Veliki jezični modeli za generiranje ugradbenih vektora

Ivanković, Goran

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Goran Ivanković

VELIKI JEZIČNI MODELI ZA GENERIRANJE UGRADBENIH VEKTORA

Diplomski rad

Voditelj rada:
doc. dr. sc. Marko Horvat (PMF)
doc. dr. sc. Marko Horvat (FER)

Zagreb, veljača, 2025.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

U današnjem digitalnom dobu, količina dostupnih informacija i podataka jako brzo raste, što dovodi do problema u obradi i pretraživanju teksta. Čovjek jednostavno ne može pročitati svaku moguću knjigu ili sve moguće dokumente vezane za temu koja ga zanima. S obzirom na to, razvijeni su sofisticirani algoritmi i modeli za pretraživanje i generiranje teksta, među kojima su ugradbeni vektori te veliki jezični modeli. Ovi pristupi omogućuju automatiziranu obradu prirodnog jezika te olakšavaju navigaciju i razumijevanje informacija u digitalnom okruženju.

Ugradbeni vektor, (eng. *embedding*), jedan je od temeljnih pojmova u obradi prirodnog jezika. Ugradbeni vektori omogućuju prikaz riječi, rečenica ili dokumenata u obliku vektora unutar višedimenzionalnog prostora, pri čemu se zadržava njihova sličnost na temelju značenja. Tako dobiveni ugradbeni vektori koriste se za lakšu daljnju obradu teksta, poput prevođenja, klasifikacije, prepoznavanja ključnih pojmova ili pretraživanja teksta. Razvoj novih arhitektura poput *transformera* [56] revolucionirao je područje obrade prirodnog jezika jer se omogućilo bolje razumijevanje teksta, uključujući njegov kontekst. Zbog toga su transformeri pronašli primjenu i u generiranju ugradbenih vektora, gdje nadmašuju tradicionalne metode pretraživanja, poput BM25 [42].

Osim za već navedene zadatke, transformeri su se pokazali izvrsnim i u generiranju teksta. U posljednje vrijeme vidjeli smo porast broja aplikacija nalik ChatGPT-u [37]. Većina koristi velike jezične modele bazirane upravo na transformerima. Međutim, modeli često ne uspijevaju odgovoriti činjenično točno ili pružiti izvor za svoj odgovor. Stoga su se pojavili napredni sustavi poput RAG-a (Retrieval-Augmented Generation) [28] koji kombiniraju generativne modele s pretraživanjem teksta. U toj arhitekturi, prvi korak uključuje pretraživanje dokumenata relevantnih za korisnikov upit, pri čemu se često koriste ugradbeni vektori. Na temelju dohvaćenih dokumenata, generativni model dohvaćene dokumente zatim obradi s korisničkim upitom te tako generira precizniji i točniji odgovor. Ova integracija postavlja temelje za razvoj modernih aplikacija u područjima inteligentnih asistenata i chatbotova.

Glavni cilj ovog diplomskog rada je istražiti i analizirati primjenu ugradbenih vektora u sustavima za pretraživanje i obradu teksta. Obradit ćemo arhitekturu

transformerskih modela [56] i opisati kako se oni treniraju, odnosno kako modeli iz podataka uče generirati odgovarajuće izlaze za dane ulazne podatke. Također ćemo prikazati razlike između jezičnih i ugradbenih modela. Objašnjavamo proces koji zovemo prijenosno učenje [38] i omogućuje prilagodbu postojećeg modela za novi zadatak uz značajno smanjenje računalnih zahtjeva u odnosu na treniranje od nule. Posebni naglasak stavljamo na sustave RAG koji koriste ugradbene modele u svrhu pretraživanja teksta. Iako se u zadatku generiranja ugradbenih vektora tradicionalno koriste manji i brži modeli [58, 29], nedavna istraživanja sugeriraju da se veliki jezični modeli mogu uspješno prilagoditi za istu svrhu [31, 59].

Naš doprinos u ovom radu uključuje razvoj jednog takvog ugradbenog modela prilagođenog hrvatskom jeziku. Za generiranje ugradbenih vektora koristili smo javno dostupan veliki jezični model Gemma2 9b [50], koji smo dodatno trenirali na našem skupu podataka kako bismo poboljšali kvalitetu ugradbenih vektora za hrvatske tekstove. Naš skup podataka sastoji se od prevedenih dostupnih korpusa na engleskom jeziku te sadržaja hrvatske Wikipedije. Usporedili smo ovaj pristup s klasičnom metodom BM25 te s modelom BERT. Na kraju smo implementirali sustav RAG prilagođen za pretraživanje i generiranje teksta na hrvatskom jeziku.

Poglavlje 1

Ugradbeni vektori

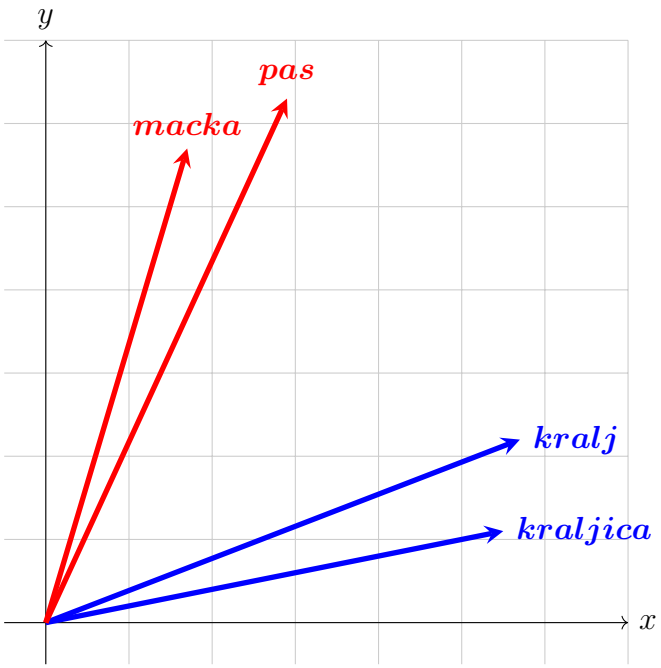
U ovom poglavlju definiramo *ugradbene vektore* i *latentni prostor*, ključne pojmove za reprezentaciju teksta u numeričkom obliku. Opisujemo metode pretraživanja teksta temeljem ugradbenih vektora te formalno definiramo ugradbeni model kao preslikavanje dokumenata u latentni prostor. Uz to, razmatramo i klasičnu metodu pretraživanja BM25.

Nakon pregleda pretraživačkih metoda, uvodimo dvije metrike: srednji recipročni rang (MRR, eng. *Mean Reciprocal Rank*) i stopu pogodaka (HR@k, eng. *Hit Rate at k*) [57], koje se koriste za procjenu kvalitete pretraživanja. Na kraju poglavlja ističemo ključne primjene ugradbenih vektora u različitim sustavima, uključujući pretraživanje informacija, sustave za preporuku sadržaja i generativne modele.

1.1 Motivacija i definicije

Odgovorimo prvo na pitanje kako se u računalu reprezentiraju različiti tipovi podataka. Brojevi se reprezentiraju u binarnom sustavu, odnosno nizom bitova. Slike se reprezentiraju nizom brojeva koji označavaju svjetlinu boja na svakom pikselu u slici. Svakom znaku pridružujemo broj prema UNICODE-u [52], a taj broj najčešće zapisujemo UTF8 kodiranjem. Riječi i rečenice su nizovi znakova.

Problem kod ove reprezentacije rečenica i slika je što nema semantičko značenje. Dvije fotografije koje prikazuju istog psa neće biti sličnog zapisa. Prvo, slike mogu biti različite veličine, pa će i zapisi biti različite dužine. Drugo, pas može izgledati drugačije i pikseli na slici mogu biti druge boje ili svjetline. Isto vrijedi i za tekst. Riječi *kralj* i *vladar* imaju potpuno drukčiji zapis, a slično značenje, dok riječi *voda* i *vođa* imaju skoro identičan zapis, ali drugačije značenje. Postavlja se pitanje možemo li zapisati ove riječi tako da se sačuva semantika. Odgovor je potvrdan. Možemo ih



Slika 1.1: Primjer reprezentacije riječi u latentnom prostoru

prikazati u prostoru vektora koji zovemo latentni prostor. Slijedi gruba definicija, a zatim govorimo o njegovim poželjnim svojstvima.

Definicija 1.1.1. ***Latentni prostor*** *definiramo kao konačno dimenzionalni vektorski prostor* $\mathbb{R}^d$ *koji reprezentira objekte iz stvarnog svijeta, gdje s* d *označavamo dimenziju pripadnog vektorskog prostora.* ***Ugradbeni vektor*** *je vektor u odabranom latentnom prostoru kojim reprezentiramo dani objekt.*

Iako vektori u latentnom prostoru ne mogu savršeno reprezentirati puno značenje objekata koje reprezentiraju, jer je stvarni svijet izuzetno složen, možemo govoriti o nekim poželjnim svojstvima latentnog prostora. Jedan primjer je očuvanje sličnosti, odnosno zahtjev da ugradbeni vektori objekta sličnog značenja imaju ili malu euklidsku udaljenost ili mali kut između vektora. Primjer je prikazan na slici 1.1. O udaljenostima ugradbenih vektora će biti više riječi u poglavlju 1.2.

Latentni prostor ne reprezentira bilo koje objekte, već samo promatrani skup objekata, pa govorimo primjerice o latentnom prostoru za slike ili latentnom prostoru za tekst. U nastavku teksta govorimo o latentnom prostoru za tekst. Počevši od reprezentacija u tom prostoru, dalje možemo klasificirati ili grupirati tekstove, pretraživati

slične tekstove, pa čak i predviđati nastavak teksta. Više o takvim primjenama govorimo u poglavlju 1.4.

Željena svojstva latentnog prostora proizlaze iz algoritma kojim objekte preslikavamo u ugradbene vektore. Prvi uspješni algoritmi koji su generirali ugradbene vektore za tekst bili su GloVe [39] i Word2Vec [35]. Danas se najviše koristi arhitektura transformera o kojima je riječ u poglavlju 2. Istraživanja [35] su pokazala da se sličnost pojmova ne očituje samo kroz mali kut između njihovih ugradbenih vektora, već i kroz složenije odnose među njima.

Primjer 1.1.2. *U latentnom prostoru, gdje su riječi, pojmovi i njihovi međusobni odnosi predstavljeni u obliku ugradbenih vektora, možemo analizirati semantičke veze među pojmovima. Primjerice, promatramo ugradbene vektore V_{Pariz}, $V_{Francuska}$, V_{Rim} i $V_{Italija}$. Vektorska operacija:*

$$V_{Francuska} - V_{Pariz} + V_{Italija}$$

rezultira novim vektorom koji je po kutnoj udaljenosti najbliži vektoru V_{Rim} u odnosu na sve druge vektore u latentnom prostoru. Ovaj primjer ilustrira kako ugradbeni vektori modeliraju analogne odnose između pojmova, poput veze između države i njezina glavnog grada.

Iako i u samom radu navode da se ovakva linearna svojstva ne zadržavaju uvijek te svakako ovise o korištenom algoritmu za ugrađivanje teksta, gornji primjer pomaže intuitivnom shvaćanju latentnih prostora. Slijede definicije osnovnih pojmova koje ćemo koristiti u nastavku rada.

Definicija 1.1.3. ***Vokabular*** *$\mathcal{V}$ je konačan neprazan skup, a njegove elemente nazivamo* ***riječi****.*

Definicija 1.1.4. ***Dokument*** *nad $\mathcal{V}$ je konačan niz riječi $D = (w_1, \ldots, w_l)$, gdje je $l \in \mathbb{N}$ te $w_i \in \mathcal{V}$. Broj riječi u dokumentu označavamo s $|D| = l$. Sa $\mathcal{V}^*$ označavamo skup svih dokumenata nad $\mathcal{V}$.*

Dokumente obično označavamo velikim slovima D ili Q, a konačni skup dokumenata s $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$, gdje je $n \in \mathbb{N}$ te $\mathcal{D} \subseteq \mathcal{V}^*$. Za konačan skup dokumenata također koristimo naziv baza dokumenata. Koristimo oznaku $\mathcal{V}^*$ kada želimo naglasiti da se radi o proizvoljnom dokumentu, a $\mathcal{D}$ kada se radi o dokumentu iz određene baze dokumenata. Vokabular $\mathcal{V}$ smatramo fiksiranim, osim kada izričito navedemo.

1.2 Pretraživanje teksta

U ovom poglavlju opisat ćemo dvije metode pretraživanja teksta, BM25 i pretraživanje pomoću ugradbenih vektora. BM25 nudi efikasno rangiranje dokumenata prema učestalosti pojmova, dok druga metoda omogućuje semantičko pretraživanje na temelju udaljenosti ugradbenih vektora tekstova.

Kad govorimo o sličnosti dokumenata, mislimo na sličnost u sadržaju dokumenata. Preciznije, definiramo funkciju sličnosti na bazi dokumenata $\mathcal{D}$ na sljedeći način:

Definicija 1.2.1. ***Funkcija sličnosti*** *na $\mathcal{D}$ je funkcija sim : $\mathcal{V}^* \times \mathcal{D} \to \mathbb{R}$. Prvi argument nazivamo* ***upit****. Vrijednost funkcije predstavlja sličnost upita i danog dokumenta, s tim da veći broj predstavlja veću sličnost.*

Konkretnu funkciju sličnosti, baziranu na kutnoj udaljenosti ugradbenih vektora, definirat ćemo u sljedećem poglavlju. Funkciju sličnosti koristimo za pretraživanje dokumenata. Za dani upit Q, bazu dokumenata $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ i definiranu funkciju sličnosti sim na $\mathcal{D}$, dokumente iz baze dokumenata možemo pretražiti i vratiti onaj dokument koji daje najveću vrijednost funkcije sličnosti. Stoga za funkciju sličnosti sim na bazi dokumenata $\mathcal{D}$ definiramo funkciju pretraživanja $\text{search}_{\text{sim}} : \mathcal{V}^* \to \mathcal{D}$ sljedećom formulom:

$$\text{search}_{\text{sim}}(Q) = \underset{D \in \mathcal{D}}{\text{argmax}}\, \text{sim}(Q, D)$$

gdje $\text{search}_{\text{sim}}(Q)$ predstavlja rezultat pretraživanja za upit Q, odnosno najrelevantniji dokument. Također definiramo funkciju rangiranja $\text{rank}_{\text{sim}} : \mathcal{V}^* \times \mathcal{D} \to \mathbb{N}$ na sljedeći način:

$$\text{rank}_{\text{sim}}(Q, D) = \mathbf{card}(\{E \mid E \in \mathcal{D}, \text{sim}(Q, E) > \text{sim}(Q, D)\}) + 1 \tag{1.1}$$

Funkcija rank_{sim} nam govori koji po redu je dokument D po sličnosti s upitom Q od svih dokumenata iz baze dokumenata $\mathcal{D}$. Ako su baza dokumenata $\mathcal{D}$ i funkcija sličnosti sim na $\mathcal{D}$ fiksirani, pišemo samo search i rank. Slijedi primjer korištenja upravo definiranih funkcija.

Primjer 1.2.2. *Zadan je upit Q, baza dokumenata $\mathcal{D} = \{D_1, D_2, D_2\}$ i funkcija rangiranja sim.*

$$\begin{aligned} Q &= \text{„}\check{S}\text{to je inflacija?”} \\ D_1 &= \text{„Ekonomija je znanost o upravljanju resursima.”} \\ D_2 &= \text{„Tehnologija mijenja način na koji radimo i komuniciramo.”} \\ D_3 &= \text{„Inflacija se odnosi na povećanje opće razine cijena.”} \end{aligned}$$

Neka je $sim(Q, D_1) = 0.6$*,* $sim(Q, D_2) = 0.2$*,* $sim(Q, D_3) = 0.9$*. Tada je rezultat pretraživanja* $search(Q)$ *jednak* D_3*. Dokumente rangiramo:*

$$rank(Q, D_1) = 2$$
$$rank(Q, D_2) = 3$$
$$rank(Q, D_3) = 1$$

Pretraživanje teksta pomoću ugradbenih vektora

Pretraživanje teksta pomoću ugradbenih vektora temelji se na udaljenostima definiranim na pripadnom latentnom prostoru. Jednom kad imamo ugradbene vektore tekstova, možemo definirati sličnost dokumenata s obzirom na udaljenost njihovih vektorskih reprezentacija (kutna ili euklidska udaljenost). U nastavku poglavlja ćemo definirati sličnost kosinusa baziranu na kutnoj udaljenosti vektora. Za ovaj postupak potreban je način za preslikavanje objekata iz stvarnog svijeta u latentni prostor. Definiramo ugradbeni model koji generira vektorske reprezentacije (ugradbene vektore) za elemente određenog skupa. Cilj modela je da elemente skupa semantički poveže na način da njihove vektorske reprezentacije imaju što veću sličnost.

Definicija 1.2.3. ***Ugradbeni model*** *(eng. Embedding Model) je funkcija* $f : \mathcal{X} \to \mathbb{R}^{d_{model}}$*. Ovdje je* $\mathbb{R}^{d_{model}}$ *latentni prostor s dimenzijom* d_{model}*, a* $\mathcal{X}$ *skup objekata koji preslikavamo.*

U nastavku rada razmatramo isključivo ugradbene modele koji generiraju ugradbene vektore za tekst, pa fiksiramo $\mathcal{X} = \mathcal{V}^*$. Ulazni podatak za model je bilo koji konačni niz riječi.

U primjeni ponekad koristimo dva modela, jedan model za upite f_Q i jedan model za dokumente f_D. Računanje vrijednosti funkcije f_Q, odnosno f_D može biti sporo na računalu, pogotovo ako su modeli bazirani na arhitekturi transformera koju opisujemo u poglavlju 2. Kako bi pretraživanje bilo brže, dokumente iz baze dokumenata koju pretražujemo pretvaramo u ugradbene vektore *jednom* na početku upotrebe koristeći f_D. Tada je za svako pretraživanje potrebno pokrenuti samo model za upit f_Q.

U latentnom prostoru možemo vršiti standardne vektorske operacije nad ugradbenim vektorima. Stoga definiramo sličnost kosinusa (eng. *Cosine Similarity*) [11] za vektore formulom:

$$\text{cossim}(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\|_2 \, \|\mathbf{d}\|_2} = \cos(\theta), \tag{1.2}$$

gdje su $\mathbf{q}$ i $\mathbf{d}$ vektori iste dimenzije, a θ kut između njih.

Vrijednost sličnosti kosinusa je između -1 i 1 te 1 predstavlja najveću sličnost, odnosno slučaj gdje su oba vektora na istom polupravcu u odnosu na ishodište.

Pokažimo kako se pomoću sličnosti kosinusa rangiraju dokumenti. Neka su ugradbeni modeli zadani kao funkcije, redom model za upite i model za dokumente s $f_Q, f_D : \mathcal{V}^* \to \mathbb{R}^{d_{model}}$, gdje je d_{model} dimenzija pripadnog latentnog prostora za tekst. Tada funkciju sličnosti za ovu metodu definiramo na sljedeći način:

$$\text{sim}(Q, D) = \text{cossim}(f_Q(Q), f_D(D))$$

Dokumente zatim sortiramo po sličnosti njihovih ugradbenih vektora s ugradbenim vektorom upita i vraćamo tim redoslijedom.

Primjer 1.2.4. *Zadan je upit Q i baza dokumenata $\mathcal{D} = \{D_1, D_2, D_2\}$. Ugradbeni modeli f_Q i f_D preslikavaju upit i dokumente u vektorski prostor dimenzije 3. Neka je zadana funkcija sličnosti sa* (1.2)*. Najrelevantniji dokument za upit zatim pronalazimo tako da računamo sličnost kosinusa između pripadnih ugradbenih vektora.*

$$\begin{aligned} f_Q(Q) &= (5, 5.5, 8) \\ f_D(D_1) &= (8, 1, 5) \\ f_D(D_2) &= (2, 9, 1) \\ f_D(D_3) &= (4, 6, 7.5) \end{aligned}$$

Sličnost s dokumentima:

$$\begin{aligned} sim(Q, D_1) &= \frac{(5 \cdot 8) + (5.5 \cdot 1) + (8 \cdot 5)}{\sqrt{5^2 + 5.5^2 + 8^2} \cdot \sqrt{8^2 + 1^2 + 5^2}} \approx 0.82530 \\ sim(Q, D_2) &= \frac{(5 \cdot 2) + (5.5 \cdot 9) + (8 \cdot 1)}{\sqrt{5^2 + 5.5^2 + 8^2} \cdot \sqrt{2^2 + 9^2 + 1^2}} \approx 0.66654 \\ sim(Q, D_3) &= \frac{(5 \cdot 4) + (5.5 \cdot 6) + (8 \cdot 7.5)}{\sqrt{5^2 + 5.5^2 + 8^2} \cdot \sqrt{4^2 + 6^2 + 7.5^2}} \approx 0.99457 \end{aligned}$$

Tada je rezultat pretraživanja $search(Q) = D_3$, jer taj dokument ima najveću sličnost 0.99457 *s upitom.*

BM25

BM25 (Best Matching 25) [42] jedna je od najčešće korištenih metoda [19] za pretraživanje teksta i temelji se na inverznoj frekvenciji po dokumentima (IDF, eng. *Inverse Document Frequency*). Iako je nastala krajem prošlog stoljeća, i dalje se koristi kao referentna točka u usporedbi s novim modelima [51]. Osim dobrih rezultata, glavna prednost je brzina izvršavanja nad korištenjem ugradbenih modela baziranih na transformerima [51]. Ugradbeni modeli trebaju prvo pretvoriti tekst u ugradbeni

vektor, a zatim računati udaljenost sa svim ugradbenim vektorima u bazi, gdje su obje operacije vrlo skupe.

Pretpostavimo da je zadan upit $Q = (q_1, q_2, \ldots, q_m)$, baza dokumenata $\mathcal{D} = \{D_1, \ldots, D_n\}$ i dokument $D = (w_1, \ldots, w_l)$. Definiramo funkciju sličnosti bm25 sa:

$$\text{bm25}(Q, D) = \sum_{i=1}^{m} \left(\text{IDF}(q_i) \cdot \frac{\text{word}(q_i, D) \cdot (k_1 + 1)}{\text{word}(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgl}}\right)} \right),$$

pri čemu su k_1 i b konstantni parametri, $k_1 = 1.2$, $b = 0.75$ [32], $\text{avgl} = \sum_{i=1}^{n} |D_i|/n$ označava prosječnu duljinu dokumenta u bazi dokumenata, a $\text{word}(q, D)$ je broj pojavljivanja riječi q u dokumentu D:

$$\text{word}(q, D) = \mathbf{card}(\{i \,|\, w_i = q,\ i \in \{1, 2, \ldots, l\}\}).$$

Također definiramo inverznu frekvenciju po dokumentima sljedećom formulom:

$$\text{IDF}(q) = \ln \left(\frac{n - \text{doc}(q) + 0.5}{\text{doc(q)}+0.5} + 1 \right),$$

gdje $\text{doc}(q)$ označava broj dokumenata u bazi dokumenata $\mathcal{D}$ koji sadrže riječ q:

$$\text{doc}(q) = \mathbf{card}(\{D \in \mathcal{D} \mid q \in D\}).$$

Često se koristi i kombinacija dvije navedene metode. BM25 pronalazi sve tekstove koji sadrže ključne riječi, dok pretraživanjem ugradbenih vektora teksta pronalazimo semantički slične tekstove. Kao funkcija sličnosti uzima se težinska kombinacija ove dvije funkcije. U sljedećem poglavlju definiramo transformerske modele koji se koriste kao ugradbeni modeli te za razne druge zadatke poput generiranja teksta.

1.3 Metrike pretraživanja

U sustavu pretraživanja informacija, model često rangira rezultate počevši od najrelevantnijih za upit do onih koji su najmanje relevantni. Ako znamo koji dokument je zaista relevantan za dani upit, uspješnost modela možemo mjeriti na temelju njegove pozicije u rangiranju. U ovom poglavlju obradit ćemo dvije metrike kojima mjerimo uspješnost rangiranja: *stopu pogodaka* i *srednji recipročni rang*. Obje metrike ćemo koristiti u evaluaciji naših modela u poglavlju 3.2.

U nastavku pretpostavljamo da je zadan skup upita $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$ i skup dokumenata $\mathcal{D} = \{D_1, \ldots, D_n\}$, gdje je D_i *relevantan* dokument za upit Q_i.

Stopa pogodaka

Stopa pogodaka za k predstavlja udio korisničkih upita koji imaju barem jedan relevantan dokument u prvih k rangiranih rezultata. Formalno, HR@k definira se kao:

$$\text{HR@}k = \frac{1}{|\mathcal{Q}|}\sum_{i=1}^{|\mathcal{Q}|}\text{relevant}_i,$$

gdje je $\text{relevant}_i = 1$ ako je za upit Q_i barem jedan od prvih k rangiranih dokumenata (onih iz skupa $\{D \mid D \in \mathcal{D}, \text{rank}(Q, D) \leq k\}$) ujedno i relevantan, a inače je $\text{relevant}_i = 0$. HR@1 je posebna inačica metrike HR@k i predstavlja udio upita kojima je prvi rangirani rezultat upravo i onaj relevantan. Slijedi primjer računanja metrika HR@1 i HR@3.

Primjer 1.3.1. *Ako imamo tri upita i relevantni rezultati su redom rangirani kao* 2.*,* 4.*, i* 1.*, tada je:*

$$\begin{aligned} HR@1 &= \frac{1}{3}(0+0+1) = \frac{1}{3}, \\ HR@3 &= \frac{1}{3}(1+0+1) = \frac{2}{3}, \end{aligned}$$

Srednji recipročni rang

Srednji recipročni rang je prosječni recipročni rang prvog relevantnog rezultata. Definiramo ga sljedećom formulom:

$$\text{MRR} = \frac{1}{|\mathcal{Q}|}\sum_{i=1}^{|\mathcal{Q}|}\frac{1}{\text{rank}_i},$$

gdje je $\text{rank}_i = \text{rank}(Q_i, D_i)$ rang prvog relevantnog odgovora za i-ti upit, dobiven formulom (1.1). Slijedi primjer s dokumentima i upitima nalik onima u stvarnosti u kojem računamo metrike MRR i HR@1.

Primjer 1.3.2. *Neka je zadana baza dokumenata* $\mathcal{D} = \{D_1, D_2, D_3\}$ *te upiti* $\mathcal{Q} = \{Q_1, Q_2, Q_3\}$*, gdje je dokument* D_i *relevantan za upit* Q_i *za* $i \in \{1, 2, 3\}$*. Upiti i dokumenti su sljedeći:*

$$\begin{aligned} Q_1 &= \text{„}\textit{Definirati ekonomiju}\text{”} \\ Q_2 &= \text{„}\textit{Što mijenja današnji svijet?}\text{”} \\ Q_3 &= \text{„}\textit{Što je inflacija?}\text{”} \end{aligned}$$

$D_1 =$ *„Ekonomija je znanost o upravljanju resursima."*
$D_2 =$ *„Tehnologija mijenja način na koji radimo i komuniciramo."*
$D_3 =$ *„Inflacija se odnosi na povećanje opće razine cijena."*

Neka su rezultati pretraživanja za upite zadani uređenim trojkama, gdje prvi element označava najbolje rangirani dokument, a zadnji element najlošije rangirani dokument. Za Q_1 imamo rezultate (D_1, D_3, D_2), za Q_2 imamo (D_3, D_2, D_1), a za Q_3 imamo (D_3, D_1, D_2). Tada je:

$$HR@1 = \frac{1}{3}(1 + 0 + 1) = \frac{2}{3} \approx 0.67$$
$$MRR = \frac{1}{3}\left(1 + \frac{1}{2} + 1\right) = \frac{5}{6} \approx 0.83$$

Ako promatramo samo prvi rezultat, možemo reći da je model u 67% dao točan odgovor. U usporedbi s HR@1, metrika MRR će uvijek imati veću vrijednost jer uzima u obzir i rezultate koji nisu najbolje rangirani.

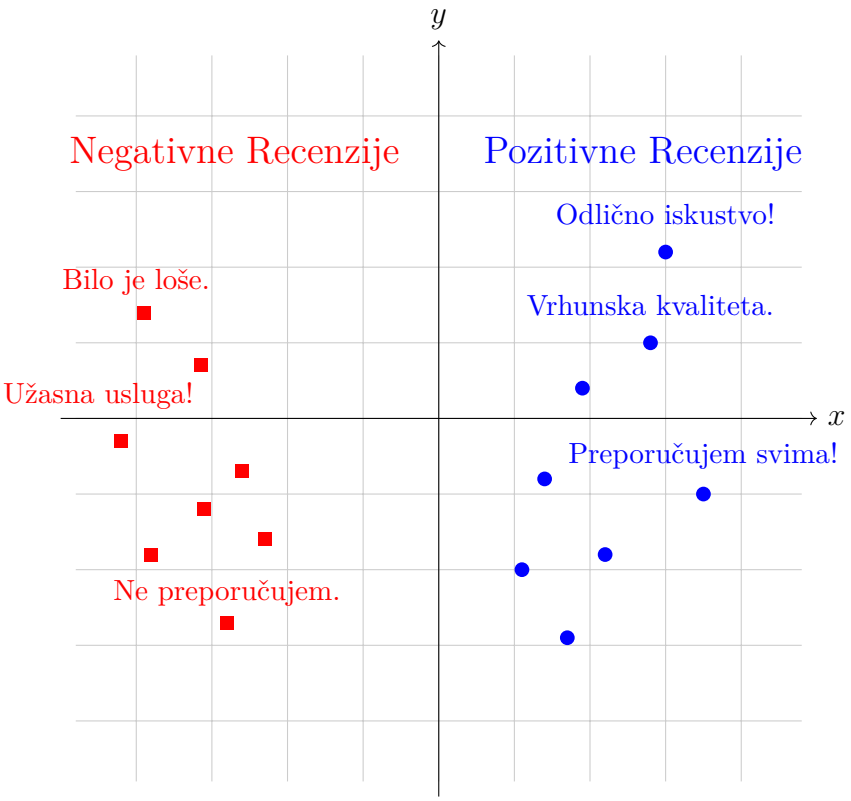
1.4 Primjena

U ovom poglavlju razmatrat ćemo različite domene primjene ugradbenih vektora:

- **Pretraživanje teksta:** Kao što je već navedeno, upit možemo pretvoriti u ugradbeni vektor, pa pretraživati dokumente po principu namanje euklidske ili kutne udaljenosti. Ovaj postupak detaljnije smo opisali u poglavlju 1.2. Pretraživanje teksta koristeći ugradbene vektore osnova je za vektorske baze poput baze Elasticsearch [19] ili baze Pinecone [24]. Vektorske baze koriste se u naprednim RAG sustavima o kojima će biti riječi u poglavlju 2.4.

- **Klasifikacija teksta:** Tekst možemo klasificirati u predodređene skupine koristeći ugradbene vektore [48]. Primjerice, tekstove recenzija možemo podijeliti na pozitivne i negativne recenzije kao što je prikazano na slici 1.2.

- **Sustavi predlaganja sadržaja:** Ugradbeni vektori koriste se i u sustavu predlaganja sadržaja poput algoritma za YouTube preporuke [7]. Ovdje se pretraživanje temelji na agregaciji ugradbenih vektora pregledanih videa i ugradbenih vektora značajki korisnika (poput dobi, lokacije i spola), a zatim se preporučuje novi sadržaj s obzirom na udaljenosti od ugradbenih vektora ostalih videa.



Slika 1.2: Primjer kako recenzije možemo pretvoriti u ugradbene vektore dvije dimenzije, a zatim klasificirati u pozitivne i negativne. Plavim točkama označene su pozitivne recenzije, a crvenim kvadratićima negativne.

- **Višemodalni latentni prostori:** Osim jednog tipa objekta, poput teksta ili slike, moguće je napraviti zajednički latentni prostor koji reprezentira i slike i tekst istovremeno [40]. Time se otvara širok spektar novih primjena. Primjerice, nove verzije ChatGPT-a mogu primati slike kao ulaz, analizirati njihov sadržaj i generirati precizne opise. S druge strane, modeli poput Stable Diffusion [43] pretvaraju tekstualni opis u ugradbeni vektor koji se zatim iterativno transformira u sliku koja prikazuje zadani tekst.

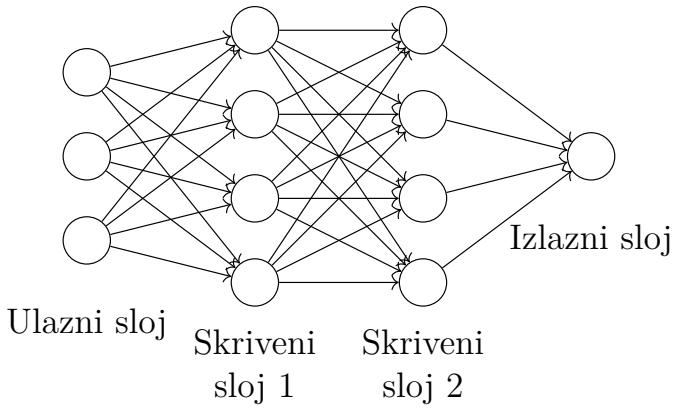
Poglavlje 2

Transformerski modeli

Duboko učenje [17] je tip strojnog učenja koji koristi višeslojne neuronske mreže za učenje iz podataka. Prva neuronska mreža [44] osmišljena je 1958. i korištena na zadatku prepoznavanja oblika na slici. Od tada je ovo područje znatno napredovalo. Neuronske mreže danas se koriste u analizi slika [20], generiranju slika [43], igranju strateških igri poput programa AlphaGo [46] i naravno u obradi prirodnog jezika [49].

Kad su se transformerski modeli [56] prvi puta pojavili 2017., donijeli su revoluciju u području obrade prirodnog jezika. Broj zadataka koji su se mogli uspješno riješiti računalom drastično se povećao. Primjeri takvih zadataka su gramatičko označavanje teksta (eng. *POS-tagging*) [10], analiza sentimenta [48], prevođenje [56] i predviđanje teksta [50]. Modele koji predviđaju tekst nazivamo *jezični modeli*. Porastom veličine modela, oni su postajali su sve boljih performansi [16]. Jezični model kojemu broj parametara brojimo u milijardama zovemo *veliki jezični model* (LLM, eng. *Large Language Model*). Tako nas je objavom aplikacije ChatGPT [37] krajem 2022. sve iznenadio ovaj brz napredak. Taj napredak ovdje nije stao — početkom 2025. tržište je potresla pojava modela DeepSeek [9] koji ima jednako dobre performanse, ali je bio mnogo jeftiniji za proizvesti. Ne zna se koja je granica u sposobnostima ovih modela, ali njihova primjena u stvarnom svijetu i aplikacijama je sve češća. Oba modela (ChatGPT i DeepSeek) su veliki jezični modeli i transformerske su arhitekture.

U ovom poglavlju opisujemo najbitnije elemente velikog jezičnog modela. Prvo navodimo razloge za korištenje dubokog učenja u analizi teksta. Zatim opisujemo tokenizaciju, odnosno postupak pretvaranja proizvoljnog teksta u tokene — jedinice s kojima model može baratati. U poglavlju 2.2 detaljno opisujemo arhitekturu transformera. Navodimo razlike jezičnih modela i ugradbenih modela baziranih na toj arhitekturi te opisujemo kako ih trenirati. Također opisujemo postupak finog podešavanja i RAG arhitekturu; oni omogućuju prilagodbu dostupnih velikih jezičnih modela za nove zadatke i primjene.



Slika 2.1: Vizualni prikaz višeslojne neuronske mreže.

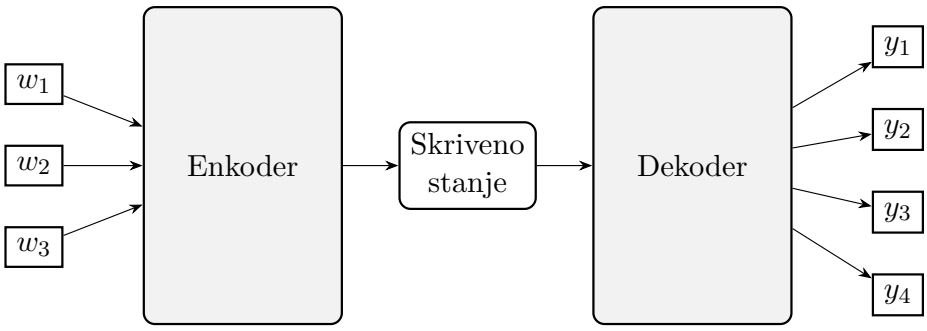
Prednost dubokog učenja u analizi teksta

Prednost dubokog učenja u analizi teksta u odnosu na klasične metode je u tome što može razumjeti kontekst i semantičke odnose unutar teksta [33].

Algoritmi poput BM25, iako učinkoviti u pretraživanju i rangiranju dokumenata, po svojoj prirodi ne uzimaju u obzir dvosmislenost riječi u kontekstu. Na primjer, ne mogu razlikovati značenja riječi *pas* (životinja) i *pas* (pojas) bez dodatnih informacija. Modeli bazirani na statistici imaju problema s rijetko korištenim riječima jer se oslanjaju na statističke značajke koje nisu dovoljno precizne za rijetke termine [17, 45].

Noviji modeli temelje se na višeslojnoj arhitekturi neuronskih mreža, prikazanih na slici 2.1. Svakim slojem ulazni podaci se obrađuju i transformiraju, omogućujući dublje razumijevanje teksta. Stoga modeli poput povratnih neuronskih mreža (eng. *Recurrent Neural Networks*) [49], a posebno arhitekture transformera, učinkovito rješavaju taj problem korištenjem mehanizma pažnje (eng. *attention*) [4]. Pomoću njega se modeli dinamički koncentriraju na relevantne dijelove ulaznog teksta i bolje razumiju kontekstualne veze između riječi. Mehanizam pažnje detaljno ćemo opisati u poglavlju 2.2.

Napomenimo da se transformeri originalno sastoje od dva dijela: enkodera i dekodera, kao na slici 2.2. Originalan transformer bio je korišten za prevođenje teksta. Enkoder je čitao tekst na originalnom jeziku, a dekoder je generirao tekst na novom jeziku. Uskoro su se pojavili modeli koji koriste samo jedan od ta dva dijela u svojoj arhitekturi pa ih dijelimo na enkoderske (BERT [10], BERTić [30]) i dekoderske (Gemma [50], Llama [53], Mistral [25]). Enkoderski modeli uglavnom su manji i koriste se za zadatke poput gramatičkog označavanja teksta, analize sentimenta te kao ugradbeni modeli. Dekoderski modeli predvode u zadatku generiranja teksta [6].



Slika 2.2: Prikaz arhitekture koja ima i enkoder i dekoder, poput originalnih transformera. U ovom radu opisujemo i koristimo samo dekoderski model. U tom slučaju nema skrivenog stanja, a ulazni podatci idu direktno u dekoder.

Cilj ovog rada, kao što je rečeno na početku, je izraditi ugradbeni dekoderski model baziran na velikom jezičnom modelu Gemma. Zbog toga, ali i zato što se dekoderski modeli danas više koriste, u poglavlju 2.2 ćemo opisati samo dekodersku arhitekturu.

Definicije

Arhitektura transformera temeljna je za izgradnju velikih jezičnih modela. S obzirom na to da je njihova glavna primjena predviđanje teksta, započinjemo s definicijom jezičnog modela.

Definicija 2.0.1. ***Jezični model*** *nad vokabularom $\mathcal{V}$ je funkcija:*

$$f : \mathcal{V}^* \to \bigcup_{n=1}^{\infty} [0,1]^{n \times |\mathcal{V}|},$$

gdje je $[0,1]^{n \times |\mathcal{V}|}$ skup matrica dimenzija $n \times |\mathcal{V}|$ čiji su elementi iz segmenta $[0,1]$ te vrijedi da za svaki konačni niz riječi x duljine n je vrijednost funkcije $f(x)$ matrica dimenzija $n \times |\mathcal{V}|$ kojoj je suma svakog retka jednaka 1.

Vrijednost funkcije $f(w_1, \ldots, w_n)_{ij}$ za $i \in \{1, \ldots, n\}$ te $j \in \{1, \ldots, |\mathcal{V}|\}$ interpretiramo kao vjerojatnost pojavljivanja riječi v_j nakon niza riječi $(w_1, \ldots, w_i)$. Vektor $f(w_1, \ldots, w_n)_n$ dimenzije $|\mathcal{V}|$ zovemo *vjerojatnosti sljedeće riječi* jer opisuje vjerojatnost pojavljivanja riječi u vokabularu nakon posljednje riječi niza $(w_1, \ldots, w_n)$. U poglavlju 2.3 ćemo opisati iterativni pohlepni pristup za generiranje teksta koji koristi vjerojatnost sljedeće riječi.

U ovom poglavlju opisujemo postupak izrade jezičnog modela temeljenog na arhitekturi transformera te način njegove prenamjene u ugradbeni model. Također razmatramo metode treniranja i finog podešavanja ovih modela.

2.1 Tokenizacija

Prvi korak u obradi teksta je tokenizacija, proces pretvaranja sirovog teksta u tokene, odnosno elemente vokabulara. Token je jedinica najčešće manja od jedne riječi; može biti samo slovo, slog, korijen riječi ili cijela riječ, a reprezentiran je prirodnim brojem kako bi model mogao s njime računati.

Sirovi tekst reprezentiramo znakovima. Označimo s $\mathcal{C}$ skup znakova, a s $\mathcal{C}^*$ skup svih konačnih nizova znakova. Matematički, *tokenizator nad znakovima $\mathcal{C}$ s vokabularom $\mathcal{V}$* je funkcija $tok : \mathcal{C}^* \to \mathcal{V}^*$.

Jedan od bitnih aspekata tokenizatora je određivanje vokabulara. On se dobiva obrađivanjem korpusa, tj. velikog skupa tekstova zapisanih znakovima. Razlikujemo tokenizatore bazirane na razmaku i one bazirane na znakovima. Tokenizatori bazirani na razmaku dobivaju vokabular iz najčešćih riječi u korpusu. Primjer problema koji se tada javlja je da su riječi u drugim padežima reprezentirane drugim tokenima, ali imaju isto značenje, primjerice *glava* i *glave*. Ovaj fenomen još je izraženiji u aglutativnim jezicima poput turskog, gdje se riječi nadograđuju različitim nastavcima, i po više odjednom. Bilo bi neefikasno svaki oblik spremati kao drugi token.

Stoga algoritmi bazirani na znakovima određuju vokabular na drugi način. Primjerice algoritam BPE (eng. *Byte Pair Encoding*) [15, 45] dobiva vokabular iterativnim postupkom. Početni vokabular jednak je skupu znakova koji koristimo. Drugim riječima, tokeniziramo čitav korpus u jednoznakovne riječi. U svakom idućem koraku, tražimo najčešću kombinaciju uzastopnih tokena te ih spajamo u novi token i dodajemo ga u vokabular. Zatim svaku pojavu te kombinacije zamjenjujemo novim tokenom. Ovaj postupak ponavljamo dok ne dođemo do određene duljine vokabulara, najčešće oko 30000 tokena. Algoritam BPE koriste mnogi veliki jezični modeli u izradi svog vokabulara [50, 25]. Napomenimo da različiti modeli i dalje imaju različit vokabular.

Kako bi modeli mogli računati s tokenima, ubuduće pretpostavljamo da je $\mathcal{V} = \{v_1, v_2, \ldots, v_{d_{vocab}}\}$ te $v_i = i$, za vokabular veličine d_{vocab}. Također, pojmovi *riječ* i *token* koristit će se kao sinonimi, a tokene i njihove tekstualne reprezentacije ćemo poistovjećivati.

Primjer 2.1.1. *Razmotrimo primjer kako se ulazni tekst tokenizira u niz brojeva.*

Ulazni tekst:

```
Osnova tokeniziranja je pretvorba teksta u tokene.
```

Tokenizirani tekst:

```
|Osnova| token|iziranja| je| pretvorba| teksta| u| token|e|.|
```

Dobiveni niz tokena:

```
(14321, 874, 912, 98, 19789, 5654, 45, 874, 32, 13)
```

2.2 Opis arhitekture

Dekoderski modeli sastoje se od tri dijela: ugradnje riječi (eng. *Input Embeddings*), dekoderskih blokova i glave transformera. Pregled arhitekture prikazan je na slici 2.3, a svaki dio će biti detaljnije opisan u nastavku ovog poglavlja. Pojašnjenja su utemeljena na originalnom radu [56].

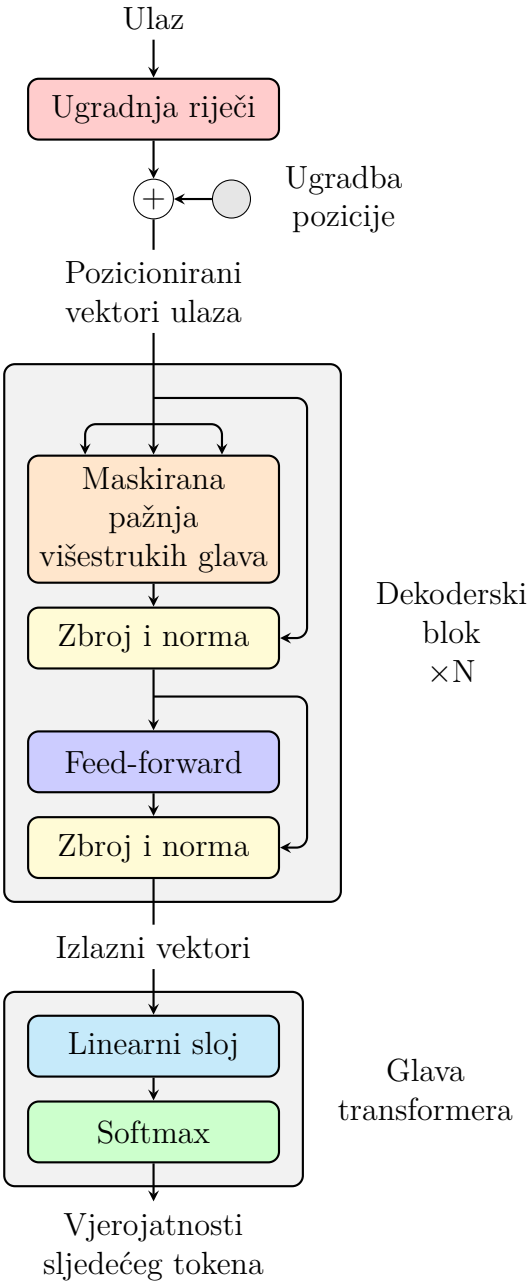
Glava transformera stvara konačne izlaze modela te ju možemo mijenjati ovisno o zadatku koji se rješava. Ostatak arhitekture (ugradnja riječi i dekoderski blokovi) ostaje nepromijenjen. U ovom poglavlju opisat ćemo dvije glave modela: glavu jezičnog modela, te glavu ugradbenog modela.

Ugradnja riječi

Prvi korak u arhitekturi transformera je pretvoriti tokene u početne ugradbene vektore. U njih zatim ugrađujemo poziciju u tekstu te ih dalje provlačimo kroz dekoderske blokove. U svakom dekoderskom bloku bit će međusobno povezani maskiranom glavom pažnje, a zatim dodatno pojedinačno obrađeni feed-forward slojem, koje ćemo kasnije objasniti.

Početni ugradbeni vektori su određeni za svaki token iz vokabulara $\mathcal{V}$, a dimenziju ugradbenog vektora jednog tokena označavamo s d_{model}. Najčešće koristimo vrijednosti $d_{model} \in \{756, 1024\}$ [58, 29]. Ovaj korak reprezentiramo funkcijom $\text{vec} : \mathcal{V} \to \mathbb{R}^{d_{model}}$. Ulaznu rečenicu $(w_1, \ldots, w_n)$ ugrađujemo po riječima, pa dobivamo niz vektora $(\text{vec}(w_1), \ldots, \text{vec}(w_n))$.

Iako rečenicu reprezentiramo uređenim nizom vektora, zbog načina obrade ovih vektora u kasnijim slojevima, potrebno je još ugraditi poziciju. U suprotnom, model ne bi znao kojim redoslijedom su složene riječi u rečenici. Poziciju ugrađujemo



Slika 2.3: Dekoderska arhitektura inspirana originalnim dijagramom iz rada [56]. Dekoderski blokovi ponavljaju se N puta, gdje se izlaz iz jednog bloka postavlja kao ulaz u sljedeći blok. Nakon posljednjeg dekoderskog bloka slijedi glava transformera koja izlazne vektore riječi pretvara u vektore vjerojatnosti za svaki token.

dodavanjem apsolutnih pozicijskih vektora. Apsolutni pozicijski vektori su definirani funkcijom $\text{pos} : \mathbb{N} \to \mathbb{R}^{d_{model}}$ na sljedeći način:

$$\text{pos}(k) = (\text{PE}(k, 0), \text{PE}(k, 1), \ldots, \text{PE}(k, d_{model} - 1)),$$

gdje je PE funkcija $\text{PE} : \mathbb{N} \times \{0, 1, 2, \ldots, d_{model} - 1\} \to \mathbb{R}$, definirana formulom:

$$\text{PE}(k, i) = \begin{cases} \sin\left(\frac{k-1}{10000^{i/d_{model}}}\right), & i \text{ je paran} \\ \cos\left(\frac{k-1}{10000^{(i-1)/d_{model}}}\right), & i \text{ je neparan} \end{cases}$$

Objedinjenjem ova dva koraka dobivamo pozicionirane ugradbene vektore. Za ulaznu rečenicu $(w_1, \ldots, w_n)$, nakon zbrajanja početnih ugradbenih vektora i pozicijskih vektora dobivamo niz vektora:

$$X = (\text{vec}(w_1) + \text{pos}(1), \ldots, \text{vec}(w_n) + \text{pos}(n))$$

Taj niz vektora predstavlja ulaz u prvi dekoderski blok i tretiramo ga kao matricu dimenzija $n \times d_{model}$. S X_i označavat ćemo i-ti redak u matrici te ih nazivamo ugradbeni vektori riječi.

Dekoderski blokovi slažu se jedan na drugi, svaki s drugim naučenim parametrima, o čemu ćemo govoriti kasnije. Oni se sastoje od maskirane pažnje višestrukih glava te feed-forward sloja. Za ulaz dimenzija $n \times d_{model}$ je izlaz oba ta sloja također dimenzija $n \times d_{model}$. Prilikom opisivanja daljnjih slojeva ćemo s X označavati ulaz u trenutni sloj, odnosno izlaz iz prethodnog sloja.

Slijedi primjer računanja pozicioniranih ugradbenih vektora.

Primjer 2.2.1. *Sljedeći primjer prikazuje ugradbu riječi i pozicije. Tokene prvo pretvaramo u početne ugradbene vektore koristeći funkciju vec ugradnje riječi. Račun je zaokružen na tri decimale.*

Ulaz:

$$(w_1, w_2, w_3, w_4) = (14321, 874, 912, 98)$$

Početni ugradbeni vektori:

$$\begin{aligned} vec(w_1) &= (0.000, \quad 1.000, \quad 0.000) \\ vec(w_2) &= (0.841, \quad 0.540, \quad 0.002) \\ vec(w_3) &= (0.909, \quad -0.416, \quad 0.004) \\ vec(w_4) &= (0.141, \quad -0.990, \quad 0.006) \end{aligned}$$

Za svaku poziciju zatim računamo pozicijske vektore. Napomenimo da oni ne ovise o ulaznom tekstu. Pozicijske vektore zatim zbrajamo s početnim ugradbenim vektorima i tako dobivamo pozicionirane ugradbene vektore.

Pozicijski vektori:

$$\begin{aligned} pos(1) &= (0.481, \quad 0.817, \quad 0.184) \\ pos(2) &= (-0.376, \quad -0.670, \quad 0.268) \\ pos(3) &= (0.837, \quad -0.696, \quad 0.545) \\ pos(4) &= (-0.381, \quad -1.000, \quad -0.798) \end{aligned}$$

Pozicionirani ugradbeni vektori:

$$\begin{aligned} X_1 &= vec(w_1) + pos(1) = (0.481, \quad 1.817, \quad 0.184) \\ X_2 &= vec(w_2) + pos(2) = (0.465, \quad -0.130, \quad 0.270) \\ X_3 &= vec(w_3) + pos(3) = (1.746, \quad -1.112, \quad 0.549) \\ X_4 &= vec(w_4) + pos(4) = (-0.240, \quad -1.990, \quad -0.792) \end{aligned}$$

Pozicionirani ugradbeni vektori (X_1, X_2, X_3, X_4) *predstavljaju ulaz u ostatak modela.*

Spomenimo još da poziciju možemo ugraditi i na druge načine, od kojih se u posljednje vrijeme koristi RoPE (Rotary Position Embedding) [47].

Maskirana pažnja višestrukih glava

Jedan dekoderski blok sastoji se od dva sloja: maskirane pažnje višestrukih glava (eng. *Masked Multi-Head Attention*) i feed-forward sloja te rezidualnih konekcija nakon oba. Uloga pažnje je da međusobno poveže vektore riječi, dok ih feed-forward sloj dodatno pojedinačno obradi. Rezidualne konekcije dodaju ulaz trenutnog sloja na njegov izlaz. Dekoderski blokovi zatim se ponavljaju više puta.

Maskirana pažnja višestrukih glava sadrži h glava pažnje tako da h dijeli d_{model}. Svaka glava pažnje računa različite odnose među riječima, primjerice promatrat će gramatičke odnose ili tražiti slične riječi. Objasnimo kako glava pažnje radi.

Za ulaz X dimenzije $n \times d_{model}$ iz prethodnog sloja, i-ta glava pažnje prvo računa tri matrice: upit Q_i, ključ K_i i vrijednost V_i, svaku dimenzija $n \times d_{head}$, pri čemu je $d_{head} = d_{model}/h$ dimenzija svake glave pažnje. One se računaju *linearnim slojevima*. Linearni sloj zadan je formulom $Y = XW$, gdje je matrica W naučen parametar, a X ulazna matrica. Za i-tu glavu pažnje računamo:

$$\begin{aligned} Q_i &= XW_i^Q, \\ K_i &= XW_i^K, \\ V_i &= XW_i^V, \end{aligned} \tag{2.1}$$

gdje su W_i^Q, W_i^K, W_i^V sve dimenzija $d_{model} \times d_{head}$. Dalje računamo izlaz glave pažnje.

$$\begin{aligned}
\text{AttentionHead}(Q, K, V) &= \text{Attention}(Q, K) \cdot V \\
\text{Attention}(Q, K) &= \text{Softmax}\left(\frac{QK^T + M}{\sqrt{d_{head}}}\right) \\
M_{i,j} &= \begin{cases} 0, & \text{ako je } i \geq j \\ -\infty, & \text{ako je } i < j \end{cases} \\
\text{Softmax}(X) &= (\text{softmax}(X_1), \ldots, \text{softmax}(X_n)),\ X \in \mathbb{R}^{n \times n} \\
\text{softmax}(x)_i &= \frac{e^{x_i}}{\sum_k e^{x_k}},\ x \in \mathbb{R}^n
\end{aligned}$$

Izlaz i-te glave pažnje je matrica $\text{AttentionHead}(Q_i, K_i, V_i)$ dimenzija $n \times d_{head}$. Sama pažnja je reprezentirana matricom $\text{Attention}(Q, K)$ dimenzija $n \times n$ te predstavlja koliko pažnje neka riječ daje drugoj riječi. Nazivnik $\sqrt{d_{head}}$ postoji radi numeričke stabilnosti [56].

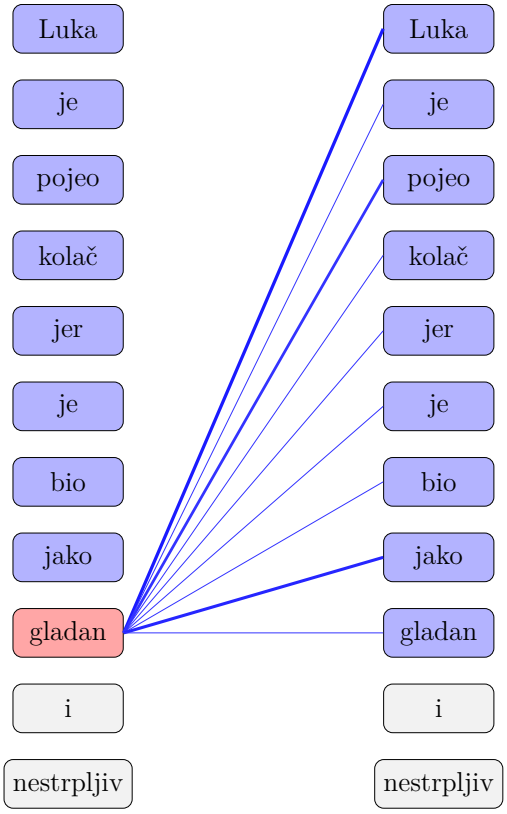
Kao što je već rečeno početkom poglavlja, jezični model predviđa vjerojatnosti sljedeće riječi nakon svake riječi. Da bi model stvarno naučio predviđati riječi, a ne samo „gledati" unaprijed, računanje ugradbenih vektora riječi ne smije ovisiti o budućim riječima. Zato prilikom računanja matrice $\text{Attention}(Q, K)$ dodajemo masku M pa tako dobivenu pažnju zovemo maskirana pažnja. Dodavanjem ove matrice, sve informacije o sljedećim riječima se poništavaju. Nakon primjene softmax funkcije, dobivena matrica ima 0 na pozicijama (i, j) gdje je $i \geq j$. Stoga se vrijednosti matrice V na j-toj poziciji prilikom računanja i-tog retka matrice izlaza glave neće uzimati u obzir.

Ovdje je dobar trenutak za napomenuti ključnu razliku enkoderskih i dekoderskih blokova. U enkoderskim blokovima prilikom računanja pažnje ne dodajemo matricu M. Enkoderski modeli rješavaju drugačije zadatke, poput analize sentimenta i gramatičkog označavanja te je stoga razumljivo omogućiti modelu da gleda cijelu rečenicu, a ne samo prethodne riječi.

Na slici 2.4 je prikazano kako zamišljamo pažnju. U sljedećem primjeru ju računamo.

Primjer 2.2.2. *Neka je matrica QK^T već određena. U sljedećem primjeru računamo vrijednost pažnje $\text{Attention}(Q, K)$ za glavu pažnje dimenzije $d_{head} = 3$. Matricu QK^T odmah uvrštavamo u formulu za pažnju.*

$$QK^T + M = \begin{bmatrix} 1.2 & 0.5 & 0.3 \\ 1.1 & 0.9 & 0.4 \\ 0.8 & 0.7 & 0.6 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -\infty \\ 0 & -\infty & -\infty \end{bmatrix} = \begin{bmatrix} 1.2 & 0.5 & 0.3 \\ 1.1 & 0.9 & -\infty \\ 0.8 & -\infty & -\infty \end{bmatrix}$$



Slika 2.4: Primjer vrijednosti pažnje za riječ „gladan”. Deblje linije označavaju veću pažnju. Najveća vrijednost pažnje je prema riječima „Luka”, „pojeo” i „jako”. Iako bi bilo gramatički ispravno da pažnja ide prema riječi „kolač”, iz konteksta je jasno da se „gladan” ne odnosi na „kolač” već na „Luka”. Sljedeće riječi su označene svjetlije jer je u dekoderskom modelu pažnja prema sljedećim riječima jednaka 0.

$$\frac{QK^T + M}{\sqrt{d_{head}}} \approx \begin{bmatrix} 0.69 & 0.29 & 0.17 \\ 0.64 & 0.52 & -\infty \\ 0.46 & -\infty & -\infty \end{bmatrix}$$

$$Attention(Q, K) = Softmax\left(\frac{QK^T + M}{\sqrt{d_{head}}}\right) \approx \begin{bmatrix} 0.44 & 0.30 & 0.26 \\ 0.53 & 0.47 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Budući da imamo h glava pažnje, nakon računanja izlaza svake glave pažnje, dobivamo h matrica $H_1, \ldots, H_h$, dimenzija $n \times d_{head}$. Njih zatim konkateniramo stupčano te tako dobivamo matricu H dimenzija $n \times d_{model}$ jer je $h \cdot d_{head} = d_{model}$. Maskirana pažnja višestrukih glava završava jednim linearnim slojem koji vraća matricu istih di-

menzija. Izlaz sloja se zbraja s prethodnim ulazom te se normalizira svaki redak, što zovemo rezidualnom konekcijom potrebnom za stabilnost treniranja mreže. Maskirana pažnja višestrukih glava i rezidualna konekcija matematički su opisani sljedećim formulama

$$\begin{aligned}
\text{AddNormMAH}(X) &= \text{Norm}(\text{MultiHeadAttention}(X) + X) \\
\text{MultiHeadAttention}(X) &= \left[H_1 \cdots H_h\right] W^O \\
H_i &= \text{AttentionHead}(Q_i, K_i, V_i),\ i \in \{1, \ldots, h\} \\
\text{Norm}(X) &= X \cdot \text{diag}(1/\|X_i\|_2,\ i = 1, \ldots, n),
\end{aligned}$$

pri čemu se Q_i, K_i i V_i računaju po formuli (2.1), a W^O je naučena matrica dimenzija $d_{model} \times d_{model}$

Feed-forward sloj

Slijedi feed-forward sloj. On se sastoji od dva potpuno povezana sloja te aktivacijske funkcije. Potpuno povezani sloj je linearni sloj sa zbrajanjem. Najčešće korištena aktivacijska funkcija je $\text{ReLU}(x) = max(x, 0)$ kojom se postiže nelinearnost modela, odnosno omogućuje se izražavanje složenijih odnosa u podatcima od onih dobivenih samo linearnim slojevima i jednostavnim matričnim množenjem. Nakon toga slijedi još jedna rezidualna konekcija u kojoj zbrajamo matricu prije primjene feed-forward sloja i onu poslije njega te dobivenu matricu normaliziramo po redcima. Ovaj korak matematički opisujemo formulama

$$\begin{aligned}
\text{AddNormFF}(X) &= \text{Norm}(\text{FeedForward}(X) + X) \\
\text{FeedForward}(X) &= (\text{ElemFF}(X_1), \ldots, \text{ElemFF}(X_n)) \\
\text{ElemFF}(x) &= \text{ReLU}(xW_1 + b_1)W_2 + b_2,
\end{aligned}$$

pri čemu se vrijednost funkcije ReLU za vektore računa za svaki element u vektoru zasebno. Matrice W_1 i W_2 su dimenzija $d_{model} \times d_{model}$, a vektori b_1 i b_2 duljine d_{model}.

Primjer 2.2.3. *Slijedi primjer računanja jednog potpuno povezanog sloja s primjenom aktivacijske funkcije ReLU. Neka su zadani vektori x i b te matrica W. Neka je potpuno povezani sloj određen formulom $xW + b$.*

$$x = (2, -1, 3),\ W = \begin{bmatrix} 1 & -2 & 0 \\ 3 & 0 & 1 \\ -1 & 4 & 2 \end{bmatrix},\ b = (1, 0, -2)$$

Računamo izlaz potpuno povezanog sloja s funkcijom ReLU.

$$\begin{aligned} ReLU(xW + b) &= ReLU((-4, 8, 5) + (1, 0, -2)) \\ &= ReLU((-3, 8, 3)) \\ &= (0, 8, 3) \end{aligned}$$

Dekoderski blok, kao što je već rečeno, sastoji se od maskirane pažnje višestrukih glava, feed-forward sloja te rezidualnih konekcija. Dekoderski blokovi ponavljaju se nekoliko puta, naravno s različitim parametrima. Pritom se izlaz iz prethodnog bloka postavlja kao ulaz u sljedeći. U originalnom radu [56] model sadrži 6 blokova, a model Gemma [50] ih sadrži 28.

Glava jezičnog modela

Poslije dekoderskih blokova, slijedi glava transformera; ona vektore dobivene iz posljednjeg dekoderskog bloka transformira u konačni izlaz modela. Odabir glave transformera ovisi o specifičnom zadatku. U ovom poglavlju usredotočit ćemo se na glavu jezičnog modela. Kasnije ćemo razmotriti i glavu ugradbenog modela, koja se koristi unutar ugradbenog modela.

Glava jezičnog modela je linearan sloj u kombinaciji s funkcijom softmax. Neka je $\mathcal{V} = \{v_1, v_2, \ldots, v_{d_{vocab}}\}$. Izlaz iz glave je matrica dimenzije $n \times d_{vocab}$. Linearni sloj opisan je naučenom matricom W^H dimenzija $d_{model} \times d_{vocab}$. Preciznije,

$$\text{LMHead}(X) = \text{Softmax}(XW^H)$$

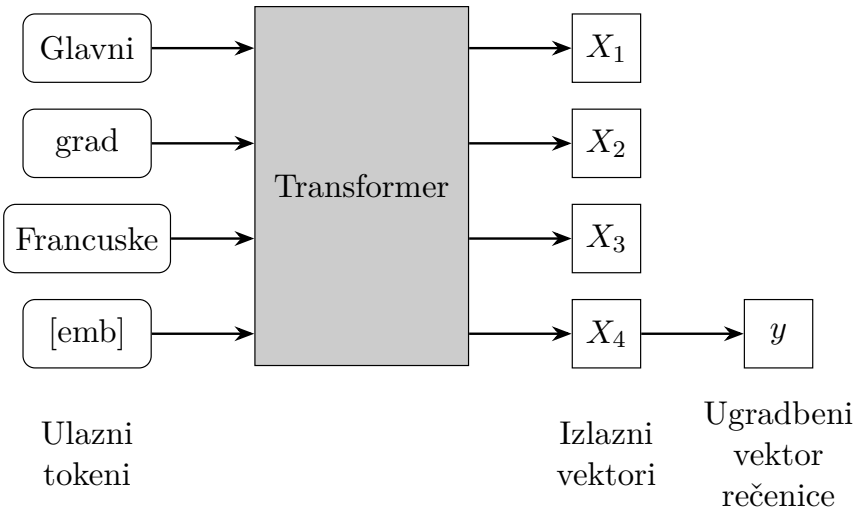
Tada je vrijednost funkcije LMHead(X) matrica dimenzija $n \times v_{d_{vocab}}$ koja zbog primjene funkcije softmax ima sume redaka jednake 1.

Primijetimo da smo time definirali jezični model. Neka je $(w_1, \ldots, w_n)$ ulazni niz riječi iz $\mathcal{V}$ u transformerski model, odnosno dosad opisane slojeve, te X izlaz iz posljednjeg dekoderskog bloka. Izlaz iz glave transformera LMHead(X) zadovoljava uvjete iz definicije 2.0.1.

Glava ugradbenog modela

Kao u prethodnom poglavlju, neka X označava izlazne vektore iz posljednjeg dekoderskog bloka. Osim predviđanja vjerojatnosti sljedeće riječi, transformerski modeli mogu se koristiti kao ugradbeni modeli. Dvije su česte opcije za glavu transformera u ovom slučaju.

Glava specijalnog tokena temelji se na tome da na kraj ulaza dodamo poseban token v_{emb} = „[emb]”. Stoga umjesto ulaznog niza riječi $(w_1, \ldots, w_n)$, u model



Slika 2.5: Prikaz glave ugradbenog modela koja koristi specijalan token. U ulaz dodajemo novi token „[emb]" na kraj rečenice. Ulazni tokeni transformiraju se u izlazne vektore nizom dekoderskih blokova. Posljednji vektor $y = X_4$ glava ugradbenog modela vraća kao ugradbeni vektor cijele rečenice.

ubacujemo niz $(w_1, \ldots, w_n, v_{emb})$. S obzirom na to da je sad specijalni token na kraju rečenice, njemu su omogućene veze pažnje prema svim prethodnim tokenima. Zbog toga bi ugradbeni vektor specijalnog tokena trebao sadržavati informacije o cijeloj rečenici. Glava jednostavno vraća taj posljednji ugradbeni vektor kao ugradbeni vektor rečenice. Neka je X matrica dimenzija $(n+1) \times d_{model}$ koja označava izlaz iz posljednjeg dekoderskog bloka.

$$\text{LastHead}(X) = X_{n+1}$$

Ovaj proces prikazan je na slici 2.5.

Glava prosjeka vraća prosjek svih ugradbenih vektora proizvedenih nakon posljednjeg dekoderskog bloka, ne bismo li dobili ugradbeni vektor koji reprezentira cijelu rečenicu. Neka je X matrica dimenzija $n \times d_{model}$ koja predstavlja izlaz iz posljednjeg dekoderskog bloka za početni ulazni niz riječi $(w_1, \ldots, w_n)$.

$$\text{AvgHead}(X) = \sum_{i=1}^{n} \frac{X_i}{n}$$

Obje ugradbene glave vraćaju vektor $y \in \mathbb{R}^{d_{model}}$. Uzmemo li d_{model} kao dimenziju latentnog prostora, u oba slučaja dobivamo ugradbeni model koji preslikava $\mathcal{V}^*$ u $\mathbb{R}^{d_{model}}$, sukladno definiciji 1.2.3.

2.3 Treniranje modela

U ovom poglavlju opisat ćemo kako se treniraju modeli, što je funkcija gubitka te koja se funkcija gubitka koristi za različite zadatke.

Definicija 2.3.1. ***Model strojnog učenja*** *je funkcija $f_\theta : \mathbf{D} \to \mathbf{K}$ definirana do na parametre θ. Pritom su $\theta \in \mathbf{P} = \mathbb{R}^p$ parametri modela dimenzije p, $\mathbf{D}$ skup mogućih ulaznih podataka u model, a $\mathbf{K}$ skup mogućih izlaznih podataka ili ciljeva.*

Parametri su u ovom slučaju sve korištene matrice W iz linearnih slojeva, vektori b iz potpuno povezanih slojeva te početni ugradbeni vektori reprezentirani funkcijom vec.

U kontekstu modela korištenih u ovom radu vrijedi $\mathbf{D} = \mathcal{V}^*$. U slučaju jezičnog modela definiranog u prethodnom poglavlju vrijedi $\mathbf{K} = \bigcup_{n=1}^{\infty} [0,1]^{n \times |\mathcal{V}|}$, gdje n predstavlja duljinu ulaza, a $[0,1]^{n \times |\mathcal{V}|}$ matricu dimenzija $n \times |\mathcal{V}|$ s elementima u rasponu $[0,1]$. U slučaju ugradbenog modela vrijedi $\mathbf{K} = \mathbb{R}^{d_{model}}$.

Postupak učenja podrazumijeva da postoji skup $\mathbf{T}$, koji nazivamo skup podataka za treniranje. U slučaju učenja s nadzorom [17] skup $\mathbf{T} \subseteq \mathbf{D} \times \mathbf{K}$ označava skup parova (x, y) gdje se za svaki ulaz x zna točan izlaz (labela) y.

Neka je je $\mathbf{F}$ skup modela strojnog učenja s parametrima iz skupa $\mathbf{P}$. Cilj treniranja modela je pronaći skup parametara $\theta^\star$ koji minimizira ukupni gubitak $L : \mathbf{F} \to \mathbb{R}$ za model strojnog učenja f:

$$\theta^\star = \underset{\theta \in \mathbf{P}}{\operatorname{argmin}} \, L(f_\theta)$$
$$L(f_\theta) = \frac{\sum_{x \in \mathbf{T}} \mathcal{L}(x, f_\theta)}{|\mathbf{T}|},$$

Funkciju $\mathcal{L} : \mathbf{T} \times \mathbf{F} \to \mathbb{R}$ zovemo funkcijom gubitka. Veći izlaz funkcije gubitka predstavlja veću grešku. To znači da će model $f_{\theta^\star}$ imati najmanju grešku na podatcima $\mathbf{T}$. Često se argument f_θ u funkciji gubitka $\mathcal{L}$ podrazumijeva, pa ga tada izostavljamo iz zapisa.

Ako su podatci raznoliki i funkcija gubitka dobro opisuje problem koji rješavamo, model će se moći primjenjivati na novim, neviđenim podatcima. Tu pojavu nazivamo *generalizacija* [55]. Dobar skup podataka za treniranje velikih jezičnih modela je jako velik. Pokazalo se da uspjeh modela logaritamski prati broj parametara modela i broj podataka za treniranje [26]. Za treniranje javno dostupnih LLama modela [53] korišteno je čak 15 bilijuna tokena.

Problem minimizacije funkcije gubitka aproksimiramo efikasnim algoritmom čestim u primjeni za rješavanje tog problema: algoritam gradijentnog spusta [36]. Ovdje ćemo navesti pseudokod, a detaljniji opis može se naći u literaturi [17].

Algoritam 1 Gradijentni spust

1: **Inicijalizacija:** Postavi početne vrijednosti parametara $\theta^{(0)}$ za model f.
2: **Za** $t = 0$ **do** $K - 1$ **radi**
3: Izračunaj gradijent: $\nabla_\theta L(f, \theta^{(t)})$
4: Ažuriraj parametre: $\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta L(f, \theta^{(t)})$
5: **Vrati:** Parametre θ^K

Broj K zovemo broj koraka treniranja, a parametar η stopom učenja (eng. *learning rate*). U primjeni se koriste razna poboljšanja ovog algoritma, od kojih je najpoznatiji stohastički gradijentni spust (eng. *Stochastic Gradient Descent*) [41, 17], koji učenje obavlja u *grupama* umjesto po cijelom skupu podataka za treniranje, te optimizator Adam (*Adaptive Moment Estimation*) [27] za isti algoritam.

Treniranje ugradbenog modela

U treniranju ugradbenog modela koristi se funkcija gubitka InfoNCE (Information Noise-Contrastive Estimation)[54]. Istu ćemo koristiti u treniranju našeg modela u poglavlju 3.2. Funkcija nastoji približiti vektorske reprezentacije dokumenata koji su semantički slični, a istovremeno udaljiti reprezentacije dokumenata koji nisu slični. Definiramo ju sljedećom formulom:

$$\mathcal{L}_{\text{InfoNCE}}(x, x^+, x_1^-, \ldots, x_k^-) = -\log \frac{e^{\text{sim}(f(x), f(x^+))/\tau}}{e^{\text{sim}(f(x), f(x^+))/\tau} + \sum_{i=1}^{k} e^{\text{sim}(f(x), f(x_i^-)/\tau)}}$$

Pritom je sim funkcija sličnosti vektora, a najčešće koristimo sličnost kosinusa. Parametar τ je hiperparametar temperature, a f ugradbeni model koji preslikava dokumente u latentni prostor. Funkcija gubitka ovisi o ulazu x, *pozitivnom primjeru* x^+ te nizu *negativnih primjera* $x_1^-, \ldots, x_k^-$. Cilj funkcije je maksimizirati sličnost ugradbenih vektora $f(x)$ i $f(x^+)$ te smanjiti sličnosti ugradbenog vektora $f(x)$ i vektora u nizu $f(x_1^-), \ldots, f(x_k^-)$. U treniranju ugradbenog modela za pretraživanje teksta, x predstavlja upit, vektor x^+ najrelevantniji dokument, a x_i^- nerelevantne dokumente za upit x. Iako su svi dokumenti koji nisu x^+ ustvari nerelevatni, uzimamo manji broj nasumično odabranih dokumenata kako računanje ne bismo provodili na cijeloj bazi dokumenata za treniranje.

Treniranje jezičnog modela

Ponovimo, jezični model računa vjerojatnost pojavljivanja sljedeće riječi nakon danog niza riječi. Neka je vokabular zadan s $\mathcal{V} = \{v_1, v_2, \ldots, v_{d_{vocab}}\}$, gdje je $v_i = i$.

Za treniranje jezičnog modela f kao funkciju gubitka koristimo *funkciju unakrsne entropije*:

$$\mathcal{L}_{\text{cross entropy}}(w_1, \ldots, w_n, w_{n+1}) = -\sum_{i=1}^{n} \log f(w_1, \ldots, w_n)_{i,w_{i+1}}, \tag{2.2}$$

pri čemu $w_i \in \mathcal{V}$. $f(w_1, \ldots, w_n)_{i,w_{i+1}}$ predstavlja izračunanu vjerojatnost pojavljivanja riječi w_{i+1} poslije niza riječi $(w_1, \ldots, w_i)$. Uočimo da je računanje funkcije gubitka ovdje vrlo efikasno jer vrijednost $f(w_1, \ldots, w_n)$ možemo izračunati samo jednom, a zatim po svakom sumandu samo pročitati komponentu $f(w_1, \ldots, w_n)_{i,w_{i+1}}$.

Predstavimo sada jedan jednostavan algoritam za generiranje teksta temeljen na jezičnom modelu f. Algoritam prima početni tokenizirani tekst kao ulaz te računa najvjerojatniji sljedeći token. Dobiveni najvjerojatniji token zatim dodajemo ulazu te ponavljamo ovaj postupak. To je *pohlepan algoritam* jer u svakom koraku bira samo trenutni najvjerojatniji token, bez razmatranja mogućnosti koje bi mogle nastati u kasnijim koracima.

Neka su zadani jezični model f, vokabular $\mathcal{V} = \{v_1, v_2, \ldots, v_{d_{vocab}}\}$ i poseban token $v_{kraj} \in \mathcal{V}$ koji predstavlja kraj teksta. Prisjetimo se da je vrijednost funkcije $f(w_1, w_2, \ldots, w_m)$ matrica dimenzija $m \times d_{vocab}$ te njezin posljednji redak predstavlja vjerojatnosti sljedeće riječi.

Algoritam 2 Pohlepno generiranje teksta

1: **Ulaz**: početni tekst $(w_1, w_2, \ldots, w_n)$
2: **Inicijalizacija**: $m \longleftarrow n$
3: **Dok** $w_m \neq v_{kraj}$ **radi**
4: Izračunaj vjerojatnosti sljedeće riječi: $y \longleftarrow f(w_1, w_2, \ldots, w_m)_m$
5: Odredi sljedeći token na pohlepan način: $j = \text{argmax}_{i \in \{1,\ldots,d_{vocab}\}} y_i$
6: Ažuriraj: $m \longleftarrow m + 1$
7: Postavi: $w_m \longleftarrow v_j$
8: **Vrati:** generirani tekst $(w_1, w_2, \ldots, w_m)$

Osim ovog načina generiranja teksta, često se koriste i unaprijeđeni algoritmi poput *top-k sampling* [21] ili *top-p sampling* [21] koji s određenom vjerojatnošću ipak odabiru neki manje vjerojatan token. Također se koristi i algoritam *tree search* [49] koji gleda u širinu istovremeno generirajući nekoliko tekstova i tek na kraju vraća onaj najvjerojatniji. Ovakvi algoritmi bazirani na vjerojatnostima generiraju tekst koji nam se često čini prirodnijim.

2.4 Prilagodba modela

Modeli su namijenjeni i istrenirani za rješavanje jednog zadatka. Primjerice jezični model može biti istreniran da govori na engleskom, njemačkom ili hrvatskom. Osim u jeziku, zadatak se može razlikovati u domeni, pa jezični model može biti namijenjen da odgovara dobro na pravne tekstove, programerska pitanja ili na pitanja općeg znanja. Prilikom rješavanja novog zadatka, moguće je iskoristiti i prilagoditi već postojeći model, pogotovo ako su zadatci slični. Model koji je već treniran i namijenjen daljnjem prilagođavanju nazivamo *predtreniran model*.

Za prilagodbu modela nudi se više tehnika poput treniranja, finog podešavanja i RAG arhitekture (eng. *Retrieval-Augmented Generation*) o kojima će biti riječ u ovom poglavlju. Svaka metoda ima svoje prednosti i mane, a odabir ovisi o namjeni modela i raspoloživim resursima.

- **Treniranje novog modela** podrazumijeva treniranje modela od početka, od nasumično odabranih početnih parametara. Ovaj pristup je najskuplji u smislu vremena i računarskih resursa [37, 26] te je treniranje velikih jezičnih modela praktički nemoguće na kućnom hardveru.

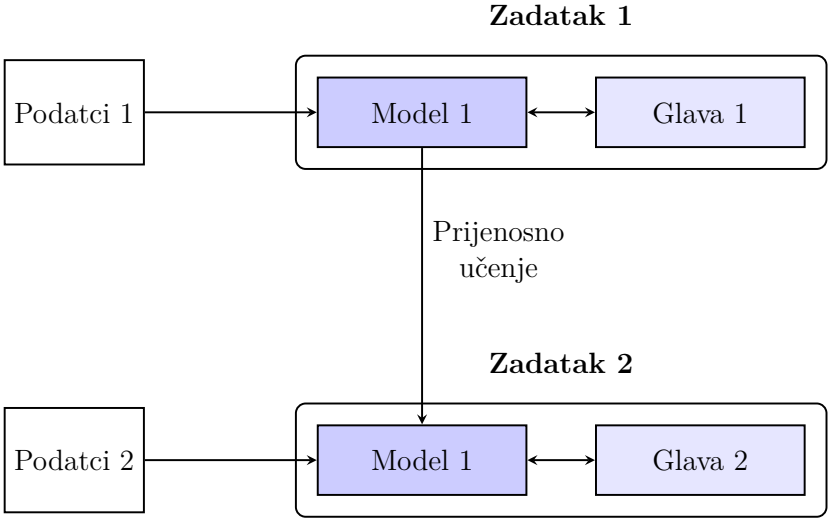
- **Fino podešavanje** uključuje predtreniran model kao osnovu umjesto nasumičnih parametara. Ponekad se ne podešavaju niti svi parametri, već nekoliko dodatnih slojeva specifičnih za novi zadatak [23, 22]. Ova metoda zahtijeva manje podataka i vremena.

- **RAG arhitektura** unaprjeđuje jezične modele tako da im omogućuje korištenje novog znanja iz tekstova spremljenih u bazi dokumenata. U ovom se slučaju jezični model uopće ne prilagođava, već je dovoljno samo dodati nove podatke u bazu te odabrati prikladni način pretraživanja tih podataka.

U ovom poglavlju detaljnije ćemo opisati metodu finog podešavanja LoRA te zatim RAG arhitekturu. RAG arhitektura često se koristi u kombinaciji s ugradbenim modelom, pa je ovo dobra prilika za pokazati korisnu primjenu takvih modela.

Fino podešavanje

Fino podešavanje tip je *prijenosnog učenja*, metode u kojoj se znanje iz prethodnog zadatka koristi u učenju novog zadatka, ne bi li proces bio brži i lakši. Prilikom finog podešavanja, uzimamo već postojeći model te ga treniramo na novim podatcima, kao što je prikazano na slici 2.6. Navodimo neke primjere gdje je fino podešavanje primjenjivo:



Slika 2.6: Grafički primjer prijenosnog učenja. Prvo treniramo model s jednim podatcima. Zatim dobivenom modelu mijenjamo glavu i treniramo ga na novim podatcima.

- Model koji već dobro prepoznaje aute na slici može se dotrenirati da prepoznaje kamione.
- Jezični model koji zna odgovoriti korisniku na nekom slavenskom jeziku, može se lakše prilagoditi da razgovara na hrvatskom nego model koji ne govori niti jedan slavenski jezik.

Fino podesiti možemo cijeli model ili samo dio modela. U drugom slučaju, većinu parametara *zamrznemo*, odnosno ne mijenjamo ih. Postoje i napredne metode finog podešavanja koji uključuju dodatke u samoj arhitekturi kako bi se model lakše trenirao i prilagodio novom zadatku. Jedna od tih metoda je LoRA [23], koju ćemo koristiti u poglavlju 3.2 prilikom finog podešavanja modela Gemma. Slijedi opis te metode.

LoRA

Umjesto da mijenjamo sve parametre modela, LoRA uvodi dodatne matrice niskog ranga [3] koje se treniraju, dok ostali parametri modela ostaju zamrznuti. Broj parametara koji se treniraju smanjuje se za više od 50% u odnosu na cijeli model, dok performanse modela ostaju iste.

U arhitekturi transformera, kao što je već rečeno, ključna komponenta je pažnja; ona koristi linearne transformacije ulaznih podataka kroz matrice upita Q, ključa K

i vrijednosti V. Prisjetimo se, originalne matrice Q, K, V u arhitekturi transformera definirane su formulama

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V.$$

LoRA mijenja matricu $W \in \{W^Q, W^K, W^V\}$ na sljedeći način:

$$W = W_{(0)} + \Delta W, \quad \Delta W = AB,$$

gdje je matrica $W_{(0)}$ originalna, zamrznuta matrica, $A \in \mathbb{R}^{d_{model} \times r}$, $B \in \mathbb{R}^{r \times d_{head}}$ ($r \ll d_{model}, d_{head}$) su pomoćne matrice niskog ranga koje se treniraju na isti način na koji smo prethodno trenirali model. Prednost treniranja njih u odnosu na cijelu početnu matricu W je upravo niski rang, što rezultira manjim brojem parametara. Istovremeno, pokazalo se da ΔW može reprezentirati sve potrebne promjene u prilagodbi modela na novi zadatak.

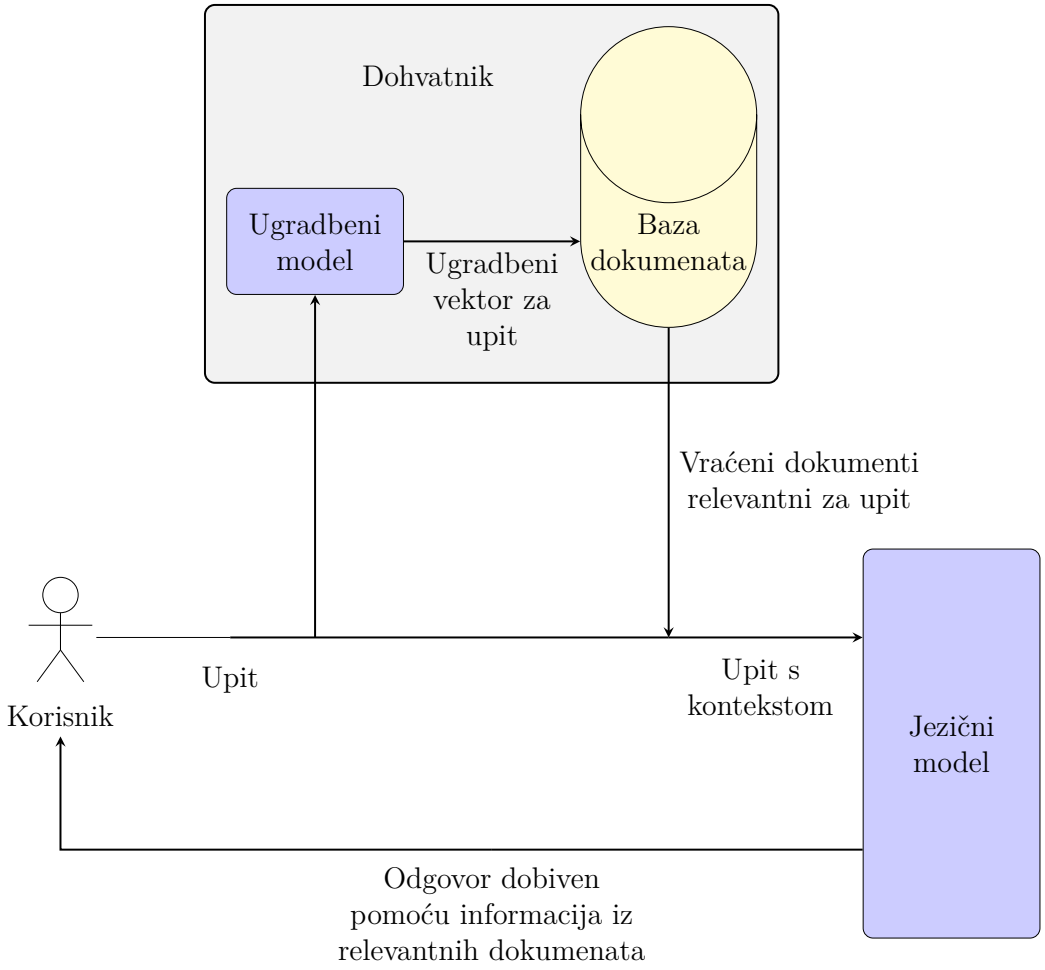
Osim prilagođavanja matrica W^Q, W^K i W^V, ova metoda prikladna je za podešavanje bilo koje matrice W korištene u modelu, primjerice matrice W^O, korištene na kraju maskirane pažnje višestrukih glava te matrica korištenih u feed-forward sloju.

Ova metoda prigodna je za fino podešavanje velikih jezičnih modela na svima lako dostupnom hardveru poput grafičke kartice s 16 GB VRAM-a (dostupne besplatno na stranici Kaggle [18] u trenutku pisanja). Iako se neki veliki jezični modeli poput Gemme mogu učitati u takve grafičke kartice, ne mogu se potpuno trenirati na njima. Prilikom treniranja, za svaki parametar koji se trenira trebaju se spremati dodatni podaci u memoriju, što bi premašilo 16 GB memorije. Korištenjem metode LoRA, ne treniraju se svi parametri pa takvo treniranje postaje moguće. Iz tog razloga ćemo koristiti ovu metodu prilikom treniranja ugradbenog modela za hrvatski.

Opis RAG arhitekture

RAG arhitektura [28] predstavlja nov pristup u načinu korištenja velikih jezičnih modela. Za razliku od klasičnih jezičnih modela koji koriste jedino parametarsko znanje, odnosno znanje stečeno kroz treniranje modela, RAG sustavi dinamički dohvaćaju relevantne dokumente iz dodatne baze podataka te ih koriste prilikom odgovaranja kako bi dali precizniji odgovor.

Sustav RAG slikovito možemo opisati na sljedeći način. Jezične modele zamišljamo kao stručnjake kojima postavljamo pitanje te nam odgovaraju napamet iz glave. Sustave RAG s druge strane možemo zamisliti kao stručnjake s pristupom knjižnici. Nakon što im postavimo pitanje, prvo će pronaći relevantnu literaturu. Zatim će pročitati literaturu te nam ponuditi odgovor na osnovu toga. Osim što je taj odgovor često precizniji, potkrijepljen je i literaturom. Također, vrlo jednostavno možemo dodati novo znanje dodavanjem novih knjiga u knjižnicu.



Slika 2.7: Grafički prikaz RAG arhitekture. Korisnik postavlja upit. Dohvatnik zatim koristeći ugradbeni model pretražuje bazu dokumenata te traži dokumente koji su relevantni za korisnikov upit. Dobiveni dokumenti se zatim stavljaju na početak upita kao kontekst. Upit s kontekstom se ubacuje u veliki jezični model koji sada može točnije odgovoriti na upit.

RAG arhitektura sastoji se od dva dijela: dohvatnika i jezičnog modela. Može se grafički prikazati slikom 2.7.

Dohvatnik je prvi sloj u RAG arhitekturi i služi za dohvaćanje relevantnih dokumenata iz baze dokumenata na temelju postavljenog upita. Dohvatnik može biti:

- *rijedak* - koristi klasične algoritme pretraživanja poput BM25,
- *gust* - koristi ugradbeni model te vektorsku bazu. Prvo pretvara upit u vektor te zatim u vektorskoj bazi pretražuje relevantne dokumente na temelju udaljenosti vektora, što je opisano u poglavlju 1.2, ili
- *hibridan* - kombinacija prethodna dva pristupa.

Jezični model je obično veliki jezični model koji koristi dohvaćene dokumente s korisničkim upitom kao ulaz te generira konačni odgovor.

Prilikom korištenja RAG arhitekture, dohvatnik prvo pronalazi relevantan dokument (ili više njih) za korisnički upit. Taj dokument stavlja se prije upita i čitav spoj šalje se jezičnom modelu. Takav spoj zovemo *upit s ugrađenim kontekstom*. Sljedeći primjer prikazuje jedan takav upit.

Primjer 2.4.1. *Razlika tipičnog upita i upita s ugrađenim kontekstom. Kontekst je neki dokument dohvaćenih dohvatnikom iz baze dokumenata.*

Tipični upit:

```
Upit: Koji su glavni faktori koji utječu na ekonomsku inflaciju?
Odgovor:
```

Upit s ugrađenim kontekstom:

```
Kontekst: Inflacija je često uzrokovana povećanjem ponude novca,
višim troškovima proizvodnje i povećanom potražnjom.
Upit: Koji su glavni faktori koji utječu na ekonomsku inflaciju?
Odgovor:
```

Rezultat ovog procesa često je precizniji odgovor. Osim toga, sustav može vratiti korisniku odakle su povučeni rezultati. RAG sustavi redovito se koriste u praksi, a primjer korištenja takvog modela prikazan je na slici 2.8.

U sljedećem poglavlju prenamjenjujemo jezični model Gemma u ugradbeni model koristeći upravo metodu LoRA. Poslije toga sastavljamo vlastiti sustav RAG te pokazujemo rezultate.

Primjer 1: Upit i odgovor GPT-3.5 Turbo

Upit:
Ukratko samo nabroji modele koji koriste RAG, te one koji ne koriste, navedi web stranice koje to dokazuju.

Odgovor:
Neki od modela koji koriste Retrieve and Generate (RAG) pristup su:

- RAG (Originalni RAG model)
- DPR (Dense Passage Retrieval) + RAG
- REALM (Retrieval-Augmented Language Model)

Modeli koji ne koriste RAG pristup uključuju većinu standardnih jezičnih modela poput BERT-a, GPT serije, itd.
Za dokaze o korištenju RAG modela možete posjetiti službene web stranice organizacija poput Facebook AI Research (FAIR) za RAG i DPR modele ili Google AI za REALM model. Osim toga, znanstveni članci, blogovi i prezentacije također mogu pružiti detaljnije informacije o primjeni ovih modela.

Primjer 2: Upit i odgovor Microsoft Copilota

Upit:
Ukratko samo nabroji modele koji koriste RAG, te one koji ne koriste, navedi web stranice koje to dokazuju.

Odgovor:
Naravno!
Modeli koji koriste RAG:

- Amazon Nova
- ChatGPT s Retrieval Pluginom
- Azure Machine Learning
- Anthropic's Constitutional AI

Modeli koji ne koriste RAG:

- Standard GPT-3
- BERT
- T5

Nadam se da ovo pomaže!

press.aboutamazon | techcommunity.microsoft.com | expertbeacon.com | developer.ibm.com

Slika 2.8: Odgovor modela bez i s ugrađenim sustavom RAG. S lijeve strane naveden je odgovor modela GPT-3.5 Turbo dostupnog na `poe.com`. GPT-3.5 Turbo ne pretražuje internet te korisnika upućuje samo na generička mjesta. S desne strane je primjer odgovora Microsoft Copilota [34]. Prije nego napiše odgovor, pretražuje internet kako bi podržao svoje tvrdnje. Korištene web stranice navodi u obliku gumba, kako bi ih korisnik mogao provjeriti.

Poglavlje 3

Fino podešavanje ugradbenog modela na hrvatskom jeziku

U ovom poglavlju bit će riječi o eksperimentalnom dijelu diplomskog rada. Kao što smo najavili u uvodu, cilj rada je fino podesiti veliki jezični model da generira ugradbene vektore za tekstove na hrvatskom jeziku. Iako se za ugradbene modele uglavnom koriste enkoderski modeli koji ujedno imaju i manji broj parametara, nedavna istraživanja [31] pokazala su da se veliki jezični modeli, koji su ujedno dekoderski modeli, mogu prenamijeniti za istu svrhu. Razlozi su sljedeći:

1. Veliki jezični model dobro reprezentira riječi iza posljednjeg dekoderskog bloka svoje arhitekture. Naime u jezičnom modelu je iz posljednjih vektorskih reprezentacija riječi potrebno predvidjeti sljedeću riječ.

2. Ako uspoređujemo veličinu modela potrebnog za kvalitetno generiranje teksta [6, 26] i ugradbenog modela namijenjenog za pretraživanje teksta [58, 29], možemo reći da je generiranje ugradbenih vektora jednostavniji zadatak jer su takvi modeli puno manji. Jezični modeli ipak moraju dublje razumijevati tekst te bi stoga trebali imati reprezentativnije ugradbene vektore u posljednjem dekoderskom bloku. Ako je tome zaista tako, i prilagodba modela kakvu opisujemo bi trebala biti jednostavna.

Eksperimenti poput ovoga mogu dati bolji uvid u način na koji funkcioniraju veliki jezični modeli u smislu reprezentiranja riječi ugradbenim vektorima te objasniti kako se mogu prilagoditi na nove jezike poput hrvatskog.

U prvom dijelu ovog poglavlja opisat ćemo proces prikupljanja podataka za fino podešavanje modela. Na tom skupu podataka smo trenirali naš model koristeći algoritam stohastičkog gradijentnog spusta s InfoNCE gubitkom te tehnikom LoRA. Eksperimentirali smo s različitim postavkama te smo napravili ukupno tri ugradbena

modela koji su namijenjeni za pretraživanje teksta pomoću ugradbenih vektora. Cijeli postupak treniranja opisat ćemo u poglavlju 3.2. Rezultate modela usporedit ćemo s klasičnom metodom dohvaćanja dokumenata BM25 i ugradbenim enkoderskim modelom BERT. Na kraju ćemo pokazati kako se naš ugradbeni model može koristiti kao dohvatnik u sustavu RAG.

U našem skupu podataka, koji opisujemo u poglavlju 3.1, postoji samo jedan relevantan dokument za svaki upit. Zato ćemo pri evaluaciji modela koristiti metrike HR@1 i MRR definirane u poglavlju 1.3

3.1 Skup podataka

Skup podataka određen je saznanjima iz prijašnjih radova [2] koji obrađuju fino podešavanje velikog jezičnog modela LLama na grčki. Za cijelo fino podešavanje koristili su 80000 odlomaka. Međutim, analiza grafa funkcije gubitka, koji prikazuje gubitak u trenutnom koraku odnosno grupi, sugerira da je model bio dovoljno istreniran već u ranijim fazama. Budući da mi radimo na jednostavnijem zadatku, kao što je napomenuto u prethodnom poglavlju, odlučili smo koristiti skup podataka od oko 20000 odlomaka.

Naš skup podataka za treniranje sastoji se od dva dijela. Jedan je prevedeni javno dostupni skup podataka MS-Marco [5], a drugi je dohvaćen iz hrvatske Wikipedije. Skup podataka koji ćemo koristiti treba imati dva stupca: upit (eng. *query*) i odgovor (eng. *passage*). Model ćemo trenirati funkcijom gubitka InfoNCE, što osigurava da ugradbeni vektori upita budu blizu po kutnoj udaljenosti ugradbenom vektoru pravog odgovora, a daleko od ugradbenih vektora odgovora na druga pitanja (nerelevantnih odgovora).

Hrvatska Wikipedija

Prvi dio skupa podataka dohvatili smo iz hrvatske Wikipedije. Preuzeli smo 10000 nasumično odabranih članaka na hrvatskom. Iz članaka smo izdvojili naslov te prvi odlomak. Upite smo sastavljali iz naslova članaka na sljedeći način:

- S 50% vjerojatnosti se sam naslov uzima kao upit.
- Inače, uzimamo nasumičnu rečenicu iz skupa te stavljamo naslov članka na mjesto *naslov*: „Što se može reći o *naslov*”?, „Kako bih saznao više o *naslov*?”, „Sažetak teme *naslov*.”, „Zašto je *naslov* važan ili zanimljiv?”, „Ključne informacije o *naslov* na jednom mjestu.”, „Objašnjenje: *naslov*.”, „Sve što treba znati o *naslov*.”, „Kratak uvod u *naslov*.”, „Kako bih mogao razumjeti *naslov*?”, „Primjer teme *naslov* i njezina važnost.”

Upit i odgovor dobiven iz hrvatske Wikipedije

Originalni naslov članka: Mars Exploration Rovers
Dobiveni upit: Kako bih mogao razumjeti Mars Exploration Rovers?
Odgovor: NASA-ina misija Mars Exploration Rover (MER) bila je robotska svemirska misija koja je uključivala dva Marsova rovera - Spirit i Opportunity - s ciljem istraživanja planeta Marsa. Započela je 2003. lansiranjem dvaju rovera: MER-A Spirit i MER-B Opportunity...

Slika 3.1: Primjer naslova, upita dobivenog našim postupkom i odgovora.

Prvi slučaj posebno je izdvojen jer upiti korisnika često nisu pune rečenice, već sadrže samo ključne riječi pretraživanja. U drugom slučaju pokušavamo postići raznolikost korisničkih upita. Kao odgovor na taj upit uzimamo prvi odlomak odgovarajućeg članka. Prvi odlomak u Wikipediji gotovo je uvijek sastavljen tako da sadrži sažetak i sve bitne informacije o temi. Primjer opisanog postupka prikazan je na slici 3.1.

Prevedeni MS-Marco

Originalni skup podataka MS-Marco sastoji se od oko 100000 parova upita i odgovora. Upiti su duljine jedne rečenice, a odgovori duljine jednog ili više odlomaka. Koristeći biblioteku googletrans preveli smo 10000 pitanja iz tog skupa na hrvatski jezik te tako dobili drugi dio podataka za treniranje.

Primjer pitanja i prijevoda prikazan je na slici 3.2. Tekst nije formatiran, već prikazan onakav kakav koristimo u treniranju. Prijevod nije najbolje kvalitete i neke riječi nedostaju, no tekst i dalje sadrži dovoljno informacija za pretraživanje. Model ionako ne treniramo da generira tekst, već samo informativnu i kompaktnu reprezentaciju teksta. Za tu svrhu je dovoljno sažeti značenje ključnih riječi.

Oba skupa (prevedeni MS-Marco i odlomke iz hrvatske Wikipedije) spajamo u jedan skup podataka te filtriramo neispravne dokumente. Radi se o onim dokumentima koji su se pojavili prilikom greške u prevođenju skupa podataka MS-Marco. Sveukupno, dohvatili smo 19814 dokumenata. Skup podataka smo podijelili u tri dijela: validacijski, testni te skup podataka za treniranje. Skup podataka za treniranje sadrži ukupno 15844 primjeraka i koristi se za treniranje kako je opisano u poglavlju 2.3. Validacijski skup sadrži 1984 primjeraka i namijenjen je za vrednovanje modela tijekom treniranja. Testni skup sadrži 1986 primjeraka te se na njemu model vrednuje pomoću metrika opisanih u prethodnom poglavlju. Kako bismo preciznije usporedili rezultate modela tijekom i nakon testiranja, evaluaciju smo proveli isključivo na testnom skupu, dok validacijski skup nije bio uključen u postupak.

Originalno MS-Marco pitanje

Upit: Why do children get aggressive?
Odgovor: At the same time, despite claiming the review demonstrates a link between playing violent video games and aggression, the authors acknowledged that some studies were inconsistent and that present research is insufficient to establish whether this can lead to criminal violence or delinquency.Kids who are bipolar, in their manic stages, very frequently become aggressive. They lose self-control, they become impulsive. On the other end of the spectrum, when they become depressed, although aggression is less common, they can become irritable, and sometimes that irritability and cantankerousness causes kids to lash out.

Prevedeno pitanje

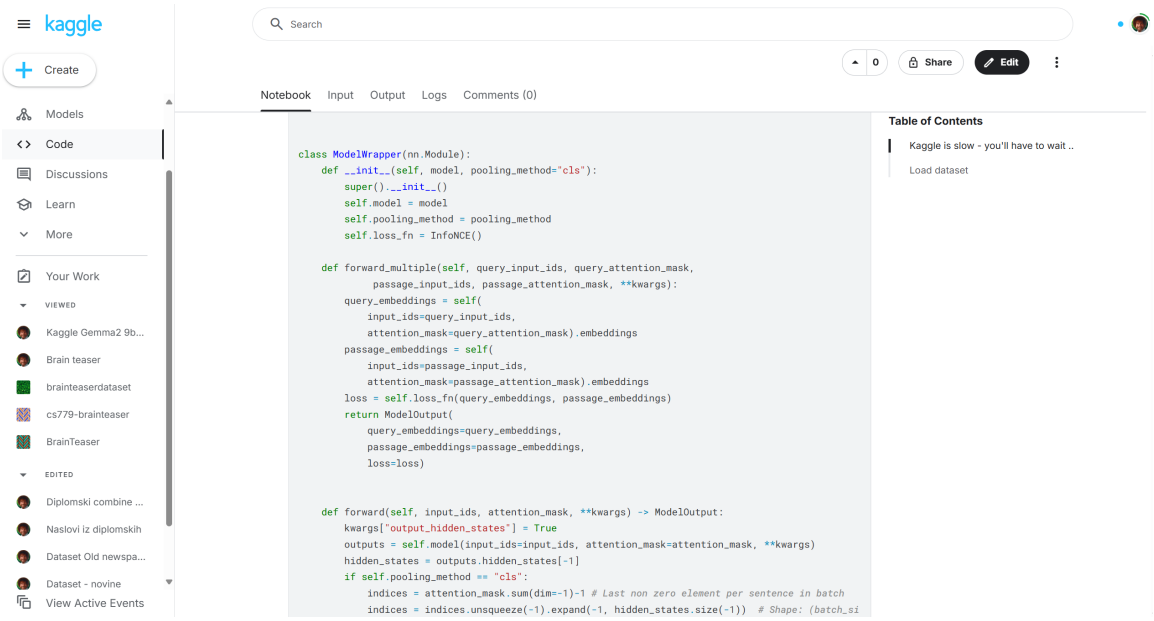
Upit: Zašto djeca postaju agresivna?
Odgovor: U isto vrijeme, unatoč tome što je tvrdio da pregled pokazuje vezu između igranja nasilnih videoigara i agresije, autori su priznali da su neke studije nedosljedne i da je sadašnje istraživanje nedovoljno da bi se utvrdilo može li to dovesti do kriminalnog nasilja ili prijestupništva., u njihovim manijačnim fazama vrlo često postaju agresivni.Gube samokontrolu, postaju impulzivni.Na drugom kraju spektra, kada postanu depresivni, iako je agresija rjeđa, mogu postati razdražljivi, a ponekad i razdražljivost i kantunkera uzrokuju djecu.

Slika 3.2: Primjer originalnog pitanja iz skupa podataka MS-Marco te pitanja prevedenog koristeći googletrans bibilioteku.

3.2 Fino podešavanje modela Gemma

Programsko okruženje

Za provedbu eksperimenata korišten je programski jezik Python, s bibliotekama Pytorch [14] za treniranje modela, transformers od tvrtke HuggingFace [13] te bibliotekom unsloth [8] za brže treniranje modela i primjenu tehnike LoRA. Preslika radne površine stranice Kaggle [18] vidljiva je na slici 3.3. Sve Kaggle bilježnice korištene za sakupljanje podataka, treniranje modela te evaluaciju dostupne su na adresi `https://github.com/rangoiv/diplomski`.



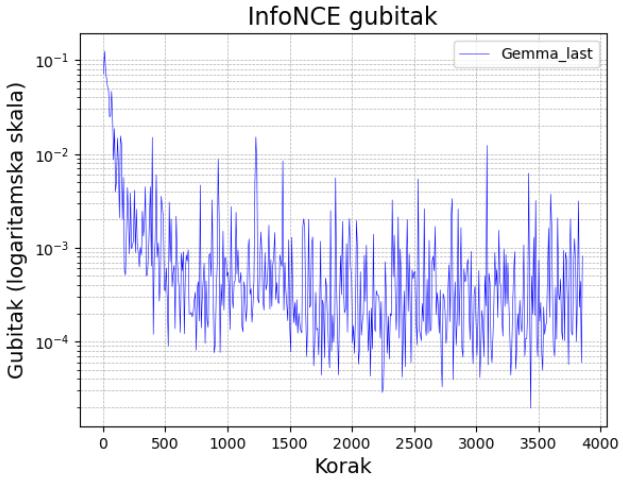
Slika 3.3: Preslika radne okoline na stranici Kaggle koja je korištena.

Odabir modela

Više popularnih javno dostupnih velikih jezičnih modela testirali smo kako bismo odredili onaj koji najbolje govori hrvatski. Svakom od modela (Mistral [25], LLama [53], Gemma [50] i Phi-3 [1]) postavili smo 10 pitanja na hrvatskom. Na temelju kvalitete odgovora, svaki smo odgovor ocijenili na skali od 1 do 5, pri čemu 1 označava izbjegavanje odgovora na hrvatskom ili vrlo loš odgovor, dok 5 označava tečan i kvalitetan odgovor. Analizom rezultata utvrdili smo da model Gemma postiže najbolje odgovore te smo ga odabrali kao početnu točku za daljnje istraživanje. U sljedečoj točki opisujemo skup podataka korišten za fino podešavanje i testiranje našeg modela.

Opis postupka

Originalni veliki jezični model Gemma smo prilagodili kao ugradbeni model na više načina. Prvi ugradbeni model, $\text{Gemma}_{\text{last}}$, smo napravili primjenom glave specijalnog tokena opisane u poglavlju 2.2. Drugi, $\text{Gemma}_{\text{avg}}$, koristi glavu prosjeka. U oba smo slučaja za fino podešavanje koristili metodu LoRA gdje su pomoćne matrice bile ranga 16. Pomoćne matrice dodali smo na sve slojeve s pažnjom te na feed-forward slojeve. Posebno, istrenirali smo model $\text{Gemma}_{\text{lastKV}}$ koji prilagođava samo matrice ključa K



Slika 3.4: Graf funkcije gubitka treniranja modela $\text{Gemma}_{\text{last}}$.

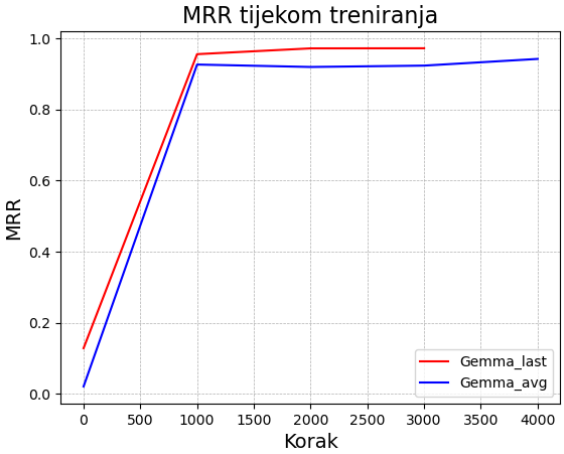
i pažnje V iz slojeva pažnje te koristi glavu specijalnog tokena.

Za moguću reproducibilnost, navodimo sve parametre treniranja. Točan naziv predtreniranog modela je gemma-2-9b-bnb-4bit. Od istog modela preuzet je i pripadni tokenizator. Za treniranje i evaluaciju je korištena grafička kartica T4 (16 GB) na stranici Kaggle (3.3). Trenirali smo model algoritmom stohastičkog gradijentnog spusta 4000 koraka sa stopom učenja 2e−4 koju smo postepeno dizali prvih 10 koraka. Veličina grupe (eng. *batch size*) bila je 2, no korištena je metoda gomilanja gradijenata (eng. *gradient accumulation*) svakih 8 koraka, što je ekvivalentno korištenju veličine grupe od 16 primjeraka.

Za skup treniranje smo koristili opisani skup iz prethodnog poglavlja. Dakle model je treniran na ukupno 8000 primjeraka, što je jednako umnošku broja koraka i veličine grupe. Korišten je AdamW optimizator [27] s parametrima $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1\text{e}{-8}$, weight decay $= 0.01$. Duljina konteksta, odnosno najveći broj riječi koji stane u model, postavljena je na 512 riječi.

Nismo fino podešavali cijeli model, već smo koristili metodu LoRA, objašnjenu u poglavlju 2.4. Za funkciju gubitka korišten je gubitak InfoNCE s negativnim primjercima unutar grupe, što znači jedan negativni primjerak za svaki par upita i relevantnog dokumenta. Evaluiran je na cijelom skupu podataka za testiranje od 1986 elemenata. Treniranje i evaluacija trajali su u prosjeku 12 sati. Graf funkcije gubitka prilikom treniranja modela $\text{Gemma}_{\text{last}}$ prikazan je na slici 3.4. Gubitak je prikazan na logaritamskoj skali. Primjećujemo da se gubitak s vremenom stabilizira.

Za usporedbu su korišteni ugradbeni enkoderski modeli BERT_{avg} i $\text{BERT}_{\text{last}}$. Oni su trenirani na isti način kao modeli Gemma. Također je uspoređen klasični model



Slika 3.5: Usporedba evaluiranih metrika tijekom treniranja modela $\text{Gemma}_{\text{last}}$ i modela $\text{Gemma}_{\text{avg}}$,

BM25, tako da su riječi u vokabularu određene razmacima. To znači da model riječi *pretraživanje* i *pretraživanja* smatra različitima. Pritom su uklonjene *zaustavne riječi*, odnosno one koje nemaju semantičko značenje. Primjer takvih riječi su: *a*, *ali*, *što*, itd. Skup od 179 zaustavnih riječi preuzet je s interneta [12].

3.3 Rezultati

Modeli su evaluirani svakih 1000 koraka, odnosno četiri puta tijekom treniranja, uključujući i početno stanje na testnom skupu podataka. Na slici 3.5 vidimo da modeli prelaze stopu pogodaka od 90% već u prvih tisuću koraka treniranja. Iako je skup podataka za testiranje u usporedbi s drugim radovima [2] malen, zanimljivo je primijetiti koliko se model brzo prilagodi novom zadatku. Konačne modele nakon završenog treniranja vrednujemo na testnom skupu podataka pomoću metrika MRR i HR@1.

Na tablici 3.1 su prikazani rezultati svih korištenih modela nakon treniranja. Najbolje rezultate postiže model $\text{Gemma}_{\text{last}}$ sa stopom pogodaka od čak 97%. Model BM25 postiže očekivano dobar rezultat, ali probleme mu postavlja način na koji je određen vokabular modela, kao što je spomenuto ranije. Modeli BERT lošiji su od BM25, no s obzirom na to da niti model BERT niti njegov vokabular nisu prilagođeni za hrvatski jezik, ovi rezultati nisu iznenađujući.

Model $\text{Gemma}_{\text{avg}}$ postiže lošiji rezultat od druga dva modela Gemma. Moguće je da je to posljedica načina na koji se generiraju ugradbeni vektor za tekst. Ta glava

Model	MRR	HR@1
BERT_{avg}	0.585	0.506
$\text{BERT}_{\text{last}}$	0.624	0.556
BM25	0.798,	0.750
$\text{Gemma}_{\text{avg}}$	0.942	0.911
$\text{Gemma}_{\text{lastKV}}$	0.951	0.923
$\text{Gemma}_{\text{last}}$	**0.972**	**0.955**

Tablica 3.1: Usporedba svih modela na testnom skupu podataka. Podebljani su najbolji rezultati.

uzima prosjek ugradbenih vektora svih riječi u rečenici, uključujući zaustavne riječi i one koje se nalaze na početku rečenice. S obzirom na to da pažnja gleda samo unazad, ugradbeni vektori riječi na početku ne mogu reprezentirati cijelu rečenicu.

Model $\text{Gemma}_{\text{lastKV}}$ je nešto bolji. Međutim, njegov učinak je ograničen parametrima koje fino podešava. Dok drugi modeli Gemma fino podešavaju sve korištene matrice u modelu tehnikom LoRA, ovaj model podešava samo matrice ključa K i vrijednosti V. Unatoč manjem broju parametara, treniranje je bilo podjednako brzo kao kod drugih modela.

Mogući daljnji smjerovi istraživanja

Iako je model $\text{Gemma}_{\text{last}}$ pokazao značajno bolje performanse od klasičnog modela BM25, treba uzeti u obzir i brzinu izvršavanja. Zanemarimo li početnu cijenu treniranja, i za kasnije pokretanje $\text{Gemma}_{\text{last}}$ je potrebna grafička kartica te je izvršavanje sporije od BM25. Računanje ugradbenih vektora koristeći velike jezične modele podrazumijeva puno matričnog množenja i računanja pažnje, a oboje su skupe operacije. Na mjestima gdje je potrebna velika brzina, korištenje velikih jezičnih modela u svrhu pretraživanja teksta stoga nije zadovoljavajuće.

S druge strane, zanimljivo je vidjeti da se modeli mogu brzo fino podesiti. Pogledamo li graf funkcije gubitka 3.4, vidimo da je on pao već nakon 500 koraka. U budućim istraživanjima bilo bi zanimljivo provjeriti koliko je najmanje koraka treniranja potrebno da se model istrenira. Naravno, poželjno je i koristiti veće skupove podataka za treniranje i evaluaciju. Kao zamjenu za BERT, treba usporediti ugradbene modele prilagođene za hrvatski, poput modela BERTić [30] i nove veće modele poput E5 [58] ili GTE [29].

Također, ključno je testirati koliko dobro model radi na drugim skupovima podataka prilikom evaluacije, odnosno koliko dobro generalizira. U ovom slučaju je

Upit i odgovor modela

Upit: Kako izbrisati anketu?

Odgovor bez RAG sustava:
Izbrisati anketu je jednostavno, ali ovisi o sustavu koji se koristi za upravljanje anketama. Evo nekoliko opcija:
1. Manualno brisanje:
- Ukoliko koristite online anketu s platformom poput SurveyMonkey, Google Forms, ili Typeform, obično se može ručno izbrisati anketa. - Ovisno o platformi, postoji opcija za brisanje ankete. - Na primjer, u SurveyMonkey, možete izbrisati anketu iz lista anketa.
2. Programiranje:
- Ako želite izbrisati anketu na više platformi, možda je potrebno programirati script koji će automatisati proces brisanja. - Ovo je kompleksna opcija, ali može biti učinljiva ako želite izbrisati mnoge ankete.
3. Usluge za brisanje anketa:
- Postoji nekoliko usluga koje nude pomoć u brisanju anketa. - Ti usluge obično za plaću, ali možete se obratiti za više informatije.

Dokument u bazi: Da biste izbrisali anketu: 1 Idite na stranicu My Surveys na vašem računu.2 Kliknite strelicu dolje s desne strane ankete koju želite izbrisati.3 Napomena: Od vas će se možda zatražiti da zatvorite sve otvorene kolekcionare prije nego što izbrišete anketu.4 Kliknite Izbriši.5 Pregledajte detalje.6 Kliknite Izbriši.Da biste izbrisali anketu: 1 Idite na stranicu My Surveys na vašem računu.2 Kliknite strelicu dolje s desne strane ankete koju želite izbrisati.3 Kliknite Izbriši.4 Pregledajte detalje.5 Kliknite Izbriši.1.Instalirajte oznaku za uklanjanje ankete XJZ.Ovo je veza s oznakom koju možete smjestiti u svoj bar s oznakom.Kada posjetite web mjesto s anketom, možete kliknuti oznaku kako biste pokušali ukloniti anketu.1 Oznaku XJZ možete pronaći na web stranici programera.2 Oznaka je dostupna besplatno.3 Jednostavno povucite i izbacite vezu u svoju oznaku kako biste stvorili oznaku.1. Posjetite web stranicu s anketom.Možete koristiti bilo koji od glavnih web preglednika za pristup funkciji Elementa Inspect.To će vam omogućiti da onemogućite odredene elemente web stranice, koji potencijalno uključuju

Odgovor s RAG sustavom
1. Idite na stranicu My Surveys na vašem računu.
2. Kliknite strelicu dolje s desne strane ankete koju želite izbrisati.
3. Kliknite Izbriši.
4. Pregledajte detalje.
5. Kliknite Izbriši.

Dodatne napomene:
* U nekim slučajevima, možda će se od vas zatražiti da zatvorite sve otvorene kolekcionare prije nego što izbrišete anketu.
* Oznaku za uklanjanje ankete XJZ možete pronaći na web stranici programera.
* Oznaka je dostupna besplatno

Slika 3.6: Odgovor modela Gemma koji koristi sustav RAG i odgovor modela koji ne koristi. Uočimo da je odgovor modela koji koristi sustav RAG puno precizniji.

skup za evaluaciju izgledao slično onome za treniranje jer su skupljani na isti način i tek na kraju podijeljeni. Još jedan smjer istraživanja bio bi provjeriti koji modeli se uopće mogu prilagoditi na nove jezike. Iako smo isprobali model Gemma koji je već u početku dobro odgovarao na pitanja iz hrvatskog, bilo bi dobro usporediti ga s modelima Mistral i Phi-3.

3.4 Sastavljanje sustava RAG

U konačnici smo sastavili sustav RAG koji odgovara na pitanja na hrvatskom jeziku. Za pretraživanje teksta koristimo ugradbeni model $\text{Gemma}_{\text{last}}$. Za jezični model koristimo model gemma-7b-it-bnb-4bit, gotovo identičan onome koji je korišten za fino podešavanje u prethodnom poglavlju. Razlika je da gemma-7b-it-bnb-4bit bolje prati upute od gemma-7b-bnb-4bit te je podešen tako da dobro odgovara na upite koji su RAG formata, odnosno sadrže upit i dokument. Na slici 3.6 vidimo primjer razgovora s jezičnim modelom gemma-7b-it-bnb-4bit bez i s ugrađenim sustavom RAG. Primijetimo da na postavljeno pitanje *Kako izbrisati anketu* odgovara s podatcima koji se odnose na konkretan sustav. U stvarnom svijetu ovakvi se sustavi mogu koristiti kao pomoć u korištenju aplikacija ili u odgovaranju na stručna pitanja. Bazu dokumenata je tada lako zamijeniti primjenjivijom za danu svrhu.

3.5 Zaključak

U ovom radu objasnili smo kako se ugradbeni vektori mogu koristiti u pretraživanju teksta te naveli ostale primjene. U drugom poglavlju obradili smo arhitekturu transformera koja je ključna za razumijevanje velikih jezičnih modela, koji su danas predmet intenzivnih istraživanja i primjene. Obradili smo mehanizam pažnje te načine na koje se modeli treniraju, uključujući tehniku LoRA. Također smo pokazali kako se ugradbeni i jezični modeli mogu koristiti zajedno u sustavu RAG te značajno povećati opseg znanja i primjenjivost velikih jezičnih modela. Navedeni sustav može se primjenjivati bez zahtjevnog finog podešavanja modela, a istovremeno posjedovati relevantno znanje o trenutnom programu ili okruženju u kojem se koristi.

Predstavili smo ugradbeni model $\text{Gemma}_{\text{last}}$ baziran na modelu Gemma, koji generira ugradbene vektore prilagođene pretraživanju teksta na hrvatskom. Eksperimentalni rezultati pokazali su da ovaj model na našem skupu podataka nadmašuje ostale metode. Cijeli proces prikupljanja podataka i treniranja detaljno je opisan u ovom radu te se može reproducirati uz umjerene zahtjeve za računalne resurse. Kao završni korak, razvili smo RAG sustav temeljen na našem modelu, sposoban za odgovaranje na pitanja na hrvatskom jeziku.

Bibliografija

[1] Marah Abdin et al., *Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone*, arXiv preprint arXiv:2404.14219 (2024).

[2] Vassilios Antonopoulos, *A Greek LLM after fine-tuning Llama2 7b*, 2024, dostupno na `https://medium.com/11tensors/a-greek-llm-after-fine-tuning-llama2-7b-678a8eca1ab3` (siječanj 2025.).

[3] Ljiljana Arambašić, Tomislav Berić i Ana Prlić, *Linearna algebra 2: vježbe*, Prirodoslovno-matematički fakultet, Zagreb, 2021.

[4] Dzmitry Bahdanau, Kyunghyun Cho i Yoshua Bengio, *Neural Machine Translation by Jointly Learning to Align and Translate*, arXiv preprint arXiv:1409.0473 (2016).

[5] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary i Tong Wang, *MS MARCO: A Human Generated MAchine Reading COmprehension Dataset*, arXiv preprint arXiv:1611.09268 (2018).

[6] Tom B. Brown et al., *Language Models are Few-Shot Learners*, arXiv preprint arXiv:2005.14165 (2020).

[7] Paul Covington, Jay Adams i Emre Sargin, *Deep neural networks for youtube recommendations*, Proceedings of the 10th ACM conference on recommender systems, 2016, str. 191–198.

[8] Michael Han Daniel Han i Unsloth team, *Unsloth*, 2023, dostupno na `http://github.com/unslothai/unsloth` (siječanj 2025.).

[9] DeepSeek-AI, *DeepSeek-V3 Technical Report*, arXiv preprint arXiv:2412.19437 (2024).

[10] Jacob Devlin, Ming Wei Chang, Kenton Lee i Kristina Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, arXiv preprint arXiv:1810.04805 (2019).

[11] Michel Marie Deza i Elena Deza, *Encyclopedia of Distances*, sv. 3, Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2014.

[12] Gene Diaz et al., *Stopwords ISO*, 2020, dostupno na `https://github.com/stopwords-iso/stopwords-iso` (siječanj 2025.).

[13] Hugging Face, *Transformers*, dostupno na `https://huggingface.co/docs/transformers` (siječanj 2025.).

[14] The PyTorch Foundation, *PyTorch*, dostupno na `https://pytorch.org` (siječanj 2025.).

[15] Philip Gage, *A new algorithm for data compression*, The C Users Journal **12** (1994), br. 2, str. 23–38.

[16] Yoav Goldberg, *Neural network methods in natural language processing*, sv. 10, Morgan & Claypool Publishers, 2017.

[17] Ian Goodfellow, *Deep learning*, sv. 196, MIT press, 2016.

[18] Google, *Kaggle*, dostupno na `https://www.kaggle.com` (siječanj 2024.).

[19] Clinton Gormley i Zachary Tong, *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*, O'Reilly Media, Inc., 2015.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren i Jian Sun, *Deep Residual Learning for Image Recognition*, arXiv preprint arXiv:1512.03385 (2015).

[21] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes i Yejin Choi, *The Curious Case of Neural Text Degeneration*, arXiv preprint arXiv:1904.09751 (2020).

[22] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan i Sylvain Gelly, *Parameter-efficient transfer learning for NLP*, International conference on machine learning, PMLR, 2019, str. 2790–2799.

[23] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang i Weizhu Chen, *LoRA: Low-Rank Adaptation of Large Language Models*, arXiv preprint arXiv:2106.09685 (2021).

[24] Pinecone Systems Inc., *Pinecone: Scalable, Serverless Vector Database*, 2021, dostupno na `https://www.pinecone.io` (veljača 2025.).

[25] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix i William El Sayed, *Mistral 7B*, arXiv preprint arXiv:2310.0682 (2023).

[26] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu i Dario Amodei, *Scaling Laws for Neural Language Models*, arXiv preprint arXiv:2001.08361 (2020).

[27] Diederik P. Kingma i Jimmy Ba, *Adam: A Method for Stochastic Optimization*, arXiv preprint arXiv:1412.6980 (2017).

[28] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel i Douwe Kiela, *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, arXiv preprint arXiv:2005.11401 (2021).

[29] Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie i Meishan Zhang, *Towards General Text Embeddings with Multi-stage Contrastive Learning*, arXiv preprint arXiv:2308.03281 (2023).

[30] Nikola Ljubešić i Davor Lauc, *BERTić – The Transformer Language Model for Bosnian, Croatian, Montenegrin and Serbian*, arXiv preprint arXiv:2104.09243 (2021).

[31] Xueguang Ma, Liang Wang, Nan Yang, Furu Wei i Jimmy Lin, *Fine-Tuning LLaMA for Multi-Stage Text Retrieval*, arXiv preprint arXiv:2310.08319 (2023).

[32] Christopher D Manning, *An introduction to information retrieval*, Cambridge University Press, 2008.

[33] Masoumzadeh, *From Rule-Based Systems to Transformers: A Journey through the Evolution of Natural Language Processing*, 2024, dostupno na `https://medium.com/@masoumzadeh/from-rule-based-systems-to-transformers-a-journey-through-the-evolution-of-natural-language-9131915e06e1` (siječanj 2025.).

[34] Microsoft, *Microsoft Copilot*, dostupno na `https://copilot.microsoft.com` (veljača 2024.).

[35] Tomas Mikolov, Kai Chen, Greg Corrado i Jeffrey Dean, *Efficient Estimation of Word Representations in Vector Space*, arXiv preprint arXiv:1301.3781 (2013).

[36] Jorge Nocedal i Stephen J Wright, *Numerical optimization*, Springer, 1999.

[37] OpenAI, *ChatGPT: Large Language Model*, 2023, dostupno na `https://openai.com` (siječanj 2025.).

[38] Sinno Jialin Pan i Qiang Yang, *A Survey on Transfer Learning*, IEEE Transactions on Knowledge and Data Engineering **22** (2010), br. 10, str. 1345–1359.

[39] Jeffrey Pennington, Richard Socher i Christopher Manning, *GloVe: Global Vectors for Word Representation*, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (Doha, Qatar) (Alessandro Moschitti, Bo Pang i Walter Daelemans, ur.), Association for Computational Linguistics, 2014, str. 1532–1543.

[40] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger i Ilya Sutskever, *Learning Transferable Visual Models From Natural Language Supervision*, arXiv preprint arXiv:2103.00020 (2021).

[41] Herbert Robbins i Sutton Monro, *A stochastic approximation method*, The annals of mathematical statistics (1951), str. 400–407.

[42] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford et al., *Okapi at TREC-3*, Nist Special Publication Sp **109** (1995), str. 0–.

[43] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser i Björn Ommer, *High-Resolution Image Synthesis with Latent Diffusion Models*, arXiv preprint arXiv:2112.10752 (2022).

[44] Frank Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain.*, Psychological review **65** (1958), br. 6, str. 386.

[45] Rico Sennrich, *Neural machine translation of rare words with subword units*, arXiv preprint arXiv:2401.12345 (2015).

[46] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray

Kavukcuoglu, Thore Graepel i Demis Hassabis, *Mastering the game of Go with deep neural networks and tree search*, Nature **529** (2016), str. 484–503.

[47] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen i Yunfeng Liu, *RoFormer: Enhanced Transformer with Rotary Position Embedding*, arXiv preprint arXiv:2104.09864 (2023).

[48] Chi Sun, Xipeng Qiu, Yige Xu i Xuanjing Huang, *How to Fine-Tune BERT for Text Classification?*, arXiv preprint arXiv:1905.05583 (2020).

[49] Ilya Sutskever, Oriol Vinyals i Quoc V. Le, *Sequence to Sequence Learning with Neural Networks*, arXiv preprint arXiv:1409.3215 (2014).

[50] Gemma Team, *Gemma 2: Improving Open Language Models at a Practical Size*, arXiv preprint arXiv:2408.00118 (2024).

[51] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava i Iryna Gurevych, *BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models*, arXiv preprint arXiv:2104.08663 (2021).

[52] The Unicode Consortium, *The Unicode Standard, Version 14.0*, 2021, dostupno na `https://www.unicode.org/versions/Unicode14.0.0` (veljača 2024.).

[53] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave i Guillaume Lample, *LLaMA: Open and Efficient Foundation Language Models*, arXiv preprint arXiv:2302.13971 (2023).

[54] Aaron van den Oord, Yazhe Li i Oriol Vinyals, *Representation Learning with Contrastive Predictive Coding*, arXiv preprint arXiv:1807.03748 (2019).

[55] Vladimir Vapnik, *The nature of statistical learning theory*, Springer science & business media, 2013.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser i Illia Polosukhin, *Attention Is All You Need*, arXiv preprint arXiv:1706.03762 (2023).

[57] Lovro Žmak, *Sustav za pretraživanje zbirke često postavljenih pitanja na hrvatskom jeziku*, diplomski rad, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, Zagreb, 2009.

[58] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder i Furu Wei, *Text Embeddings by Weakly-Supervised Contrastive Pre-training*, arXiv preprint arXiv:2212.03533 (2024).

[59] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder i Furu Wei, *Improving Text Embeddings with Large Language Models*, arXiv preprint arXiv:2401.00368 (2024).

Sažetak

Veliki jezični modeli donijeli su značajan napredak u području obrade prirodnog jezika, omogućujući dublje razumijevanje teksta putem ugradbenih vektora.

U ovom radu analizirali smo arhitekturu transformera, koja predstavlja temelj velikih jezičnih modela i ima ključnu ulogu u suvremenim tehnološkim rješenjima obrade jezika. Istražili smo metode pretraživanja teksta koristeći ugradbene vektore, kao i klasične pristupe poput BM25. Također smo razmatrali kako se ugradbeni vektori u kombinaciji s jezičnim modelima mogu koristiti unutar sustava RAG, čime se značajno proširuje raspon znanja i povećava primjenjivost velikih jezičnih modela.

Naš doprinos obuhvaća razvoj ugradbenog modela $\text{Gemma}_{\text{last}}$, temeljenog na arhitekturi Gemma, koji generira ugradbene vektore prilagođene pretraživanju teksta na hrvatskom jeziku. Eksperimentalni rezultati pokazali su da ovaj model na našem skupu podataka postiže nešto bolje rezultate u usporedbi s ostalim metodama. Kao završni korak, implementirali smo sustav RAG koristeći razvijeni model.

Summary

Large language models have brought significant advancements in the field of natural language processing, enabling a deeper understanding of text through embedding vectors.

In this study, we analyzed the transformer architecture, which serves as the foundation of large language models and plays a crucial role in modern language processing technologies. We explored text retrieval methods using embedding vectors, as well as classical approaches such as BM25. Additionally, we examined how embedding vectors, in combination with language models, can be utilized within the RAG system, significantly expanding the knowledge scope and increasing the applicability of large language models.

Our contribution includes the development of the embedding model $\text{Gemma}_{\text{last}}$, based on the Gemma architecture, which generates embedding vectors tailored for text retrieval in the Croatian language. Experimental results have shown that this model achieves slightly better performance on our dataset compared to other methods. Finally, we implemented a RAG system using the developed model.

Životopis

Roden sam u Zagrebu 14. prosinca 2000. godine. Osnovnoškolsko obrazovanje završio sam u školi Augusta Šenoe u Zagrebu. Nakon toga sam upisao XV. gimnaziju. Tijekom srednje škole aktivno sam se bavio natjecateljskim programiranjem i matematikom, gdje sam sudjelovao na dva državna natjecanja.

Akademske godine 2019./2020. sam upisao preddiplomski sveučilišni studij Matematika na Prirodoslovno-matematičkom fakultetu u Zagrebu. Nakon završetka preddiplomskog studija, akademske godine 2022./2023. sam upisao diplomski sveučilišni studij Računarstvo i matematika.

Moji profesionalni interesi uključuju proučavanje velikih jezičnih modela i ostale metode dubokog učenja. U slobodno vrijeme volim putovati, kuhati te uređivati bonsai drvca.