i manje uobičajene odgovore. Osim toga, pristranost se može ublažiti i kroz selekciju i uravnoteženje podataka te tehnikama učenja modela.

4.3. Kompromisi i međuovisnosti

Pouzdanost LLM-ova ne proizlazi iz skupine neovisnih svojstava, već iz složene međusobne povezanosti transparentnosti, objašnjivosti, pristranosti, pravednosti i povjerenja [32, 26]. Poboljšanja u jednom području često utječu na ostala, ponekad i na neočekivane načine.

Na primjer, nastojanja da se modeli učine transparentnijima, otkrivanjem njihovih unutarnjih mehanizama ili davanjem detaljnih objašnjenja, često otkrivaju prijašnje skrivene pristranosti, što zatim potiče razvoj novih strategija za njihovo ublažavanje. Ipak, davanje prednosti transparentnosti ili objašnjivosti može dovesti do pojednostavljenja modela ili ograničenja njegove složenosti, čime se može umanjiti njegova ukupna učinkovitost ili sposobnost generalizacije. S druge strane, optimizacija modela za maksimalnu točnost predviđanja ili robusnost može rezultirati manje objašnjivim ili manje pravednim ponašanjem, jer model uči iskorištavati obrasce u podacima koji nisu lako objašnjivi ni opravdani.

Intervencije radi postizanja pravednosti, poput uravnoteživanja podataka ili ciljane redukcije pristranosti, mogu smanjiti određene vrste pristranosti, ali istovremeno uvesti nove nedosljednosti ili nepredvidivosti u odgovore modela, posebno na rijetke ili dvosmislene ulaze. Slično tome, pokušaji povećanja pouzdanosti kalibriranjem izlaza ili filtriranjem nesigurnih odgovora mogu nenamjerno prikriti sustavne pogreške ili zanemariti važne rubne slučajeve.

Zbog ovih međuovisnosti, razvoj vjerodostojnih LLM-ova zahtijeva ne samo tehnička rješenja za svako pojedino svojstvo, već i pažljivu procjenu kako promjene u jednoj dimenziji utječu na druge.

5. Veliki jezični modeli kao inteligentni agenti

Veliki jezični modeli više nisu ograničeni na pasivno generiranje teksta, već se sve češće primjenjuju kao autonomni inteligentni agenti. U toj ulozi, LLM-ovi komuniciraju s dinamičnim okruženjima, donose odluke, planiraju radnje od više koraka i prilagođavaju svoje ponašanje ovisno o kontekstu i povratnim informacijama.

5.1. Komunikacija agenata

U sustavima agenata temeljenima na LLM-ovima, komunikacija se provodi kroz razmjenu poruka. Svaka poruka prenosi određenu ulogu ili namjeru, a ukupno ponašanje sustava proizlazi iz strukturirane međudjelatnosti tih poruka.

Vrste poruka Tipična komunikacija među agentima oslanja se na nekoliko jasno definiranih vrsta poruka:

- **Poruke korisnika**: Upiti ili upute krajnjeg korisnika.
- **Poruke razvojnog okruženja (sustava)**: Promptovi i konfiguracijski detalji koje osigurava razvojni inženjer; nevidljive su korisniku, ali ključne za usmjeravanje ponašanja agenta. U to spadaju početni sustavni prompt (koji LLM uvijek prvi prima), upute za stil odgovora i pravila ponašanja agenta
- **Poruke alata/funkcija**: Zahtjevi prema vanjskim alatima ili API-jima i odgovori iz njih, obično u strukturiranom formatu.
- **Poruke asistenta**: Izlazi LLM-a, uključujući odgovore u prirodnom jeziku ili strukturirane odgovore.

Strukturirani izlaz Strukturirani izlaz odnosi se na odgovore LLM-a koji slijede unaprijed definiran, strojno čitljiv format, poput JSON-a, parova ključ–vrijednost ili specifičnih jezika kao što je SQL. Za razliku od slobodnog teksta, takav izlaz može se izravno analizirati i koristiti u drugim programskim komponentama.

U sustavima agenata temeljenima na LLM-ovima, strukturirani izlaz ključan je za integraciju s vanjskim alatima, automatizaciju procesa i smanjenje dvosmislenosti u komunikaciji. Kada se od agenta zatraži generiranje strukturiranog izlaza, primjerice SQL upita ili poziva API-ja, time se nastoji osigurati[1] da odgovor odgovara jasno definiranom formatu. To omogućuje pouzdanije parsiranje, rukovanje pogreškama i obradu u daljnjim koracima, te olakšava robusnu interakciju između LLM-a i drugih modula.

Zahtijevanje strukturiranog izlaza pomaže agentima da preciznije izvršavaju složene zadatke te omogućuje automatiziranu evaluaciju njihovih odgovora.

5.2. Korištenje alata

Jedna od ključnih mogućnosti suvremenih agenata temeljenih na velikim jezičnim modelima je interakcija s vanjskim alatima, API-jima i servisima, čime se proširuju njihove izvorne sposobnosti zaključivanja i pristupa informacijama. Korištenjem vanjskih alata agent može dohvatiti podatke, izvoditi izračune, slati naredbe ili pristupati specijaliziranim funkcionalnostima koje nadilaze ono što je usvojio tijekom učenja.

Svijest o alatima i njihov odabir Kako bi agent mogao koristiti vanjske alate, potrebno je LLM-u unaprijed dostaviti informacije o tome koji su alati dostupni, čemu služe i na koji se način pozivaju. To se najčešće postiže putem eksplicitnih poruka sustava ili razvojnog okruženja koje definiraju popis alata i uključuju upute za korištenje unutar samog prompta.

Iako LLM-ovi nisu izvorno učeni za korištenje vanjskih alata, mogu se na to usmjeriti odgovarajućim oblikovanjem prompta. Agent unutar konteksta svake interakcije stavlja popis dostupnih alata, uključujući njihova imena, ulazne parametre i očekivani format

[1] U praksi, strogo pridržavanje zadanom formatu nije zajamčeno jer su LLM-ovi probabilistički modeli i ponekad mogu proizvesti neispravan ili nepotpun izlaz. Često su potrebni dodatni mehanizmi za parsiranje, validaciju i ispravljanje pogrešaka.

izlaza zajedno s preciznim uputama i primjerima korištenja. Takav pristup potiče model da, kada je to primjereno, generira pozive alata u ispravnom formatu. Napredniji sustavi dodatno unapređuju ovo ponašanje finim podešavanjem LLM-ova na skupovima podataka koji uključuju primjere pozivanja alata.

Proces pozivanja alata Kada LLM procijeni da je potrebno koristiti neki alat, on sam ne izvršava nikakvu radnju. Umjesto toga, generira strukturiranu poruku koja opisuje željeni poziv funkcije ili alata, primjerice zahtjev prema API-ju ili upit prema bazi podataka. Sam agent je odgovoran za prepoznavanje takve poruke, izvođenje tražene funkcije u vanjskom okruženju i vraćanje rezultata natrag LLM-u kao novu poruku. Agent može više puta ponavljati ovaj postupak, kombinirajući rezultate poziva alata kako bi razradio svoje zaključke i formulirao odgovor.

5.3. Arhitektura agenta

Suvremeni sustavi agenata temeljeni na velikim jezičnim modelima organizirani su kao modularne arhitekture koje koordiniraju komunikaciju, zaključivanje, korištenje alata i upravljanje kontekstom. Njihov rad obično slijedi iterativni ciklus: primanje ulaza, planiranje ili zaključivanje uz pomoć LLM-a, generiranje poziva alata ili odgovora, obrada rezultata i ažuriranje konteksta ili memorije za sljedeći korak.

Tijek rada agenta Uobičajeni tijek rada agenta odvija se kroz sljedeće korake:

1. Zaprimanje unosa (korisnička poruka, uputa sustavu ili rezultat alata)
2. Konstrukcija prompta za LLM i obrada njegovog izlaza
3. Ako je izlaz poziv alata, izvršavanje traženog alata i vraćanje rezultata LLM-u
4. Ako je izlaz poruka asistenta, vraćanje odgovora korisniku ili sljedećoj komponenti sustava
5. Ažuriranje internog stanja, konteksta ili memorije agenta prema potrebi
6. Ponavljanje procesa za daljnje unose ili radnje

Arhitekturni obrasci U organizaciji rada LLM-agenta pojavilo se nekoliko uobičajenih obrazaca:

- **Planiranje i izvođenje** (engl. *Plan and Execute*): Agent najprije generira plan od više koraka (npr. popis radnji), a zatim ga provodi, pozivajući alate ili generirajući odgovore prema potrebi [34].
- **Naizmjenično zaključivanje i djelovanje** (engl. *Reason and Act*, ReAct): Agent ne planira sve unaprijed, već se kreće kroz petlju u kojoj najprije razmišlja, zatim djeluje. Ova strategija omogućuje prilagodbu na temelju trenutnih rezultata, što je korisno u dinamičnim situacijama. [18].
- **Refleksija** (engl. *Reflection*): Agent ulazi u cikluse samoprocjene, analizirajući vlastite izlaze, prepoznajući pogreške ili neizvjesnosti te ih ispravlja kroz ponovljeno zaključivanje ili korištenje alata [20].

Integracija komponenti Ove arhitekture omogućuju koordinaciju obrade poruka, strukturiranih izlaza, korištenja alata i upravljanja kontekstom u svrhu podrške složenog ponašanja agenata. U praksi, uspješni agenti često kombiniraju elemente planiranja, zaključivanja i refleksije kako bi mogli učinkovito reagirati na raznolike i nepredvidive zadatke.

6. Implementacije agenata temeljenih na velikim jezičnim modelima

6.1. Načela transparentnog oblikovanja agenata

Veliki jezični modeli danas su među najsposobnijim alatima za zaključivanje, no često se kritiziraju zbog svoje "crne kutije", složenosti, netransparentnosti i teškoća u ispravljanju ili nadzoru. Takva netransparentnost može otežati ne samo povjerenje i prihvaćanje modela, već i otkrivanje pogrešaka te poboljšanje performansi sustava.

Jedna od ključnih strategija za prevladavanje tih izazova jest oblikovanje agenata čiji su procesi zaključivanja i donošenja odluka transparentni i mogu se pratiti u svakoj fazi rada. Umjesto postavljanja jednog složenog problema modelu s nadom da će ponuditi točan odgovor, zadatak se razlaže u niz manjih, usmjerenih podzadataka, prilagođenih tako da ih model može uspješno riješiti. Ova modularizacija donosi nekoliko važnih prednosti:

- **Kontrola konteksta:** Svaka komponenta agenta prima samo relevantne ulazne podatke i kontekst koji su joj potrebni, čime se smanjuje rizik od distrakcije ili preopterećenja informacijama.
- **Strukturirano rezoniranje:** Korištenjem strukturiranih izlaza (npr. planova, poziva alata, SQL upita) olakšava se parsiranje, provjera i tumačenje međukoraka u rezoniranju modela.
- **Integracija alata:** Omogućavanjem pristupa vanjskim alatima ili API-jima (poput baza podataka ili internetskog pretraživanja), agent može temeljiti svoje zaključke na ažurnim informacijama, čime proširuje svoje sposobnosti izvan granica skupa

podataka za učenje.

- **Transparentnost i mogućnost revizije:** U svakoj fazi rada moguće je precizno vidjeti koje je informacije agent primio, koje je odluke donio, koje je alate koristio i kakav je izlaz generirao. Time se omogućuje potpuna revizija lanca zaključivanja, pri čemu se pogreške ili neočekivana ponašanja mogu povezati s konkretnim odlukama ili odgovorima modela.

Modularnim i transparentnim oblikovanjem udaljavamo se od koncepta jezičnih modela kao nedokučivih sustava, te umjesto toga gradimo agente čije zaključivanje postaje razumljivo i nadzirano. Svaki podzadatak, od ekstrakcije informacija do validacije odgovora, strukturiran je tako da ne samo poboljša ukupnu točnost, već i omogući uvid u tok zaključivanja. Takva arhitektura omogućuje praćenje i dijagnostiku pojedinačnih odluka koje model donosi, čime se LLM sustav pretvara iz crne kutije u predvidljiv i pouzdan mehanizam zaključivanja [35].

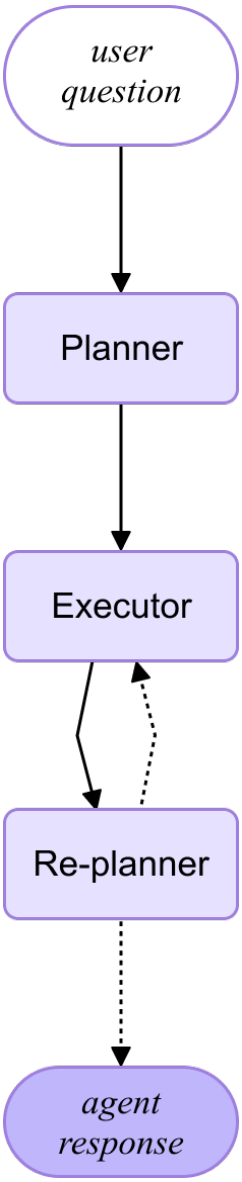
Sljedeći odjeljci prikazuju konkretna ostvarenja ovih načela u praksi. Kroz pažljivo strukturirane primjere prikazano je kako se transparentnost, objašnjivost i postupno zaključivanje mogu ostvariti u agentskim sustavima te kako te osobine pridonose pouzdanom radu i smislenoj analizi složenih zadataka.

6.2. Agent za planiranje i izvođenje (*Plan and Execute*)

Kako bi se prikazalo zaključivanje i transparentno donošenje odluke korak po korak, prvi primjer fokusira se na agenta koji sustavno dijeli korisnički cilj na pojedinačne korake i rješava ih redom. Agent prima korisnički cilj, formulira jednostavan plan razdvajanjem cilja na jasno definirane korake, a zatim ih izvršava jedan po jedan. Rezultat svakog koraka prenosi se u sljedeći, što agentu omogućuje izgradnju konteksta i prilagodbu zaključivanja u procesu postizanja konačnog odgovora.

6.2.1. Oblikovanje agenta i tijek rada

Arhitektura ovog agenta sastoji se od tri glavne komponente: *Planner*, *Executor* i *Replanner*.



Slika 6.1. Tijek rada agenta za planiranje i izvođenje

Planner Planner inicijalno oblikuje plan tako da korisnički cilj razlaže na najmanji mogući broj logički povezanih i smislenih koraka. Svaki korak mora biti svrhovit i doprinositi ostvarenju ukupnog zadatka.

Executor Executor izvršava svaki korak plana redom, provodeći potrebne radnje i dohvaćajući informacije prema potrebi. Koristi pristup u stilu ReAct, što znači da može pozivati vanjske alate poput internetskog pretraživanja te uključivati dohvaćene rezultate u vlastito zaključivanje. Nakon što izvrši pojedini korak, bilježi njegov ishod i prenosi ga dalje kao kontekst za sljedeće korake.

Re-planner Re-planner analizira rezultate izvršenih koraka i preostali dio plana. Na temelju te analize odlučuje jesu li potrebne dodatne akcije ili je agent spreman generirati konačan odgovor. Ako su potrebni daljnji koraci, ažurira plan i vraća kontrolu natrag *Executoru*. Ako je cilj postignut, generira završni odgovor i zaključuje tijek rada. Sva objašnjenja odluka i izmjena u planu bilježe se radi transparentnosti.

Iako ove komponente imaju različite uloge, sve koriste istu instancu jezičnog modela. Razlika među njima proizlazi iz specifičnih predložaka promptova, kontekstnih poruka i dostupnih alata koji se primjenjuju u svakoj fazi. Ovakav modularni pristup omogućuje dosljedno zaključivanje, smanjuje složenost implementacije i podržava fleksibilno ponašanje putem inženjeringa prompta.

6.2.2. Primjer zadatka i tijeka zaključivanja agenta

Kako bismo prikazali transparentan način zaključivanja i postupni rad agenta, uzimamo sljedeće korisničko pitanje (vidi sliku 6.2.).

Korisnički upit

What is the population of the hometown of the winner of the men's singles at the 2025 Australian Open?

Slika 6.2. Korisnički upit koji zahtijeva pristup aktualnim podacima i više koraka zaključivanja.

Ovo pitanje je odabrano jer se ne može riješiti isključivo temeljem podataka dostupnih jezičnom modelu tijekom učenja. Budući da je GPT-4.1 učen na podacima dostupnima do lipnja 2024., ne posjeduje informacije o turniru iz 2025. godine. Zbog toga agent mora koristiti alate za pretraživanje interneta kako bi u svakom koraku prikupio aktualne podatke, što mu omogućuje donošenje informiranih odluka. Time se demonstrira kako agent kombinira vanjske izvore s kontekstnim zaključivanjem kroz niz povezanih koraka pri rješavanju složenih upita.

Kako bi učinkovito pristupio rješavanju upita, Planner najprije generira plan koji definira potrebne korake i obrazlaže pristup koji će koristiti (vidi sliku 6.3.).

Planerov odgovor

Objective: What is the population of the hometown of the winner of the men's singles at the 2025 Australian Open?

Plan:

- Determine the winner of the men's 2025 Australian Open tennis tournament.
- Identify the hometown of the winner.
- Find the current population of the winner's hometown.

Rationale: The user wants to know the population of the hometown of the men's 2025 Australian Open winner. To answer this, we need to identify the winner, their hometown, and then find the current population of that place.

Slika 6.3. Odgovor Planera koji generira korake potrebne za rješavanje kompleksnog korisničkog upita.

Executor se usredotočuje isključivo na trenutačni korak plana, koristeći poziv alata za pretraživanje interneta kako bi pribavio potrebne informacije prije nego što generira svoj odgovor (vidi sliku 6.4.).

Executorov odgovor

Task: Determine the winner of the men's 2025 Australian Open tennis tournament.

Result: The winner of the men's 2025 Australian Open tennis tournament is Jannik Sinner.

Slika 6.4. Odgovor Executora temeljen na pretraživanju informacija za prvi korak plana.

Nakon što sazna da je pobjednik Jannik Sinner, agent koristi postojeće znanje modela o poznatim sportašima. U ovom slučaju, model prepoznaje rodni grad Jannika Sinnera bez potrebe za dodatnim pretraživanjem. Kao rezultat toga, tijekom ponovnog planiranja agent inteligentno preskače korak "Identify the hometown of the men's 2025 Australian Open winner." i ažurira plan u skladu s tim (vidi sliku 6.5.).

Re-plannerov odgovor

Objective: What is the population of the hometown of the winner of the men's singles at the 2025 Australian Open?

Plan:

- Determine the current population of Sexten, Italy (Jannik Sinner's hometown)

Rationale: Now that the winner and his hometown have been determined, the next necessary step is to find the population of Jannik Sinner's hometown, Sexten, Italy.

Slika 6.5. Re-planiranje koje preskače poznate informacije i fokusira se na sljedeći nepoznati korak.

Executor zatim koristi internetski alat kako bi pretražio „current population of Sexten (Sesto), Italy" te dobiva relevantan odgovor (vidi sliku 6.6.).

Executorov odgovor

Task: Determine the current population of Sexten, Italy (Jannik Sinner's hometown).

Result: The population of Sexten (Sesto), Italy is approximately 1,822 people, based on preliminary 2025 data.

Slika 6.6. Odgovor Executora na zadatak pronalaženja broja stanovnika Sextena.

Prikupljanjem svih nužnih informacija, Re-planner zaključuje da je vrijeme za konačni odgovor korisniku (vidi sliku 6.7.).

Odgovor korisniku

The population of Sexten (Sesto), Italy—the hometown of Jannik Sinner, the winner of the men's singles at the 2025 Australian Open—is approximately 1,822 people, based on preliminary 2025 data.

Slika 6.7. Konačan odgovor korisniku na temelju svih prikupljenih informacija.

Ovaj primjer prikazuje kako agent dinamički integrira vanjske informacije, prilagođava svoj plan i pruža transparentan, objašnjiv odgovor.

6.2.3. Transparentnost, objašnjivost i ograničenja

Gornji primjer pokazuje kako pristup agenta temeljenog na planiranju i izvođenju omogućuje transparentnost i objašnjivost u praksi. Na svakom pojedinom koraku moguće je točno pregledati što je predano jezičnom modelu i kakav je rezultat generiran. Takva

razina uvida olakšava prepoznavanje trenutka kada agentovo rasuđivanje ne uspije ili dovede do pogrešnih zaključaka. Iako ovako detaljno praćenje može djelovati pretjerano kod jednostavnih ili manjih agenata, ono postaje izuzetno vrijedno pri razvoju ili otklanjanju grešaka u većim, složenijim sustavima.

Iako se pristup planiranja i izvođenja pokazuje vrlo učinkovitim za zadatke koji se mogu podijeliti na sekvencijalne i jasno definirane korake, primjerice pretraživanje činjenica, on nije optimalan za složenije zadatke koji uključuju kreativno rasuđivanje, integraciju širokog znanja ili otvorene probleme. U takvim slučajevima, ovaj pristup često zahtijeva dodatne mehanizme i dublje promišljeno oblikovanje.

6.3. Agent za pretvorbu teksta u SQL

Zadatak pretvorbe teksta u SQL može se jednostavno opisati: uz zadanu shemu baze podataka i upit izražen prirodnim jezikom, cilj je generirati SQL upit koji će iz baze dohvatiti točan odgovor. Na prvi pogled, ovo može djelovati trivijalno, najnaivniji pristup bio bi jednostavno proslijediti cijelu shemu i upit LLM-u i nadati se ispravnom SQL odgovoru.

Cilj ovog primjera nije izgradnja potpunog agenta za pretvorbu teksta u SQL, već prikaz kako razlaganje zadatka na manje korake povećava točnost i objašnjivost. Umjesto naivnog pristupa s jednim upitom, modularno oblikovanje, inspirirano rješenjima poput CHESS [36] i CHASE-SQL [37], omogućuje bolju kontrolu, uvid u rad sustava te lakše otkrivanje pogrešaka. Riječ je o složenim agentskim arhitekturama koje predstavljaju vrhunske sustave iz stvarnog svijeta te se ubrajaju među najbolje rangirane pristupe na evaluacijama poput BIRD Bench (Big-Bench for Large-Scale Database Grounded Text-to-SQL). Unatoč svojoj snazi i složenosti, ovi sustavi pokazuju da se i najsuvremeniji pristupi mogu razložiti na jednostavne, transparentne komponente, čime se olakšava njihovo razumijevanje, testiranje i prilagodba.

Primjeri i djelomična implementacija u nastavku pokazuju kako svaki modul agenta doprinosi boljoj vidljivosti procesa rada LLM-a. Razlaganjem zadatka, bilježenjem međukoraka i provjerom rezultata, modularni pristup povećava transparentnost i pouzdanost sustava.

6.3.1. Oblikovanje agenta i tijek rada

Agent za pretvorbu teksta u SQL strukturiran je kao niz modularnih komponenti, od kojih svaka ima jasno definiranu ulogu u postupnoj transformaciji upita na prirodnom jeziku u formalni SQL upit:

- **Filtar sheme:** Analizira upit i identificira tablice i stupce baze podataka koji su najrelevantniji za njegovo rješavanje.
- **Generator kandidata:** Na temelju korisničkog upita i filtrirane sheme generira više potencijalnih SQL upita koji bi mogli odgovoriti na postavljeno pitanje.
- **Evaluator upita:** Procjenjuje generirane upite korištenjem ispita formuliranih u prirodnom jeziku te odabire onaj koji najbolje odgovara korisničkoj namjeri.

Ovakva modularna arhitektura omogućuje transparentnost svakog koraka u procesu, čime se olakšava praćenje i interpretacija ponašanja agenta tijekom generiranja odgovora.

Filtar sheme

Jedan od najvećih praktičnih izazova u zadacima pretvorbe teksta u SQL je sama veličina i složenost stvarnih baza podataka. Industrijske baze često sadrže desetke, pa i stotine tablica s brojnim stupcima, što rezultira s tisućama elemenata sheme i potencijalno desecima tisuća ulaznih tokena. Prosljeđivanje tako opsežne, nefiltrirane sheme jezičnom modelu ne samo da preopterećuje njegovu pažnju, već ujedno otežava rasuđivanje i gotovo onemogućuje ispravno otklanjanje pogrešaka za korisnika.

Filtar sheme rješava ovaj problem kroz višestupanjski proces kojim se shema baze agresivno reducira na uski podskup koji sadrži samo one tablice i stupce koji su relevantni za korisnički upit. Ovakva modularna redukcija ključna je za ostvarivanje izvedivosti zadatka s jezičnim modelima, jer njihovu pažnju usmjerava isključivo na nužne informacije.

Na svakoj razini, odluke i obrazloženja se bilježe, a filtrirana shema reprezentira se u konzistentnom, strojno čitljivom JSON formatu. Kako se tablice i stupci izbacuju, svi odnosi na izbačene elemente (primjerice, veze vanjskih ključeva) također se ažuriraju ili

brišu, čime shema ostaje koherentna i samostalna. Primjer takve strukturirane reprezentacije prikazan je na slici 6.8.

Reprezentacija sheme baze podataka

```
{
  "table": "employee",
  "description": "Table of all employees at the company",
  "columns": [
    {
      "column": "employee_id",
      "data_type": "INTEGER",
      "primary_key": true,
      "description": "Unique identifier for each employee",
      "foreign_keys": [
        {
          "referenced_table": "department",
          "referenced_column": "id"
        }
      ],
      "referenced_by": [],
      "enum_values": []
    },
    ...
  ]
}
```

Slika 6.8. JSON reprezentacija sheme baze podataka za tablicu `employee`.

Prednost ovakvog sekvencijskog i modularnog pristupa jest ta što omogućuje primjenu različitih jezičnih modela na različite faze zadatka. Lakši i brži modeli zaduženi su za filtriranje, dok se najjači modeli koriste za završne korake, što doprinosi efikasnosti i većoj preglednosti procesa.

Korak 1: Filtriranje stupaca. Proces započinje procjenom relevantnosti svakog stupca u odnosu na korisnički upit (za primjer vidi sliku 6.9.). Metapodaci i opis svakog stupca povezuju se s upitom te se evaluiraju pomoću manjih jezičnih modela kao jednostavan

klasifikacijski zadatak. Ovakvo filtriranje je izrazito paralelizirano i prikladno za učinkovite modele poput GPT-4.1-mini ili 4.1-nano, čime se smanjuju i troškovi i vrijeme odaziva. Stupci koji sadrže primarne ili strane ključeve uvijek se zadržavaju kako bi se spriječilo da jednostavniji modeli, koji su skloniji pogreškama, ne izbace ključne dijelove tablica. Njihova je uloga ograničena na filtriranje samo onih stupaca za koje smo sigurni da nisu potrebni za daljnje izvođenje upita.

Primjer dnevnika modela koji filtrira stupce

```
Table job, column id is relevant: False, reasoning: ...
Table job, column name is relevant: True, reasoning: ...
Table job, column created_at is relevant: False, reasoning: ...
...
```

Slika 6.9. Izlaz modela koji procjenjuje relevantnost pojedinih stupaca tablice u odnosu na korisnički upit.

Korak 2: Odabir tablica. Nakon što su nebitni stupci uklonjeni, napredniji jezični model pregledava korisnički upit i reduciranu shemu te odabire samo one tablice koje su doista potrebne za odgovor na upit. Ovim korakom dodatno se precizira kontekst i uklanja nepotrebni šum.

Korak 3: Dodatna selekcija stupaca. U završnom koraku, svaka preostala tablica ponovno se analizira korištenjem snažnog jezičnog modela (primjerice GPT-4.1), kako bi se osiguralo da su zadržani isključivo stupci koji su zaista ključni. Na taj način mogu se prepoznati suptilnosti koje su izmakle prethodnim fazama, čime se dobiva minimalna, ali informacijski potpuna shema pogodna za robusno generiranje upita.

Shema tijekom svih faza ostaje u istom standardiziranom JSON formatu, a prilikom uklanjanja nevažnih tablica ili stupaca, svi povezani odnosi automatski se ažuriraju.

Generator kandidata

Nakon što je Filtar sheme izgradio relevantan podskup sheme, Generator kandidata preuzima zadatak. Njegova je uloga prevesti korisnički upit na prirodnom jeziku i reduciranu shemu u jedan ili više mogućih SQL upita.

Korak 1: Inicijalno generiranje upita. Za svaku odabranu strategiju, generator upućuje jezičnom modelu korisnički upit zajedno s minimalnim kontekstom sheme. Precizno definiran kontekst usmjerava model na relevantne podatke, što u pravilu rezultira većom točnošću i boljom objašnjivošću generiranih SQL upita.

Korak 2: Izvršavanje i revizija upita. Nakon što su inicijalni kandidati za SQL upite generirani, oni se izvršavaju nad bazom podataka (ili nad zaštićenom ispitnom instancom). Ako pojedini upit ne uspije zbog pogreške ili vraća prazan rezultat, Generator kandidata ulazi u petlju revizije: jezični model tada dobiva problematičan upit, pripadajuće poruke o pogrešci te izvorni upit korisnika, uz zadatak da revidira svoj prijedlog. Ovaj mehanizam samopopravka omogućuje agentu automatsko ispravljanje uobičajenih pogrešaka, čime se povećava vjerojatnost dobivanja ispravnog i smislenog upita.

Umjesto oslanjanja na samo jedan pristup, Generator kandidata može koristiti više različitih strategija za generiranje SQL upita. To uključuje izravno generiranje, lanac razmišljanja, postupno planiranje i izvođenje te podjelu zadatka na manje dijelove kod posebno složenih upita (engl. *divide and conquer*). Primjenom više strategija, agent se može bolje prilagoditi i jednostavnim i zahtjevnijim pitanjima, odabirom brzih, jednostavnih pristupa za lakše slučajeve, te strukturiranijeg rasuđivanje kad je potrebno postupno razlaganje zadatka.

Evaluator upita

Završni korak u ovom modularnom sustavu je Evaluator upita, čija je uloga odabrati najboljeg SQL kandidata, ne samo prema sintaktičkoj ispravnosti, već i prema semantičkoj točnosti. Cilj je osigurati da odabrani upit doista odgovara na korisničko pitanje.

Korak 1: Evaluacija i selekcija. Polazeći od izvornog pitanja i skupa generiranih kandidata, Evaluator upita upućuje jezični model da generira ciljane ispitne slučajeve u prirodnom jeziku. Svaki ispitni slučaj posebno je osmišljen kako bi razlikovao ispravno konstruirane upite od onih koji sadrže suptilne pogreške. Primjerice, ispitni slučaj može zadati očekivani raspon vrijednosti, provjeravati ispravnost agregacije ili potvrđivati postojanje određene relacije. Model se potiče na kreiranje ispitnih slučajeva koji će natjerati pogrešan ili nepotpun SQL upit da padne barem na jednom provjeravanju, čime se omo-

gućuje razlikovanje sličnih upita.

Korak 2: Bodovanje i odabir. Svaki kandidat se ocjenjuje prema broju uspješno položenih diskriminativnih ispitnih slučajeva. Upit koji zadovolji najviše ispitnih slučajeva bira se kao konačni rezultat, čime se postiže dodatna semantička sigurnost, izvan osnovne provjere ispravnosti izvršavanja.

Ovaj završni modul predstavlja najsloženiji i najmanje trivijalan dio cijelog sustava. Upravo ovdje dolazi do izražaja kreativnost i sposobnost modela da razumije nijanse korisničkog upita te razlikuje suptilne pogreške u generiranim SQL upitima. Evaluator upita značajno dobiva na učinkovitosti kada je dodatno fino podešen na pažljivo odabranim primjerima evaluacije i selekcije, čime se povećava preciznost i pouzdanost završnog odabira upita.

6.3.2. Primjer zadatka i tijek rada agenta

Kako bi se ilustrirao rad agenta, razmotrimo sljedeći korisnički upit (vidi sliku 6.10.).

Korisnički upit

Which department has the highest average salary?

Slika 6.10. Korisnički upit koji agent pokušava riješiti analizom baze podataka.

Najprije **Filtar sheme** prima korisnički upit i kompletnu shemu baze podataka. On uklanja sve nevažne tablice i stupce te generira sažetu, zadatku prilagođenu shemu (vidi sliku 6.11.).

Filtrirana shema (JSON format)

```
{
  "table": "department",
  "columns": ["id", "name"]
},
{
  "table": "employee",
  "columns": ["id", "department_id", "salary"]
}
```

Slika 6.11. Izlaz filtra sheme: smanjena shema relevantna za postavljeni upit.

Nakon toga, **Generator kandidata** koristi filtriranu shemu i korisnički upit. On generira jedan ili više SQL upita koji mogu dati odgovor na postavljeno pitanje (vidi sliku 6.12.).

SQL kandidati

```
SELECT department.name, AVG(employee.salary) AS avg_salary
FROM employee
JOIN department ON employee.department_id = department.id
GROUP BY department.name
ORDER BY avg_salary DESC
LIMIT 1;
...
```

Slika 6.12. Jedan od SQL upita koji kandidatno odgovara na korisnički upit.

Nakon generiranja upita, **Evaluator upita** dobiva izvorni korisnički upit i SQL kandidate. Automatski kreira ciljane ispitne slučajeve kako bi razlikovao ispravan upit od onih koji sadrže suptilne pogreške (vidi sliku 6.13.).

Generirani ispitni slučajevi

Test 1: Does the query output include both the department name and its average salary?

Test 2: Does the query return exactly one department?

Slika 6.13. Jedinični testovi generirani za procjenu točnosti SQL upita.

Svaki kandidat se procjenjuje prema ovim ispitnim slučajevima. Na kraju, agent korisniku prikazuje odgovor (vidi sliku 6.14.):

Agentov odgovor

The department with the highest average salary is **Engineering**.

Slika 6.14. Konačan odgovor koji agent prikazuje korisniku nakon evaluacije svih kandidata.

6.3.3. Transparentnost, objašnjivost i ograničenja

Ovakvo modularno oblikovanje agenta koji pretvara tekst u SQL namjerno je usmjeren prema transparentnosti i lakšem tumačenju rada sustava. Svaki modul, od Filtra sheme do Generatora kandidata i Evaluatora upita, ostavlja jasne zapise svojih odluka i proizvodi strukturirane rezultate, omogućujući detaljno praćenje cijelog procesa. Time se korisnicima pruža mogućnost da brzo identificiraju i isprave greške ili nesporazume.

Unatoč navedenim prednostima, sustav se suočava s određenim izazovima. Filtriranje sheme zahtijeva pažljivo usklađivanje između jasnoće i pokrivenosti: pretjerano stroga filtracija može rezultirati gubitkom bitnih informacija, dok preširoka selekcija može dovesti do viška podataka i nepoželjne složenosti. Također, učinkovitost i pouzdanost cjelokupnog rješenja uvelike ovise o sposobnostima korištenih jezičnih modela te o dostupnosti računalnih resursa, osobito kada agent primjenjuje više strategija ili višestruke pokušaje ispravljanja upita. Konačno, iako modularni pristup olakšava pronalaženje pogrešaka, slabosti u bilo kojoj pojedinoj komponenti mogu utjecati na točnost cijelog procesa i otežati naknadno popravljanje pogrešaka kod zahtjevnijih upita.

6.4. Korištene tehnologije

Rad je izrađen u programskom jeziku Python, odabranom zbog njegove fleksibilnosti i široke podrške za rad s velikim jezičnim modelima. U implementaciji agenta korištene su sljedeće ključne biblioteke:

- **LangChain** – za orkestraciju modela i izgradnju agentske arhitekture,
- **LangGraph** – za definiranje tijeka rada agenta kroz graf stanja,
- **OpenAI API** – za integraciju s OpenAI modelima,

- **Tavily** – za dohvat aktualnih informacija putem pretraživanja interneta.

7. Zaključak

Integracija velikih jezičnih modela u inteligentne agentske sustave otvara brojne mogućnosti, ali i postavlja važna pitanja. Kako LLM-ovi postaju sve sposobniji za složeno zaključivanje, planiranje i prirodnu komunikaciju, tako dolaze do izražaja i njihova ograničenja u transparentnosti, pouzdanosti i pravednosti. Izazov više nije samo kako učiniti modele pametnijima, već kako osigurati da su njihovi procesi zaključivanja vidljivi, razumljivi i pouzdani.

Jedan od ključnih zaključaka ovog rada je važnost modularne i pregledne arhitekture agenata. Kada se koraci razmišljanja, korištenje alata i donošenje odluka razdvoje na jasno definirane i pregledne faze, znatno je lakše otklanjati pogreške, prepoznati izvore pristranosti i pružiti korisnicima razumljive odgovore. Ovakav pristup ne samo da povećava povjerenje u sustav, već omogućuje i novi oblik iterativnog razvoja: poboljšanja se ne postižu isključivo finim podešavanjem modela, nego i unapređenjem načina na koji se razlažu zadtaci, kako se upravlja kontekstom i kako se vrednuju izlazi modela.

Unatoč značajnom napretku, brojna otvorena pitanja još uvijek ostaju. Veliki jezični modeli i dalje preuzimaju pristranosti iz svojih skupova podataka za učenje, a njihovi odgovori često su skloni generičkim ili sigurnim obrascima umjesto prilagođenim i specifičnim rješenjima. Napori usmjereni na smanjenje pristranosti, povećanje pravednosti i jačanje pouzdanosti zahtijevaju stalnu pažnju, ne samo putem tehničkih poboljšanja, nego i kroz pažljiv odabir podataka za učenje, transparentno vrednovanje te, gdje god je moguće, uključivanje čovjeka u nadzor i evaluaciju.

U sklopu rada razvijen je modularni agent temeljen na paradigmi planiranja i izvršavanja. Agent generira plan, izvršava pojedinačne korake te ih prema potrebi revidira na temelju dobivenih rezultata. Djelomično je implementiran i agent za pretvorbu teksta u

SQL, s ciljem prikaza kako se razlaganjem složenog zadatka na manje, povezane korake može postići veća točnost i objašnjivost zaključivanja.

U budućnosti će razvoj modela zahtijevati pažljivo usklađivanje njihovih naprednih sposobnosti s jasnim mehanizmima nadzora i upravljanja. Veći modeli i sofisticiranije tehnike promptanja omogućuju složenije oblike zaključivanja, ali uspješna primjena u praksi pretpostavlja transparentnost i odgovornost na svim razinama sustava. Sve je važnije razvijati agentske sustave koji jasno obrazlažu svoje odluke, učinkovito reagiraju na povratne informacije te pouzdano rješavaju izazove nesigurnosti i višeznačnosti.

U konačnici, kako agenti temeljeni na velikim jezičnim modelima preuzimaju sve zahtjevnije i kritičnije uloge, nužno je preusmjeriti pažnju s isključivo operativne učinkovitosti prema odgovornom, transparentnom i pravednom donošenju odluka. Ostvarenje tog cilja zahtijeva ne samo daljnji razvoj modela, već i trajnu predanost otvorenosti, mogućnosti revizije te kontinuiranu suradnju između inženjera, korisnika i šire društvene zajednice kojoj su ovi sustavi namijenjeni.

Literatura

[1] S. J. Russell i P. Norvig, *Artificial intelligence: a modern approach*. Pearson Education Limited, Malaysia, 2016.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, i D. Amodei, "Language models are few-shot learners", u *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, i H. Lin, Ur., sv. 33. Curran Associates, Inc., 2020., str. 1877–1901. [Mrežno]. Adresa: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[3] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, i I. Babuschkin, "Gpt-4 technical report", 2024. [Mrežno]. Adresa: https://arxiv.org/abs/2303.08774

[4] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, i et al., "On the opportunities and risks of foundation models", Stanford Center for Research on Foundation Models, teh. izv., 2021., technical Report. [Mrežno]. Adresa: https://crfm.stanford.edu/assets/report.pdf

[5] R. Sennrich, B. Haddow, i A. Birch, "Neural machine translation of rare words with subword units", u *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, 2016., str. 1715–1725. https://doi.org/10.18653/v1/P16-1162

[6] M. Schuster i K. Nakajima, “Japanese and korean voice search”, u *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2012., str. 5149–5152.

[7] T. Mikolov, K. Chen, G. Corrado, i J. Dean, “Efficient estimation of word representations in vector space”, 2013. [Mrežno]. Adresa: https://arxiv.org/abs/1301.3781

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, i I. Polosukhin, “Attention is all you need”, u *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, i R. Garnett, Ur., sv. 30. Curran Associates, Inc., 2017. [Mrežno]. Adresa: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[9] J. Devlin, M.-W. Chang, K. Lee, i K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding”, u *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 2019., str. 4171–4186. https://doi.org/10.18653/v1/N19-1423

[10] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, i I. Sutskever, “Language models are unsupervised multitask learners”, OpenAI, teh. izv., 2019., technical report. [Mrežno]. Adresa: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[11] D. P. Kingma i J. Ba, “Adam: A method for stochastic optimization”, u *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015. [Mrežno]. Adresa: https://arxiv.org/abs/1412.6980

[12] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, i Q. V. Le, “Finetuned language models are zero-shot learners”, u *Advances in Neural Information Processing Systems*, sv. 35, 2022., str. 22 165–22 179. [Mrežno]. Adresa: https://doi.org/10.48550/arXiv.2109.01652

[13] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback", *Advances in neural information processing systems*, sv. 35, str. 27 730–27 744, 2022.

[14] J. Wei, M. B. Du, D. S. Wang, Q. Le, E. H. Chi, i D. Zhou, "Emergent abilities of large language models", *Transactions on Machine Learning Research*, 2022. [Mrežno]. Adresa: https://arxiv.org/abs/2206.07682

[15] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, i D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models", u *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., 2022.

[16] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, i D. Zhou, "Self-consistency improves chain of thought reasoning in language models", u *International Conference on Learning Representations (ICLR)*, 2023. [Mrežno]. Adresa: https://openreview.net/forum?id=1PL1NIMMrw

[17] M. Nye, A. J. Andreassen, G. Gur-Ari, H. Michalewski, J. Austin, D. Bieber, D. Dohan, i A. Lewkowycz, "Show your work: Scratchpads for intermediate computation with language models", 2021. [Mrežno]. Adresa: https://arxiv.org/abs/2112.00114

[18] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, i Y. Cao, "React: Synergizing reasoning and acting in language models", u *International Conference on Learning Representations (ICLR)*, 2023.

[19] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, i K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models", u *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, i S. Levine, Ur., sv. 36. Curran Associates, Inc., 2023., str. 11 809–11 822. [Mrežno]. Adresa: https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf

[20] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, i S. Yao, “Reflexion: language agents with verbal reinforcement learning”, u *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, i S. Levine, Ur., sv. 36. Curran Associates, Inc., 2023., str. 8634–8652. [Mrežno]. Adresa: https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf

[21] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, i D. Kiela, “Retrieval-augmented generation for knowledge-intensive nlp tasks”, u *Advances in Neural Information Processing Systems*, sv. 33. Curran Associates, Inc., 2020., str. 9459–9474. [Mrežno]. Adresa: https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

[22] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, i R. Salakhutdinov, “Transformer-xl: Attentive language models beyond a fixed-length context”, u *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019., str. 2978–2988. https://doi.org/10.18653/v1/P19-1285

[23] I. Beltagy, M. E. Peters, i A. Cohan, “Longformer: The long-document transformer”, 2020. [Mrežno]. Adresa: https://arxiv.org/abs/2004.05150

[24] J. Maynez, S. Narayan, B. Bohnet, i R. McDonald, “On faithfulness and factuality in abstractive summarization”, u *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*. Online: Association for Computational Linguistics, 2020., str. 1906–1919. [Mrežno]. Adresa: https://aclanthology.org/2020.acl-main.173

[25] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, i P. Liang, “Lost in the middle: How language models use long contexts”, *Transactions of the Association for Computational Linguistics*, sv. 12, str. 157–173, 2024. https://doi.org/10.1162/tacl_a_00638

[26] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik,

A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, i F. Herrera, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai", *Information Fusion*, sv. 58, str. 82–115, 2020. https://doi.org/https://doi.org/10.1016/j.inffus.2019.12.012

[27] E. M. Bender, T. Gebru, A. McMillan-Major, i S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?" u *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (FAccT '21)*. Virtual Event, Canada: Association for Computing Machinery, 2021., str. 610–623. https://doi.org/10.1145/3442188.3445922

[28] S. Barocas, M. Hardt, i A. Narayanan, *Fairness and Machine Learning: Limitations and Opportunities*, 1. izd. fairmlbook.org, 2019. [Mrežno]. Adresa: https://fairmlbook.org

[29] J. Vig, "A multiscale visualization of attention in the transformer model", u *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, M. R. Costa-jussà i E. Alfonseca, Ur. Florence, Italy: Association for Computational Linguistics, srpanj 2019., str. 37–42. https://doi.org/10.18653/v1/P19-3007

[30] M. Sundararajan, A. Taly, i Q. Yan, "Axiomatic attribution for deep networks", u *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017., str. 3319–3328.

[31] Y. Belinkov i J. Glass, "Analysis methods in neural language processing: A survey", *Transactions of the Association for Computational Linguistics*, sv. 7, str. 49–72, 2019. https://doi.org/10.1162/tacl_a_00254

[32] Z. C. Lipton, "The mythos of model interpretability", *Communications of the ACM*, sv. 61, br. 10, str. 36–43, 2018. https://doi.org/10.1145/3233231

[33] S. L. Blodgett, S. Barocas, H. Daumé III, i H. Wallach, "Language (technology) is power: A critical survey of "bias" in NLP", u *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, D. Jurafsky, J. Chai,

N. Schluter, i J. Tetreault, Ur. Online: Association for Computational Linguistics, srpanj 2020., str. 5454–5476. https://doi.org/10.18653/v1/2020.acl-main.485

[34] L. Wang, W. Xu, Y. Lan, Z. Hu, Y. Lan, R. K. Lee, i E. Lim, "Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models", u *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL 2023)*. Association for Computational Linguistics, 2023. https://doi.org/10.18653/v1/2023.acl-long.147

[35] F. Doshi-Velez i B. Kim, "Towards a rigorous science of interpretable machine learning", 2017. [Mrežno]. Adresa: https://arxiv.org/abs/1702.08608

[36] S. Talaei, M. Pourreza, Y.-C. Chang, A. Mirhoseini, i A. Saberi, "Chess: Contextual harnessing for efficient sql synthesis", 2024. [Mrežno]. Adresa: https://arxiv.org/abs/2405.16755

[37] M. Pourreza, H. Li, R. Sun, Y. Chung, S. Talaei, G. T. Kakkar, Y. Gan, A. Saberi, F. Özcan, i S. O. Arık, "Chase–sql: Multi-path reasoning and preference-optimized candidate selection in text-to-sql", u *Proceedings of the 2025 International Conference on Learning Representations (ICLR)*, 2025., poster Paper. [Mrežno]. Adresa: https://openreview.net/forum?id=CvGqMD5OtX

Sažetak

Veliki jezični modeli za optimizaciju, planiranje i objašnjivost u inteligentnim agentskim sustavima

Antun Tišljar

Primjena velikih jezičnih modela (LLM-ova) u izgradnji inteligentnih agenata donosi iznimne mogućnosti, no istovremeno postavlja izazove vezane uz transparentnost i razumljivost procesa zaključivanja. Ovaj rad istražuje kako pažljivo osmišljena arhitektura agenata može omogućiti jasnoću, sljedivost i objašnjivost u radu LLM-ova. Ovakvo modularno oblikovanje, s jasno definiranim koracima i transparentnim bilježenjem procesa, omogućuje precizno praćenje i provjeru svakog generiranog odgovora. Kao praktični prikaz ovog pristupa, predstavljeni su agent za planiranje i izvođenje te agent za automatsko generiranje SQL upita iz pitanja postavljenih prirodnim jezikom.

Ključne riječi: veliki jezični modeli; inteligentni agenti; objašnjivost; transparentnost

Abstract

Large language models for optimization, planning and explainability in intelligent agent systems

Antun Tišljar

While large language models (LLMs) have made intelligent agents more capable than ever, their internal reasoning often remains opaque, especially as systems grow in complexity. This study shows how agent architecture can be designed to make LLM reasoning explicit and explainable. By breaking down tasks into modular, auditable steps and logging each decision, it becomes possible to trace how answers are constructed. Two agent prototypes, one that plans and executes stepwise solutions and another that translates questions to SQL queries, illustrate these principles in practice.

Keywords: large language models; intelligent agents; explainability; transparency

Privitak A: Promptovi

Kod uporabe strukturiranih izlaza, nije potrebno izravno uključivati propisani JSON format u sam prompt, budući da sustav ili API automatski osigurava očekivanu strukturu odgovora. Ipak, često se doda primjer odgovora, poput:

```
Respond like this:
{
...
}
```

Ovo je posebno često kod duljih ili složenijih promptova, kako bi format izlaza bio što jasniji ljudskim čitateljima. Ovakav pristup posebno olakšava rad programerima koji rade s opsežnim promptovima jer unaprijed jasno postavlja očekivanja te olakšava razumijevanje i otklanjanje pogrešaka, iako sam model zapravo ne zahtijeva eksplicitni primjer u promptu.

Na slikama A1., A2. i A3. se nalaze promptovi koje koristi agent za planiranje i izvođenje.

Prompt za Planera

For the given objective, come up with a simple, step-by-step plan.
`{objective}`

Each step should be necessary and sufficient to achieve the objective. Avoid adding any superfluous steps. The result of the final step should be the final answer. Focus on what to do, not how to do it. Do not skip steps.

Slika A1. Prompt koji se koristi za generiranje plana iz zadanog cilja.

Prompt za Executora

For the following plan:

`{plan}`

You are tasked with executing task:

`{task}`

Focus only on this task!

Slika A2. Prompt koji definira kontekst za izvršavanje pojedinog koraka plana.

Prompt za Re-plannera

For the given objective, come up with a simple, step-by-step plan. Each step should be necessary and sufficient to achieve the objective. Avoid adding any superfluous steps. The result of the final step should be the final answer. Focus on what to do, not how to do it. Do not skip steps.

Objective:

`{input}`

Completed Steps and their results:

`{past_steps_and_results}`

Update your plan accordingly. If no more steps are needed and you can return to the user, then use Response. Otherwise, add only the steps that still need to be done. Do not include previously completed steps in the updated plan.
If you choose to re-plan, provide a reason for the re-plan.

Slika A3. Prompt koji agent koristi za re-planiranje na temelju prethodnih rezultata.

Privitak B: Kod agenta za planiranje i izvođenje

```python
class SystemState(BaseModel):
    """
    Holds the entire state of the 'Plan and Execute' workflow.
    """

    input: str = Field(..., description="The raw text from the user.")
    plan: Optional[Plan] = Field(
        default=None,
        description="The current Plan object, including rationale and steps.",
    )

    # Each entry is (executed_step_description, result_text).
    past_steps_and_results: Annotated[list[tuple[str, str]], operator.add] = Field(
        default_factory=list, description="Record of executed steps and their results."
    )

    rationale: Optional[str] = Field(
        default=None, description="Explanation of why we still need more steps."
    )
    response: Optional[str] = Field(
        default=None,
        description="A final or intermediate response to the user, if completed.",
    )
```

Slika B1. Interno stanje agenta za planiranje i izvođenje

```python
import operator
from typing import Annotated, Literal, Optional

from pydantic import BaseModel, Field


class PlanStep(BaseModel):
    """
    An individual step within a Plan.
    """

    goal: str = Field(
        ..., description="A brief statement of what needs to be achieved."
    )
    additional_context: Optional[str] = Field(
        None, description="Optional clarifications or hints for the Executor."
    )


class Plan(BaseModel):
    """
    A sequence of high-level steps to achieve the user's objective.
    """

    type: Literal["Plan"] = "Plan"
    rationale: str = Field(
        ...,
        description=(
            "Explanation for why we need these steps or why we are replanning, "
        ),
    )
    steps: list[PlanStep] = Field(
        ..., description="A list of high-level steps to achieve the user's objective."
    )


class Respond(BaseModel):
    """
    The final or intermediate response to the user.
    """

    type: Literal["Respond"] = "Respond"
    response: str = Field(
        ..., description="A conversational response to the user's query."
    )


class Action(BaseModel):
    """
    Represents the agent's next action, which can either be:
    - a new or updated Plan to execute, or
    - a Respond object to finalize and return an answer to the user.
    """

    action: Plan | Respond = Field(
        ...,
        discriminator="type",
        description=(
            "Action to perform. If you want to respond to the user, use Respond. "
            "If you need more steps (tools, data, etc.), use Plan."
        ),
    )
```

Slika B2. Podatkovni modeli agenta za planiranje i izvođenje

```python
import textwrap
import uuid
from typing import Any
from openai import OpenAI
import logging
from langchain_openai import ChatOpenAI

from langchain.callbacks.tracers import LoggingCallbackHandler
from langchain.tools.retriever import create_retriever_tool
from langchain_community.tools.tavily_search import TavilySearchResults
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables.config import RunnableConfig
from langgraph.checkpoint.memory import MemorySaver
from langgraph.graph import END, START, StateGraph
from langgraph.graph.state import CompiledStateGraph
from langgraph.prebuilt import create_react_agent

from agent.plan_and_execute.model import (
    Action,
    Plan,
    SystemState,
    Respond,
)

# The proposed Plan-and-Solve (PS) Prompting consists of two components: devising a plan
# to divide the task into smaller subtasks and then executing the subtasks according to the plan.
#
# Inspired by "Plan-and-Solve Prompting for Zero-Shot Reasoning" (see https://arxiv.org/abs/2305.04091).
# I suggest reading the paper to understand the limitations and potential improvements of this prompting approach.


class PlanAndExecuteAgent():

    _planner_prompt = ...

    _replanner_prompt = ...

    _executor_prompt = ...

    def __init__(self, model_name: str | None = "gpt-4.1") -> None:
        if not model_name in [model.id for model in OpenAI().models.list().data]:
            raise ValueError(f"Model {model_name} does not exist.")

        self._model = ChatOpenAI(
            model=model_name, use_responses_api=True
        )

        agent_logger = logging.getLogger("agentLogger")
        self._graph_logger = logging.getLogger("agentGraphLogger")

        callbacks = LoggingCallbackHandler(agent_logger)
        self._config: RunnableConfig = RunnableConfig(
            configurable={"thread_id": str(uuid.uuid4())},
            callbacks=[callbacks],
            recursion_limit=15,
        )

        self._planner = self._planner_prompt | self._model.with_structured_output(Plan)
        self._replanner = self._replanner_prompt | self._model.with_structured_output(
            Action, method="function_calling"
        )
        tools = [TavilySearchResults(include_answer=True)]
        self._executor = create_react_agent(self._model, tools)

        self._graph: CompiledStateGraph = self._build_graph()

    def _build_graph(self) -> CompiledStateGraph:
        builder = StateGraph(SystemState)

        builder.add_node("planner", self._plan_step)
        builder.add_node("executor", self._execute_step)
        builder.add_node("re-planner", self._replan_step)

        builder.add_edge(START, "planner")
        builder.add_edge("planner", "executor")
        builder.add_edge("executor", "re-planner")

        builder.add_conditional_edges(
            "re-planner",
            self._should_end,
            [END, "executor"],
        )

        return builder.compile(MemorySaver())
```

Slika B3. Implementacija agenta za planiranje i izvođenje (1. dio)

```
def _execute_step(self, state: SystemState):
    self._graph_logger.info("=== executor called ===")

    assert state.plan is not None, "plan should never be None at this point."
    plan_enumerated: str = "\n".join(
        f"{i+1}. {step.goal}"
        + (f" ({step.additional_context})" if step.additional_context else "")
        for i, step in enumerate(state.plan.steps)
    )
    task_text: str = state.plan.steps[0].goal + (
        f" ({state.plan.steps[0].additional_context})"
        if state.plan.steps[0].additional_context
        else ""
    )

    agent_response = self._executor.invoke(
        self._executor_prompt.invoke({"plan": plan_enumerated, "task": task_text}),
    )

    self._graph_logger.info(
        f"past_steps_and_results: {[(task_text, agent_response['messages'][-1].content)]}"
    )
    return {
        "past_steps_and_results": [
            (task_text, agent_response["messages"][-1].content)
        ],
    }

def _plan_step(self, state: SystemState):
    self._graph_logger.info("=== planner called ===")

    plan: Plan = Plan.model_validate(
        self._planner.invoke({"messages": state.input})
    )

    self._graph_logger.info(f"plan: {plan}")
    return {"plan": plan}

def _replan_step(self, state: SystemState):
    self._graph_logger.info("=== re-planner called ===")

    enumerated_past_steps = "\n".join(
        f"{i+1}. {step[0]}: {step[1]}"
        for i, step in enumerate(state.past_steps_and_results)
    )

    action: Action = Action.model_validate(
        self._replanner.invoke(
            {
                "input": state.input,
                "past_steps_and_results": enumerated_past_steps,
            }
        )
    )

    if isinstance(action.action, Respond):
        self._graph_logger.info(f"response: {action.action.response}")
        return {"response": action.action.response}
    elif isinstance(action.action, Plan):
        self._graph_logger.info(
            f"plan: {action.action.steps}, rationale: {action.action.rationale}"
        )
        return {
            "plan": action.action,
            "rationale": action.action.rationale,
        }

def _should_end(self, state: SystemState):
    if state.response is not None:
        return END
    else:
        return "executor"

def send_message(self, message: str) -> str:
    state = SystemState(input=message)
    output: dict[str, Any] = self._graph.invoke(state, self._config)
    return output.get("response", "")
```

Slika B4. Implementacija agenta za planiranje i izvođenje (2. dio)