

Primjena i usporedba LLM-ova u evaluaciji kôda

Mrčela, Judita

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split, Faculty of Science / Sveučilište u Splitu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:166:838963>

Rights / Prava: [In copyright](#)/[Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2024-12-24**

Repository / Repozitorij:

[Repository of Faculty of Science](#)



SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET

DIPLOMSKI RAD

**Primjena i usporedba LLM-ova u evaluaciji
kôda**

Judita Mrčela

Split, Rujan 2024.

Temeljna dokumentacijska kartica

Diplomski rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za Informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

Primjena i usporedba LLM-ova u evaluaciji kôda

Judita Mrčela

SAŽETAK

Ovo istraživanje za cilj ima istražiti primjenu i usporedbu velikih jezičnih modela u evaluaciji procjene koda. Usredotočeno je na tri modela: OpenAI-jev GPT-4, Code Llama 7B Instruct i fino podešenu verziju CodeT5+. Modeli se ocjenjuju na temelju njihove sposobnosti procjene koda u različitim kategorijama, uključujući matematičke i algoritamske zadatke, strukture podataka i web programiranje. Uspoređujući izvedbu ovih modela, nastoje se identificirati njihove prednosti i slabosti u pružanju točnih i korisnih povratnih informacija za poboljšanje kvalitete koda. Ovo istraživanje naglašava potencijal velikih jezičnih u pojednostavljivanju procesa ocjenjivanja programskog koda, a istovremeno prepoznaje potrebu za daljnjim napretkom u njihovom finom podešavanju i primjeni.

Ključne riječi: Veliki jezični modeli, evaluacija kôda, usporedba modela, fino podešavanje

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: [66] stranica, [38] grafičkih prikaza, [3] tablica i [44] literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: **Doc. dr. sc. Goran Zaharija**, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Ocjenjivači: **Doc. dr. sc. Goran Zaharija**, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Doc. Dr. sc. Divna Krpan, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Doc. Dr. sc. Monika Mladenović, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: **Rujan 2024.**

Basic documentation card

Thesis

University of Split
Faculty of Science
Department of Informatics
Ruđera Boškovića 33, 21000 Split, Croatia

Application and Comparison of LLMs in Code Evaluation

Judita Mrčela

ABSTRACT

This research aims to investigate the application and comparison of large language models in the evaluation of code estimation. It focuses on three models: OpenAI's GPT-4, Code Llama 7B Instruct, and a fine-tuned version of CodeT5+. Models are evaluated based on their ability to evaluate code in a variety of categories, including mathematical and algorithmic tasks, data structures, and web programming. By comparing the performance of these models, we seek to identify their strengths and weaknesses in providing accurate and useful feedback to improve code quality. This research highlights the potential of large languages in simplifying the process of evaluating software code, while recognizing the need for further progress in their fine-tuning and application.

Key words: large language models, code evaluation, model comparison, fine-tuning

Thesis is deposited in the library of Faculty of Science, University of Split

Thesis consists of: [66] pages, [38] figures, [3] tables and [44] references

Original language: Croatian

Supervisor: **Goran Zaharija, Ph.D.** *Professor of Faculty of Science, University of Split*

Reviewers: **Goran Zaharija, Ph.D.** *Professor of Faculty of Science, University of Split,*

Divna Krpan, Ph.D. *Assistant professor of Faculty of Science, University of Split,*

Monika Mladenović, Ph.D. *Assistant professor of Faculty of Science, University of Split,*

Thesis accepted: **September 2024**

IZJAVA

kojom izjavljujem s punom materijalnom i moralnom odgovornošću da sam diplomski rad s naslovom Primjena i usporedba LLM-ova u evaluaciji kôda izradila samostalno pod voditeljstvom doc. dr. sc. Goran Zaharija. U radu sam primijenila metodologiju znanstvenoistraživačkog rada i koristila literaturu koja je navedena na kraju diplomskog rada. Tuđe spoznaje, stavove, zaključke, teorije i zakonitosti koje sam izravno ili parafrazirajući navela u diplomskom radu na uobičajen, standardan način citirala sam i povezala s fusnotama s korištenim bibliografskim jedinicama. Rad je pisan u duhu hrvatskog jezika.

Student/ica

Judita Mrčela

Sadržaj

Uvod	1
1. Pregled literature.....	2
2. Veliki jezični modeli (engl. <i>Large language models</i> , LLM).....	6
2.1. Rani razvoj jezičnih modela	6
2.2. Arhitektura <i>Transformers</i> modela	7
2.3. Evolucija LLM-a	8
2.3.1. GPT (engl. <i>Generative Pre-trained Transformer</i>)	8
2.3.2. LLaMA (engl. <i>Large Language Model Meta AI</i>).....	9
2.3.3. PaLM (engl. <i>Pathways Language Model</i>).....	10
2.3.4. Ostali značajni LLM-i	10
3. Primjena velikih jezičnih modela	11
3.1. Obrada prirodnog jezika (engl. <i>Natural Language Processing</i> , NLP).....	11
3.1.1. Generiranje teksta	11
3.1.2. Strojno prevođenje.....	12
3.1.3. Sažimanje teksta	12
3.1.4. Analiza osjećaja (engl. <i>Sentiment Analysis</i>)	12
3.1.5. Prepoznavanje imenovanog entiteta	13
3.1.6. Klasifikacija teksta	13
3.1.7. Odgovaranje na pitanje	13
3.1.8. Agenti za razgovor (engl. <i>Chatbot</i>)	13
3.2. Generiranje i analiza koda	14
3.2.1. Dovršavanje koda (engl. <i>Code Completion</i>).....	14
3.2.2. Otkrivanje i ispravljanje pogrešaka	14
3.2.3. Dokumentacija koda i sažetak	15
3.3. Kreativno pisanje i stvaranje sadržaja	16

3.3.1.	Generacija priča i pjesama.....	16
3.3.2.	Marketinški tekst i oglašavanje	16
3.3.3.	Izrada oglasa.....	17
3.4.	Obrazovanje i e-učenje	17
3.4.1.	Personalizirane upute.....	17
3.4.2.	Izrada udžbenika i priručnika	18
3.4.3.	Interaktivne vježbe i kvizovi	18
3.4.4.	Multimedijski obrazovni resursi.....	18
3.4.5.	Prilagodba i lokalizacija	18
3.5.	Zdravstvo i medicinska istraživanja	19
3.5.1.	Analiza medicinske dokumentacije	19
3.5.2.	Prediktivna analitika.....	20
3.5.3.	Otkrivanje lijekova i stvaranje istraživačkih radova	20
4.	Metodologija.....	22
4.1.	Istraživačka pitanja.....	22
4.2.	Podaci korišteni za transfer učenja	22
4.3.	Tehnike analize.....	24
4.4.	Pristup.....	25
4.4.1.	Pripremljeni skupa podataka	25
4.4.2.	Odabir velikog jezičnog modela.....	26
4.4.3.	Koraci pretprocesiranja.....	27
4.4.4.	Treniranje modela.....	27
4.4.5.	Hiperparametarsko podešavanje (engl. <i>Hyperparameter tuning</i>)	28
4.4.6.	Ulazni podaci za procjenu	30
5.	Praktični dio.....	32
5.1.	Uključene biblioteke.....	32

5.2.	Priprema podataka	33
5.3.	Fino podešavanje modela	34
5.4.	Rezultati.....	37
5.4.1.	Podešavanje hiperparametara	38
5.4.2.	Evaluacija Llama i Chat GPT modela	38
5.4.3.	Evaluacija fino podešenog modela	47
5.5.	Rasprava	50
5.5.1.	Mjerila evaluacije modela	51
5.5.2.	Predviđene procjene	52
5.5.3.	Rezultati Llama i GPT modela	53
5.5.4.	Ograničenja istraživanja	57
5.5.5.	Upute za poboljšanje	58
	Zaključak	59
	Popis slika.....	60
	Popis tablica.....	62
	Literatura	63

Uvod

Brzi napredak umjetne inteligencije i strojnog učenja doveo je do značajnih inovacija u raznim područjima, uključujući obradu prirodnog jezika (engl. *Natural Language Processing*, NLP). Jedan od najutjecajnijih razvoja u NLP-u je stvaranje velikih jezičnih modela (engl. *Large language models*, u ostatku rada LLM), koji su sposobni razumjeti i generirati tekst sličan ljudskom. Modeli poput OpenAI-jevog GPT-3 i njegovih nasljednika napravili su revoluciju u različitim područjima, imajući utjecaja na sve, od stvaranja sadržaja do korisničke službe. U području razvoja softvera, ovi modeli pokazali su obećavajući potencijal u automatiziranju procjene koda, ključnog zadatka za osiguranje kvalitete, učinkovitosti i ispravnosti koda. Tradicionalne metode procjene koda često se oslanjaju na ručni pregled što može biti dugotrajno i podložno ljudskim pogreškama. Pojava LLM-ova nudi transformativni pristup iskorištavanjem njihove sposobnosti razumijevanja i analize semantike koda i konteksta, pružajući učinkovitije i skalabilnije rješenja. Međutim, i dalje je ključno procijeniti i usporediti učinkovitost različitih LLM-ova u ovoj specifičnoj primjeni. (Brown, i dr., 2020)

Ovaj rad ima za cilj provjeriti može li se LLM fino podesiti s primjerenim podacima da bi imao što bolje rezultate za zadatak procjene koda i usporediti istaknuti LLM s već uspostavljenim LLM-ovima u njihovoj sposobnosti procjene kvalitete programskog koda. Provođenjem temeljite analize i usporedbe, ovaj rad nastoji identificirati prednosti i slabosti svakog modela, pružajući uvide za njihovu primjenu u procjeni koda. Istraživanje će se usredotočiti na ključne pokazatelje performansi kao što su točnost, preciznost i upotrebljivost, osiguravajući sveobuhvatnu procjenu.

Struktura ovoga rada podijeljena je na dva dijela, teorijski okvir i praktični dio. U teorijskom okviru proći će se kroz pregled relevantnih istraživanja, detaljni pregled LLM-a s detaljima njihove arhitekture i praćenjem njihove evolucije tijekom vremena. Zatim će se predstaviti metodologija istraživanja, nakon čega slijede rezultati i rasprava. Na kraju, rad završava sažetkom nalaza i njihovih širih implikacija.

1. Pregled literature

Područje strojnog učenja primijenjenog na programski kod u proteklom je desetljeću doživjelo značajan rast, s brojnim istraživanjima koje istražuju različite aspekte analize, generiranja i procjene izvornog programskog koda. Ovo poglavlje daje pregled relevantne literature koja se fokusira na istraživanja LLM-a i njihove primjene u procjeni koda.

Pretraživanje na Google Scholaru korištenjem ključnih riječi kao što su "evaluacija koda", "veliki jezični modeli", "strojno učenje", "analiza izvornog koda" i "duboko učenje" prvotno je vratilo 4.280.000 rezultata. Primjenom filtera: ograničavanjem raspona publikacija na 2016.–2024., fokusiranjem samo na recenzirane članke i sortiranjem prema broju citata—ovaj je broj sužen na 84.000 radova. Među njima su najrelevantnije studije odabrane na temelju toga koliko su njihovi naslovi bili u skladu s temom istraživanja.

U području automatizirane procjene koda, modeli strojnog učenja korišteni su za procjenu kvalitete i ispravnosti programskog koda. Istraživanje iz 2022. "*Speeding Up Automated Assessment of Programming Exercises*" predlaže pristup pomoću strojnog učenja za automatizirano predviđanje točnosti programskih zadataka, čime se smanjuje potreba za ručnim ocjenjivanjem i potreba za ručnim pokretanjem programskog koda u drugom okruženju. Njihova je metoda uključivala izdvajanje značajki iz predanog koda i primjenu algoritama klasifikacije kako bi se utvrdilo zadovoljava li kod tražene specifikacije. Izdvojene značajke uključuju sintaktičke elemente, strukturna svojstva i druge relevantne karakteristike koje se mogu analizirati. Treniranjem ovih algoritama na velikom skupu podataka predanih kodova, model uči razlikovati točan od netočnog koda na temelju identificiranih značajki. Istraživanje pokazuje da takvi modeli strojnog učenja mogu točno predvidjeti ispravnost podnošenja koda uz visoku učinkovitost. Ovaj pristup ne samo da ubrzava proces ocjenjivanja, već također osigurava dosljednost i objektivnost u ocjenjivanju, što su ključni čimbenici u obrazovnim okruženjima. Rezultati istraživanja ukazuju na značajno smanjenje radnog opterećenja za nastavnike, dopuštajući im da se više usredotoče na pružanje personaliziranih povratnih informacija i podrške. (Sarsa, Leinonen, Koutchme, & Hellas, 2022)

Slično tome, istraživanje Zhanga i suradnika u "*A Survey of Large Language Models*" istražuje različite primjene LLM-a, uključujući procjenu koda. Autori su pregledali postojeće LLM-ove i njihovu izvedbu na zadacima kao što su sažimanje koda, otkrivanje

pogrešaka i automatsko ocjenjivanje. Istaknuli su izazove i prilike u ovom području, napominjući da LLM-ovi mogu značajno pojednostaviti proces pružanjem točnih i pravovremenih povratnih informacija, čime se povećava učinkovitost u praksi kodiranja i poboljšava kvaliteta baze podataka programskog koda (Zhang i ostali, 2023)

Rad iz 2021. pod nazivom „*Evaluating Large Language Models Trained on Code*“ pruža empirijsku procjenu LLM-ova na zadacima povezanim s programskim kodom. Autori predstavljaju model *Codex* kojeg su procijenili na temelju mjerila za generiranje, dovršavanje i prijevod koda. Rezultati ovog istraživanja značajni su za područje programskog inženjerstva. Postizanjem visoke točnosti u programskim zadacima, *Codex* i slični LLM-ovi mogu se integrirati u razvojni proces kako bi se povećala produktivnost i smanjilo vrijeme potrebno za ručno kodiranje. Specifično, u području procjene koda, ovi modeli mogu pomoći u automatiziranom ocjenjivanju programskih zadataka, pružajući direktnu povratnu informaciju studentima, time smanjujući teret ocjenjivanja za nastavnike. Ovo istraživanje Chena i suradnika naglašava transformativni potencijal LLM-ova treniranih za generiranje i procjenjivanje programskog koda. Iskorištavanjem prednosti ovih modela može se postići veća učinkovitost i točnost u zadacima programiranja. Rad pruža snažnu osnovu za daljnje istraživanje mogućnosti LLM- u programskom inženjerstvu. (Chen, Tworek, Jun, Qiming Yuan, & Pinto, 2021)

Rad Browna i suradnika iz 2020. godine pod nazivom „*Language Models are Few-Shot Learners*“ predstavlja GPT-3, autoregresivni jezični model velikih razmjera koji je treniran OpenAI-jem. GPT-3, sa svojih 175 milijardi parametara, pokazao je sposobnost obavljanja širokog spektra zadataka s minimalnim finim podešavanjem (engl. *fine tuning*), uključujući generiranje i dovršavanje koda. Ovaj rad naglašava svestranost LLM-a i njihovu primjenjivost u različitim domenama, uključujući i programiranje. (Brown, i dr., 2020)

Rad iz 2022. autora F. Barchija i suradnika „*Deep Learning Approaches to Source Code Analysis for Optimization of Heterogeneous Systems: Recent Results, Challenges and Opportunities*“ istražuje različite modele dubokog učenja (engl. *Deep learning*), kao što su konvolucijske neuronske mreže (engl. *Convolutional neural networks*, CNN), rekurentne neuronske mreže (engl. *Recurrent neural networks*, RNN) i transformatorske (engl. *Transformers*) arhitekture, te njihovu primjenjivost u analizi izvornog programskog koda. Rad pregledava kako duboko učenje može pomoći u optimizaciji koda za te sustave. U radu se zaključuje da su velike mogućnosti dubokog učenja za analizu izvornog koda i

optimizaciju sustava, iako postoje značajni izazovi s kojima se treba pozabaviti, (Barchi, Parisi, Bartolini, & Acquaviva, 2022)

Istraživanje Allamansa i suradnika 2018. „*A Survey of Machine Learning for Big Code and Naturalness*“ daje pregled različitih pristupa strojnog učenja za analizu koda i kako se modeli strojnog učenja primjenjuju na velike baze podataka programskog koda. Ovo istraživanje naglašava različite pristupe poboljšanja analize koda, kao što je korištenje probabilističkih modela i tehnika dubinskog učenja za poboljšanje čitljivosti koda. Ističe važnost razumijevanja 'prirodnosti' koda i kako se statistička svojstva koda mogu iskoristiti za automatizaciju zadataka poput otkrivanja grešaka i sinteze koda. Autori daju primjere kako su modeli strojnog učenja primijenjeni na različite zadatke programiranja, kao što su dovršavanje koda, otkrivanje grešaka, refaktoriranje i sažimanje koda. Naglašavaju važnost skupova podataka u treniranju i evaluaciji ovih modela, ističući izazove u dobivanju visokokvalitetnih i reprezentativnih skupova podataka. (Allamanis & Sutton, 2018.)

U radu „*Are Deep Neural Networks the Best Choice for Modeling Source Code?*“ autori Hellendoorn i Devanbu žele procijeniti jesu li duboke neuronske mreže (engl. *Deep Neural Networks*, DNN), poznate po svom uspjehu u raznim domenama poput prepoznavanja slike i govora, prikladne za modeliranje izvornog koda. Istraživači su analizirali izvedbu DNN-ova u zadacima kao što su predviđanje završetka koda, identificiranje grešaka i predlaganje promjena koda. Iako su DNN-ovi pokazali obećavajuće rezultate u određenim aspektima modeliranja izvornog koda, istraživanje zaključuje da njihova učinkovitost varira ovisno o specifičnom zadatku i skupu podataka. Za zadatke koji zahtijevaju dublje semantičko razumijevanje ili razmišljanje o ponašanju programa (kao što je otkrivanje grešaka ili procjena kvalitete koda), DNN-ovi su pokazali ograničenja zbog složenosti razumijevanja konteksta i ovisnosti u kodu. (Hellendoorn & Devanbu, 2017)

Rad iz 2019. „*What do you learn from context? Probing for sentence structure in contextualized word representations*“, uspoređuje modele temeljene na mrežama dugog kratkoročnog pamćenja (engl. *Long short-term memory*, skraćeno LSTM) s modelima temeljenim na *Transformers* arhitekturi za zadatke otkrivanja grešaka i sažimanja koda. Rezultati istraživanja pokazuju da su modeli temeljeni na *Transformers* arhitekturi, koji koriste *self-attention* mehanizme, nadmašili LSTM modele i u otkrivanju grešaka i u sažimanju koda, pokazujući napredniju sposobnost hvatanja složenih ovisnosti i kontekstualnih informacija. Rezultati su podržali prijelaz na LLM-ove temeljene na

Transformers arhitekturi za učinkovitiju analizu koda i procjenu, naglašavajući njihov potencijal u poboljšanju kvalitete softvera.

Komparativna istraživanja LLM-ova za procjenu koda dala su značajne uvide u mogućnosti i ograničenja ovih modela. Prijelaz s tradicionalnih statističkih metoda na sofisticirane arhitekture dubokog učenja, posebice modele temeljene na *Transformers* arhitekturi, značajno je poboljšao izvedbu zadataka povezanih s kodom. Međutim, potrebno je pozabaviti se izazovima kao što su kvaliteta podataka i računalni resursi kako bi se u potpunosti mogao ostvariti potencijal LLM-a u procjeni koda.

2. Veliki jezični modeli (engl. *Large language models, LLM*)

Veliki jezični modeli podskup su modela umjetne inteligencije dizajniranih za obradu i generiranje teksta prividnom ljudskom. Ove modele karakterizira njihovo opsežno treniranje na veoma velikim količinama tekstualnih podataka, što im omogućuje učenje složenih obrazaca, struktura i nijansi prirodnog jezika. LLM se prvenstveno koriste u zadacima obrade prirodnog jezika (engl. *Natural language processing*, skraćeno NLP) kao što su generiranje teksta, prijevod, sažimanje i još mnogo toga. (Minaee i ostali, 2024)

2.1. Rani razvoj jezičnih modela

Napredak prema velikim jezičnim modelima započeo je s jednostavnijim statističkim metodama za modeliranje jezika, kao što su n-grami. Ovi modeli predviđaju sljedeću riječ u nizu na temelju prethodnih n riječi, gdje je n obično mali broj. Iako su učinkoviti za kratke kontekste, n-grami imaju problema s brzinom obrade i povećanjem performansi nad velikim skupom podataka. (Li, Ca, Xu, & Watanabe, 2022)

S pojavom neuronskih mreža razvijeni su sofisticiraniji modeli poput rekurentnih neuronskih mreža (engl. *recurrent neural networks*, RNN) i mreža dugog kratkoročnog pamćenja (engl. *Long short-term memory*, skraćeno LSTM). RNN-ovi obrađuju nizove podataka održavajući skriveno stanje koje bilježi informacije o prethodnim elementima u nizu, čineći ih prikladnijima za zadatke koji uključuju sekvencijalne podatke. Međutim, RNN-ovi su se suočavali s izazovima kao što su iščezavanje i eksplozivni gradijenti, što je ometalo njihovu sposobnost učinkovitog hvatanja dugotrajnih ovisnosti. (Pešut, 2019)

LSTM-ovi riješili su neka od ovih ograničenja uključivanjem mehanizama usmjeravanja koji reguliraju protok informacija, dopuštajući modelu da održi stabilnije skriveno stanje tijekom duljih sekvenci. Unatoč ovim naprecima, i RNN-ovi i LSTM-ovi bili su računalno intenzivni i još uvijek su se borili s vrlo velikim ovisnostima (Wikipedia, Long short-term memory, 2024).

2.2. Arhitektura *Transformers* modela

Veliki napredak u jezičnom modeliranju došao je s uvođenjem *Transformers* arhitekture koju predstavlja Vaswanij i suradnici 2017. u svom radu "*Attention is all you need*". Model *Transformers* revolucionirao je NLP eliminirajući potrebu za ponavljajućim vezama i umjesto toga u potpunosti se oslanjajući na mehanizme samoopažanja (engl. *self-attention mechanism*). Ova se arhitektura sastoji od strukture koder-dekoder, iako mnogi veliki jezični modeli, kao što je GPT, koriste samo dekoderski dio za zadatke poput generiranja teksta (Vaswani, i dr., 2017).

Ključne inovacije arhitekture *Transformersa* uključuju:

- *Self-attention mechanism* - ovaj mehanizam omogućuje modelu da odvagane važnost različitih riječi u rečenici u odnosu jedne na drugu.
- *Scaled Dot-Product Attention* - mehanizam *Scaled Dot-Product Attention* izračunava rezultate pažnje uzimajući točkasti produkt upita i ključnih vektora i skalirajući ih obrnutim kvadratnim korijenom dimenzije ključnih vektora.
- *Positional Encoding* - pozicijsko kodiranje se dodaje ulaznim ugrađivanjima (engl. *embeddings*) kako bi omogućilo da učinkovito koriste informacije o redoslijedu, budući da *Transformers* ne obrađuju sekvence na linearan način.
- *Multi-Head Attention* - ova komponenta omogućuje modelu da se usredotoči na različite dijelove rečenice istovremeno, poboljšavajući njegovu sposobnost razumijevanja konteksta iz više perspektiva.
- Arhitektura koder-dekoder – *Transformers* model slijedi arhitekturu koder-dekoder. Koder obrađuje ulaznu sekvencu i generira prikaz temeljen na pažnji, koju dekoder zatim koristi za generiranje izlazne sekvence. Svaki sloj koder i dekoder sastoji se od niza identičnih slojeva, od kojih svaki sadrži *Multi-head attention* mrežu za prijenos podataka naprijed.
- *Feed-Forward* mreže - *Transformers* arhitektura uključuje potpuno povezane *feed-forward* mreže primijenjene neovisno na svaku poziciju. Te se mreže sastoje od dvije linearne transformacije s ReLU aktivacijom između (Vaswani, i dr., 2017).

2.3. Evolucija LLM-a

Nakon uspjeha *Transformers* arhitekture, razvijeno je nekoliko iteracija LLM-ova, od kojih svaka pomiče granice onoga što ti modeli mogu postići. Ovaj odjeljak detaljno opisuje tri istaknute obitelji LLM-a: GPT, LLaMA i PaLM (Minaee i sur. 2024).

2.3.1. GPT (engl. *Generative Pre-trained Transformer*)

GPT (engl. *Generative Pre-trained Transformers*) je veliki jezični model temeljen na dubokom učenju, koji koristi dekodirajuću arhitekturu temeljenu na *Transformersima*. Njegova je svrha obraditi tekstualne podatke i generirati tekstualni izlaz koji nalikuje ljudskom jeziku. Glavne značajke GPT modela su: *Generative*, *Pre-Trained* i *Transformers*. *Generative* značajka naglašava sposobnost modela da generira tekst razumijevanjem i odgovaranjem na dati uzorak teksta. GPT modeli treniraju se korištenjem autoregresivnog jezičnog modeliranja, gdje se modeli unose nizom riječi, a model pokušava pronaći najprikladniju sljedeću riječ koristeći distribucije vjerojatnosti za predviđanje najvjerojatnije riječi ili izraza. Prethodno treniran (engl. *Pre-trained*) značajka odnosi se na model strojnog učenja koji je prethodno treniran na velikom skupu podataka primjera prije nego što je implementiran za određeni zadatak. U slučaju GPT-a, model se trenira na opsežnom korpusu tekstualnih podataka korištenjem pristupa učenja bez nadzora. *Transformers* predstavlja vrstu arhitekture neuronske mreže, koja je opisana prethodno (Yendur, i dr., 2024).

Sada će se proći kroz evoluciju GPT modela od GPT-1 do trenutno najnovijeg modela.

2.3.1.1 GPT-1

Uveden od strane OpenAI-ja, GPT-1 je bio prvi model koji je mogao čitati tekst i odgovoriti na upite. Što se tiče dizajna, GPT-1 koristi *Transformers* arhitekturu dekodera od 12 slojeva sa sustavom samoopažanja za trening. GPT je prethodno treniran na velikom skupu podataka *BooksCorpus* za učenje općih jezičnih značajki, praćen finim podešavanjem specifičnih zadataka radi prilagodbe određenim aplikacijama. Ova sposobnost je pokazala da se modeliranje generativnog jezika može koristiti za generalizaciju modela u kombinaciji s uspješnim prethodnim treniranjem (Yendur, i dr., 2024).

2.3.1.2 GPT-2

GPT-2 značajno je povećao broj parametara (1,5 milijardi) i veličinu skupa podataka za treniranje, pokazujući sposobnost modela da generira koherentan i kontekstualno relevantan tekst u različitim temama. Njegovo izdanje potaknulo je rasprave o potencijalu i rizicima snažnih modela umjetne inteligencije, posebno u vezi s njihovom sposobnošću generiranja obmanjujućeg ili štetnog sadržaja (Minaee, i dr., 2024, Yendur, i dr., 2024).

2.3.1.3 GPT-3

Označavajući značajan skok u mogućnostima, GPT-3 sadrži 175 milijardi parametara, što ga čini jednim od najvećih i najmoćnijih jezičnih modela. GPT-3 pokazuje pojavnu sposobnost učenja u kontekstu, što znači da se GPT-3 može primijeniti na sve nizvodne zadatke bez ikakvih ažuriranja gradijenata ili finog podešavanja, sa zadacima i demonstracijama u nekoliko snimaka specificiranih isključivo putem tekstualne interakcije s modelom. Izvedba GPT-3 u različitim NLP zadacima učvrstila je svoj status svestranog i moćnog alata u AI zajednici (Minaee, i dr., 2024 Mehra, 2023).

2.3.1.4 GPT-4:

Najnovija iteracija, GPT-4, nastavlja nadograđivati svoje prethodnike s još više parametara i refiniranih tehnika treniranja. GPT-4 pokazuje poboljšane performanse u razumijevanju i generiranju teksta, s poboljšanjima u područjima kao što su kontekstualno razumijevanje, dosljednost i rukovanje dvosmislenim upitima. Njegovi širi podaci za treniranje i napredne metode finog podešavanja dodatno poboljšavaju njegove mogućnosti (Minaee, i dr., 2024, Mehra, 2023).

2.3.2. LLaMA (engl. *Large Language Model Meta AI*)

Obitelj LLaMA, koju je razvila Meta, usmjerena je na optimizaciju učinkovitosti i performansi modela, posebno za implementaciju u okruženjima s ograničenim resursima. LLaMA modeli dizajnirani su da budu skalabilni i fleksibilni, osiguravajući visoke performanse čak i uz ograničene računalne resurse. To ih čini prikladnima za širi raspon aplikacija, od mobilnih uređaja do velikih implementacija u oblaku. Naglasak na učinkovitosti bez ugrožavanja performansi naglašava potencijal LLaMA-e za široku upotrebu u različitim okruženjima (Minaee i ostali, 2024).

2.3.3. PaLM (engl. *Pathways Language Model*)

PaLM, koji je razvio Google, predstavlja značajan napredak u mogućnostima LLM-a. PaLM koristi novu arhitekturu dizajniranu za učinkovito rukovanje ogromnim količinama podataka. Iskorištava sustav *Pathways*, koji modelu omogućuje generalizaciju na različite zadatke koristeći manje resursa. PaLM je pokazao iznimne performanse u raznim mjerilima, pokazujući svoju sposobnost razumijevanja i generiranja teksta s visokim stupnjem točnosti i kontekstualne relevantnosti. Svestranost i učinkovitost ovog modela čine ga ključnim igračem u sljedećoj generaciji LLM-a (Minaee i ostali, 2024).

2.3.4. Ostali značajni LLM-i

Nekoliko drugih LLM-a dalo je značajan doprinos ovom području:

BERT (engl. *Bidirectional Encoder Representations from Transformers*) je razvio Google. BERT je uveo dvosmjerno uvježbavanje transformatora, poboljšavajući modelovo razumijevanje konteksta iz oba smjera u rečenici. (Toutanova, 2019) T5 (engl. *Text-to-Text Transfer Transformer*) je također razvio Google. T5 uokvirio je sve NLP zadatke kao problem s tekстом u tekst, demonstrirajući svestranost transformatora u različitim aplikacijama (Raffel i ostali, 2019.). Ovaj inovativni pristup omogućuje T5 ujednačeno tretiranje različitih NLP zadataka pretvarajući ih u format teksta u tekst, gdje su i ulaz i izlaz u obliku teksta. To između ostalog uključuje zadatke kao što su prijevod, sažimanje i odgovaranje na pitanja. Objedinjavanjem ovih zadataka u jedan okvir, T5 pojednostavljuje proces treniranja i primjene NLP modela, pokazujući snagu i fleksibilnost transformatorske arhitekture u širokom rasponu izazova obrade jezika. Prilagodljivost i učinkovitost ovog modela učinile su ga značajnom prekretnicom u razvoju velikih jezičnih modela, jačajući potencijal transformatora u pomicanju granica onoga što NLP sustavi mogu postići (Wang, i dr., 2023). LLM-ovi su revolucionirali polje NLP-a, pokazujući izvanredne sposobnosti u razumijevanju i generiranju ljudskog teksta. Iako nude ogroman potencijal, ključno je rješavanje etičkih izazova, izazova povezanih s resursima i tehničkih izazova koje predstavljaju. Tekuća istraživanja i inovacije nastavit će usavršavati ove modele, proširujući njihove primjene i poboljšavajući njihovu usklađenost s ljudskim vrijednostima i potrebama.

3. Primjena velikih jezičnih modela

Veliki jezični modeli mogu se primijeniti u razne svrhe. Neke od njih su obrada prirodnog jezika, generiranje programskog koda, kreativno pisanje i stvaranje sadržaja, obrazovanje i e-učenje, zdravstvo i medicinska istraživanja (Jean Kaddour, 2023). U ovom poglavlju će se objasniti i proći kroz navedene primjene velikih jezičnih modela.

3.1. Obrada prirodnog jezika (engl. *Natural Language Processing*, NLP)

Obrada prirodnog jezika (engl. *Natural language processing*, skraćeno NLP) ključna je domena unutar umjetne inteligencije, fokusirana na interakciju između računala i ljudskog jezika. Ovo polje obuhvaća niz zadataka čiji je cilj omogućiti strojevima da razumiju, interpretiraju i generiraju ljudski jezik na smislen način. Pojava velikih jezičnih modela značajno je revolucionirala NLP, nudeći značajna poboljšanja u raznim zadacima. LLM-ovi su dramatično poboljšali generiranje teksta, omogućujući stvaranje koherentnog i kontekstualno relevantnog teksta. Modeli poput GPT-3 i GPT-4, koje je razvio OpenAI, pokazuju te mogućnosti generiranjem ljudskog teksta s visokom točnosti i razumijevanjem konteksta, što uvelike pomaže aplikacijama u automatiziranoj izradi sadržaja, chatbotovima i virtualnim pomoćnicima (OpenAI i ostali, 2023; Radford i ostali, 2019).

3.1.1. Generiranje teksta

Generiranje teksta uključuje stvaranje koherentnog i kontekstualno relevantnog teksta na temelju danog unosa. Veliki jezični modeli, posebno GPT obitelj, pokazali su izvanredne sposobnosti u ovom području. GPT-3, koji je razvio OpenAI, može generirati tekst sličan ljudskom. Sposobnost modela da razumije kontekst i proizvede različite odgovore postavila je novo mjerilo u ovom području. Na primjer, može generirati eseje, priče, pa čak i tehničke dokumente uz minimalnu ljudsku intervenciju, pokazujući potencijal LLM-a u automatizaciji stvaranja sadržaja (Brown i ostali, 2020).

3.1.2. Strojno prevođenje

Strojno prevođenje je proces u kojem računalni program analizira tekst na jednom jeziku (polazni tekst), i potom proizvodi tekst na drugom jeziku, bez učešća čovjeka u tom procesu (Wikipedia, 2020). Tradicionalno se ovaj zadatak oslanjao na statističke metode i pristupe temeljene na pravilima, što je često rezultiralo neurednim i netočnim prijevodima. Uvođenje neuronskih mreža i, u novije vrijeme, *Transformers* modela kao što su GPT-3 i BERT, dramatično je poboljšalo kvalitetu prijevoda. Ovi modeli mogu uhvatiti nijanse jezika, rukovati idiomatskim izrazima bolje od svojih prethodnika (Vaswani, i dr., 2017) (Toutanova, 2019). Napredak u strojnom prevođenju omogućen velikim jezičnim modelima otvorio je nove mogućnosti za međujezičnu komunikaciju i razumijevanje, čineći ih nezamjenjivim alatima u globalnoj komunikaciji i višejezičnim aplikacijama.

3.1.3. Sažimanje teksta

Sažimanje teksta ima za cilj sažimanje dugih dokumenata u kraće verzije uz očuvanje izvornog značenja. Dva su primarna pristupa: ekstraktivno (engl. *Extractive*) i apstraktno sažimanje. Ekstraktivno sažimanje odabire ključne rečenice izravno iz teksta, dok apstraktno sažimanje generira nove rečenice koje sažimaju glavne ideje dokumenta. LLM-i, posebno modeli poput BART-a, izvrsni su u obje metode. BART je pokazao najsuvremeniju izvedbu učinkovitim razumijevanjem i prepisivanjem sadržaja kako bi proizveo sažetke. Ove su mogućnosti osobito korisne u područjima kao što su novinarstvo, pregled pravnih dokumenata i akademska istraživanja, gdje je brz pristup bitnim informacijama od ključne važnosti (Jacob Murel Ph.D., 2024)(Lewis i ostali, 2020.).

3.1.4. Analiza osjećaja (engl. *Sentiment Analysis*)

Analiza osjećaja (engl. *sentiment analysis*) uključuje određivanje emocionalnog tona iza dijela teksta. Naširoko se koristi u praćenju društvenih medija, analizi povratnih informacija kupaca i istraživanju tržišta. Veliki jezični modeli poboljšavaju analizu osjećaja točnim tumačenjem suptilnih nijansi i konteksta unutar teksta. Modeli kao što je RoBERTa (engl. *Robustly Optimized BERT Approach*) fino su podešeni na velikim skupovima podataka kako bi se poboljšala njihova točnost u zadacima klasifikacije sentimenta. Ovi modeli mogu razlikovati pozitivne, negativne i neutralne osjećaje s visokom preciznošću, pružajući

vrijedne uvide za tvrtke i istraživače koji žele procijeniti javno mnijenje ili zadovoljstvo kupaca (Yinhan Liu, 2019) (Lei Zhang, 2018).

3.1.5. Prepoznavanje imenovanog entiteta

Prepoznavanje imenovanih entiteta (engl. *Named Entity Recognition*, NER) proces je identificiranja i klasificiranja vlastitih imenica (imena, organizacija, lokacija itd.) unutar teksta. LLM-ovi su značajno poboljšali točnost i učinkovitost NER sustava. Na primjer, biblioteka SpaCy, koja integrira modele transformatora, pokazala je značajna poboljšanja u identificiranju i kategoriziranju entiteta u različitim domenama. Precizan NER ključan je za sustave za traženje informacija, zadatke odgovaranja na pitanja i konstrukciju grafikona znanja (Takyar, 2024).

3.1.6. Klasifikacija teksta

Klasifikacija teksta uključuje kategorizaciju teksta u unaprijed definirane kategorije. Ovaj je zadatak temeljan za razne aplikacije, uključujući otkrivanje neželjene pošte, kategorizaciju tema i analizu mišljenja. LLM poboljšavaju klasifikaciju teksta koristeći svoje duboko razumijevanje jezičnih nijansi i konteksta. Modeli poput XLNet i ALBERT pokazali su značajna poboljšanja u zadacima klasifikacije, pružajući precizne i pouzdane rezultate (EITICA, 2023).

3.1.7. Odgovaranje na pitanje

Sustavi s odgovorima na pitanja pružaju precizne odgovore na upite korisnika na temelju danog teksta ili šire baze znanja. LLM kao što su BERT i GPT-3 veoma dobro su podešeni za pitanje-odgovor zadatke, pokazujući svoju sposobnost razumijevanja i generiranja točnih odgovora. BERT se posebno ističe u razumijevanju konteksta i dohvaćanju relevantnih informacija iz teksta, što ga čini moćnim alatom za razvoj inteligentnih tražilica i virtualnih pomoćnika. (Ojokoh & Adebisi, 2019)

3.1.8. Agenti za razgovor (engl. *Chatbot*)

Na razvoj *chatbota*, agenata za razgovor, uvelike je utjecao napredak u LLM-u. Ovi modeli omogućuju chatbotovima da se uključe u prirodnije i kontekstualno relevantnije razgovore s korisnicima. Primjer toga je ChatGPT, koji može održavati koherentne i smislene dijaloge,

što ga čini prikladnim za korisničku službu, virtualnu pomoć i druge interaktivne aplikacije. Sposobnost ovih modela da generiraju ljudske odgovore otvorila je nove puteve za poboljšanje korisničkog iskustva na različitim digitalnim platformama (Medium, 2023).

3.2. Generiranje i analiza koda

Na području generiranja i analize koda veliki jezični modeli imali su značajan utjecaj. Ovi modeli iskorištavaju ogromne količine podataka prilikom treniranja za generiranje, dovršavanje i analizu koda, donoseći novu razinu automatizacije i inteligencije razvoju softvera. U ovom odjeljku će se proći kroz ključna područja u kojima su LLM-ovi bili posebno utjecajni, neki od kojih su: dovršavanje koda, otkrivanje i popravljavanje grešaka te dokumentacija i sažimanje koda (Mazoor, 2024).

3.2.1. Dovršavanje koda (engl. *Code Completion*)

Dovršavanje koda, poznato i kao automatsko dovršavanje koda, značajka je koja predviđa sljedeći niz koda na temelju trenutnog konteksta, pomažući programerima da učinkovitije pišu kod. Tradicionalni alati za dovršavanje koda uvelike su se oslanjali na podudaranje uzoraka i sintaktičku analizu. Međutim, uvođenje LLM-a, posebice modela poput OpenAI-jevog Codexa, značajno je poboljšalo mogućnosti ovih alata. Codex, model koji stoji iza GitHub Copilota, treniran je na različitim programskim jezicima i može razumjeti i generirati isječke koda s izuzetnom točnošću. Nudi prijedloge u kontekstu (engl. *context-aware suggestions*) koji nadilaze jednostavnu sintaksu i uključuju složenu logiku i biblioteke specifične za razvojno okruženje. Ova sposobnost ne samo da ubrzava kodiranje, već također pomaže programerima da se pridržavaju najboljih praksi i učenja za nove jezike i okvire (Chen i ostali, 2021).

3.2.2. Otkrivanje i ispravljanje pogrešaka

Otkrivanje i ispravljanje pogrešaka kritični su aspekti razvoja softvera koji često zahtijevaju značajno vrijeme i stručnost. LLM-ovi su pokazali obećanje u automatizaciji ovih zadataka identificiranjem potencijalnih grešaka i predlaganjem popravaka. Modeli poput BERT-a i GPT-3, kada su podešeni za analizu koda, mogu detektirati anomalije i obrasce koji ukazuju na greške (Chen i ostali, 2021). LLM modeli mogu se osposobiti za prepoznavanje uobičajenih programskih pogrešaka i davanje prijedloga za ispravljanje. Ovi modeli

analiziraju kontekst koda i povijest popravljavanja grešaka kako bi predvidjeli i riješili probleme. Ovaj pristup ne samo da povećava točnost otkrivanja pogrešaka, već također smanjuje vrijeme potrebno za uklanjanje pogrešaka, čime se poboljšava ukupna kvaliteta softvera. Sličan model razvio je Facebook AI Research nazvan Aroma (Sifei Luan, 2019), koji koristi duboko učenje za preporučivanje promjena koda analizom uzoraka u velikim bazama koda. Napredni LLM-ovi mogu pomoći u razumijevanju i popravljanju složenijih logičkih pogrešaka koje nije lako otkriti tradicionalnim alatima za statičku analizu. Razumijevanjem namjere i logike koda, ovi modeli mogu predložiti smislene ispravke, čineći proces otklanjanja pogrešaka učinkovitijim i manje sklonim pogreškama (Miltiadis Allamanis, 2018).

3.2.3. Dokumentacija koda i sažetak

Dokumentacija koda ključna je za održavanje i razumijevanje softvera, nekada i zadatak koji programeri mogu zanemariti. LLM nudi rješenje automatskim generiranjem dokumentacije i sažetaka za baze kodova. Ovi modeli mogu analizirati kod i proizvesti čovjeku čitljive opise, olakšavajući programerima razumijevanje i rad s velikim i složenim bazama kodova. Modeli poput BART-a učinkovito su korišteni za sažimanje teksta i mogu se prilagoditi za dokumentaciju koda. BART može generirati sveobuhvatne opise funkcija koda, klasa i modula razumijevanjem njihove svrhe i funkcionalnosti (Yang Liu, 2019). Ova je sposobnost posebno vrijedna u projektima otvorenog koda i okruženjima za suradnju gdje je jasna dokumentacija ključna za učinkovit timski rad. Osim toga, alati kao što je GitHub Copilot (Friedman, 2022) mogu generirati ugrađene komentare i nizove dokumenata dok programeri pišu kod. Ova pomoć u stvarnom vremenu osigurava da je dokumentacija ažurna i relevantna, smanjujući opterećenje programera i poboljšavajući čitljivost koda.

Integracija velikih jezičnih modela u zadatke generiranja koda i analize značajno je poboljšala učinkovitost i točnost procesa razvoja softvera. Od dovršavanja koda i otkrivanja grešaka do dokumentacije i sažimanja, LLM-ovi su pokazali svoj potencijal za automatizaciju i poboljšanje različitih aspekata kodiranja.

3.3. Kreativno pisanje i stvaranje sadržaja

LLM-ovi trenirani na različitim skupovima podataka mogu generirati priče, pjesme, marketinške tekstove i oglase s razinom sofisticiranosti koja blisko oponaša ljudsku kreativnost. Ovaj odjeljak istražuje kako LLM-ovi transformiraju kreativno pisanje kroz generiranje priča i pjesama te njihov utjecaj na marketinški tekst i stvaranje oglasa.

3.3.1. Generacija priča i pjesama

Jedna od fascinantnijih primjena LLM-a je njihova sposobnost generiranja kreativnih tekstova kao što su priče i pjesme. Ovi modeli mogu proizvesti narative koji su koherentni, zanimljivi i stilski raznoliki, što ih čini vrijednim alatima za pisce i pripovjedače. Veliki jezični modeli, kao što je OpenAI GPT-3 i njegovi nasljednici, pokazali su izvanrednu vještinu u generiranju priča iz upita (Zhuohan Xie, 2023). Ovi modeli mogu stvoriti zamršene zaplete, razviti likove i održati dosljednost pripovijedanja tijekom dugih odlomaka. Na primjer, GPT-3 može generirati kratku priču na temelju nekoliko rečenica koje je dao korisnik, nastavljajući pripovijest na način koji je u skladu s početnim stilom i tonom. Ova je mogućnost posebno korisna za razmišljanje, prevladavanje spisateljske blokade i istraživanje različitih narativnih mogućnosti (Eric Nichols, 2020). Slično tome, LLM-ovi su pokazali sposobnost stvaranja poezije. Ovi modeli mogu oponašati različite poetske forme, uključujući sonete, haikue i slobodne stihove, razumijevanjem i repliciranjem strukture, ritma i tema tipičnih za te stilove. Na primjer, GPT-3 može proizvesti sonet koji se pridržava tradicionalnog jambskog pentametra i sheme rime, sve iz jednostavne tematske upute (Lamar i Chambers, 2019).

3.3.2. Marketinški tekst i oglašavanje

Utjecaj LLM-a na marketing i oglašavanje je velik, jer ti modeli mogu generirati uvjerljiv i prilagođen sadržaj koji se sviđa određenoj publici. Ova primjena ne samo da povećava produktivnost, već također omogućuje viši stupanj personalizacije u marketinškim proizvodima. (Milton-Barker, 2024.) Na primjer, GPT-3 se može koristiti za izradu više verzija opisa proizvoda, od kojih je svaka prilagođena različitim segmentima publike. Ova personalizacija se postiže putem upita koji određuju ton, stil i ključne poruke koje treba prenijeti. Sposobnost modela da razumije i generira sadržaj koji odgovara glasu brenda osigurava dosljednost u različitim marketinškim kanalima. (Milton-Barker, 2024.)

3.3.3. Izrada oglasa

U području oglašavanja, LLM-ovi mogu generirati sadržaj koji privlači pozornost i potiče angažman. LLM-ovi pružaju kreativnu osnovu za učinkovito oglašavanje. Korištenje LLM-ova u oglašavanju također se proširuje na automatizirano generiranje sadržaja za dinamičke oglase. Takvi oglasi mogu mijenjati svoj sadržaj na temelju korisničkih interakcija ili preferencija, pružajući personalizirano iskustvo koje povećava angažman i stope konverzije. (Scripted, 2023)

Primjena velikih jezičnih modela u kreativnom pisanju i stvaranju sadržaja predstavlja značajan napredak u našem pristupu tim zadacima. Od stvaranja uvjerljivih priča i pjesama do izrade uvjerljivog marketinškog teksta i oglasa, LLM nudi alate koji povećavaju kreativnost, učinkovitost i personalizaciju.

3.4. Obrazovanje i e-učenje

Primjena velikih jezičnih modela (LLM) u obrazovanju i e-učenju revolucionarizira način na koji se znanje prenosi i stječe. Ovi modeli trenirani su na opsežnim skupovima podataka koji obuhvaćaju širok raspon predmeta i stilova. Oni su sposobni su pružiti automatizirano podučavanje, generirati obrazovni sadržaj i personalizirati iskustva učenja. Ovaj odjeljak bavi se specifičnim ulogama LLM-ova u sustavima automatiziranog podučavanja i generiranju obrazovnog sadržaja. (Wang, i dr., 2024, Gan, Qi, Wu, & Lin, 2023)

3.4.1. Personalizirane upute

LLM-ovi mogu analizirati studentove odgovore i obrasce učenja kako bi se prilagodila nastava. Ova prilagodljivost osigurava da učenici dobiju pravu razinu izazova i podrške, potičući učinkovitiji proces učenja. U automatiziranom sustavu podučavanja, LLM također mogu ponuditi trenutne povratne informacije o zadacima i kvizovima. Analizom točnosti odgovora može dati detaljnu povratnu informaciju, ističući pogreške i predlažući poboljšanja. Ovim neposrednim primanjem povratnih informacija učenicima pomaže da uče na svojim pogreškama. (Gan, Qi, Wu, & Lin, 2023)

3.4.2. Izrada udžbenika i priručnika

LLM mogu izraditi sveobuhvatne udžbenike i vodiče za učenje prilagođene različitim obrazovnim razinama i predmetima. Obradujući ogromne količine informacija, ovi modeli mogu sintetizirati sadržaj u koherentna poglavlja, odjeljke i sažetke. Na primjer, LLM može izraditi udžbenik biologije koji pokriva stanične strukture, genetiku i ekosustave, zajedno s dijagramima, primjerima i pitanjima za ponavljanje. Uz tradicionalne udžbenike, LLM mogu izraditi specijalizirane materijale za učenje koji se fokusiraju na određene ispite ili kurikulume. Ovi materijali mogu uključivati pitanja za vježbanje, savjete za učinkovito učenje i detaljna objašnjenja ključnih pojmova, pomažući učenicima da se učinkovitije pripreme. (Wang, i dr., 2024)

3.4.3. Interaktivne vježbe i kvizovi

LLM mogu osmisлити interaktivne vježbe i kvizove koji odgovaraju različitim stilovima učenja i razinama težine. Ove vježbe mogu varirati od pitanja s višestrukim izborom i aktivnosti popunjavanja praznina do složenih zadataka rješavanja problema i otvorenih pitanja. Omogućavajući različite vrste pitanja, LLM podržavaju različite potrebe učenja i potiču aktivno sudjelovanje. Također, LLM mogu generirati trenutne povratne informacije i savjete za svako pitanje, vodeći studente kroz proces učenja. Na primjer, ako učenik netočno odgovori na matematički problem, sustav može ponuditi savjete korak po korak kako bi im pomogao da dođu do ispravnog rješenja, jačajući njihovo razumijevanje. (Wang, i dr., 2024)

3.4.4. Multimedijски obrazovni resursi

Osim tekstualnog sadržaja, LLM mogu pomoći u stvaranju multimedijских obrazovnih resursa, uključujući videozapise, animacije i interaktivne simulacije. Generiranjem skripti, naslova i opisa, LLM olakšavaju proizvodnju zanimljivog vizualnog sadržaja koji poboljšava razumijevanje. Na primjer, LLM može skriptirati obrazovni video o ciklusu vode kako bi se osigurala točnost i jasnoća, čineći koncept pristupačnijim vizualnim učenicima. (Gan, Qi, Wu, & Lin, 2023)

3.4.5. Prilagodba i lokalizacija

LLM također omogućuju prilagodbu i lokalizaciju obrazovnog sadržaja kako bi se zadovoljile potrebe različitih studentskih populacija. Sadržaj se može prilagoditi tako da

odražava kulturni kontekst, jezične preferencije i regionalne obrazovne standarde. Na primjer, LLM može generirati lekcije iz povijesti koje uključuju lokalne perspektive i primjere, čineći materijal relevantnijim i zanimljivijim za studente u različitim regijama. (Gan, Qi, Wu, & Lin, 2023)

Integracija velikih jezičnih modela u obrazovanje i e-učenje mijenja način na koji se znanje isporučuje i konzumira. Putem automatiziranih sustava podučavanja, LLM-ovi pružaju personalizirana, interaktivna i prilagodljiva iskustva učenja, povećavajući angažman i postignuća studenata. U stvaranju obrazovnog sadržaja, ovi modeli proizvode raznolike, kvalitetne materijale koji zadovoljavaju različite potrebe i sklonosti učenja. Kako LLM-ovi nastavljaju napredovati, očekuje se da će njihov utjecaj na obrazovanje rasti.

3.5. Zdravstvo i medicinska istraživanja

Integracija velikih jezičnih modela (LLM) u zdravstvu i medicinskim istraživanjima stvara revolucionarni napredak. Ovi modeli, sa svojom sposobnošću obrade i analize velikih količina podataka, povećavaju učinkovitost i točnost analize medicinskih dokumentacija, otkrivanja lijekova i generiranja istraživačkih radova. Ovaj odjeljak bavi se specifičnim primjenama LLM-ova u tim područjima, ističući njihov utjecaj i potencijal.

3.5.1. Analiza medicinske dokumentacije

Medicinska dokumentacija sadrži mnoštvo informacija ključnih za njegu pacijenata i medicinska istraživanja. Međutim, sama količina i složenost ovih zapisa predstavljaju značajne izazove. LLM-ovi, trenirani na opsežnim skupovima medicinskih podataka, revolucioniraju analizu medicinskih zapisa automatizirajući ekstrakciju, obradu i interpretaciju podataka. LLM-ovi mogu izvući relevantne informacije iz nestrukturirane medicinske dokumentacije, uključujući kliničke bilješke, laboratorijske rezultate i radiološka izvješća. Ova mogućnost usmjerava proces sastavljanja sveobuhvatne povijesti pacijenata, što je bitno za točnu dijagnozu i planiranje liječenja. Na primjer, LLM može identificirati lijekove, alergije i prošle medicinske postupke iz kartona pacijenta, objedinjujući ih u strukturirani format. (Chen, i dr., 2024)

3.5.2. Prediktivna analitika

LLM-ovi mogu se koristiti za predviđanje ishoda pacijenata na temelju povijesnih podataka. Analizirajući trendove i obrasce u medicinskoj dokumentaciji, ovi modeli mogu predvidjeti vjerojatnost ponovnog prijema, komplikacija ili ponavljanja bolesti. Ova mogućnost predviđanja podržava proaktivno upravljanje zdravstvenom skrbi, omogućujući pružateljima usluga provedbu preventivnih mjera i personaliziranih planova liječenja. Na primjer, LLM može predvidjeti rizik ponovnog prijema za pacijenta sa zatajenjem srca, omogućujući kliničarima da poduzmu preventivne korake za ublažavanje tog rizika. (Chen, i dr., 2024)(Nazi & Peng, 2024)

3.5.3. Otkrivanje lijekova i stvaranje istraživačkih radova

Proces otkrivanja lijeka veoma je dugotrajan i skup, često zahtijeva više od desetljeća i milijarde dolara da se novi lijek dovede na tržište. LLM transformiraju ovo područje ubrzavajući različite faze procesa otkrivanja lijekova i pomažući u stvaranju istraživačkih radova koji šire nova otkrića. LLM-ovi mogu značajno ubrzati početne faze otkrivanja lijekova, kao što su identifikacija cilja i validacija, obradom i analizom golemih skupova podataka bioloških i kemijskih informacija. Ovi modeli mogu identificirati potencijalne mete za lijekove filtriranjem složenih biomedicinskih podataka, predviđanjem molekularnih interakcija i sugeriranjem novih spojeva s terapijskim potencijalom. Na primjer, modeli temeljeni na BERT-u fino su podešeni za razumijevanje biomedicinske literature, omogućujući izdvajanje relevantnih podataka i hipoteza iz znanstvenih tekstova. (Lee, i dr., 2019)

Također, LLM olakšavaju izradu i generiranje istraživačkih radova automatiziranjem sinteze informacija iz različitih izvora. Ova automatizacija pomaže istraživačima da ostanu u tijeku s najnovijim razvojem i spoznajama na terenu, ubrzavajući širenje znanja i potičući suradnju između različitih istraživačkih timova. Na primjer, modeli poput SciBERT-a korišteni su za sažimanje i generiranje sveobuhvatnih pregleda znanstvene literature, osiguravajući da su kritične informacije lako dostupne znanstvenoj zajednici. (Beltagy, Lo, & Cohan, 2019)

Primjena LLM-a u zdravstvu i medicinskom istraživanju uvode u novu eru učinkovitosti i inovacija. Automatizirajući analizu medicinskih zapisa, LLM poboljšavaju skrb o pacijentima kroz precizno izdvajanje podataka, razumijevanje prirodnog jezika i prediktivnu analitiku. U otkrivanju lijekova i stvaranju istraživačkih radova, ovi modeli ubrzavaju razvoj

novih terapija i olakšavaju širenje znanstvenih spoznaja. Kako se LLM-ovi nastavljaju razvijati, očekuje se da će se njihov utjecaj na zdravstvenu skrb i medicinska istraživanja proširiti, nudeći još sofisticiranije alate za poboljšanje ishoda pacijenata i unaprjeđenje medicinske znanosti.

4. Metodologija

U ovom poglavlju opisati ćemo metodologiju istraživanja korištenu u ovom istraživanju, detaljno opisujući pristup, istraživačka pitanja, metode prikupljanja podataka i korištene tehnike analize. Primarni fokus ovog istraživanja je procijeniti i usporediti izvedbu fino podešenog (engl. *Fine tune*) modela za procjenu izvornog programskog koda u odnosu na dobro razvijene jezične modele kao što su ChatGPT-4 i Code Llama. Usporedba uključuje ručnu procjenu različitih rezultata kroz različite zadatke kako bi se odredile prednosti i slabosti svakog modela. Istraživanje usvaja komparativni eksperimentalni pristup. To uključuje treniranje postojećeg velikog jezičnog modela na određenom skupu podataka kako bi se fino prilagodio (engl. *fine tune*) za ciljane zadatke i usporedbu njegove izvedbe s već postojećim, korištenim modelima kao što su ChatGPT-4 i Code Llama. Proces finog podešavanja usmjeren je na optimizaciju modela za specifičnu primjenu, pružajući osnovu za procjenu poboljšanja u odnosu na jezične modele opće namjene.

4.1. Istraživačka pitanja

Ovo istraživanje vodi se slijedećim istraživačkim pitanjima:

1. Može li se postojeći veliki jezični model fino prilagoditi (engl. *fine tune*) da bi se koristio u svrhe detaljne procijene izvornog programskog koda?
2. Ima li prilagođeni model bolje performanse od već uspostavljenih velikih jezičnih modela?

4.2. Podaci korišteni za transfer učenja

Skup podataka koji je korišten za transfer učenja sastoji se od parova kod (engl. *code*, naziv korišten prilikom provođenja istraživanja) i procjena (engl. *assessment*, naziv korišten prilikom provođenja istraživanja). Stupac kod je ulazni podatak, koji sadrži isječke programskog koda napisan u JavaScript programskom jeziku. Ovi uzorci koda uključuju jednostavne metode i algoritme kao što je pronalaženje najvećeg broja u nizu i izvođenje osnovnih aritmetičkih operacija poput zbrajanja, oduzimanja i dijeljenja. Stupac procjena, izlazni podatak, daje odgovarajuću procjenu za svaki isječak koda. Ova procjena uključuje naznaku je li kod točan ili netočan, detaljno objašnjenje razloga koji stoje iza ove procjene i prijedloge za poboljšanje i daljnju praksu. Detaljne povratne informacije imaju za cilj

usmjeriti programere u razumijevanju svojih pogrešaka i učenju najboljih praksi kodiranja. Skup podataka odabran je iz kombinacije izvora, uključujući vlastite autorske primjere i isječke koda prikupljene s različitih web stranica za programiranje kao što su *FreeCodeCamp*, *GeeksforGeeks* i *Codecademy*. Ovi primjeri su odabrani i modificirani kako bi uključili ispravne implementacije i uobičajene pogreške, što je pružilo osnovu za generiranje procjena tijekom finog podešavanja modela. Na slici 1 je prikazan primjer prvih pet parova koda i procjene. Kako bi se osigurao raznolik skup primjera, pripremljeni su podaci od početnih do naprednih razina, pokrivajući široki spektar programskih konstrukcija i logičkih obrazaca. Neki su isječci koda namjerno izmijenjeni kako bi uključili sintaktičke pogreške, logičke pogreške i druge uobičajene pogreške programera. Prije korištenja skupa podataka za treniranje velikog jezičnog modela, podaci su prošli nekoliko koraka predobrade. Ti koraci su uključivali: standardizaciju formata koda, kako bi se osigurala dosljednost u svim primjerima, ručno pregledavanje i označavanje isječaka koda, kako bi se osiguralo da je svaka pogreška ispravno identificirana i objašnjena, te osiguravanje uravnoteženog skupa podataka s jednakom zastupljenošću različitih vrsta pogrešaka i ispravnih implementacija kako bi se spriječila bilo kakva pristranost u procesu učenja modela.

index	code
0	<pre>function addNumbers(num1, num2) { return num1 + num2; } var result = addNumbers("10", "20"); console.log("The result is:" + result);</pre>
1	<pre>function addNumbers(num1, num2) { return num1 + num2; } var result = addNumbers(10, 20); console.log("The result is: " + result);</pre>
2	<pre>class Stack { constructor() { this.items = []; } push(element) { this.items.push(element); } pop() { if (this.isEmpty()) return "Underflow"; return this.items.pop(); } isEmpty() { return this.items.length === 0; } } const stack = new Stack(); console.log(stack.pop());</pre>
3	<pre>function updateUserProfile(userId, userData) { fetch("https://api.example.com/users/\${userId}", { method: 'POST', headers: { 'Content-Type': 'application/json' }, body: JSON.stringify(userData), }) .then(response => response.json()) .then(data => console.log(data)) .catch(error => console.error('Error:', error)); } updateUserProfile(123, {name: 'John Doe', email: 'john.doe@example.com'});</pre>
4	<pre>function mergeArrays(arr1, arr2) { return arr1.concat(arr2).sort((a, b) => a - b); } console.log(mergeArrays([3, 1, 4], [2, 5, 6]));</pre>

Slika 1 Primjer skupa podataka - *Code* stupac

This code contains mistakes due to a mismatch in data types. The function `addNumbers` is designed to take numerical inputs but is called with string arguments. As a result, JavaScript will perform string concatenation instead of numeric addition, leading to incorrect behavior. To fix this, convert the strings to numbers using `parseFloat()` or `Number()` before adding. Additionally, ensure that the code is robust by validating input types. Review type conversion and implicit coercion in JavaScript for better understanding.

This code is correct. The function `addNumbers` properly takes two numerical inputs and returns their sum. The code is concise and efficient, and the result is correctly logged to the console. No errors or issues were found, and the logic demonstrates a clear understanding of JavaScript functions and arithmetic operations. You can experiment further with different types of numbers, such as floats, and ensure your function handles edge cases like NaN or infinity.

This code contains a minor issue with the `pop` method logic. The method checks if the stack is empty, but the response "Underflow" might be ambiguous in real-world applications. It would be more appropriate to handle such cases with an exception or error message explaining the condition, such as "Stack is empty, cannot pop". Additionally, you could improve the method by returning `null` or `undefined` to signify an empty stack. Consider reviewing error handling and edge case management in data structures to improve code resilience.

This code is correct and efficiently handles a typical POST request to update user profile data. The function `updateUserProfile` correctly uses the Fetch API to send the user data in JSON format and handles the response appropriately. Error handling with `catch()` is also well implemented. You could enhance the function further by adding more error handling, for example, checking the HTTP status code before converting the response to JSON. This will make the function more robust in real-world applications.

This code is mostly correct, but it previously used an incomplete sorting approach. By default, JavaScript's `sort()` treats elements as strings. The added comparison function `(a, b) => a - b` correctly sorts numbers in ascending order. The function now concatenates two arrays and sorts the merged array correctly. This demonstrates a clear understanding of array manipulation. Consider also exploring different sorting algorithms and understanding their performance implications for large datasets.

Slika 2 Primjer skupa podataka - *Assessment* stupac

4.3. Tehnike analize

Analiza je usmjerena na kvantitativne i kvalitativne metode procjene za procjenu izvedbe i primjenjivosti jezičnih modela. Kvantitativna procjena oslanja se na uspostavljene metrike kao što su BLEU i ROUGE za mjerenje kvalitete generiranog teksta. (Mark, Jerry, Qiming, Pinto, & Oliveira, 2021) BLEU ili *Bilingual Evaluation Understudy* mjeri sličnost između generiranog teksta i referentnog teksta. BLEU se prvenstveno koristi u strojnom prevođenju, ali je također primjenjiv na bilo koji zadatak generiranja teksta gdje je potrebno procijeniti sličnost između generiranog teksta i referentnog teksta. BLEU izračunava preklapanje n-grama (nizova od n riječi) između generiranog teksta i jednog ili više referentnih tekstova. Rezultat se kreće od 0 do 1, gdje 1 označava savršeno podudaranje, a 0 označava da nema preklapanja. BLEU rezultat bliži 1 označava višu kvalitetu i točnost generiranog teksta. Ocjenjuje preciznost uzimajući u obzir udio n-grama u generiranom tekstu koji su također prisutni u referentnom tekstu. Metrika ROUGE, skraćeno od *Recall-Oriented Understudy for Gisting Evaluation*, mjeri kvalitetu sažetaka koje nastane modelom u usporedbi s referentnim sažetcima. Procjenjuje preklapanje n-grama između generiranog sažetka i referentnih sažetaka. Vrijednost ROUGE metrike proteže se između 0 i 1, gdje veća vrijednost označava bolju kvalitetu generiranog teksta.

Kvalitativna analiza uključuje ljudsku procjenu, kojom se procjenjuje generirani tekst na temelju različitih atributa kao što su koherentnost, kreativnost, relevantnost i točnost. Ovaj pristup pruža nijansiranije razumijevanje praktične primjenjivosti modela u scenarijima stvarnog svijeta. Koherentan tekst trebao bi predstavljati ideje u jasnom i logičnom slijedu, čineći ga lakim za razumijevanje. Koherentnost generiranog teksta procijenjena je provjerom logičnog tijeka i dosljednosti generiranog teksta. Kreativnost mjeri sposobnost modela da

proizvede originalan i inovativan sadržaj. Visoki rezultati kreativnosti pokazuju da model može generirati zanimljiv i privlačan tekst. Relevantnost procjenjuje koliko je dobro generirani tekst usklađen s danim upitom ili kontekstom. Relevantan tekst izravno se bavi temom i ispunjava očekivane ciljeve. Točnost uključuje provjeru činjenične točnosti i preciznosti informacija predstavljenih u generiranom tekstu. Precizan tekst pruža pouzdane i provjerljive informacije.

Kombinacija kvantitativnih i kvalitativnih metoda omogućuje sveobuhvatniju procjenu velikih jezičnih modela. Dok kvantitativne metrike poput BLEU i ROUGE daju objektivne mjere performansi, kvalitativne procjene daju dublji uvid u realnu korisnost modela i korisničko iskustvo.

4.4. Pristup

Kako bi dobili odgovor na prvo istraživačko pitanje - može li se postojeći veliki jezični model fino podesiti za određene svrhe – fino podešavanje (engl. *fine tuning*) provedeno je na *CodeT5+ 220M Bimodal Model* unutar Jupyter Notebook okruženja. Odabrani LLM, CodeT5+, potječe iz Hugging Face-a, dobro poznate platforme za modele strojnog učenja. CodeT5+ dio je obitelji CodeT5 koja se sastoji od velikih modela jezika otvorenog koda posebno dizajniranih za razumijevanje i generiranje programskog koda. Razvijen na arhitekturi kodera-dekoder, CodeT5+ je poboljšana za rad u različitim načinima rada za razne nizvodne zadatke putem mješavine ciljeva, postignutih prethodnim treniranjem na unimodalnim i bimodalnim podacima. CodeT5+ je prethodno treniran na devet programskih jezika: Java, JavaScript, Python, Go, PHP, Ruby, C#, C++ i C. Ova raznolikost u podacima za treniranje omogućuje modelu dobru generalizaciju na različite programske jezike, što ga čini robusnim izborom za fino podešavanje. (Wang i ostali, 2023)

4.4.1. Pripremljeni skupa podataka

Skup podataka koji se koristi za fino podešavanje uključuje ulazne vrijednosti programskog koda i odgovarajuće podatke o procjeni teksta kao izlaz. Ovaj skup podataka pažljivo je odabran kako bi obuhvatio širok raspon programskih zadataka i vrsta pogrešaka kako bi se osiguralo što kvalitetnije učenje za model. Primjeri uključuju tri kategorije programskih zadataka, matematički i algoritamski zadatci, strukture podataka i web programiranje. Kategorija matematičkih i algoritamskih zadataka uključivala je isječke koda koji izvode

numeričke izračune i implementiraju algoritme. Na primjer, jedan isječak uključivao je pronalaženje najvećeg broja u nizu, dok se drugi fokusirao na osnovne aritmetičke operacije kao što su zbrajanje, oduzimanje i dijeljenje. Kategorija strukture podatka uključuje implementaciju stog strukture i jednostavnu implementaciju *Bubble sort* algoritma za sortiranje. Kategorija web programiranja uključuje isječke relevantne za web razvoj, kao što je rukovanje HTML elementima i upućivanje API poziva. Skup podataka podijeljen je na podskupove za treniranje i testiranje u omjeru 80:20. Skup za treniranje, koji se sastoji od 80% podataka, koristi se za treniranje modela, omogućujući mu da nauči obrasce i odnose unutar podataka. Preostalih 20% služi kao skup za testiranje, koji se koristi za procjenu izvedbe modela i mogućnosti generalizacije na novim podacima.

4.4.2. Odabir velikog jezičnog modela

Odabir odgovarajućeg modela velikog jezika za ovo istraživanje uključivao je sveobuhvatnu evaluaciju nekoliko kandidata. Razmotrena su tri LLM-a: Code Llama 7B, ChatGPT-2 i CodeT5+.

Prvi model koji je bio razmatran je Code Llama 7B iz Hugging Face domene. Code Llama je u početku bio privlačan zbog svoje arhitekture i svoje reputacije izvrsnih performansi u zadacima generiranja koda. Međutim, veliki računalni zahtjevi predstavljali su značajan izazov. Konkretno, Code Llama 7B zahtijeva značajnu količinu RAM-a i snažan GPU za učinkovit rad. S obzirom na ograničenja dostupnog hardvera i nepraktičnost pribavljanja dodatnih resursa, na kraju je odlučeno da se ovaj model neće koristiti.

Drugi razmotreni model bio je ChatGPT-2. Dok ChatGPT-2 znatno manje zahtijeva resurse u usporedbi s Code Llama 7B, predstavlja vlastiti niz ograničenja. ChatGPT-2 je ranija i manja verzija serije GPT, kojoj nedostaju napredne mogućnosti njegovih nasljednika, GPT-3 i GPT-4. Nije posebno dizajniran za zadatke kao što su analiza i generiranje koda, što je dovelo do neoptimalnih performansi u našim preliminarnim testovima. Model se borio sa zamršenostima sintakse i logike koda, što ga je činilo neprikladnim za potrebe ovog istraživanja.

Nakon procjene prethodnih modela, odabran je model CodeT5+ kao najprikladnija opcija. CodeT5+ uspostavlja ravnotežu između računalne učinkovitosti i performansi. Posebno je prilagođen za zadatke povezane s kodom, koristeći arhitekturu koder-dekoder za rukovanje podacima. Sposobnost modela da upravlja različitim programskim jezicima i niži računalni

zahtjevi učinili su ga praktičnim izborom za ovo istraživačko okruženje. CodeT5+ prethodno je treniran na različitim skupovima podataka, uključujući više programskih jezika kao što su Java, JavaScript, Python i C++, osiguravajući sveobuhvatno razumijevanje i generiranje koda. Na odluku o korištenju CodeT5+ utjecala je i njegova isplativost u smislu korištenja računalnih resursa. Model se može učinkovito fino podesiti i njime se može upravljati unutar postojećih hardverskih mogućnosti ovog istraživanja.

4.4.3. Koraci pretprocesiranja

Svaki isječak koda u skupu podataka standardiziran je kako bi se osigurala dosljednost u oblikovanju, što je ključno za točno učenje modela. Osim toga, isječci koda su ručno pregledani kako bi se potvrdila ispravnost primjedbi i danih objašnjenja, osiguravajući dosljednost skupa podataka. Zatim, je skup podataka podijeljen na podskupove za treniranje i testiranje u omjeru 80:20. Skup za treniranje, koji se sastoji od 80% podataka, koristi se za treniranje modela, omogućujući mu da nauči obrasce i odnose unutar podataka. Preostalih 20% služi kao skup za testiranje, koji se koristi za procjenu izvedbe modela i mogućnosti generalizacije na nevidljivim podacima. Nakon što je skup podataka pripremljen, podvrgnut je tokenizaciji. Tokenizacija je ključni korak pretprocesiranja u kojem se isječci koda i tekstualne evaluacije pretvaraju u format koji model može razumjeti. Ovaj proces uključuje rastavljanje koda i teksta na manje jedinice, poznate kao tokeni, koje se zatim unose u model. Korišteni tokenizator posebno je dizajniran za model CodeT5+, optimiziran za učinkovito rukovanje programskim jezicima. Ovaj tokenizator ne samo da obrađuje sintaksu programiranja, već također hvata semantičko značenje, omogućujući modelu da razumije i učinkovito generira kod. Ovakva sveobuhvatna priprema skupa podataka, u kombinaciji s prethodnom obradom i uravnoteženim treniranjem, postavlja snažne temelje za postizanje točnih i pouzdanih rezultata u sljedećim fazama evaluacije.

4.4.4. Treniranje modela

Nakon tokenizacije, model je trenirani uz pomoću klase *Trainer* iz biblioteke *Hugging Face Transformers*. Klasa *Trainer* olakšava proces treniranja upravljanjem različitim aspektima kao što su učitavanje podataka, optimizacija i evaluacija. Sljedeći argumenti postavljeni su *Trainer* klasi:

- Model: unaprijed trenirani model CodeT5+.

- Argumenti treniranja: Konfiguracijski parametri koji definiraju proces treniranja, kao što su stopa učenja, veličina serije i broj epoha.
- Podaci o treniranju: Tokenizirani skup treniranih podataka
- Izračun metrike: Funkcije za izračunavanje metrike procjene, kao što su rezultati BLEU i ROUGE.
- Podaci testiranja: Tokenizirani skup podataka testiranja koji se koristi za procjenu izvedbe modela.

Postavljanje optimalnih argumenata treninga ključan je faktor koji utječe na rezultate treninga. Ključni parametri uključuju: veličinu seriju (engl. *batch size*), stopa učenja (engl. *learning rate*) i broj epoha treniranja (engl. *number of epochs*). Veličina serije određuje broj primjera treniranja koji se koriste u jednoj iteraciji. Manja veličina serije koristi manje memorije i omogućuje češće ažuriranje težine modela, što potencijalno dovodi do bolje konvergencije. Suprotno tome, veća veličina serije pruža stabilnije procjene gradijenta i brže uvježbavanje po epohi, ali zahtijeva više memorije i može dovesti do suboptimalne konvergencije ako je prevelika (mljourney, 2024). Stopa učenja određuje veličinu koraka u svakoj iteraciji dok se kreće prema minimumu funkcije gubitka. Niska stopa učenja dovodi do stabilnije i preciznije konvergencije, ali zahtijeva više epoha da se postigne optimum. Visoka stopa učenja ubrzava treniranje, ali može premašiti minimume, potencijalno dovodeći do nestabilnog treniranja i slabe konvergencije. (Medium, Medim, 2024) Broj epoha je broj puta kada je cijeli skup podataka za treniranje prošao kroz model. Manji broj epoha može dovesti do nedovoljnog prilagođavanja kada model nije dovoljno naučio iz podataka o treniranju, dok veliki broj epoha može rezultirati prekomjernim prilagođavanjem (pretreniranje) gdje model uči šum i detalje iz podataka o treniranju, smanjujući njegovu izvedbu na nevidljivim podacima. (Rsvmukhesh, 2023)

4.4.5. Hiperparametarsko podešavanje (engl. *Hyperparameter tuning*)

Na performanse fino podešenog modela CodeT5+ uvelike utječu hiperparametri, argumenti odabrani tijekom treninga, kao što je navedeno u prethodnome odjeljku. Da se odaberu najbolji hiperparametri koristi se proces hiperparametarskog podešavanja (engl. *Hyperparameter tuning*). Podešavanje hiperparametara provedeno je pomoću Optune, okvira za optimizaciju hiperparametara koji omogućuje učinkovito pretraživanje optimalnih postavki. Ključni hiperparametri uključuju: stopu učenja (engl. *learning rate*), veličina

serije (engl. *batch size*) i broj epoha (engl. *number of epochs*). Proces je uključivao sljedeće korake: definiranje funkcije cilja, izrada i pokretanje studije i dohvaćanje najboljih hiperparametara. Ciljna funkcija enkapsulira proces treniranja modela i vraćanje metrike procjene koju koristi Optuna za usporedbu različitih skupova hiperparametara. Objekt studije upravlja procesom optimizacije, izvodeći funkciju cilja više puta s različitim vrijednostima hiperparametara. Optuna automatizira traženje optimalnih hiperparametara, osiguravajući sveobuhvatno istraživanje prostora hiperparametara. Nakon završetka procesa optimizacije, najbolji skup hiperparametara je dohvaćen i korišten za konačno treniranje. Praktična implementacija ovog pristupa osigurava da je proces treniranja podataka optimiziran za najbolje performanse fino podešenog modela CodeT5+. Nakon finog podešavanja hiperparametara, model se uvježbava do kraja, a regulirajuća izvedba se procjenjuje kako bi se odredila učinkovitost optimizacije. Ovaj metodički pristup pomaže u usavršavanju mogućnosti modela, čineći ga prikladnijim za specifične programerske zadatke i scenarije otkrivanja pogrešaka. (Najib, 2024)

Kako bi se dobio odgovor na drugo istraživačko pitanje - ima li prilagođeni model bolju izvedbu od uspostavljenih velikih jezičnih modela - provela se komparativna procjena. To je uključivalo testiranje fino podešenog modela CodeT5+ u odnosu na Chat GPT-4 i Code Llama koristeći iste ulazne podatke. Proces je uključivao sljedeće korake: priprema koda za unos, unos podataka u svaki model, usporedba izlaza.

Za ovu procjenu odabrana su četiri različita isječka koda, svaki napisan u JavaScriptu, kako bi se osiguralo ujednačeno testiranje. Kao što je opisano u prethodnom odijeljku veliki jezični model CodeT5+ je fino podešen i treniran na podacima isječaka koda iz različitih kategorija zadataka. Time su odabrane kategorije za testiranje: matematički i algoritamski zadaci, koji uključuju numeričke izračune, i web programiranje. Svaka kategorija imala je i potpuno ispravnu verziju i verziju s namjernom sintaksnom ili logičkom pogreškom. Testiranjem u ovim različitim kategorijama, želi se procijeniti svestranost modela i sposobnost rukovanja različitim vrstama programskih zadataka. Za kvalitativnu analizu korišten je LM Studio, okruženje koje olakšava lokalno izvođenje različitih modela strojnog učenja, i sučelje OpenAI chat. LM Studio nudi opsežnu biblioteku jezičnih modela i nudi sučelje jednostavno za korištenje za pokretanje i usporedbu modela. Ovaj je alat bio osobito koristan za provođenje detaljne kvalitativne analize generiranih rezultata.

Ljudski faktor korišteni su za usporedbu rezultata triju modela: fino podešenog CodeT5+, Chat GPT-4 i Code Llama. Rezultati izlaznih podataka procijenjeni su na temelju atributa kao što su ispravnost, koherentnost, relevantnost, sposobnost rješavanja pogrešaka i sveukupne povratne informacije. Svaki isječak koda procijenjen je kako bi se utvrdilo koliko su modeli identificirali pogreške i pružili korisne povratne informacije. Ova usporedna analiza omogućila je da se identificiraju prednosti i slabosti svakog modela u rješavanju različitih vrsta programskih zadataka.

4.4.6. Ulazni podaci za procjenu

Za procjenu, ulazni podaci se dijele u tri kategorije: matematički i algoritamski zadaci, web programiranje i strukture podataka. U matematičkoj i algoritamskoj kategoriji primjeri uključuju zadatke poput ispravnog izračuna najvećeg broja u nizu i netočne metode zbrajanja brojeva. Za kategoriju web programiranja odabrali smo ispravno implementiran GET zahtjev i primjer HTML koda. U kategoriju struktura podataka uključili smo ispravno implementiran algoritam *Bubble* sortiranja i neispravno implementiranu stog strukturu podataka. Slika pruža primjer strukture ulaznih podataka.

```
""Give me assessment on this code:
function findMax(arr)
{
  let max = arr[0];
  for (let i = 1; i < arr.length; i++) {
    if (arr[i] > max) {
      max = arr[i];
    }
  }
  return max;
}
console.log(findMax([1, 2, 3, 4, 5]));""
```

Slika 3 Primjer ulaznog podataka za procjenu

Ljudskom procjenom uspoređeni su rezultati triju modela: fino podešenog CodeT5+, Chat GPT-4 i Code Llama. Rezultati izlaznih podataka procijenjeni su na temelju atributa kao što su ispravnost, koherentnost, relevantnost i sposobnost rješavanja pogrešaka. Svaki isječak koda procijenjen je kako bi se utvrdilo koliko su modeli identificirali pogreške i pružili korisne povratne informacije. Ova usporedna analiza omogućila nam je da identificiramo prednosti i slabosti svakog modela u rješavanju različitih vrsta programskih zadataka. Slijedeći ovakav pristup procjene, cilj je bio pružiti temeljitu procjenu fino podešenog modela CodeT5+ u usporedbi s uspostavljenim modelima kao što su Chat GPT-4 i Code Llama. Ovaj proces ne samo da naglašava praktičnu primjenjivost fino podešenog modela,

već također pruža uvid u područja u kojima se ističe ili zahtijeva daljnje poboljšanje. Time je osigurana sveobuhvatna i objektivna procjena fino podešenog modela CodeT5+ u usporedbi s drugim istaknutim velikim jezičnim modelima.

5. Praktični dio

Za svrhu ovog istraživanja fino je podešen model CodeT5+ unutar jupyter okruženja. U slijedećim odlomcima opisati će se proces finog podešavanja modela CodeT5, od pripreme podataka i tokenizacije do optimizacije hiperparametara i evaluacije modela, pružajući opsežan vodič sa tehničkog aspekta istraživanja. Nakon kojih će slijediti rezultati te rasprava dobivenih rezultata.

5.1. Uključene biblioteke

U procesu finog podešavanja modela CodeT5 koristi se nekoliko biblioteka za pojednostavljenje rukovanja podacima, treniranje modela i evaluaciju. U ovom odjeljku će se proći kroz nekoliko najbitnijih korištenih biblioteka. Biblioteka *Pandas* koristi se za manipuliranje i spremanje skupa podataka, što olakšava upravljanje isječcima koda i procjenama u tabličnom formatu. Za pripremu podataka za treniranje, *Scikit-learn* funkcija *train_test_split* koristi se za podjelu skupa podataka u skupove za treniranje i validaciju, osiguravajući da se izvedba modela može ispravno procijeniti. Biblioteka *Hugging Face Transformers* pruža pristup unaprijed treniranim modelima za zadatke obrade prirodnog jezika. Iz *Hugging Face Transformers* biblioteke koriste se slijedeće funkcije: *AutoTokenizer* za pretvaranje koda u tokene koje model može obraditi, *AutoModel* za pristup prethodno treniranom modelu CodeT5, te *TrainingArguments* za konfiguriranje parametara za treniranje i *Trainer* API, koji pojednostavljuje treniranje modela i proces evaluacije. Biblioteka *Optuna* se koristi za automatizaciju traženja najboljih hiperparametara, kao što su stopa učenja i veličina serije. *Evaluate* biblioteka pruža ključne metrike kao što su BLEU i ROUGE za procjenu kvalitete izlaza modela u odnosu na referentni kod, osiguravajući da fino podešeni model zadovoljava željene kriterije izvedbe.

```
[ ] import pandas as pd
import numpy as np
import logging
from sklearn.model_selection import train_test_split
import evaluate
import gdown
import torch
from transformers import AutoModel, AutoTokenizer, Trainer, TrainingArguments
from datasets import Dataset
import optuna
```

Slika 4 Korištene biblioteke

5.2. Priprema podataka

Nakon odabira korištenih biblioteka, idući korak uključivao je pripremu skupa podataka za fino podešavanje modela CodeT5. U formatu Python rječnika stvoren je skup podataka koji sadrži više isječaka koda i njihove odgovarajuće procjene. Svaki unos u skupu podataka uključivao je isječak koda i njegovu procjenu. Skup podataka je zatim pretvoren u *Pandas DataFrame* radi lakšeg rukovanja, te je naknadno spremljen kao CSV datoteka.

```
df = pd.DataFrame(data)

#stvaranje CSV
csv_file_path = "./code_assessment_dataset_with_correct_examples3.csv"
df.to_csv(csv_file_path, index=False)
print(f"Dataset exported to {csv_file_path}")
```

Slika 5 Priprema skupa podataka 1

Kako bi se olakšao proces finog podešavanja, skup podataka je podijeljen na skupove za treniranje i validaciju pomoću podjele 80-20. Skup za treniranje korišten je za fino podešavanje modela, dok je skup za validaciju korišten za procjenu njegove izvedbe tijekom treniranja.

```

# Podjela podataka na suka za treniranje i testiranje
train_df, val_df = train_test_split(df, test_size=0.2, random_state=42)

# Pretvaranje DataFrame u Hugging Face Dataset
train_dataset = Dataset.from_pandas(train_df)
val_dataset = Dataset.from_pandas(val_df)
print(train_dataset)
print(val_dataset)

Dataset({
  features: ['code', 'assessment', '__index_level_0__'],
  num_rows: 24
})
Dataset({
  features: ['code', 'assessment', '__index_level_0__'],
  num_rows: 7
})

```

Slika 6 Priprema skupa podataka 2

Na podijeljenom skupu podataka zatim je izvršena tokenizacija. Tokenizacija podataka ključan je korak koji transformira neobrađene isječke koda u format koji model može razumjeti i obraditi. Iz *Hugging Face* biblioteke *Transformers* korištena je funkcija *AutoTokenizer*, koja je posebno prilagođen zahtjevima modela CodeT5.

```

# Odabir najbolje max_length vrijednosti prema skupu podataka
df['code_length'] = df['code'].apply(lambda x: len(tokenizer(x)['input_ids']))
print(df['code_length'].describe())
max_length = int(df['code_length'].quantile(0.95))

# Tokenizacija podataka
def tokenize_function(examples):
    code_tokens = tokenizer(examples["code"], truncation=True, padding='max_length', max_length=max_length, add_special_tokens=True)

    tokenized_assessment = tokenizer(examples["assessment"], truncation=True, padding='max_length', max_length=max_length, add_special_tokens=True)["input_ids"]

    labels = [(label if label != tokenizer.eos_token_id else -100) for label in tokenized_assessment]

    return {
        "input_ids": code_tokens["input_ids"],
        "attention_mask": code_tokens["attention_mask"],
        "labels": labels
    }

tokenized_train_dataset = train_dataset.map(tokenize_function, batched=True, remove_columns=train_dataset.column_names)
tokenized_val_dataset = val_dataset.map(tokenize_function, batched=True, remove_columns=val_dataset.column_names)

tokenized_train_dataset.set_format("torch", columns=["input_ids", "attention_mask", "labels"])
tokenized_val_dataset.set_format("torch", columns=["input_ids", "attention_mask", "labels"])

```

Slika 7 Tokenizacija podataka

5.3. Fino podešavanje modela

Model CodeT5 je fino podešen korištenjem prethodno obrađenog skupa podataka. Fino podešavanje provedeno je pomoću *Hugging Face Trainer* klase, koja pojednostavljuje procese treniranja i evaluacije. Hiperparametri modela optimizirani su pomoću Optune, okvira za optimizaciju hiperparametara. Metoda *optuna.create_study* korištena je za pronalaženje optimalne stope učenja, broja epoha i veličine serije.

```

def objective(trial):
    learning_rate = trial.suggest_float("learning_rate", 1e-5, 1e-4, log=True)
    num_train_epochs = trial.suggest_int("num_train_epochs", 3, 10)
    per_device_train_batch_size = trial.suggest_categorical("per_device_train_batch_size", [2, 4, 8])

    training_args = TrainingArguments(
        output_dir="./results",
        eval_strategy="epoch",
        save_strategy="epoch",
        learning_rate=learning_rate,
        per_device_train_batch_size=per_device_train_batch_size,
        per_device_eval_batch_size=2,
        num_train_epochs=num_train_epochs,
        weight_decay=0.01,
        logging_dir="./logs",
        gradient_accumulation_steps=8,
        fp16=True,
        logging_steps=10,
        report_to=["tensorboard"],
        disable_tqdm=False,
        save_steps=1000,
        save_total_limit=3,
        load_best_model_at_end=True,
    )

    model = AutoModel.from_pretrained("Salesforce/codet5p-220m-bimodal", trust_remote_code=True)
    model.gradient_checkpointing_enable()

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_train_dataset,
        eval_dataset=tokenized_val_dataset,
        compute_metrics=compute_metrics,
    )

    trainer.train()
    eval_results = trainer.evaluate()
    return eval_results["eval_loss"]

study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=20)

```

Slika 8 Optimizacija arugmenta za treniranje

Nakon optimizacije hiperparametara, model je fino podešen koristeći najbolje parametre koje je identificirala Optuna. Fino podešeni model je zatim spremljen za kasniju upotrebu.

```

# Definiranje argumenata za treniranje
training_args = TrainingArguments(
    output_dir="./results",
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=9.554263192753116e-05,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=8,
    weight_decay=0.01,
    logging_dir="./logs",
    gradient_accumulation_steps=8,
    fp16=True,
    logging_steps=10,
    report_to=["tensorboard"],
    disable_tqdm=False,
    save_steps=1000,
    save_total_limit=3,
    load_best_model_at_end=True,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train_dataset,
    eval_dataset=tokenized_val_dataset,
    compute_metrics=compute_metrics,
)

```

Slika 9 Treniranje modela

Izvedba modela procijenjena je pomoću metrika kao što su BLEU i ROUGE, koje se obično koriste za mjerenje kvalitete generiranog teksta. Procjena je provedena na validacijskom skupu kako bi se utvrdilo koliko je model generaliziran izvan podataka za treniranje.

```

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    if isinstance(logits, tuple):
        logits=logits[0]

    labels=np.where(labels != -100, labels, tokenizer.pad_token_id)

    predictions = np.argmax(logits, axis=-1)

    decoded_preds = tokenizer.batch_decode(predictions, skip_special_tokens=True)
    decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)

    decoded_labels = [[label] for label in decoded_labels]

    metrics = {}
    #BLEU Score
    try:
        bleu = bleu_metric.compute(predictions=decoded_preds, references=decoded_labels)
        print(f"BLEU: {bleu}")
        metrics["bleu"] = bleu["bleu"]
    except Exception as e:
        print(f"Error computing BLEU: {e}")
        metrics["bleu"] = float('nan')
    #ROUGE Score
    try:
        rouge = rouge_metric.compute(predictions=decoded_preds, references=decoded_labels)
        print(f"ROUGE: {rouge}")
        if isinstance(rouge, dict):
            metrics["rouge1"] = rouge["rouge1"].mid.fmeasure
            metrics["rouge2"] = rouge["rouge2"].mid.fmeasure
            metrics["rougeL"] = rouge["rougeL"].mid.fmeasure
        else:
            metrics["rouge1"] = float('nan')
            metrics["rouge2"] = float('nan')
            metrics["rougeL"] = float('nan')
    except Exception as e:
        print(f"Error computing ROUGE: {e}")
        metrics["rouge1"] = float('nan')
        metrics["rouge2"] = float('nan')
        metrics["rougeL"] = float('nan')
    return metrics

```

Slika 10 Metrike za evaluaciju modela

```

# Evaluacija modela
eval_results = trainer.evaluate()
print(f"Evaluation results: {eval_results}")

```

Slika 11 Evaluacija modela

5.4. Rezultati

U sljedećim odjeljcima prikazani su rezultati fino podešene izvedbe modela, optimizacije hiperparametara i modela Llama i GPT.

5.4.1. Podešavanje hiperparametara

U ovom istraživanju veliki jezični model CodeT5+ fino je podešen na skupu podataka isječaka JavaScript koda. Kako bi se identificirali najučinkovitiji argumenti treniranja korištena je optimizacija hiperparametara putem biblioteke Optuna. Model je fino podešen za različite kategorije programskih zadataka, uključujući matematičke i algoritamske zadatke, strukture podataka i web programiranje. Primarni cilj bio je procijeniti sposobnost modela da generira točne procjene isječaka koda, identificirajući ispravne implementacije i one koje sadrže sintaktičke ili logičke pogreške. U procesu optimizacije hiperparametara fino su podešeni argumenti stopa učenja, broj epoha treniranja i veličina serije. Provedeno je 20 ciklusa, a optimalni parametri identificirani su kao stopa učenja od $9,55e-05$, 8 epoha treniranja i veličina serije od 2. Ti hiperparametri odabrani su na temelju njihove sposobnosti da minimiziraju gubitak evaluacije tijekom treninga, osiguravajući da model postigne najbolju moguću izvedbu na zadanim zadacima.

5.4.2. Evaluacija Llama i Chat GPT modela

Za procjenu velikih jezičnih modela, Llama i GPT-4, korištena su posebna sučelja. Model Code Llama procijenjen je pomoću LM Studio okruženja, dok je GPT-4 procijenjen putem sučelja OpenAI Chat. Svaki je model prošao evaluaciju u tri različite kategorije: matematički i algoritamski zadaci, strukture podataka i web programiranje. Radi dosljednosti, svaka je kategorija uključivala dvije ulazne vrijednosti: jednu koja predstavlja ispravnu implementaciju, a drugu koja sadrži namjernu pogrešku kako bi se testirala sposobnost modela da identificiraju i procijene pogreške. Oba modela dobila su jednake ulazne vrijednosti za izravnu usporedbu. U prvoj kategoriji, matematički i algoritamski zadaci, modeli su procijenjeni na dva primjera: funkcija findMax kao ispravna implementacija i funkcija addNumber, koja je sadržavala namjernu pogrešku, kao neispravan primjer.

```
Give me assesement on this code: function addNumbers(num1, num2) {  
    return num1 + num2;  
}  
var result = addNumbers("10", "20");  
console.log("The result is:" + result);
```

Slika 12 Ulazni podatak za evaluaciju - Matematički i algoritamski zadatci 1

```
Give me assesement on this code:
function addNumbers(num1, num2)
{ return num1 + num2; }
var result = addNumbers("10", "20");
console.log("The result is:" + result);
```

Slika 13 Ulazni podatak za evaluaciju - Matematički i algoritamski zadatci 2

Zatim je procijenjena kategorija struktura podataka. Za ulazne podatke unesena je neispravna implementacija struktura stog i ispravno implementiran *Bubble sort* algoritam za sortiranje.

```
give me assessment on this code: class Stack {
  constructor() {
    this.items = [];
  }
  push(element) {
    this.items.push(element);
  }
  pop() {
    if(this.isEmpty()) return "Underflow";
    return this.items.pop();
  }
  isEmpty() {
    return this.items.length == 0;
  }
}
const stack = new Stack();
console.log(stack.pop());
```

Slika 14 Ulazni podatak za evaluaciju - Strukture podataka 1


```
Give me assesement on this code: function bubbleSort(arr) {
  let n = arr.length;
  for (let i = 0; i < n - 1; i++) {
    for (let j = 0; j < n - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        let temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
  return arr;
}
console.log(bubbleSort([64, 34, 25, 12, 22, 11, 90]));
```

USER

Slika 15 Ulazni podatak za evaluaciju - Strukture podataka 2

Zadnja procijenjena kategorija je kategorija web programiranja. Za ulazni podatak ispravne implementacije unesena je funkcija *fetch* za dohvaćanje, a za ulazni podatak neispravne implementacije unesen je HTML kod .

```
Give me assessment on this code: fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.log('Error:', error));
```

USER


Slika 16 Ulazni podatak za evaluaciju - Web programiranje 1

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Incorrect Example</title>
</head>
<body>
  <button id="myButton">Click me</button>
  <script>
    const button = document.getElementById('myButton');
    button.addEventListener('click', showAlert());
    function showAlert() {
      alert('Button was clicked!');
    }
  </script>
</body>
</html>
```

USER

Slika 17 Ulazni podatak za evaluaciju - Web programiranje 2

Rezultati modela *Code Llama* i *GPT-4* prikazani su na slijedećim slikama.

 The code you've written is a simple and correct implementation to find the maximum value in an array of numbers. Here's an assessment of your code:

Strengths:

1. **Correct Logic:** The function initializes the `max` variable with the first element of the array, and then iterates through the remaining elements to compare and update the `max` variable whenever a larger value is found. It works as intended.
2. **Readability:** The code is easy to read and understand. The use of variable names like `max` and `arr` makes the logic clear.
3. **Efficiency:** The time complexity of this approach is $O(n)$, which is optimal for this problem since every element in the array must be checked to find the maximum.

Slika 18 *ChatGpt4* rezultat - Matematički i algoritamski zadatci 1



The code you've written is a simple and correct implementation to find the maximum value in an array of numbers. Here's an assessment of your code:

Strengths:

1. **Correct Logic:** The function initializes the `max` variable with the first element of the array, and then iterates through the remaining elements to compare and update the `max` variable whenever a larger value is found. It works as intended.
2. **Readability:** The code is easy to read and understand. The use of variable names like `max` and `arr` makes the logic clear.
3. **Efficiency:** The time complexity of this approach is $O(n)$, which is optimal for this problem since every element in the array must be checked to find the maximum.

Slika 19 *ChatGpt4* rezultat - Matematički i algoritamski zadatci 2

Issues:

1. Type Coercion:

- The arguments `"10"` and `"20"` are passed as **strings**, not numbers. JavaScript is dynamically typed and will attempt to coerce the strings into numbers for the `+` operation. In this case, the function works because JavaScript will convert `"10"` and `"20"` to numbers (10 and 20), and the result will be `30`.
- However, this behavior is implicit and can cause confusion or bugs in more complex code or when different types of input are passed. For example, if one of the inputs were a string that doesn't represent a valid number (e.g., `"abc"`), it would result in `NaN` (Not a Number).

2. **String Concatenation Risk:** The `+` operator can also concatenate strings if one or both of the inputs are strings. So if a different type of string input is passed (e.g., `"10" + "abc"`), it will concatenate the strings instead of adding numbers.

Slika 20 *ChatGpt4* rezultat - Matematički i algoritamski zadatci 2



The code you provided is a basic and valid example of how to fetch data from an API using JavaScript's `fetch()` function. It includes proper handling for success and error cases, but there are several improvements that could be made for robustness, readability, and real-world usage.

Slika 21 *ChatGpt4* rezultat - Web programiranje 1

Strengths:

1. Correct Use of Fetch API:

- The `fetch()` function is used correctly to send a GET request to the specified URL (`https://api.example.com/data`).
- The `response.json()` method is used to parse the JSON data returned by the API.
- The `then()` blocks handle the success case by logging the parsed data, and the `catch()` block handles any errors that occur during the request or the parsing process.

2. Error Handling:

- The code properly catches errors with a `.catch()` block, which is essential for handling network failures or issues with parsing JSON.

3. Simplicity:

- The code is concise and easy to read, making it a good starting point for basic API requests.

Slika 22 ChatGpt4 rezultat - Web programiranje 1



The code you provided defines a `Stack` class in JavaScript and implements basic stack operations: `push()`, `pop()`, and `isEmpty()`. The code generally works well, but there are some areas where improvements can be made, particularly in edge case handling and error messaging.

Slika 23 ChatGpt4 rezultat - Strukture podataka 1

Strengths:

1. Correct Implementation of Stack Operations:

- The `push()` method correctly adds an element to the stack.
- The `pop()` method correctly removes and returns the top element from the stack, or returns `"Underflow"` if the stack is empty.
- The `isEmpty()` method checks whether the stack is empty, which is useful for handling the pop operation safely.

2. Encapsulation of Stack Data:

- The stack's internal data (`this.items`) is encapsulated within the class, ensuring that only the methods of the `Stack` class can directly modify the stack.

3. Simplicity and Readability:

- The code is simple, readable, and adheres to object-oriented programming principles. It's clear and easy to understand for anyone familiar with stacks.

Slika 24 ChatGpt4 rezultat - Strukture podataka 1

The code you provided contains an issue with how the `addEventListener()` method is used. Here's a detailed assessment of the code:

Issue:

1. Incorrect Function Call in Event Listener:

- The line:

```
javascript Copy code  
button.addEventListener('click', showAlert());
```

is incorrect because it **immediately invokes** the `showAlert()` function as soon as the script runs, instead of assigning it as an event handler. In JavaScript, using `()` after a function name calls the function right away, so the alert will appear as soon as the page loads, not when the button is clicked.

- The event listener expects a **function reference**, not the result of calling the function.

Slika 25 ChatGpt4 rezultat - Web programiranje 2



This implementation of the **Bubble Sort** algorithm is correct and follows a standard approach. Here's an assessment of the code:

Positive Aspects:

1. **Correct Logic:** The code correctly implements the Bubble Sort algorithm. It uses nested loops where the inner loop compares adjacent elements and swaps them if they are out of order, ensuring the largest unsorted element "bubbles up" to its correct position after each pass of the outer loop.
2. **No Issues with Edge Cases:** The code can handle edge cases like an empty array or a single-element array without issues.
3. **Readable Code:** The code is well-structured and easy to read with clear variable names and logic.



Slika 26 ChatGpt4 rezultat - Strukture podataka 2

AI

This code is a valid JavaScript function that takes an array as input and returns the maximum value in the array. Here's a breakdown of the code:

- * `function findMax(arr) { ... }` declares a function called `findMax` that takes an array `arr` as input.
- * `let max = arr[0];` initializes a variable `max` to the first element of the array `arr`.
- * `for (let i = 1; i < arr.length; i++)` loops through each element of the array `arr`, starting from the second element (`i = 1`).
- * `if (arr[i] > max) { max = arr[i]; }` checks if the current element is greater than the previous maximum value, and if so, updates the maximum value to the current element.
- * `return max;` returns the maximum value found in the array.

The function can be called with an array as input, such as `[1, 2, 3, 4, 5]`, and it will return the maximum value in the array, which is `5` in this case.

Overall, this code is a valid implementation of a function to find the maximum value in an array in JavaScript.

Slika 27 Code Llama rezultat - Matematički i algoritamski zadatci 1

AI

This code is a valid JavaScript program that defines a function called `addNumbers` that takes two numbers as input and returns their sum. The function can be called with any two numbers, such as `addNumbers(10, 20)`, and it will return the sum of those numbers.

Here's a breakdown of the code:

- * `function addNumbers(num1, num2) { ... }` declares a function called `addNumbers` that takes two arguments `num1` and `num2`.
- * `return num1 + num2;` returns the sum of the two numbers.
- * `var result = addNumbers("10", "20");` calls the `addNumbers` function with the arguments `"10"` and `"20"`, and assigns the return value to a variable called `result`.
- * `console.log("The result is:" + result);` logs the string `"The result is:"` followed by the value of `result` to the console using `console.log()`.

Overall, this code defines a function that adds two numbers and calls it with specific arguments, which results in logging the sum of those numbers to the console.

Slika 28 Code Llama rezultat - Matematički i algoritamski zadatci 2

AI

This code defines a function called `bubbleSort` that takes an array as input and sorts it in ascending order using the bubble sort algorithm. The function iterates over the array, comparing adjacent elements and swapping them if they are in the wrong order. It then returns the sorted array.

The code also logs the result of calling the `bubbleSort` function with an example input array to the console using `console.log()`.

Overall, this code defines a simple sorting algorithm that uses the bubble sort method and demonstrates how to call it in JavaScript.

Slika 29 *Code Llama* rezultat - Strukture podatka 1

AI

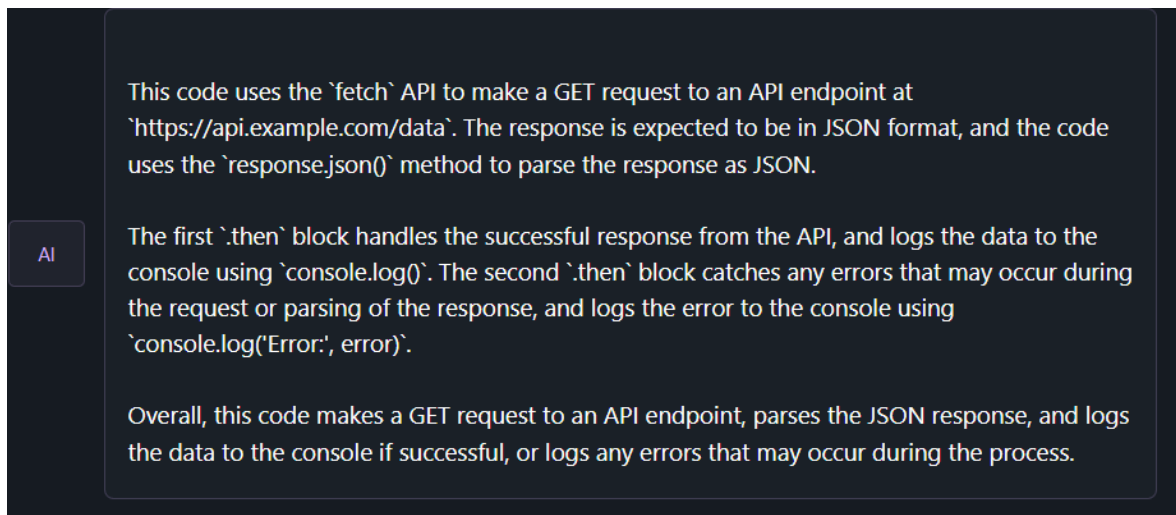
This code defines a class called `Stack` that has three methods: `push`, `pop`, and `isEmpty`. The `push` method adds an element to the stack, the `pop` method removes the top element from the stack, and the `isEmpty` method checks if the stack is empty.

The constructor of the class initializes a property called `items` that stores the elements of the stack. The `push` method pushes an element onto the `items` array, and the `pop` method removes the last element from the `items` array. The `isEmpty` method checks if the `items` array is empty.

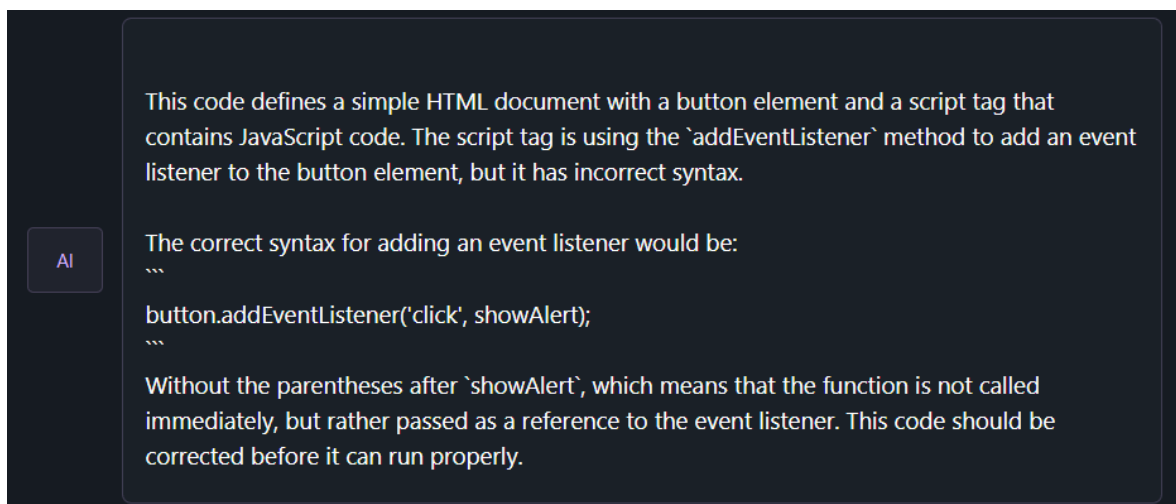
The code also defines a variable called `stack` that is an instance of the `Stack` class. The `pop` method is called on the `stack` object, and the result is logged to the console using `console.log()`.

Overall, this code defines a simple stack data structure with `push`, `pop`, and `isEmpty` methods, and demonstrates how to create an instance of the class and use its methods.

Slika 30 *Code Llama* rezultat - Strukture podatka 2



Slika 31 *Code Llama* rezultat - Web programiranje 1



Slika 32 *Code Llama* rezultat - Web programiranje 2

5.4.3. Evaluacija fino podešenog modela

Za procjenu izvedbe fino podešenog modela korišteni su BLEU i ROUGE kao primarne metrike procjene. Ove su metrike odabrane zbog svoje sposobnosti procjene kvalitete teksta generiranog modelom u usporedbi s referentnim procjenama. Dobiveni BLEU rezultat za model iznosio je na 0,404. ROUGE metrika dodatno je potvrdila izvedbu modela, s rezultatom od 0,62 za ROUGE-1, a 0,41 za ROUGE-2. ROUGE-L rezultat, koji uzima u obzir najduži zajednički podsljed, iznosio je 0,60.

Gledajući konkretna predviđanja model je testiran na nizu isječaka koda u tri kategorije, uključujući ispravne i pogrešne implementacije. Za prvu kategoriju, matematički i

algoritamski zadatci, ulazni podatci su funkcija findMax() za pronalazak najvećeg broja i funkcija addNumbers() za zbrajanje brojeva. Na slikama 32 i 33 su prikazani uneseni podaci.

```
"""Give me assessment on this code:
function findMax(arr)
{
  let max = arr[0];
  for (let i = 1; i < arr.length; i++) {
    if (arr[i] > max) {
      max = arr[i];
    }
  }
  return max;
}
console.log(findMax([1, 2, 3, 4, 5]));"""
```

Slika 33 Ulazni podatci CodeT5- matematički i algoritamski zadatci 1

```
"""Give me assessment on this code:
function addNumbers(num1, num2) {
  return num1 + num2;
}
var result = addNumbers("10", "20");
console.log("The result is:" + result);"""
```

Slika 34 Ulazni podaci CodeT5 - matematički i algoritamski zadatci 2

Predviđena procjena modela za prvu ispravnu implementaciju je sljedeća: „*This code is correct: The function findMax correctly calculates the maximum of the given array and logs it.*“

Za drugu neispravnu implementaciju model je predvidio sljedeće: „*This code is correct: The function addNumbers correctly adds two numbers and logs the result. Keep practicing functions and functions with the same arguments.*“

Zatim je testirana druga kategorija, strukture podataka. Uneseni testni podaci su neispravna implementacija stog strukture podataka, te ispravno implementirano korištenje strukture niza. Na slikama 34 i 35 su prikazani uneseni podaci.

```

"""Give me assessment on this code:
class Stack {
  constructor() {
    this.items = [];
  }
  push(element) {
    this.items.push(element);
  }
  pop() {
    if (this.isEmpty()) return "Underflow";
    return this.items.pop();
  }
  isEmpty() {
    return this.items.length === 0;
  }
}
const stack = new Stack();
console.log(stack.pop());""",

```

Slika 35 Ulazni podaci CodeT5 - Strukture podataka 1

```

"""Give me assesement on this code: function bubbleSort(arr) {
  let n = arr.length;
  for (let i = 0; i < n - 1; i++) {
    for (let j = 0; j < n - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        let temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
  return arr;
}
console.log(bubbleSort([64, 34, 25, 12, 22, 11, 90]));""",

```

Slika 36 Ulazni podaci CodeT5 - strukture podataka 2

Za neispravnu implementaciju stog strukture podataka dobivena je procjena: „*This code is correct: The `pop` method should take a string argument, not a number. Review the structure of a stack and check if it is empty before attempting to remove an element.*“ Drugi uneseni testni podatak, ispravno implementirana funkcija korištenja *Bubble sorta*, imala je slijedeću procjenu: „*This code is correct: The `bubbleSort` function correctly sorts an array of numbers and logs the result. Keep practicing sorting of strings and arrays.*“

Zadnja testirana kategorija, web programiranje, za testne unosne podatke imala je ispravno implementiranu *fetch* metodu i neispravno implementirano korištenje akcija unutar HTML koda. Uneseni podaci mogu se vidjeti na slikama 36 i 37.

```

"""Give me assessment on this code:
fetch('https://api.example.com/data') |
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.log('Error:', error));"""

```

Slika 37 Ulazni podaci CodeT5 - Web programiranje 1

```

"""Give me assessment on this code: <!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Incorrect Example</title>
</head>
<body>
<button id="myButton">Click me</button>

<script>
const button = document.getElementById('myButton');

// Incorrect function syntax
button.addEventListener('click', showAlert());

function showAlert() {
  alert('Button was clicked!');
}
</script>
</body>
</html>"""

```

Slika 38 Ulazni podaci CodeT5 - Web programiranje 2

Za ispravnu implementaciju model je dao ovu procjenu: „*This code is correct: The `fetch` function correctly handles the response and logs the result. Keep practicing handling API requests and logging results.*“. Za neispravnu implementaciju dao je slijedeći odgovor: „*This code is correct: The `showAlert` function correctly handles the alert and logs the result. Review the API response and response processing code.*“

5.5. Rasprava

Rezultati procesa podešavanja hiperparametara korištenjem biblioteke Optuna dali su skup optimalnih parametara za koje se očekuje da maksimiziraju performanse fino podešenog modela CodeT5+. Ti hiperparametri uključuju stopu učenja, broj epoha treniranja i veličinu serije po uređaju tijekom treniranja. Najbolja stopa učenja koju je utvrdila Optuna je otprilike $9,55e-05$. Stopa učenja ključni je hiperparametar koji kontrolira veličinu koraka u svakoj iteraciji dok se kreće prema minimumu funkcije gubitka. Stopa učenja koja je previsoka može uzrokovati prebrzu konvergenciju modela do suboptimalnog rješenja ili može uzrokovati nestabilnost treniranja. S druge strane, preniska stopa učenja može učiniti proces

treniranja izuzetno sporim, a model bi mogao zapeti u lokalnim minimumima. (EITCA, 2024) Dobivena stopa učenja od $9,55e-05$ je relativno niska, što sugerira da su ažuriranja modela mala i postupna, što može pomoći u pronalaženju preciznijeg i stabilnijeg minimuma za funkciju gubitka, posebno u kontekstu finog podešavanja gdje model treba kako bi se prilagodio specifičnim nijansama novih podataka bez gubitka znanja koje je već stekao. Nadalje, proces podešavanja je utvrdio da je 8 epoha optimalan broj za trening. Epoha je jedan potpuni prolaz kroz cijeli skup podataka za treniranje. Odabir od 8 epoha sugerira da model ima koristi od višestrukih iteracija nad podacima, što mu omogućuje učenje i pročišćavanje svojih predviđanja sa svakim prolazom. Uvježbavanje za premali broj epoha može rezultirati nedovoljnim prilagođavanjem, gdje model ne uči dovoljno iz podataka, dok previše epoha može dovesti do prekomjernog prilagođavanja, gdje model postaje previše prilagođen podacima za treniranje i ima lošu izvedbu na nevidljivim podacima. Osam epoha uspostavlja ravnotežu, pružajući dovoljno mogućnosti modelu za učenje dok se rizik od prekomjernog opremanja smanjuje na minimum. Utvrđeno je da je optimalna veličina serije 2. Veličina serije odnosi se na broj primjera treniranja koji se koriste u jednoj iteraciji. Manja veličina serije, poput 2, znači da se parametri modela češće ažuriraju, što može dovesti do brže konvergencije i može pomoći u izbjegavanju prekomjernog prilagođavanja uvođenjem oblika regulacije kroz šum u procjenama gradijenta. Međutim, manje serije također mogu dovesti do veće varijacije u ažuriranjima, što može učiniti proces treniranja manje stabilnim. Općenito, proces podešavanja naglašava da je pažljivo balansiranje ovih parametara ključno za postizanje učinkovitog učenja i generalizacije modela.

5.5.1. Mjerila evaluacije modela

BLEU rezultat iznosio je 0,404, što ukazuje da model generira tekst koji je razumno usklađen s referencom, iako ima prostora za poboljšanje, posebno u hvatanju složenijih n-grama. ROUGE rezultati, posebno ROUGE-1 i ROUGE-L imaju rezultate iznad 0,60. Takvi rezultati naglašavaju da model ima dobro razumijevanje sadržaja niza i strukture, što je ključno za generiranje povezanog i kontekstualno točnog teksta. Relativno niža ocjena ROUGE-2, od 0,416 sugerira da bi se model mogao više mučiti s hvatanjem detalja na razini fraze, što bi moglo utjecati na sposobnosti modela da duboko razumije kontekst. Performanse fino podešenog modela CodeT5 otkrivaju područja potencijala i poboljšanja. Dok model uspijeva uhvatiti određene važne elemente referentnog teksta, njegova izvedba je ometena nedostatkom dubljeg razumijevanja na razini fraze i strukturne povezanosti.

Kako bi se poboljšala njegova sposobnost generiranja kontekstualno točnijih i koherentnijih rezultata, možda bi bilo potrebno daljnje treniranje s raznolikijim i složenijim skupovima podataka. Ovo bi vjerojatno poboljšalo ne samo rezultat BLEU nego i rezultat ROUGE, posebno za bigrame i dulje nizove. Rezultati evaluacije fino podešenog modela CodeT5 ukazuju na njegove prednosti i područja u kojima bi moglo biti potrebno poboljšanje, što se odražava u procjenama temeljenim na rezultatima BLEU i ROUGE. Ovakvi rezultati sugeriraju da fino podešeni model dobro funkcionira u generiranju teksta koji je kontekstualno i strukturno usklađen s referencom. Međutim, za daljnje poboljšanje kvalitete, osobito na razini izraza i rečenice, detaljnije treniranje modela bilo bi potrebno.

5.5.2. Predviđene procjene

Gledajući konkretna predviđanja model je testiran na nizu isječaka koda u tri kategorije, uključujući ispravne i pogrešne implementacije. Za prvu kategoriju, matematički i algoritamski zadatci, ulazni podatci su funkcija *findMax* za pronalazak najvećeg broja i funkcija *addNumbers()* za zbrajanje brojeva. Za ispravne implementacije koda, kao što je funkcija *findMax*, model je pružio točne i pozitivne povratne informacije. Međutim, procjeni modela nedostajali su detalji, procjena je bila funkcionalna, ali pretjerano pojednostavljena, pružajući minimalan uvid osim navođenja očite svrhe funkcije. Za kod s namjernim pogreškama, kao što je funkcija *addNumbers*, model nije uspio prepoznati problem neusklađenosti tipa. Umjesto toga, pružio je netočnu procjenu, navodeći da je funkcija ispravno dodala brojeve. Što se tiče druge kategorije zadataka, strukture podataka, ulazni podaci su: neispravna implementacija stog strukture podataka i ispravna implementacija *Bubble* algoritma za sortiranje. Za implementaciju klase *Stack*, koja je uključivala metode kao što su *push*, *pop* i *isEmpty*, model je točno identificirao opću strukturu, ali je dao donekle pogrešne savjete. Sugerirao je da *pop* metoda zahtijeva argument niza, što je bilo netočno. Što se tiče ispravne implementacije, funkcija *bubbleSort* ispravno je procijenjena u smislu njezine funkcionalnosti. Međutim, povratne informacije o modelu ponovno su bile poprilično osnovne, propuštajući priliku za raspravu o učinkovitosti algoritma, ili potencijalnim pozitivnim povratnim informacijama. Zadnja kategorija zadataka, web programiranje, za ulazne podatke prima funkciju *fetch()* i primjer HTML koda. Model je ispravno procijenio funkciju dohvaćanja, koja upućuje HTTP zahtjev na određeni URL, obrađuje odgovor i obrađuje pogreške. Funkcija ispravno obrađuje obećanje vraćeno dohvaćanjem, pretvarajući odgovor u JSON i bilježeći rezultat ili moguće pogreške.

Procjena pokazuje da model može točno procijeniti asinkroni kod, posebno upotrebu obećanja i rukovanje pogreškama u JavaScriptu. Predviđanje odražava razumijevanje načina na koji funkcionira rukovanje HTTP zahtjevima i odgovorima. Što se tiče drugog ulaznog podatka, HTML koda. Model je ispravno prepoznao svrhu funkcije, no nije uspješno prepoznao greške, gdje se funkcija *showAlert* poziva odmah umjesto da se proslijedi kao povratni poziv. Ovo pokazuje ograničenu sposobnost modela da otkrije i istakne pogreške, osobito u složenijim isječcima koda.

Ukupna izvedba fino podešenog modela, kao što je naznačeno rezultatima BLEU i ROUGE, pokazuje da je model razumno učinkovit u stvaranju procjena koje su kontekstualno relevantne i točne u identificiranju funkcionalnosti koda. No, njegova sposobnost pružanja detaljnih i nijansiranih povratnih informacija, kao što je prepoznavanje potencijalnih pogrešaka ili nuđenje prilagođenijih savjeta je ograničena. Ovo ukazuje na potrebu za daljnjim poboljšanjima, dodatno fino podešavanje ili dodatni trening podataka koji se fokusiraju na složenije ili raznolikije povratne informacije.

5.5.3. Rezultati Llama i GPT modela

Rezultati modela *Code Llama* i Chat GPT-4 pokazuju jasne i uočljive razlike. Svaki modeli za unosni podatak imao je šest različitih programskih kodova, koji su bili podijeljeni u tri kategorije programskih zadataka s ispravnom i neispravnom implementacijom. Chat GPT-4, najnoviji i najnapredniji model u GPT obitelji, daje rezultate koji su impresivno detaljni i sveobuhvatni. Za svaki ulazni kod pruža dubinsku procjenu, nudeći uvid u točnost ili netočnost koda. Osim što identificira potencijalne probleme, već također predlaže kako se izvorni kod može ispraviti ili poboljšati. U mnogim slučajevima Chat GPT-4 čak uključuje primjer ispravljenog koda, nakon čega slijede jasne upute za poboljšanje i sažeti sažetak cijelog procesa. Takva je izvedba u skladu s očekivanjima, posebno s obzirom na značajan napredak u GPT tehnologiji u posljednjih nekoliko mjeseci. Model iskorištava goleme računalne resurse, radeći na iznimno snažnoj infrastrukturi, što pridonosi njegovoj sposobnosti da proizvede ove visokokvalitetne, dobro zaokružene odgovore. Nasuprot tome, model *Code Llama* predstavlja drugačije iskustvo. Njegovi su odgovori obično manje detaljni i nedostaje im dubina analize koju Chat GPT-4 pruža. Dok su odgovori općenito sažetiji i jasniji, nedostaju u pogledu temeljitosti. Kada je riječ o identificiranju grešaka ili grešaka u kodu, model pokazuje djelomičnu točnost. Od tri netočna primjera koda, *Code*

Llama je uspio otkriti i objasniti pogrešku samo u jednom slučaju, koji je uključivao primjer web programiranja s HTML-om. Čak i u ovom slučaju, iako je model ispravno identificirao problem, njegovo je objašnjenje bilo kratko i nedostajalo mu je dubljeg uvida koji bi se mogao očekivati od naprednije analize. Razlika u performansama između dva modela posebno je očita kada se uspoređi ukupna kvaliteta njihovih odgovora. Chat GPT-4 dosljedno daje vrhunske rezultate, nudeći ne samo točnost u otkrivanju grešaka, već i detaljnije obrazloženje i praktična rješenja. Dok s druge strane *Code Llama*, pruža ograničen pogled, kako u smislu identificiranja problema tako i u ponudi smislenih prijedloga za poboljšanje. Takvu razliku u kvaliteti naglašava napredak koji je Chat GPT-4 napravio u razumijevanju i obradi složenih zadataka programiranja.

Fino podešeni model CodeT5+, u usporedbi s GPT-4 i *Code Llama*, pokazuje srednje performanse s prednosti u općoj identifikaciji funkcionalnosti, ali i nedostacima u pružanju detaljnih povratnih informacija. Dok GPT-4 nudi duboku, sveobuhvatnu analizu, pa čak i ispravljene primjere koda, CodeT5+ ima tendenciju generiranja jednostavnijih procjena kojima često nedostaje dubine, posebno za složenije ili pogrešne isječke koda. CodeT5+ model ispravno identificira osnovne funkcionalnosti koda, kao što je pronalaženje maksimalne vrijednosti u nizu, ali često ne rješava rubne slučajeve ili potencijalne optimizacije, slično sažetijim i nepotpunim odgovorima *Code Llama*. Međutim, CodeT5+ ima tendenciju ponuditi nešto korisnije savjete od *Code Llama*, iako mu nedostaje dubina i jasnoća koju pruža GPT-4. CodeT5+ se također suočava s izazovima prepoznavanja grešaka u kodu i nijansiranom analizom koda, gdje GPT-4 dosljedno nadmašuje pružajući ne samo precizno otkrivanje grešaka, već i sveobuhvatnije obrazloženje i smjernice. Za razliku od *Code Llama*, kojoj su odgovori često prekratki i izostavljaju važne pojedinosti, CodeT5+ nudi ravnotežu između kratkoće i detalja, no ne uspijeva uskladiti sofisticirani rezultat GPT-4.

Model	Rezultat	Ocjena
Code Llama	<p>This code is a valid JavaScript program that defines a function called <code>addNumbers</code> that takes two numbers as input and returns their sum. The function can be called with any two numbers, such as <code>addNumbers(10, 20)</code>, and it will return the sum of those numbers. Here's a breakdown of the code: <code>function addNumbers(num1, num2) { ... }</code> declares a function called <code>addNumbers</code> that takes two arguments <code>num1</code> and <code>num2</code>. <code>return num1 + num2;</code> returns the sum of the two numbers. <code>var result = addNumbers("10", "20");</code> calls the <code>addNumbers</code> function with the arguments <code>"10"</code> and <code>"20"</code>, and assigns the return value to a variable called <code>result</code>. <code>console.log("The result is:" + result);</code> logs the string <code>"The result is:"</code> followed by the value of <code>result</code> to the console using <code>console.log()</code>.</p>	Neutralno
Chat GPT	<p>The code you've written works, but it has some potential issues and areas for improvement. Here's a detailed assessment:</p> <ol style="list-style-type: none"> 1. Strengths: <p>Basic Functionality: The function <code>addNumbers(num1, num2)</code> correctly returns the sum of <code>num1</code> and <code>num2</code>. The function itself is simple and works as expected for numeric inputs.</p> <p>Readability: The code is easy to read and understand, with clear function and variable names.</p> 2. Issues: <ol style="list-style-type: none"> 1. Type Coercion: <ul style="list-style-type: none"> ○ The arguments <code>"10"</code> and <code>"20"</code> are passed as strings, not numbers. JavaScript is dynamically typed and will attempt to coerce the strings into numbers for the <code>+</code> operation. In this case, the function works because JavaScript will convert <code>"10"</code> and <code>"20"</code> to numbers (10 and 20), and the result will be 30. ○ However, this behavior is implicit and can cause confusion or bugs in more complex code or when different types of input are passed. For example, if one of the inputs were a string that doesn't represent a valid number (e.g., <code>"abc"</code>), it would result in NaN (Not a Number). 2. String Concatenation Risk: The <code>+</code> operator can also concatenate strings if one or both of the inputs are strings. So if a different type of string input is passed (e.g., <code>"10" + "abc"</code>), it will concatenate the strings instead of adding numbers. 	Dobro
CodeT5+	<p><i>This code is correct: The function <code>addNumbers</code> correctly adds two numbers and logs the result. Keep practicing functions and functions with the same arguments.</i></p>	Loše

Tablica 1 Rezultati neispravnih primjera - matematički i algoritamski zadatci

Model	Rezultat	Ocjena
Code Llama	This code defines a function called `bubbleSort` that takes an array as input and sorts it in ascending order using the bubble sort algorithm. The function iterates over the array, comparing adjacent elements and swapping them if they are in the wrong order. It then returns the sorted array. The code also logs the result of calling the `bubbleSort` function with an example input array to the console using `console.log()`. Overall, this code defines a simple sorting algorithm that uses the bubble sort method and demonstrates how to call it in JavaScript.	Dobro
Chat GPT	This implementation of the Bubble Sort algorithm is correct and follows a standard approach. Here's an assessment of the code: <ol style="list-style-type: none"> Positive Aspects: Correct Logic: The code correctly implements the Bubble Sort algorithm. It uses nested loops where the inner loop compares adjacent elements and swaps them if they are out of order, ensuring the largest unsorted element "bubbles up" to its correct position after each pass of the outer loop. No Issues with Edge Cases: The code can handle edge cases like an empty array or a single-element array without issues. Readable Code: The code is well-structured and easy to read with clear variable names and logic. 	Dobro
CodeT5+	<i>This code is correct: The `bubbleSort` function correctly sorts an array of numbers and logs the result. Keep practicing sorting of strings and arrays..</i>	Dobro

Tablica 2 Rezultati ispravnih primjera - strukture podataka

Model	Rezultat	Ocjena
Code Llama	This code uses the `fetch` API to make a GET request to an API endpoint at `https://api.example.com/data`. The response is expected to be in JSON format, and the code uses the `response.json()` method to parse the response as JSON. The first `.then` block handles the successful response from the API, and logs the data to the console using `console.log()`. The second `.then` block catches any errors that may occur during the request or parsing of the response, and logs the error to the console using `console.log('Error:', error)`. Overall, this code makes a GET request to an API endpoint, parses the JSON response, and logs the data to the console if successful, or logs any errors that may occur during the process.	Dobro
Chat GPT	The code you provided contains an issue with how the `addEventListener()` method is used. Here's a detailed assessment of the code: <ol style="list-style-type: none"> Issue: 	Dobro

	<p>2. Incorrect Function Call in Event Listener:</p> <ul style="list-style-type: none"> ○ The line: <code>button.addEventListener('click', showAlert());</code> is incorrect because it immediately invokes the <code>showAlert()</code> function as soon as the script runs, instead of assigning it as an event handler. In JavaScript, using <code>()</code> after a function name calls the function right away, so the alert will appear as soon as the page loads, not when the button is clicked. ○ The event listener expects a function reference, not the result of calling the function. 	
CodeT5+	<i>This code is correct: The `showAlert` function correctly handles the alert and logs the result. Review the API response and response processing code.</i>	Loše

Tablica 3 Rezultati neispravnih primjera - web programiranje

5.5.4. Ograničenja istraživanja

Tijekom ovog istraživanja naišlo se na nekoliko ograničenja, koja su utjecala na opseg i ishode procesa finog podešavanja i evaluacije modela. Jedno od ograničenja istraživanja bilo je ograničenje računalnim resursima. Dostupni hardver za ovo istraživanje uključivao je računalo s ograničenom snagom RAM-a i CPU-a, što je ograničavalo mogućnost učinkovitog finog podešavanja modela velikih razmjera. Ovo ograničenje je imalo izravan utjecaj na proces treniranja, jer je ograničavalo veličinu i složenost modela kojima se moglo rukovati, kao i mogućnost eksperimentiranja s naprednijim arhitekturama ili tehnikama optimizacije. Time je proces finog podešavanja bio sporiji i manje iscrpan od idealnog. Nadalje, postojala je i vremenska ograničenost prikupljana podataka. Ovo je ograničenje vjerojatno doprinijelo poteškoćama modela u generiranju nijansiranih i detaljnijih povratnih informacija, budući da podaci za treniranje nisu bili složeni i raznovrsni potrebni za dublje učenje. Opsežniji skup podataka, uključujući širok raspon isječaka koda iz različitih domena i razina težine, omogućio bi robusniji proces treniranja i model s boljom izvedbom. Još jedno ključno ograničenje bila je nedostupnost dobro odabranih skupova podataka otvorenog koda posebno dizajniranih za zadatke procjene koda. Model je uvježban na podacima koji možda nisu bili u potpunosti reprezentativni za složenost uključenu u procjenu koda. Bez pristupa specifičnijim skupovima podataka, proces finog podešavanja mogao je propustiti ključne aspekte razumijevanja koda, posebno za složenu logiku ili rubne slučajeve. Ova ograničenja značajno su utjecala na istraživački proces, ograničavajući i dubinu treniranja modela i opseg evaluacije.

5.5.5. Upute za poboljšanje

Identificirano je nekoliko područja za poboljšanje, što bi moglo dovesti do boljih rezultata u budućem radu. Kako bi se prevladala ograničenja s kojima se suočava ovo istraživanje i poboljšala izvedba modela procjene koda, predlažu se slijedeći savjeti. Skup podataka korišten u ovom istraživanju vjerojatno je bio ograničen i veličinom i složenošću, što je doprinijelo poteškoćama modela u generiranju nijansiranih povratnih informacija. Budući rad trebao bi dati prioritet pohranjivanju sveobuhvatnijeg skupa podataka koji uključuje različite isječke koda u različitim programskim jezicima, domenama i razinama težine. Ovaj skup podataka također bi trebao sadržavati složenije i rubne scenarije koji zahtijevaju dublje razumijevanje koda, kao što su izazovi kodiranja u stvarnom svijetu ili veći, višenamjenski projekti. Kombinacija ljudskog i strojno generiranog koda mogla bi pružiti bolji uvid u sposobnost modela da procijeni kvalitetu i ispravnost koda. Dok je CodeT5 bio prikladan izbor za ovo istraživanje, eksperimentiranje sa snažnijim i naprednijim modelima moglo bi dati bolju izvedbu. Modeli kao što su GPT-4, Codex ili bilo koje novije varijante posebno dizajnirane za razumijevanje i generiranje koda trebaju se uzeti u obzir za buduće eksperimente. Ovi bi modeli mogli ponuditi bolje razumijevanje sintakse, logike i složenih struktura koda. Jedno od značajnih ograničenja s kojima se suočavalo tijekom ovog istraživanja bio je nedostatak računalne snage. Nadogradnja na hardver s većim RAM-om, snažnijim CPU-ima i pristupom GPU-ovima ili TPU-ovima omogućila bi brže i učinkovitije treniranje modela. Korištenje rješenja ili platformi temeljenih na oblaku kao što su Google Cloud ili Azure, koje nude opširnije računalne resurse, omogućilo bi budućem istraživanju eksperimentiranje s većim modelima i složenijim podešavanjem hiperparametara, što bi potencijalno moglo dovesti do boljih performansi.

Zaključak

U ovom istraživanju željeli smo odgovoriti na dva ključna istraživačka pitanja: (1) Može li se postojeći veliki jezični model, fino podesiti za detaljnu procjenu izvornog koda? i (2) Nadmašuje li prilagođeni fino podešeni model utvrđene modele velikih jezika u smislu procjene koda? Istraživanje je uključivalo niz koraka, od prikupljanja podataka i predobrade, preko treniranja modela i evaluacije, do podešavanja hiperparametara korištenjem biblioteke Optuna. Rezultati ovog istraživanja sugeriraju da, iako se fino podešeni model CodeT5 može prilagoditi za procjenu ispravnosti izvornog koda, on se bori s pružanjem detaljnih i nijansiranih povratnih informacija. BLEU rezultat modela od 0,404 i ROUGE-1 rezultat od 0,62 pokazuju da je učinkovit u prepoznavanju ispravnog koda, ali njegove povratne informacije ostaju uglavnom na površinskoj razini, nedostaje im dublja analiza ili identifikacija suptilnih problema, kao što su implicitne pretvorbe tipa podataka. Ovo sugerira da, iako se model može fino podesiti za osnovnu procjenu koda, postizanje detaljnijih procjena zahtijeva daljnja poboljšanja u raznolikosti podataka za treniranje i sposobnosti generiranja povratnih informacija. Što se tiče drugog istraživačkog pitanja, iako je prilagođeni fino podešeni model pokazao neka poboljšanja u odnosu na osnovni model CodeT5, osobito u rukovanju osnovnim isječcima koda i procjenama ispravnosti, nije presudno nadmašio postojeće modele u pružanju sveobuhvatnih povratnih informacija. Utvrđeni veliki jezični modeli, poput GPT-a, mogu ponuditi detaljne povratne informacije bez finog podešavanja zbog svoje izloženosti raznolikim i velikim skupovima podataka. Nasuprot tome, fino podešeni model CodeT5 pokazao je znakove pretjeranog prilagođavanja jednostavnijim obrascima u podacima za treniranje, ograničavajući njegovu sposobnost da ponudi konstruktivnu kritiku ili da se nosi sa složenijim scenarijima kodiranja. Dok je fino podešavanje postojećeg velikog jezičnog modela kao što je CodeT5 za procjenu koda izvedivo, prilagođeni model zahtijeva daljnje usavršavanje kako bi uistinu briljirao u generiranju detaljnih povratnih informacija.

Popis slika

Slika 1 Primjer skupa podataka - <i>Code</i> stupac	23
Slika 2 Primjer skupa podataka - <i>Assessment</i> stupac	24
Slika 3 Primjer ulaznog podataka za procjenu	30
Slika 4 Korištene biblioteke	33
Slika 5 Priprema skupa podataka 1.....	33
Slika 6 Priprema skupa podataka 2.....	34
Slika 7 Tokenizacija podataka.....	34
Slika 8 Optimizacija arugmenta za treniranje.....	35
Slika 9 Treniranje modela.....	36
Slika 10 Metrike za evaluaciju modela	37
Slika 11 Evaluacija modela	37
Slika 12 Ulazni podatak za evaluaciju - Matematički i algoritamski zadatci 1.....	38
Slika 13 Ulazni podatak za evaluaciju - Matematički i algoritamski zadatci 2.....	39
Slika 14 Ulazni podatak za evaluaciju - Strukture podataka 1	39
Slika 15 Ulazni podatak za evaluaciju - Strukture podataka 2.....	40
Slika 16 Ulazni podatak za evaluaciju - Web programiranje 1	40
Slika 17 Ulazni podatak za evaluaciju - Web programiranje 2.....	41
Slika 18 <i>ChatGpt4</i> rezultat - Matematički i algoritamski zadatci 1	41
Slika 19 <i>ChatGpt4</i> rezultat - Matematički i algoritamski zadatci 2	42
Slika 20 <i>ChatGpt4</i> rezultat - Matematički i algoritamski zadatci 2	42
Slika 21 <i>ChatGpt4 rezultat</i> - Web programiranje 1	42
Slika 22 <i>ChatGpt4</i> rezultat - Web programiranje 1.....	43
Slika 23 <i>ChatGpt4</i> rezultat - Strukture podataka 1	43
Slika 24 <i>ChatGpt4</i> rezultat - Strukture podataka 1	43

Slika 25 <i>ChatGpt4</i> rezultat - Web programiranje 2.....	44
Slika 26 <i>ChatGpt4</i> rezultat - Strukture podataka 2	44
Slika 27 <i>Code Llama</i> rezultat - Matematički i algoritamski zadatci 1	45
Slika 28 <i>Code Llama</i> rezultat - Matematički i algoritamski zadatci 2	45
Slika 29 <i>Code Llama</i> rezultat - Strukture podataka 1.....	46
Slika 30 <i>Code Llama</i> rezultat - Strukture podatka 2	46
Slika 31 <i>Code Llama</i> rezultat - Web programiranje 1	47
Slika 32 <i>Code Llama</i> rezultat - Web programiranje 2.....	47
Slika 33 Ulazni podaci CodeT5- matematički i algoritamski zadatci 1	48
Slika 34 Ulazni podaci CodeT5 - matematički i algoritamski zadatci 2	48
Slika 35 Ulazni podaci CodeT5 - Strukture podataka 1	49
Slika 36 Ulazni podaci CodeT5 - strukture podataka 2.....	49
Slika 37 Ulazni podaci CodeT5 - Web programiranje 1	50
Slika 38 Ulazni podaci CodeT5 - Web programiranje 2	50

Popis tablica

Tablica 1 Rezultati neispravnih primjera - matematički i algoritamski zadatci	55
Tablica 2 Rezultati ispravnih primjera - strukture podataka	56
Tablica 3 Rezultati neispravnih primjera - web programiranje	57

Literatura

- Barchi, F., Parisi, E., Bartolini, A., & Acquaviva, A. (2022). Deep Learning Approaches to Source Code Analysis for Optimization of Heterogeneous Systems: Recent Results, Challenges and Opportunities.
- Beltagy, I., Lo, K., & Cohan, A. (2019). SCIBERT: A Pretrained Language Model for Scientific Text.
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., . . . McCan, S. (2020). Language Models are Few-Shot Learners.
- Chen, X., Xiang, J., Lu, S., Liu, Y., He, M., & Shi, D. (2024). Evaluating large language models in medical applications: a survey.
- EITCA. (Svibanj 2024). *Šta znači podešavanje hiperparametara?* Dohvaćeno iz EITCA: <https://bs.eitca.org/umjetne-inteligencije/eitc-ai-gcml-google-ma%C5%A1insko-u%C4%8Denje-u-oblaku/uvod-a/%C5%A1ta-je-ma%C5%A1insko-u%C4%8Denje/%C5%A1ta-zna%C4%8Di-pode%C5%A1avanje-hiperparametara-a/>
- EITICA. (2023). *What is text classification and why is it important in machine learning?* Dohvaćeno iz EUROPEAN INFORMATION TECHNOLOGIES CERTIFICATION ACADEMY - ATTESTING YOUR PROFESSIONAL DIGITAL SKILLS: <https://eitca.org/artificial-intelligence/eitc-ai-tff-tensorflow-fundamentals/text-classification-with-tensorflow/preparing-data-for-machine-learning/examination-review-preparing-data-for-machine-learning/what-is-text-classification-and-why-is-it-importan>
- Eric Nichols, L. G. (2020). Collaborative Storytelling with Large-scale Neural.
- Friedman, N. (2022). *GitHub blog*. Dohvaćeno iz Introducing GitHub Copilot: your AI pair programmer: <https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/>
- Gan, W., Qi, Z., Wu, J., & Lin, J. C.-W. (2023). Large Language Models in Education:.
- GeeksForGeeks. (2024). *Dropout vs weight decay*. Dohvaćeno iz GeeksForGeeks: <https://www.geeksforgeeks.org/dropout-vs-weight-decay/>

- Jacob Murel Ph.D., E. K. (Svibanj 2024). *IBM*. Dohvaćeno iz What is text summarization?:
<https://www.ibm.com/topics/text-summarization>
- Kaddour, J., Harris, J., Mozes, M., Bradley, H., Raileanu, R., & McHardy, R. (2023).
Challenges and Applications of Large Language Models.
- Lee, J., Yoon, W., Kim, S., Kim, D., So, C. H., & Kang, J. (2019). BioBERT: a pre-trained
biomedical language representation model for biomedical text mining.
- Lei Zhang, B. L. (2018). Deep Learning for Sentiment Analysis: A Survey.
- Li, H., Ca, D., Xu, J., & Watanabe, T. (2022). N-gram Is Back: Residual Learning of Neural
Text Generation with n-gram Language Model. Hong Kong.
- Mark, C., Jerry, T., Qiming, J. H., Pinto, Y., & Oliveira, H. P. (2021). Code, Evaluating
Large Language Models Trained on Code.
- Mazoor, K. (2024). *Code Generation with Large Language Models (LLMs)*. Dohvaćeno iz
Linked in: <https://www.linkedin.com/pulse/code-generation-large-language-models-llms-kashif-manzoor-n8bvf>
- Medium. (2023). *llms and chatbots a match made in tech heaven*. Dohvaćeno iz Medium:
<https://medium.com/@interacly/llms-and-chatbots-a-match-made-in-tech-heaven-dc548583146c>
- Medium. (2024). *Medim*. Dohvaćeno iz Learning Rate and Its Strategies in Neural Network
Training: <https://medium.com/thedeephub/learning-rate-and-its-strategies-in-neural-network-training-270a91ea0e5c>
- Mehra, A. (2023). *KDnuggets*. Dohvaćeno iz A Deep Dive into GPT Models: Evolution &
Performance Comparison: <https://www.kdnuggets.com/2023/05/deep-dive-gpt-models.html>
- Miltiadis Allamanis, E. T. (2018). A Survey of Machine Learning for Big Code and
Naturalness.
- Milton-Barker, A. K. (2024.). *innov8digitalmedia*. Dohvaćeno iz Large Language Models:
The Future of Marketing?: <https://www.innov8digitalmedia.com/blog/advanced-technologies-development/large-language-models>
- Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., & Gao, J.
(2024). Large Language Models: A Survey.

- mljourney. (2024). *ML Journey*. Dohvaćeno iz How Does Batch Size Affect Training?: <https://mljourney.com/how-does-batch-size-affect-training/>
- Najib, T. (2024). *Medium*. Dohvaćeno iz Hyperparameter Tuning Using Optuna: <https://medium.com/@taefnajib/hyperparameter-tuning-using-optuna-c46d7b29a3e>
- Nazi, Z. A., & Peng, W. (2024). LARGE LANGUAGE MODELS IN HEALTHCARE AND MEDICAL DOMAIN: A REVIEW.
- Ojokoh, B., & Adebisi, E. (2019). A Review of Question Answering Systems.
- Pešut, L. (Studen 2019). REKURENTNE NEURONSKE MREZE. Zagreb, Hrvatska.
- Rsvmukhesh. (2023). *Medium*. Dohvaćeno iz Determining the Number of Epochs: <https://medium.com/@rsvmukhesh/determining-the-number-of-epochs-d8b3526d8d06>
- Sarsa, S., Leinonen, J., Koutcheme, C., & Hellas, A. (2022). Speeding Up Automated Assessment of Programming Exercises.
- Schmidhuber, S. H. (15. 11 1997). Long Short-Term Memory in Neural Computation.
- Scripted. (Kolovoz 2023). *Scripted*. Dohvaćeno iz Understanding LLMs is the Secret to Marketing Content That Performs: <https://www.scripted.com/ai/understanding-llms>
- Sifei Luan, D. Y. (2019). Aroma: Code Recommendation via Structural Code Search.
- Takyar, A. (2024). *LeewayHertz - AI Development Company*. Dohvaćeno iz named-entity-recognition: <https://www.leewayhertz.com/named-entity-recognition/#Key-components-of-named-entity-recognition>
- Toutanova, J. D.-W. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., & Polosukhin, I. (2017). Attention Is All You Need.
- Wang, S., Xu, T., Li, H., Zhang, C., Liang, J., Tang, J., . . . Wen, Q. (2024). Large Language Models for Education: A Survey and Outlook.
- Wang, Y., Le, H., Gotmare, A. D., Bui, N. D., Li, J., & Hoi, S. C. (2023). CodeT5+: Open Code Large Language Models for Code Understanding and Generation.

Wikipedia. (2020). *Wikipedia*. Dohvaćeno iz Strojno prevođenje:
https://hr.wikipedia.org/wiki/Strojno_prevo%C4%91enje

Wikipedia. (2024). *Long short-term memory*. Dohvaćeno iz
https://en.wikipedia.org/wiki/Long_short-term_memory#cite_note-hochreiter1991-2

Yang Liu, M. L. (5. Rujan 2019). Text Summarization with Pretrained Encoders.

Yendur, G., M, R., G, C. S., Y, S., Srivastava, G., Maddikunta, P. K., . . . Gadekallu, T. R. (2024). GPT (Generative Pre-trained Transformer) – A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions.

Yinhan Liu, M. O. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach.

Zhuohan Xie, T. C. (2023). The Next Chapter: A Study of Large Language Models in Storytelling.