**UNIVERZITET U BEOGRADU**

**FAKULTET ORGANIZACIONIH NAUKA**

**ZAVRŠNI RAD**

# Razvoj RAG za unapređivanje odgovora velikih jezičkih modela

**Mentor**

**Dr Sandro Radovanović,**

**Asistent**

**Student**

**Zoran Mahovac 2020/0109**

**Beograd, 2024. godine**

# APSTRAKT

Cilj ovog rada je razvoj i implementacija aplikacije za razogovor sa knjigom *An Introduction to Statistical Learning with Application in Python* [28]. Kroz primenu metode *Retrieval-Augmented Generation* (RAG), aplikacija koristi sofisticirane mehanizme za pretragu informacija i generisanje odgovora kako bi krajnjem korisniku bili omogućeni precizni i kontekstualno relevantni odgovori. Ispitane su različite metodologije parsiranja PDF dokumenta implementirane u okviru *unstructured* biblioteke i *nougat* modela, razmotren je korpus različitih mehanizama podele dokumenta na segmente. U toku rada na arhitekturi aplikacije analizirana je prosečna brzina odgovora na evaluacionom setu upita upotrebom *Chroma* i *FAISS* vektorskih skladišta podataka, pri čemu je baza podataka sa kraćim prosečnim vremenom upotrebljena u analizi različitih arhitektura pretraživača, poput instanci klase *MultiVectorRetriever*, *ParentDocumentRetriever*, *ContextualCompressionRetriever* i *EnsembleRetriever*. Evaluirane su performanse ovih instanci pretraživača tokom pretrage za četiri različite strategije dekompozicije originalnog dokumenta upotrebom evaluacionih metrika kontekstni odziv i preciznost konteksta sa referencom uz asistiranje velikog jezičkog modela biblioteke *ragas.* U domenu transformacije korisničkog upita preispitani su različiti mehanizmi, kao što su transformacija originalnog upita u više različitih perspektiva originala, transformacija "korak unazad", transformacija upita kreiranjem hipotetičkog dokumenta, kao i mehanizmi dekompozicije korisničkog upita pri rekurzivnom i individualnom generisanju finalnog odgovora. Za pretragu slika, grafika i formula fino su podešeni ColBERT modeli i za svaki od njih kreiran je indeks nad kojim su evaluirane sposobnosti njegove pretrage. Konačna arhitektura aplikacije kreirana je implementacijom instance pretraživača sa najvišim vrednostima evaluacionih metrika (pretraživač je instanca klase *MultiVectorRetriever* i napunjen je sažecima originalnih dokumenata u vektorskom skladištu podataka) odgovarajuće strategije za separatisanje dokumenta (Strategija 1) uz primenu najbolje metode za transformaciju korisničkog upita (transformacija "korak unazad").

# SADRŽAJ

# 1 UVOD

Cilj ovog rada je razvoj i implementacija chatbot aplikacije zasnovane na knjizi *An Introduction to Statistical Learning with Application in Python* kroz primenu metode *Retrieval-Augmented Generation* (RAG) [48]. Ova literatura predstavlja jedan od vodećih obrazovnih resursa za početnike i stručnjake koji žele da steknu znanja o tehnikama i primenama statističkog učenja, posebno u Python okruženju. Međutim, pristup informacijama u okviru knjige može biti izazovan zbog njenog obima, posebno za korisnike koji traže brze i precizne odgovore na specifične upite.

Rad na implementaciji *chatbot* aplikacije sa pomenutom literaturom takođe je izazovan zbog različitih struktura podataka koje se u njoj nalaze (tekst, tabele, slike, kod u programskom jeziku Python), zbog čega će u radu biti istraženi i mehanizmi da se ovakve strukture transformišu u oblik pogodan za integrisanje u RAG lanac.

Kroz ovo istraživanje ispitaće se različiti aspekti razvoja aplikacije, uključujući metode za parsiranje PDF sadržaja, razdvajanje teksta na segmente, kao i izbor najboljih tehnika za vektorsko pretraživanje i optimizaciju retrievera. Evaluacija različitih pristupa i parametara pretrage, transformacija korisničkog upita kao i analiza vremena odgovora i tačnosti informacija, predstavljaju važne aspekte istraživanja koji će u ovom radu biti razmotreni ponaosob.

U radu će biti opisani i teorijski koncepti na kojima se zasniva celokupna implementacija aplikacije, kako bi se detaljnije razumeli odabiri odgovarajućih gradivnih elemenata u njenoj arhitekturi. Teorijski koncepti opisuju modele koji su korišćeni prilikom parsiranja PDF dokumenta, trening odabranih modela, evaluacione metrike korišćene pri merenju performansi kao i ideje na kojima se zasnivaju mehanizmi transformacije korisničkih upita i skladištenja podataka za pretragu u pretraživač.

# 2 PARSIRANJE DOKUMENTA

## 2.1 PARSER DOKUMENATA BIBLIOTEKE UNSTRUCTURED

*Unstructured* [17] obezbeđuje platformu i alate za unos i obradu nestrukturiranih dokumenata u kontekstu sistema zasnovanih na RAG metodologiji i fino podešavanje modela. Ova platforma omogućava automatizovano procesiranje velikih količina neorganizovanih podataka, kao što su PDF dokumenti, tekstualni fajlovi, slike, i drugi oblici nestrukturiranih podataka, kako bi se ovi podaci transformisali u strukture pogodne za dalje pretraživanje i analiziranje. Kroz alate za parsiranje i segmentaciju, *Unstructured* omogućava izdvajanje ključnih informacija, poput entiteta, ključnih reči i semantički relevantnih fraza, što doprinosi visokom kvalitetu podataka za treniranje modela. Osim toga, primenom ovih alata, korisnici mogu optimizovati procese pretrage i generisanja odgovora, čime se značajno poboljšava efikasnost i preciznost RAG sistema i modela za jezičku obradu. Integracija ovih alata u RAG sisteme posebno je korisna za zadatke poput pretrage dokumenata, kontekstualne analize i generisanja odgovora, gde je od suštinske važnosti kvalitetan unos i obrada podataka. [17]

### 2.1.1 YOLOX model

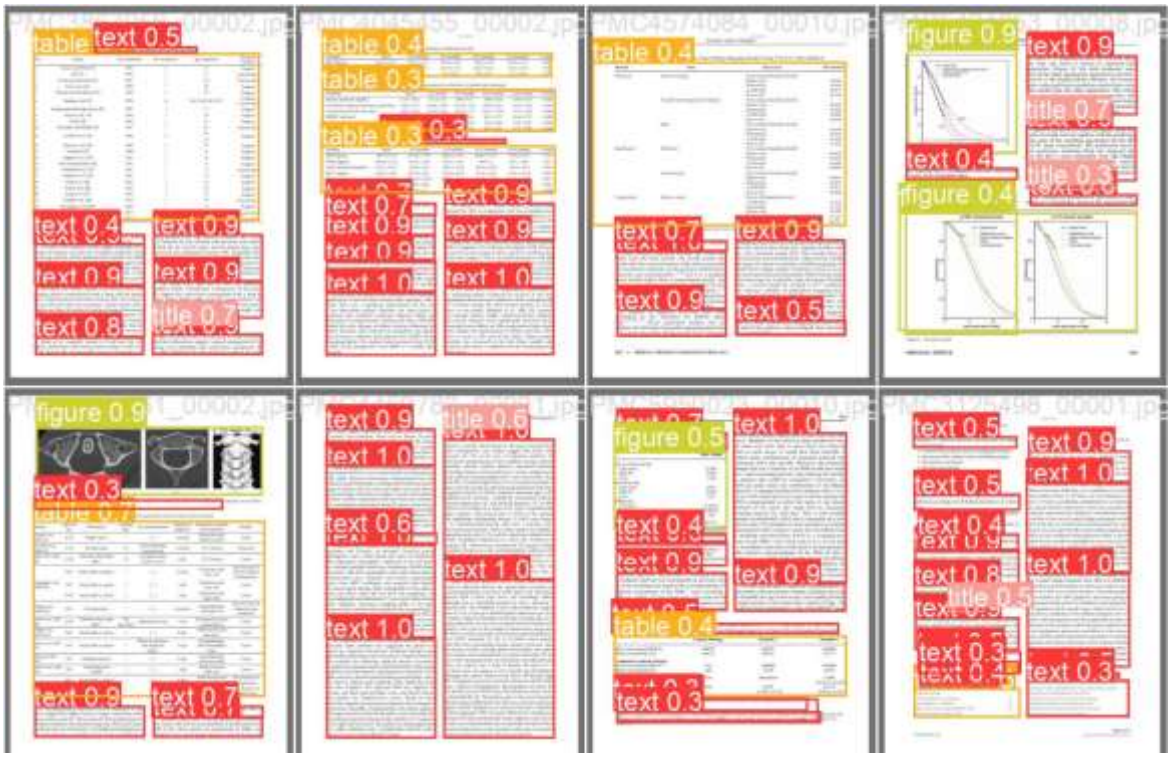
*Unstructured* biblioteka obezbeđuje *partition_pdf* metodu za parsiranje PDF dokumenta u elemente, čije su klase implementirane u *unstructured.documents* modulu. *Text* klasa pomaže u ekstrakciji teksta iz PDF dokumenta; *Formula* klasa opisuje element koji sadrži formulu u dokumentu; *CompositeElement* klasa generiše instance kombinacijom jednog ili više elemenata klase *Element* prilikom povezivanja ili separatisanja (u zavisnosti od parametra koji definiše broj karaktera tekstualnog segmenta) teksta u tekstualne segmente; *FigureCaption* klasa koristi se za beleženje tekstualnog opisa povezanog sa slikama; *NarrativeText* klasa predstavlja element konstituisan od više rečenica koje obuhvataju i naslove, zaglavlja, podnožja, opise; *ListItem* klasa čuva narativni tekst koji je deo neke nabrajajuće strukture; *Title* klasa čuva naslove; *Address* klasa čuva adrese; *Image* klasa beleži izvučene slike; *PageBreak* klasa skladišti stranične prekide; *Table* klasa čuva tabele; *Header, Footer* klase čuvaju sadržaje zaglavlja i podnožja, respektivno; *CodeSnippet* klasa

čuva segmente koda prepoznate u PDF dokumentu; *FormKeysValues* klasa čuva "ključ - vrednost" rečnike; *PageNumber* klasa skladišti brojeve stranica; *EmailAddress* klasa koristi se za čuvanje tekstualnog sadržaja u obliku imejl adresa. [18]

U odnosu na potrebe korisnika, biblioteka *unstructured* obezbeđuje modele koji su bazirani na OCR tehnologiji ili Transformer arhitekturi ne bi li se detektovali kompleksni rasporedi elemenata u dokumentu i predvideo tip elemenata. Ukoliko se korisnik odluči za *hi_res* strategiju, model za parsiranje PDF fajla koristi arhitekturu dubokog učenja ne bi li razumeo raspored elemenata na stranici. Metoda *_partition_pdf_or_image_loacl()* koja se poziva pri odabiru ove strategije prima podatke o PDF fajlu koji je potrebno parsirati, kao i odabiru korisnika da li je potrebna ekstrakcija slika i tabela. Kada se PDF fajl parsira *hi_res* strategijom, potrebno je postaviti DPI (*Dots Per Inch*) parametar koji opisuje gustinu tačaka koje čine sliku na ekranu. Veće vrednosti DPI parametra omogućuju da slika bude detaljnija i oštrija jer ima više tačaka po jedinici dužine. Podrazumevano, DPI parametar ima vrednost 200. Promenljive u koje se skladište rezultati primene modela detekcije (*od_model_layout_dumper* – skladišti rezultat primene modela za detekciju objekata; *ocr_layout_dumper* – skladišti rezultat primene OCR modela). Pozivom metode *process_data_with_model,* koja prima putanju do fajla, naziv modela korišćenog tokom primene *hi_res* strategije, kao i vrednost broja tačaka po inču. Ova metoda vraća raspored elemenata PDF dokumenta stranicu po stranicu, upotrebom metode *get_elements_with_detection_model,* biblioteke *unstructured_inference,* u modulu *inference.* U modulu *unstructured_inference.models* u fajlu *yolox.py* nalazi se potrebne metode za primenu YOLOX modela. [49] Ovaj model ima važnu ulogu u parsiranju dokumenata jer omogućava preciznu analizu rasporeda dokumenata *(document layout analysis*). To znači da model može da prepozna i klasifikuje različite tipove elemenata unutar dokumenta, kao što su pasusi, slike, tabele, formule, i drugi elementi nabrojanih klasa, što je ključni korak u daljoj obradi i ekstrakciji informacija. Ukoliko YOLOX model prepozna tabelu, onda je za parsiranje sadržaja tabele zadužen *TableTransformerForObjectDetection* model biblioteke *transformer.* [49]

Već je rečeno da se za potrebe parsiranja PDF fajla koristi *YOLOX* model ne bi li bio prepoznat raspored elemenata odgovarajućih klasa, čije su labele nabrojane u *yolox.py* fajlu biblioteke *unstructured – inference.* Rečnik *MODEL_TYPES* sadrži tri konfiguracije

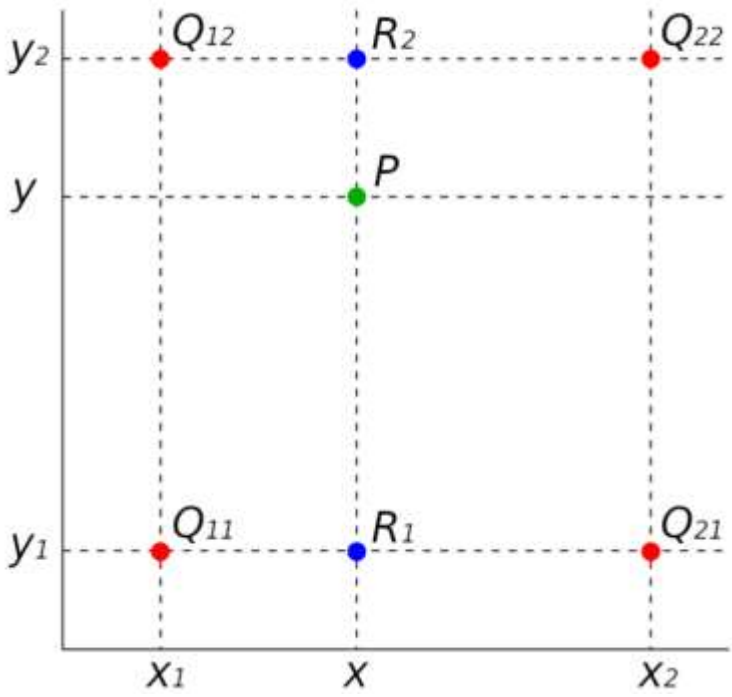
YOLOX modela – standardnu verziju, manju verziju YOLOX modela i kvantizovanu verziju YOLOX modela, optimizovanu za bolje performanse na uređajima sa ograničenim resursima. Ovaj model implementiran je pomoću klase sa atributima *model_path* – parametar opisuje putanju do modela, *model* – krerira se sesija za izvođenje *ONNX* modela koristeći biblioteku *onnxruntime, layout_classes* – rečnik koji čuva nazive klasa koje opisuju elemente različitih tipova. Metoda *initialize* pokreće sesiju za izvođenje modela, pomoću jednog od provajdera (1) *TensorrtExecutionProvider* – koristi TensorRT kao NVIDIA platformu za optimizaciju modela dubokog učenja i upotrebu njihovih GPU jedinica; (2) *CUDAExecutionProvider* – koristi CUDA API za izvođenje na NVIDIA GPU jedinicama; (3) *CPUExecutionProvider* – koristi CPU jedinice za upotrebu modela, što je uvek dostupna i najsporija opcija. Metoda *predict* kao parametar prima sliku dokumenta klase *PILImage.Image* i koristi metodu *predict()* instance klase *UnstructuredYoloXModel*, tako da je rezultat ove metode raspored elemenata dokumenta. Glavna metoda koja poziva *YOLOX* model za detekciju rasporeda elemenata koja vraća *PageLayout* parametre je *image_processing* u istom fajlu. Klasa *PageLayout* u sebi ima sačuvane sledeće atribute – *element_probs* – niz verovatnoća koje predstavljaju poverenje modela u klasifikaciju svakog elementa; *element_class_ids* – niz koji sadrži identifikatore klase svakog elementa; *element_class_id_map* – mapa koja povezuje identifikatore klase svakog elementa sa nazivima klasa, kao i atribute nasleđene iz klase *TextRegions: element_coords* –niz koordinata svakog elementa u formatu ($x_1$, $x_2$, $y_1$, $y_2$), *texts* – niz tekstova povezanih sa svakim elementom; *source* – u slučaju *YOLOX* modela vrednost parametra data je sa *Source.YOLOX*. Nakon naknade obrade sirovog izlaza nastalog primenom YOLOX modela, dobijaju se koordinate okvira u obliku *(x_center, y_center, width, height),* kao i vrednosti sigurnosti modela da je neki objekat zaista detektovan u okviru. Ostali elementi vektora opisuju verovatnoće po klasama za svaki objekat (za klase *CAPTION, FOOTNOTE, FORMULA, LIST_ITEM, PAGE_FOOTER, PAGE_HEADER, PICTURE, SECTION_HEADER, TABLE, TEXT, TITLE*). [49]



**Slika 1. Vizuelni prikaz rezultata primene YOLOX modela za zadatak detektovanja rasporeda elemenata**

Tokom pripreme slike za model, potrebno je izračunati faktor skaliranja r, kao minimalni odnos između ciljne i trenutne visine, kao i ciljne i trenutne širine, ne bi li slika bila uklopljena unutar ciljne veličine. Pomoću ovog faktora čuva se proporcionalnost stranica slike, ali se menjaju visina i širina, pri čemu će jedna od dimenzija ulazne slike biti jednaka jednoj od ciljnih dimenzija nakon skaliranja ovim faktorom. Ostatak slike popunjava se neutralnom, sivom bojom (vrednost piksela je 114). Prilikom primene YOLOX modela ciljne dimenzije koje se žele postići su (1024, 768). [49] Ako je ulazna slika, na primer, dimenzija (924, 412), onda će minimalni fakotr r biti 1.10822, nakon čega se množenjem originalnih dimenzija ovim faktorom omogućuje da slika "raste" do veličine prve dimenzije 1024 piksela visine, kada će nove dimenzije biti (1024, 456.5866). Prilikom promene dimenzionalnosti slike potrebno je primeniti odgovarajući metod interpolacije, a tokom preprocesiranja slike za primenu YOLOX modela koristi se *cv2.INTER_LINEAR* metod. [49] Ovaj tip interpolacije koristi algoritam bilinearne interpolacije. Bilinearna interpolacija je definisana kao linearna interpolacija na dve ose (x i y osa). Pretpostavimo da je definisan set tačaka, $(x_k, y_k)$, gde je k = 1,2. Ove koordinate definišu pozicije tačaka $Q_{11}$, $Q_{21}$, $Q_{12}$ i $Q_{22}$. Za svaku tačku definisanu koordinatama x, y koje su locirane između tačaka $Q_{11}$, $Q_{21}$,

$Q_{12}$ i $Q_{22}$, upotrebom tehnike bilinearne interpolacije može se odrediti P tačka, opisana preko x i y koordinata.




**Slika 2. Princip rada bilinearne interpolacije**

Da bi se pronašla tačka P(x, y) kroz bilinearnu interpolaciju, potrebno je izvršiti dve linearne interpolacije po x – osi kako bi se odredile tačke $R_1$ (određivanjem duži određene tačkama $Q_{11}$, $Q_{21}$, kada je $y_1$ konstantno) i $R_2$ (određivanjem duži definisane tačkama $Q_{12}$, $Q_{22}$, kada je $y_2$ konstantno), kao i jednu linearnu interpolaciju po y osi da bi se pronašla tačka P (određivanjem duži definisane tačkama $R_1$ i $R_2$, kada je x konstantna vrednost, pa se vrši skaliranje po y). Ove tačke definisane su sledećim formulama:

$$R_1(x,y) = Q_{11} * \frac{(X_2 - X)}{(X_2 - X_1)} + Q_{21} * \frac{(X - X_1)}{(X_2 - X_1)} \quad (1)$$

$$R_2(x,y) = Q_{12} * \frac{(X_2 - X)}{(X_2 - X_1)} + Q_{22} * \frac{(X - X_1)}{(X_2 - X_1)} \quad (2)$$

Interpolacija P(x, y) data je kao:

$$P(x,y) = R_1 * \frac{(Y_2 - Y)}{(Y_2 - Y_1)} + R_2 * \frac{(Y - Y_1)}{(Y_2 - Y_1)} \quad (3)$$

[50] Nakon primene bilinearne transformacije, vrši se permutacija dimenzija originalne slike (*visina, širina, broj kanala*) u oblik (*broj kanala, visina, širina*), koji je pogodan za primenu modela dubokog učenja kao što je YOLOX. Nakon transponovanja, potrebno je da se niz vrati u kontinualni prikaz, kada su njegovi elementi memorijski organizovani u redosledu koji je linearan i bez prekida, što je narušeno pomenutom transformacijom, kada su promenjene dimenzije niza. Model *YOLOX* sačuvan je u *ONNX (Open Neural Network Exchange)* formatu i pokreće se pomoću instance *InferenceSession* klase, kojoj se prosleđuje putanja do direktorijuma modela, kao i provajder. [49] Kada je reč o obradi izlaza nakon primene YOLOX modela, izlazni tenzor ima dimenzije (N, M, C), gde je N broj nivoa mrežnih ćelija (1 za jednu mrežu slike), M je broj ćelija mreže, a C je dimenzija izlaznog vektora data sa 2 * 5 + C, gde je C vektor koji opisuje verovatnoću pripadnosti svake od 10 klasa za određeni okvir u kojem se nalazi objekat. Dimenzija pet opisuje koordinate centra, visinu, širinu okvira i ocenu objektivnosti (da li se u okviru nalazi objekat ili ne). Na primer, ako je za vrednost parametra *strides,* koji opisuje dimenziju jedne mrežne ćelije u obliku liste dimenzija, dato [8, 16, 32], to znači da model *YOLOX* vrši predviđanja na svim veličina okvira definisanim ovim parametrom; za vrednost osam, biće definisano 12896 ćelija slike (1024, 768) dimenzija, za vrednost 16, biće 3072 ćelije, a za vrednost 32 ukupno 768 ćelija, što je 16128 ćelija za sve tri vrednosti veličine, pa bi dimenzija sirovog izlaza bila (1, 16128, 20). [49] Koordinate centra koje su dobijene upotrebom YOLOX modela transformišu se iz relativnih u apsolutne vrednosti, o čemu će biti reči kasnije.

YOLOX (You Only Look Once Extended) predstavlja napredak u razvoju objektnog detekcionog modela koji se temelji na seriji YOLO algoritama. Kao jedna od najsavremenijih arhitektura za detekciju objekata u realnom vremenu, YOLOX integriše ključne mehanizme koji poboljšavaju tačnost i brzinu prepoznavanja objekata. Ovaj model koristi pristup koji se oslanja na direktno predviđanje pozicija bez oslanjanja na prethodno postavljene tačke ili referentne okvire, koji pojednostavljuje proces detekcije, smanjuje potrebu za ručnim postavljanjem okvira i omogućava direktno predviđanje granica objekata. Pored toga, YOLOX implementira dinamičku programabilnost i poboljšane tehnike augmentacije podataka, čime se značajno povećava robustnost modela i njegova sposobnost

generalizacije na nepoznatim skupovima podataka. Ove karakteristike čine YOLOX pogodnim za različite primene, uključujući autonomna vozila, nadzor i analizu video sadržaja. [23]

YOLO (*You Only Look Once*) model kao input prima sliku, a kao output daje $\{b_1, b_2, \ldots, b_n\}$ okvirne kutije za n detektovanih objekata, kao i $\{c_1, c_2, \ldots, c_n\}$ labele klasa za svih n detektovanih objekata. Algoritmi pre YOLO modela, poput RCNN su dvofazne mreže koje koriste na prvom nivou CNN (naziva se i Backbone) ne bi li generisali okvirne kutije, a zatim u drugoj fazi vrše klasifikaciju detektovanih okvira, koja je bazirana na generisanim konvolucionim mapama atributa, koje sadrže semantičke informacije o slikama. Nakon kreiranja konvolucionih mapa atributa, primenjuje se RPN za detekciju okvira (u tom trenutku još se ne zna labela objekta koji je klasifikovan), a zatim se primenjuju iste konvolucione mape atributa da bi se upotrebio algoritam klasifikacije ili regresije koji na osnovu informacija o koordinatama okvirnih kutija klasifikuje objekat. [24] Nedostaci ovakvog pristupa leže u broju faza koje je potrebno izvršiti, pojedinačnom treningu pomenutih komponenti, kompleksnosti upotrebe u svakodnevnim aplikacijama, kao i nemogućnosti generalizacije na ostale domene. Glavna ideja YOLO modela je jedinstvena mreža za detekciju objekata koja je zadužena za oba zadatka (predviđanje okvirnih kutija i zadatak predviđanja labela objekata), tako da se dvofazni zadatak svede na jednofazni regresioni zadatak. [2]

Da bi se opisali inovativni pristupi u implementaciji YOLOX modela, potrebno je opisati osnovnu arhitekturu i mehanizme koji su zadržani još od prve verzije, YOLOv1 modela. Početna slika originalnih dimenzija se prilagođava fiksnoj veličini 448 x 448, deli se u S x S ćelija (u originalnom radu S = 7) tako da svaka ćelija bude veličine 64 x 64 piksela. To znači da je svaka ćelija odgovorna za predikciju samo jednog objekta unutar granica od 64 x 64 piksela mreže. Stvarne vrednosti date su za svaki objekat u obliku (x, y, w, h). Umesto predviđanja tačnih vrednosti za koordinate centra, visinu i širinu, neuronska mreža vrši predikciju relativno u odnosu na ćeliju mreže; umesto predviđanja vrednosti (x, y, w, h), vrednosti koje se predviđaju su ($\Delta x, \Delta y, \Delta w, \Delta h$), koje su date sa

$$\Delta x = \frac{x - x_\alpha}{64} \tag{4}$$

$$\Delta y = \frac{y - y_\alpha}{64} \tag{5}$$

$$\Delta w = \frac{w}{448} \tag{6}$$

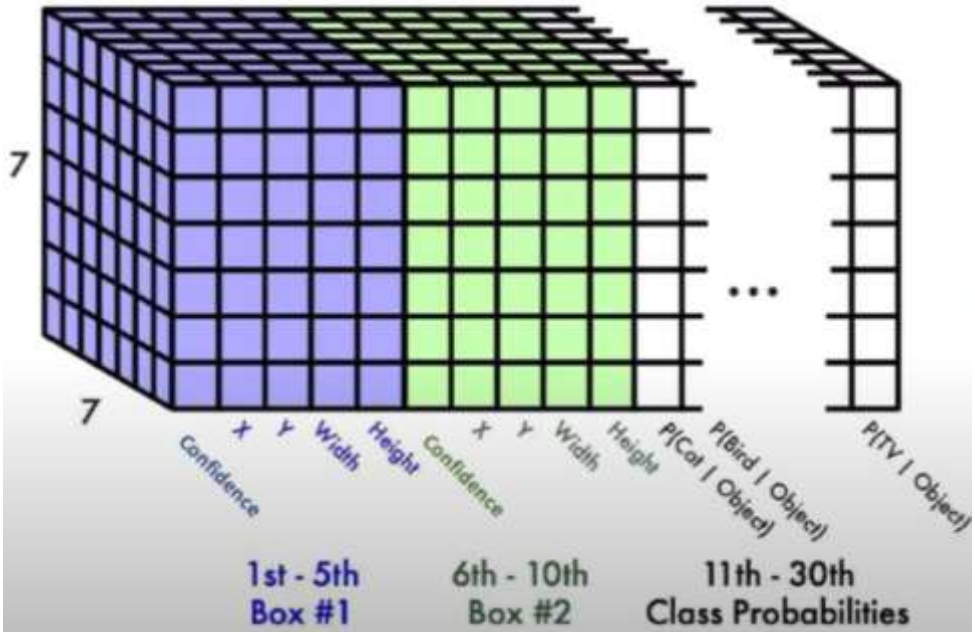
$$\Delta h = \frac{h}{448} \tag{7}$$

gde su x i y koordinate tačke gornjeg levog ugla stvarnog objekta; w i h – širina i visina okvirne kutije stvarnih vrednosti; $x_\alpha$, $y_\alpha$ - koordinate gornjeg levog ugla ćelije mreže. [2], [24]

Zadatak neuronske mreže jeste da izračuna ciljne vrednosti u odnosu na gornji levi ugao jedne ćelije mreže, pri čemu su visina i širina nezavisne u odnosu na koordinate gornjeg levog ugla, dok vrednosti Δx, Δy zavise od ćelije. Izračunate vrednosti Δx, Δy, Δw, Δh nazivamo relativnim stvarnim vrednostima zato što uzimaju u obzir koordinate gornjeg levog ugla ćelije mreže slike. To su vrednosti koje su mnogo manje od originalnih (na primer, (0.13, 0.87, 0.31, 0.56)) koje su generisane kako bi neuronska mreža lakše učila ciljne vrednosti (apsolutne vrednosti su, na primer, date sa (200, 311, 142, 250)). Za svaku od 49 ćelija mreže slike računaju se relativne ciljne vrednosti samo za one ćelije koje u sebi sadrže koordinate centra stvarnog objekta; za ostale ćelije, ove vrednosti biće jednake nuli. Za ćelije koje ne sadrže centar stvarnih objekta, postavlja se vrednost ocene objektivnosti na nula, u suprotnom vrednost je jedan. Za svaku ćeliju, kreira se vektor jedinica i nula (*OHE* vektor) koji označava pripadnost odgovarajućoj klasi koji je popunjen vrednošću jedan na mestu koje odgovara tačnoj klasi koju je potrebno prepoznati i vrednostima nula na svim ostalim mestima. [24]

Svaka ćelija mreže slike 7 x 7 predviđa dve okvirne kutije. To znači da će za svaku ćeliju mreže postojati tačno (2 x 5 + dimenzija *OHE* vektora) parametara (broj 5 odnosi se na Δx, Δy, Δw, Δh, c vrednosti). Na primer, ako je dimenzija *OHE* vektora 20, onda će izlazni sloj mreže biti 7 x 7 x 30 i opisan je na sledećoj slici. Na slici se takođe vidi da svaka ćelija mreže slike od 64 x 64 piksela ima svoje vrednosti za parametre *Confidence, X, Y, Width,*

*Height* za dve okvirne kutije kao i jedan *OHE* vektor dužine 20. Postoji samo jedan *OHE* vektor i najčešće je to vektor one okvirne kutije koji ima veću ocenu poverenja. [24]




**Slika 3.** Vizuelni prikaz izlaznog tenzora YOLO modela dimenzija 7 x 7 x 30 [24]

Kada je reč o parsiranju izlaznog vektora, potrebno je relativne vrednosti transformisati u apsolutne za obe okvirne kutije. Apsolutne vrednosti dobijamo obrnutom transformacijom u odnosu na fomrule za relativne vrednosti i one su date sa:

$$x_1 = \Delta x_1 * 64 + x_{\alpha,} \quad x_2 = \Delta x_2 * 64 + x_\alpha \qquad (8), (9)$$

$$y_1 = \Delta y_1 * 64 + y_{\alpha.} \quad y_2 = \Delta y_2 * 64 + x_\alpha \qquad (10), (11)$$

$$w_1 = \Delta w_1 * 448, \quad w_2 = \Delta w_2 * 448 \qquad (12), (13)$$

$$h_1 = \Delta h_1 * 448, \qquad h_2 = \Delta h_2 * 448 \qquad (14), (15)$$

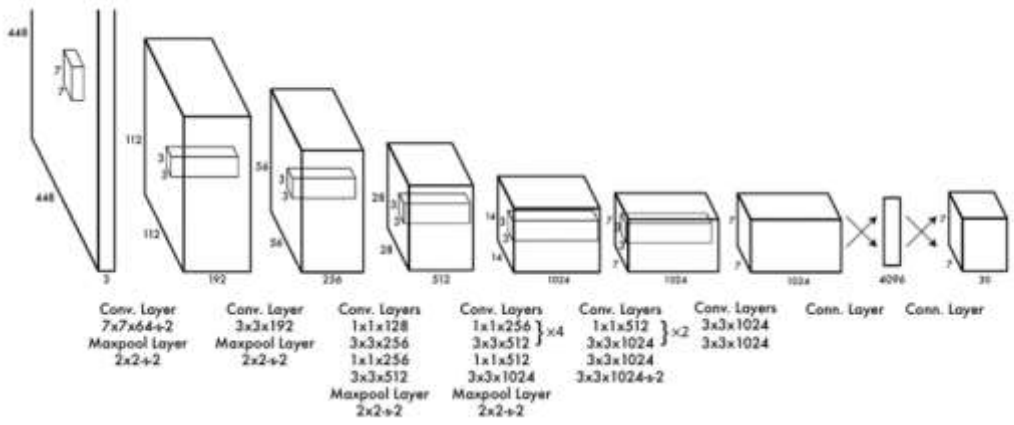
Za svaku okvirnu kutiju pronalazi se maksimalna vrednost verovatnoće u *OHE* vektorima, koje se koriste u izračunavanju vrednosti u literaturi poznatih pod nazivom uslovno poverenje, koje su opisane sa

$$p = \max(p_1, \dots, p_{20}) \tag{16}$$

$$\hat{C}_1 = C_{1^*} p \tag{17}$$

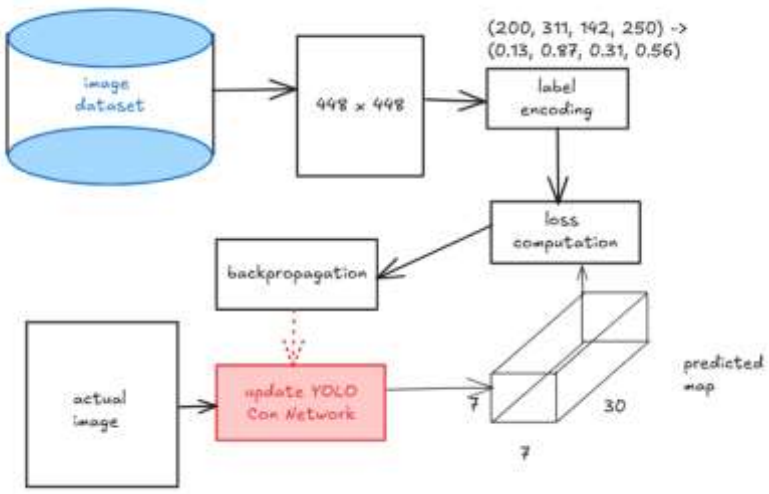
$$\hat{C}_2 = C_{2^*} p \tag{18}$$

gde je p maksimalna vrednost verovatnoće *OHE* vektora, $c_1$ i $c_2$ su vrednosti ocene poverenja za dve okvirne kutije, $\hat{c}_1$, $\hat{c}_2$ predstavljaju vrednosti uslovnog poverenja. Najčešće se zadržava ona okvirna kutija koja ima veću ocenu poverenja. Arhitektura YOLO modela inspirisana je *GoogleNet* modelom neuronske mreže sa 2 konvoluciona i 2 potpuno povezana sloja. Finalni rezultat je 7 x 7 x 30 mapa atributa nakon prolaska kroz 2 potpuno povezana sloja. [24]




**Slika 4.** Arhitektura YOLO modela [2]

Trening YOLO modela prikazan je na sledećoj slici. Tokom treninga korišćen je *Pascal VOC* skup podataka koji se sastoji od 20 klasa, dok je mreža prethodno trenirana na *Imagenet* skupu podataka na 224 x 224 rezoluciji slika, a sam trening je sproveden na slikama dimenzija 448 x 448 na *VOC* skupu podataka. [2]




**Slika 5.** Opis procesa treninga YOLO modela

Funkcija gubitka izračunava se kao zbir gubitaka svih ćelija rezolucije S x S mreže slike. Takođe je moguća situacija da u okviru neke ćelije ne postoji nijedan objekat. Zato se značaj ćelija koje nemaju nijedan objekat mora smanjiti i to se odražava na funkciju gubitka; ona se deli na dva dela, pri čemu se prvi deo odnosi na ćelije koje sadrže objekat, dok se drugi deo funkcije odnosi na ćelije koje ne sadrže objekte, koji se množi parametrom čija je vrednost manja od jedan ne bi li se umanjio njihov značaj. U originalnom radu koji predstavlja YOLO model, ovaj parametar iznosi 0.5 (*lambda parametar*). Takođe, u funkciji cilja postoje vrednosti $1^{obj}$ i $1^{no\_obj}$ koji opisuju da li ćelija ima ili nema objekat; ako ćelija sadrži objekat, onda će njen gubitak da se izračunava po prvom delu funkcije gubitka (vrednost $1^{obj}$ biće jednaka jedan, a vrednost $1^{no\_obj}$ biće nula), a u suprotnom funkcija gubitka izračunavaće se za odgovarajuću ćeliju po drugom delu jer će prvi deo funkcije biti nula (nemoguće je da jedna ćelija u isto vreme sadrži i ne sadrži objekat, odnosno da vrednosti $1^{obj}$ i $1^{no\_obj}$ obe budu jednake jedan). Formula kojom se opisuje izračunavanje vrednosti funkcije gubitka data je sa

$$L = \sum_{i=1}^{S^2} 1_i^{obj} * L_{i,obj} + \lambda_{no\_obj} \sum_{i=1}^{S^2} 1_i^{no\_obj} * L_{i,no\_obj} \tag{19}$$

gde je prva suma u funkciji gubitka deo opisan sintagmom “prvi deo funkcije gubitka”, a druga suma pomnožena prametrom lambda “drugi deo funkcije gubitka”, $L_{i,obj}$ funkcija

gubitka za ćelije koje sadrže neki objekat, $L_{i,no\_obj}$ funkcija gubitka za ćelije koje ne sadrže objekat. Kada je reč o gubitku za ćelije koje sadrže objekat, njihov gubitak jednak je zbiru gubitaka za koordinate okvirne kutije, gubitaka za vrednosti skora poverljivosti i gubitka za samu klasifikaciju. Prvom gubitku (gubitak koji se odnosi na koordinate okvirnih kutija) dodat je koeficijent koji dodatno povećava težinu ovog gubitka (pri treningu YOLO modela korišćena je vrednost pet). [2], [25]

$$L_{i,obj} = \lambda_{coord} * L_{i,obj}^{box} + L_{i,obj}^{conf} + L_{i,obj}^{cls} \tag{20}$$

Gubitak okvirne kutije za ćeliju koja sadrži objekat može se izračunati kao

$$L_{i,obj}^{box} = (\Delta x_i^* - \Delta \hat{x}_i)^2 + (\Delta y_i^* - \Delta \hat{y}_i)^2 + (\sqrt{\Delta w_i^*} - \sqrt{\Delta \hat{w}_i})^2 + (\sqrt{\Delta h_i^*} - \sqrt{\Delta \hat{h}_i})^2 \tag{21}$$

pri čemu se vrednosti anotirane pomoću "^" odnose na predikcije modela, dok se vrednosti anotirane pomoću "*" odnose na stvarne vrednosti za koordinate kutije. Na Δw, i Δh vrednosti dodat je kvadratni koren zato što oni mogu brojčano varirati u odnosu na vrednosti Δx i Δy, koje su mnogo manje. [2], [25]

Gubitak za ocene poverenja ćelija koje sadrže objekat može se izračunati kao kvadrat razlike između vrednosti stvarne ocene poverenja i predviđene ocene poverenja:

$$L_{i,obj}^{conf} = (c_i^* - \hat{c}_i)^2 \tag{22}$$

Ako je predviđena vrednost poverenja 0.9, to znači da je model 90% siguran da je u ćeliji objekat. [2]

Klasifikacioni gubitak može se izračunati kao suma kvadrata razlike stvarnih vrednosti i modelom predviđenih verovatnoća za sve klase. [2]

$$L_{i,obj}^{cls} = (p_{i,1} - \hat{p}_{i,1})^2 + \cdots + (p_{i,14} - \hat{p}_{i,14})^2 + \cdots + (p_{i,20} - \hat{p}_{i,20})^2 \tag{23}$$

Gubitak ćelija koje ne sadrže objekte izračunava se kao kvadrat razlike između ocene objektivnosti ciljne i modelom predviđene vrednosti. Za ciljnu vrednost, ova ocena ima uvek vrednost nula (zato što se u ćelijama ne nalaze objekti). Ovaj gubitak opisan je formulom [2]

$$\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{i,j}^{noobj} (C_i - \hat{C}_i)^2 \tag{24}$$

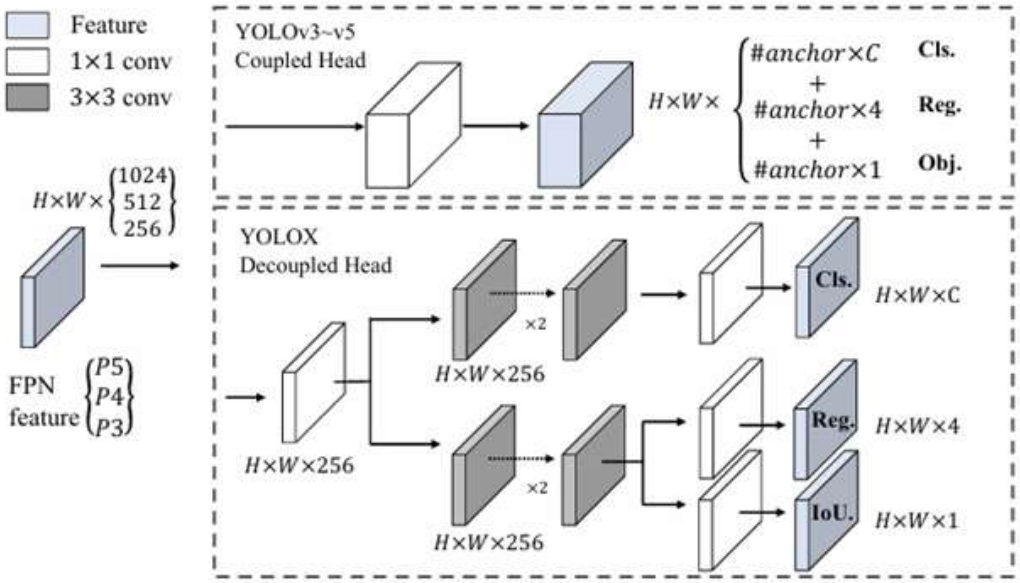
YOLOX model uvodi dva ključna poboljšanja: (1) eliminaciju predefinisanih kutija i prelazak na detekciju bez kutija sa prethodno postavljenim koordinatama, što pojednostavljuje proces predikcije i smanjuje kompleksnost modela, jer uklanja potrebu za ručnim podešavanjem parametara ovih kutija, čime model direktno predviđa pozicije objekata; (2) predikciju objekata kroz višestruke skale mreže, kako bi bolje detektovao objekte različitih veličina. Kada se na slici nalaze objekti raznih dimenzija, model koristi tri različite skale predikcije (npr. 80 x 80, 40 x 40, i 20 x 20 za slike od 640 x 640 piksela), što omogućava preciznije detekcije malih, srednjih i velikih objekata. [1]

U originalnom radu koji predstavlja YOLOX model, autori napominju da deo modela za detekciju granica objekata i njihovu klasifikaciju integrisan u jednu komponentu modela, tzv. spojena detekciona glava može narušiti performanse modela. U YOLOX verziji modela, glava koja se odnosi na deo mreže koji vrši predikcije klasa objekata i lokacije granica odvojene su u različite podsisteme modela. Ovaj pristup ubrzava konvergenciju, model uči efikasnije i dostiže zadovoljavajuće performanse u kraćem vremenskom okviru. Prema tabeli u kojoj su prikazani rezultati merenja AP metrike na COCO skupu podataka, razdvajanje glava korisno je za održavanje visokih performansi; pri arhitekturi modela sa spojenom glavom performanse opadaju za 4.2% AP, dok pristupom sa odvojenim glavama pad u performansama biva značajno manji, svega 0.8% AP. [1]

Tabela 1: Pad u vrednosti AP metrike na COCO skupu podataka pri poređenju pristupa *Coupled Head* i *Decoupled Head* [1]

| Model | Coupled Head | Decoupled Head |
|---|---|---|
| Vanilla YOLO | 38.5% | 39.6% |
| End-to-end YOLO | 34.3% (-4.2) | 38.8% (-0.8) |

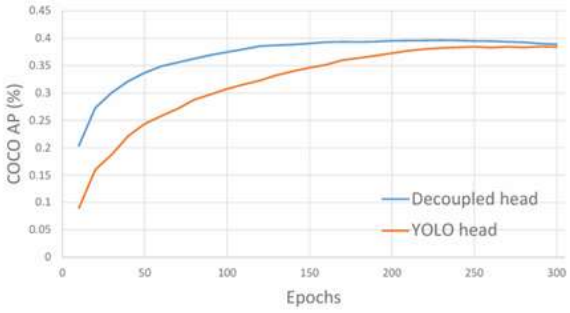
Zato je u novom YOLOX modelu glava za detekciju razdvojena, sadrži 1 x 1 konvolucioni sloj koji redukuje dimenziju kanala atributa na 256 i praćen je paralelnim slojevima sa dva 3 x 3 konvoluciona sloja, za zadatke klasifikacije i regresije, respektivno. Na particiju koja je zadužena za regresioni zadatak, dodata je IoU grana. [1]




**Slika 6.** Vizuelni prikaz arhitektura YOLOv3-v5 koje imaju implementiran *coupled head* mehanizam i YOLOX modela sa *decoupled head* implementacijom [1]

Na slici su prikazana dva pristupa, za YOLOv3 – v5 arhitekture korišćen je pristup sa zajedničkom glavom, dok je za YOLOX arhitekturu korišćen pristup sa odvojenim glavama, koji odvaja zadatke klasifikacije (predviđanje klase objekta) i regresije (detekcija okvirnih kutija objekta). Mreža piramide atributa *(FPN)* je arhitektura dubokih neuronskih mreža koja se koristi za ekstrakciju višeslojnih karakteristika iz slike, pri čemu se detektoru

obezbeđuju informacije sa različitih nivoa rezolucije slike, što pomaže u detekciji objekata različitih veličina. Niži nivoi sadrže fine detalje visoke rezolucije, dok viši nivoi sadrže globalne informacije sa niskom rezolucijom. Za svaki nivo FPN – a usvaja se 1 x 1 konvolucioni sloj za redukovanje dimenzionalnosti. [26], [1]




**Slika 7.** Konvergencija modela tokom treninga upotrebom *coupled head* i *decoupled head* pristupa [1]

Mereno na COCO validacionom skupu podataka, na svakih 10 epoha, očigledni su rezultati da pristup sa odvojenim glvama konvergira mnogo brže nego arhitektura koja se bazira na zajedničkoj glavi i na kraju postiže bolje rezultate. [1]

Strategije augmentacije podataka koje su dodate pri treningu YOLOX modela su *Mozaik* i *Strategija mešanja podataka. Mozaik* kombinuje četiri nasumično odabrane slike trening skupa podataka, koje se kombinuju u jednu novu sliku, koja se sastoji od četiri kvadranta, pri čemu svaka četvrtina dolazi od jedne od izabranih slika. *Strategija mešanja podataka* kombinuje dve nasumično odabrane slike iz skupa podataka i linearno ih kombinuje ne bi li se stvorila nova slika. Ako su date dve slike $x_i$ i $x_j$ sa labelama $y_i$ i $y_j$, tada se nova slika $x'$ i labela $y'$ dobija kao po sledećim formulama, pri čemu je lambda slučajna vrednost iz beta distribucije B(α, α), pri čemu α kontroliše težinu mešanja. [1] Objašnjeni pristup opisan je formulama

$$x' = \lambda * x_i + (1 - \lambda) * x_j \tag{25}$$

$$y' = \lambda * y_i + (1 - \lambda) * y_j \tag{26}$$

Mehanizam prethodnog određivanja koorrdinata kutija ima nekoliko problema: (1) da bi se sproveo, potrebno je sprovesti analizu klastera kako bi se odredio set optimalnih kutija pre treninga modela, koji su manje generalni i domenski specifični; (2) mehanizam povećava kompleksnost detekcionih glava, broj predikcija po slici, kao i latenciju pri prenosu velikih količina predikcija sa NPU na CPU, što rezultuje kašnjenjima u sistemu i padu u efikasnosti. Mehanizam koji ne koristi unapred definisane tačke prilikom predikcije objekata, pri čemu se broj parametara koji se moraju konfigurisati i optimizovati smanjuje olakšava dizajn YOLOX modela. [1]

SimOTA, kao napredna strategija dodele oznaka u detekciji objekata, predstavlja značajan napredak u oblasti optimizacije dodele predikcija i stvarnih vrednosti. Ova metoda prvo izračunava međusobnu povezanost između predikcija i stvarnih vrednosti, koristeći meru gubitka ili kvaliteta za svaku par predikcija - stvarna vrednost. [1] Konkretno, gubitak između stvarne vrednosti $g_i$ i predikcije $p_j$ definiše se kao suma klasifikacione i regresione gubitničke funkcije, odnosno

$$c_{ij} = L_{cls} + L_{reg} \tag{27}$$

Lambda predstavlja koeficijent ravnoteže. U SimOTA, za svaku stvarnu vrednost $g_i$ biramo top-k predikcija sa najmanjim gubicima unutar unapred definisanog centralnog regiona, što omogućava selekciju pozitivnih uzoraka. Ćelije slike koje odgovaraju tim pozitivnim predikcijama se dodeljuju kao pozitivni, dok ostali postaju negativni uzorci. Ova strategija takođe uvodi dinamičan broj pozitivnih ankora, što se naziva dinamički top-k, a k varira u zavisnosti od specifične stvarne vrednosti. SimOTA se temelji na principima optimalnog transporta (OT), omogućavajući efikasniju dodelu oznaka koja je svesna gubitaka i kvaliteta, uz prioritet centra i globalni pregled. Iako je korišćenje Sinkhorn-Knopp algoritma značajno poboljšalo performanse, povećavajući vreme treninga za 25% tokom 300 epoha, SimOTA pojednostavljuje ovaj proces izbacujući dodatne hiperparametre koji su potrebni za rešavanje OT problema. Kao rezultat, SimOTA je poboljšala tačnost detektora sa 45,0% AP na 47,3% AP, nadmašujući SOTA performanse ultralytics-YOLOv3 za 3,0% AP, čime se potvrđuje snaga ove napredne strategije dodele oznaka. [1]

### 2.1.2 Implementacija i parsiranje u Python – u

Za parsiranje pdf dokumenta korišćena je biblioteka unstructured, koja se koristi za particionisanje nestrukturiranih dokumenata poput PDF, HTML, Word dokumenata i fino podešavanje modela. [15] PDF dokument je transformisan u listu elemenata klase *Element* upotrebom metode *partition_pdf* paketa *unstructured.partition.pdf.* Ova metoda kao parametre prima: (1) *strategy* – strategija particionisanja PDF dokumenta, gde su validne strategije pri parsiranju *hi_res, ocr_only* i *fast* strategija. *Hi_res* strategija koristi model za detekciju rasporeda elemenata kako bi identifikovala elemente dokumenta. Kada je reč o *ocr_only* strategiji, vrši se ekstrakcija teksta iz dokumenta pomoću OCR (*Optical Character Recognition*) tehnologije, koja omogućuje prepoznavanje i konvertovanje teksta iz slika ili skeniranih dokumenata u digitalni, mašinski čitljiv tekst. Jedan od najpoznatijih alata za OCR je *Tesseract,* koji je solucija otvorenog koda i podržan je u *Python* - u. *Fast* strategija je korišćena za direktno izdvajanje teksta iz PDF dokumenta; (2) *infer_table_structure* – *boolean* indikator koji omogućuje ekstrakciju tabela iz PDF dokumenta i njihovo mapiranje u element klase *Table*. Ovaj parametar obezbeđuje polje metapodataka pod nazivom *text_as_html*, koji omogućuje modelu da razume strukturu tabele kreirajući njenu reprezentaciju u obliku HTML tagova (veliki jezički modeli trenirani su na HTML podacima). Ukoliko je vrednost ovog parametra postavljena na *True*, onda je obavezna upotreba *hi_res* strategije particionisanja PDF dokumenta; (3) *extract_images_in_pdf* – parametar koji ima Boolean vrednosti i podešen je na *True* ukoliko je zahtevana ekstrakcija slika iz PDF dokumenata. Tada je obavezno koristiti *hi_res* strategiju particionisanja; (4) *model_name* – parametar prima naziv modela koji se koristi za detekciju objekata, a vrednosti koje prima su *yolox* i *detectron2_onnx.* [15], [16] Pri particionisanju PDF dokumenta u radu je korišćen opisani YOLOX model.

```
In [ ]: elements = partition_pdf(
            filename='../data/ISiP.pdf',  # putanja do pdf fajla
            strategy="hi_res", # strategija za obradu pdf dokumenta
            infer_table_structure=True, # prepoznaje da u dokumentu postoji tabela i strukturira je
            model_name="yolox", # model koji cemo koristiti za prepoznavanje i analizu objekata u slikama
            extract_images_in_pdf=True, # omogucuje ekstrakciju slika iz pdf - a u folder figures/
        )
```

**Slika 8.** Implementacija koda za particionisanje PDF fajla uz upotrebu YOLOX modela, *hi_res* strategije uz paralelnu ekstrakciju tabelarnih podataka i slika [27]

Parser nije dobro identifikovao sve elemente PDF dokumenta. Na primer, pogrešno je identifikovao Header i Footer elemente kao elemente klase ListItem i slično. Ove greške u parsiranju dovele su do toga da je kontekst tekstualnih elemenata koji su rezultat procesa podele teksta u segmente narušen jer je kod paragrafa koji se prelamaju na više strana narušen kontekst njihovim odvajanjem zbog Header i Footer elemenata koji su narušili njihov semantički kontekst. O rešenju ovog problema biće reči kasnije.

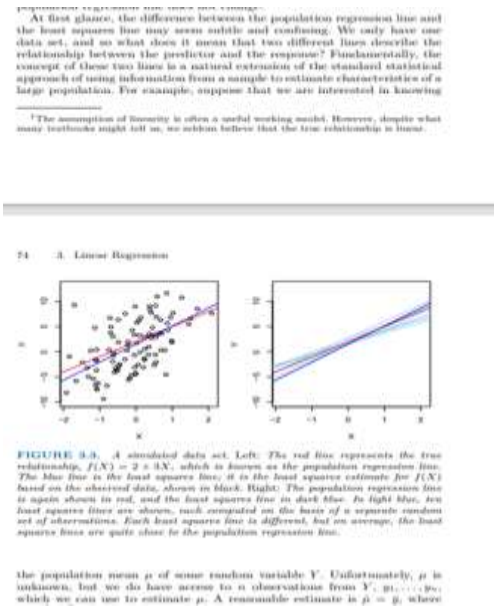
Kada je reč o načinima podele dokumenta, korišćene su različite strategije deobe originalnog PDF fajla. Funkcija *chunk_by_title* koristi se za deljenje dokumenta na sekcije i to na osnovu naslova ili pri promeni metapodataka (pri promeni broja strane). Sekcija se prekida kada broj karaktera postane veći od *max_characters*. Metoda prima listu elemenata i omogućuje dodatna prilagođavanja parametara: (1) *elements* - lista elemenata koji se obrađuju, obično dobijena iz neke funkcije za particionisanje dokuemenata (na primer *partition_pdf*); (2) *combine_text_under_n_chars* - parametar određuje koliko kratki delovi teksta (na primer, serije naslova) mogu biti spojeni zajedno dok ne dosegnu određenu dužinu u karakterima. Ako je vrednost parametra 0, onda se onemogućuje spajanje malih delova. Ako se specificira, spajanje će se vršiti do tog broja karaktera, u suprotnom spajanje se vrši do *max_characters; (3) max_characters* - maksimalna dužina teksta u karakterima za svaki novi deo tj. sekciju. Ako tekst prelazi ovu dužinu, on će biti podeljen; (4) *new_after_n_chars* - definiše dužinu sekcija nakon koje će se započeti nova sekcija. Ako nije postavljen, njegova vrednost je *max_characters*. Ako je 0, svaki element biće samostalna sekcija, iako se tekst duži od *max_characters* deli na više sekcija; (5) *include_orig_elements* - ako je true, originalni elementi pre deljenja biće uključeni u metapodatke novih delova (sekcija) koje se formiraju. Ovo onemogućuje pristup metapodacima kada se oni gube nakon deljenja. (6) *multipage_sections* - true, sekcije mogu da prelaze više stranica, u suprotnom je false; (7) *overlap* - dužina niza karaktera koji će biti prekopiran iz prethodnog dela i dodat na početak sledećeg dela kako bi se očuvao i povezao kontekst chunkova. Ovo se primenjuje samo na deljenje tekstova koji su predugi i podeljeni su na više delova; (8) *overlap_all* – za vrednost true, primenjuje preklapanje između normalnih delova koji nisu podeljeni tekstualno, treba ih koristiti oprezno jer oni narušavaju granice sekcija. [15], [16]

Pri particionisanju .pdf dokumenta kao što je knjiga, praksa pokazuje da se najčešće deoba vrši strategijom *chunk_by_title,* imajući u vidu da je ona strukturisana po naslovima i

podnaslovima do odgovarajućeg nivoa, a oni samim tim čine jednu semantičku celinu. Naslovi su pri particionisanju odvojeni "\n\n" separatorom, kao i znak za novu stranicu. Problem u separatisanju ove knjige *unstructured* parserom jeste pogrešno parsiranje Header i Footer elemenata kao elemenata klase ListItem na prelasku između dve stranice, u situacijama kada pasus nije završen na kraju jedne stranice, već se nastavlja na drugoj. Tada se drugi deo gubi prilikom podele na segmente. Zato bi uklanjanje ovih elemenata pomoglo u parsiranju dokumenta.

### 2.1.3 Klasifikacija Header i Footer elemenata

Kao što je već naznačeno, tokom parsiranja PDF fajla pojavio se problem pogrešnog parsiranja Header i Footer elemenata u ListItem elemente, kao i parsiranje ostalih *unstructured* elemenata u instance klasa Header ili Footer, što je učinilo transformaciju originalnog teksta u chunkove neispravnom, pa instanca pretraživača nekada nije mogla da pronađe odgovarajući pasaž teksta, u situacijama kada se jedan paragraf ne završava na kraju jedne strane, već prelazi i na sledeću stranu, a između njih je Footer element koji je parser prepoznao kao ListItem element. Na primer, u isečku teksta na slici broj sedam, unstructured parser je prepoznao tekst na vrhu stranice, Footer koji je pogrešno prepoznat kao ListItem, dat sa "[1]The assumption of linearity is often a useful working model. However, despite what many textbooks might tell us, we seldom believe that the true relationship is linear." pa Header dat sa "74 3. Linear Regression", koji je takođe prepoznat kao ListItem, koji je ispraćen opisom slike "FIGURE 3.3 …", a zatim nastavkom paragrafa. Povezivanjem unstructured elemenata u CompositeElement, između povezanih tekstualnih elemenata bili bi umetnutni tekst Footera, Headera i opis slike, što je onemogućilo ispravnu pretragu. U nekim situacijama, javio bi se problem podele ovog sadržaja na dva chunk - a, od kojih prvi sadrži prvi deo pasusa i Footer, a drugi sadrži Header i drugi deo pasusa.



**Slika 9.** Primer iz podataka u kojem se jedan pasus ne završava na jednoj strani, prva stranica sadrži fusnotu, Header, sliku i opis slike, nakon čega se nastavlja započeti pasus [28]

Kao jedno od rešenja ovog problema koje je dalo zadovoljavajuće rezultate bilo je primena XGBoost modela za klasifikaciju elemenata na osnovu njihovog sadržaja u one koji će biti izbrisani pre kreiranja instance CompositeElement klase (pre procesa podele teksta na segmente) i ostale koji će biti zadržani. Kako je unstructured parser pogrešno prepoznao većinu Header / Footer elemenata kao ListItem elemente, onda ćemo za obuku modela iskoristiti sve ListItem, Header i Footer elemente.

Skup podataka sastoji se od tri kolone: tekst – tekstualni sadržaj parsiranog elementa; *content_type* – tip elementa koji je prepoznao unstructured parser, *junk* – indikator koji opisuje da li je tekstualni element prepoznat kao "junk" (pogrešno klasifikovani Header / Footer elementi prepoznati kao ListItem elementi) ili je regularni ListItem element (ListItem instance koje su ispravno klasifikovane). Kolonu *content_type* izdvojili smo samo da bismo uvideli kako je parser prepoznao odgovarajući tekstualni element, dok će se tokom treninga koristiti samo *text* i *junk* kolone.

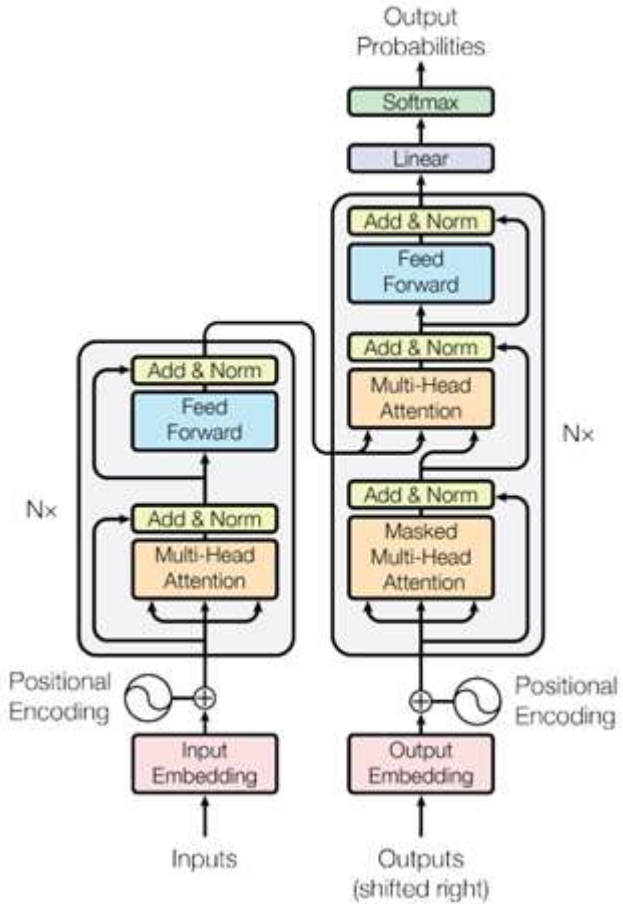
| text | content_type | junk |
|---|---|---|
| viii Preface | Header | yes |
| 1 Introduction | ListItem | no |
| x Contents | ListItem | yes |
| 3.2.2 Some Important Questions. ..... 83 | ListItem | no |
| Contents xi | ListItem | yes |
| xii Contents | ListItem | yes |
| Contents xiii | Header | yes |
| xiv Contents | Header | yes |
| Contents xv | Header | yes |
| 2 1. Introduction | Header | yes |
| 1. Introduction 3 | ListItem | yes |
| 4 1. Introduction | Header | yes |
| 1. Introduction 5 | ListItem | yes |
| 6 1. Introduction | Header | yes |
| 1. Introduction 7 | ListItem | yes |
| 1. Many statistical learning methods are relevar | ListItem | no |

**Slika 10.** Skup podataka za klasifikaciju Header, Footer i ListItem elemenata u klase *yes, no* kolone *junk*

Da bi XGBoost klasifikator bio istreniran, potrebno je pretvoriti tekst u vektorske reprezentacije, koje mogu da identifikuju sintaksne i semantičke veze u tekstu. Primena jednostavnih transformacija teksta u vektore pomoću *OHE* – a neće biti dovoljna za identifikaciju veza između reči. Vektorske reprezentacije koje su dobijene treningom neuronskih mreža svaku tekstualnu sekvencu predstavljaju kao vektor u kojem svaka dimenzija može predstavljati sintaksnu ili semantičku karakteristiku. Za kreiranje vektorskih reprezentacija teksta koristićemo *all-MiniLM-L6-v2* model biblioteke *sentence-transformers*, gde se svaki tekst predstavlja kao vektor od 384 atributa koji ga opisuju, a sličnost između vektora može da se opiše pomoću distance između dve tačke 384 – dimenzionog prostora. Ovaj model koristi pojednostavljenu verziju transformer arhitekture kako bi postigao balans između efikasnosti i performansi, što ga čini pogodnim za primene koje zahtevaju brzo procesiranje teksta, Koristi mehanizam pažnje, što omogućuje modelu da obrađuje reči u kontekstu ostalih reči u rečenici. Baziran je na BERT – arhitekturi, koja je jedan od najpoznatijih tipova transformera, ali ima šest slojeva, odakle dolazi L6 oznaka, što je manje u poređenju sa BERT modelom, koji ima dvanaest slojeva. Takođe, koristi vektorske reprezentacije sa 384 dimenzije, što je manje od standardnih 768 dimenzija BERT-base modela. [29]

### 2.1.4 Kratak pregled transformer arhitekture

Arhitektura modela zasniva se na enkoder – dekoder strukturi. Uopšteno, enkoder mapira ulaznu sekvencu simbola $x = (x_1, ..., x_n)$ u sekvencu kontinualnih reprezentacija $z = (z_1, \ldots, z_n)$ - vektore realnih brojeva. Za ovakvu reprezentaciju z, dekoder generiše izlaznu sekvencu $y = (y_1, \ldots, y_m)$ simbola, jedan po jedan, pri čemu je model auto - regresivan, koristeći prethodno generisane simbole kao dodatni ulaz prilikom generisanja sledećeg simbola. [3]




**Slika 11.** Arhitektura transformer modela [3]

Enkoder je sastavljen od 6 identičnih slojeva, pri čemu svaki sloj ima dva podsloja: (1) mehanizam višestruke samopažnje: omogućuje modelu da obradi različite delove ulazne sekvence paralelno, pri čemu se procesiranje vrši u više "glava", što modelu omogućuje da istovremeno uči različite aspekte i odnose između reči ili simbola na ulazu, što pomaže pri fokusiranju na relevantne informacije iz cele sekvence; (2) potpuno povezana jednosmerna,

unapred prosleđujuća, neuronska mreža: jednostavna neuronska mreža koja se primenjuje na svaki token sekvence pojedinačno, pri čemu svaki token prolazi kroz istu mrežu. Oko svakog podsloja koristi se rezidualna veza, koja omogućuje da se informacije iz ulaza prenesu direktno do izlaza podsloja, pomažući modelu da izbegne problem gubitka informacija tokom obrade. Kada se rezultat podsloja (Podsloj(x)) doda ulazu x, mreža može lakše da uči jer se fokusira na razlike između ulaza i izlaza umesto na učenje svega ispočetka. Radi stabilizacije učenja i ubrzane konvergencije, vrši se normalizacija sloja. Uopšteno, izlaz iz svakog podsloja dat je sa *SlojNormalizacije(x + Podsloj(x))*, gde je Podsloj(x) funkcija implementirana od strane podsloja. Svi podslojevi u modelu imaju dimenziju 512, za sve izlaze modela. [3]

Dekoder je sastavljen od šest identičnih slojeva, međutim u okviru svakog podsloja postoji i treći podsloj, koji primenjuje mehanizam višestruke pažnje nad izlazom iz enkodera. To znači da dekoder može da pristupa informacijama koje je enkoder generisao. Ovaj mehanizam omogućuje dekoderu da se fokusira na relevantne delove ulazne sekvence dok generiše izlaz. Na primer, kada dekoder generiše reč, on ima uvid u ceo ulaz koji je obradio enkoder i uzima u obzir relevantne delove, što poboljšava tačnost i koherentnost izlaza. Slično kao i u arhitekturi enkodera, koriste se rezidualne veze oko svakog od podslojeva, praćene normalizacijom sloja. Takođe, modifikuje se podsloj samopažnje u dekoderu ne bi li se sprečilo da tekuće pozicije obraćaju pažnju na sledeće pozicije. Ovo maskiranje u kombinaciji sa činjenicom da su izlazne reprezentacije pomerene za jednu poziciju osigurava da predikcije za poziciju zavise samo od poznatih izlaza na poziciji manjoj od tekuće. [3]

Pre nego što ulazna sekvenca pristupi enkoderu, vriši se apsolutno poziciono enkodiranje, čija je primarna svrha da modelu pruži informacije o redosledu elemenata u sekvenci. Metod pozicionog enkodiranja sinusnom funkcijom generiše jedinstveni vektor za svaku poziciju u sekvenci koristeći sinusne funkcije. Ovaj vektor ima iste dimenzije kao i vektorska reprezentacija tokena, što omogućuje njihovo jednostavno sabiranje ne bi li se spojile informacije o poziciji i značenju. Za svaki token u ulaznoj sekvenci definišu se vrednosti koje odgovaraju vrednostima sinusne funkcije različitih perioda. Na primer, za sve tokene jednog ulaza prvi realni broj u vrednosti vektorskih reprezentacija biće definisan zajedničkom sinusnom funkcijom, tako da će postojati različite vrednosti pozicionih

enkodiranja. Na isti način definisaće se i vrednosti pozicionog enkodiranja za realne brojeve na ostalim pozicijama za svaki token, samo što će sinusna funkcija imati drugačiju periodu. Kako su sinusna i kosinusna funkcija periodične, moguće je da će neki tokeni imati iste vrednosti pozicionog enkodiranja. Ove vrednosti uvek su iz opsega od -1 do 1. Svaka ulazna sekvenca uvek će imati jedinstveni skup vrednosti za vrednosti pozicionog enkodiranja. [30] Najčešći pristup je upotreba sinusnih i kosinusnih funkcija različitih frekvencija:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}) \tag{28}$$

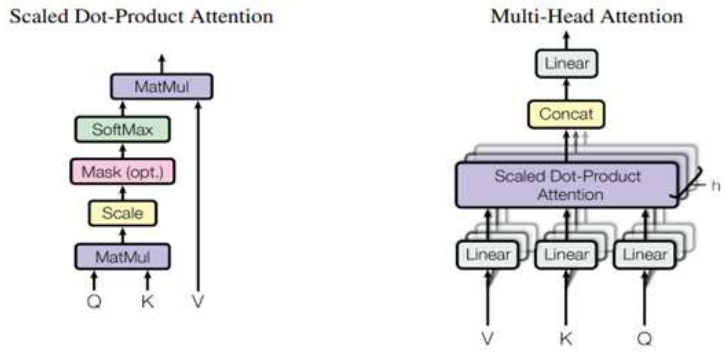
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}}) \tag{29}$$

gde je sa $d_{model}$ označena dimenzija skrivenog sloja u mreži enkodera, iznosi 768 za BERT base model. [3]

Mehanizam samopažnje izračunava sličnost svake reči u rečenici sa drugim rečima u rečenici i sa samom sobom. Ako neka reč ima veću metriku sličnosti sa tekućom rečju za koju se izračunava vektorska reprezentacija, onda će samim tim ta reč imati veći uticaj u enkodiranju tekuće reči u vektorsku reprezentaciju. Uobičajeno, vrednosti pažnje se računaju simultano na setu "upit" vrednosti smeštenih u matricu Q, dok su K i V matrice "ključ" i "vrednost" reprezentacija. Ove matrice predstavljaju različite vektorske reprezentacije početne vektorske reprezentacije odgovarajućeg tokena u arhitekturi enkodera nad kojima se primenjuje mehanizam samopažnje, o čemu će biti reči kasnije. Oznakom $d_k$ biće predstavljena dimenzija *"*ključ*"* vektorske reprezentacije. Za dugačke sekvence teksta, dimenzija $d_k$ postaje velika, isti efekat odražava se na skalarni proizvodi dva vektora, pa softmax funkcija postaje veoma osetljiva pri čemu neki izlazi iz ove funkcije postaju veoma mali, što rezultuje malim vrednostima izvoda pri ažuriranju težina čime se usporava ili zaustavlja učenje. Zato se vrši skaliranje vrednosti skalarnog proizvoda. [30] Matrica izlaza data je sa

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{30}$$

[3] Svaki realan broj vektorske reprezentacije jednog tokena množi se sa odgovarajućim težinama, pa se nove vrednosti sabiraju. Nakon sabiranja, nove vektorske reprezentacije tokena imaće istu dimenziju kao početne reprezentacije. Nove vrednosti vektorskih reprezentacija poznatije su u literaturi kao ”upit” vrednosti. Nad istim tokenom i ostalim tokenima koristi se drugi skup težina nad kojim se ponavlja isti proces i dobijaju se “ključ” i “vrednost” reprezentacije. Ove vrednosti koriste se ne bi li se izračunala sličnost između reči čija je vektorska reprezentacija označena sa “upit” i svih ostalih reči čije su vektorske reprezentacije označene sa “ključ”. Originalni token čija se sličnost sa ostalim tokenima ispituje ima i “ključ” i “vrednost” reprezentaciju, zato što mehanizam samopažnje podrazumeva da se izračunava i sličnost odgovarajuće reči sa samom sobom. Metrika sličnosti koja se koristi za poređenje između reči je skalarni proizvod ili neka druga metrika. Za prevođenje sličnosti između odgovarajućih tokena koristi se softmax funkcija, koja sličnosti skalira na vrednosti od nula do jedan. Na primer, ako je sličnost tekućeg tokena sa nekim drugim nakon transformacije softmax funkcijom jednaka jedinici, to znači da će se koristiti 100% tog tokena za enkodiranje tekućeg. Početne vektorske reprezentacije uvećane za vrednosti pozicionog enkodiranja množe se sa odgovarajućim težinama, dobijene vrednosti se sabiraju i tako se dobijaju nove vektorske reprezentacije tokena (tzv. “vrednost” reprezentacije) koje se množe sa izlazima iz softmax funkcije. Konačno, dobijene vrednosti nazivaju se vrednosti samopažnje za tekući token. Iako nakon primene mehanizma samopažnje dimenzije vektorskih reprezentacija ostaju iste kao i pre, novonastale vrednosti obuhvataju i informaciju o vrednostima svih ostalih vektora tokena, što tekućoj reči daje kontekst. Povezivanjem blokova samopažnje dobija se sloj višestruke samopažnje (najčešće osam blokova samopažnje). Sabiranjem prvobitnih vektorskih reprezentacija koje su uvećane za vrednosti pozicionog enkodiranja sa vrednostima samopažnje dobijaju se vrednosti rezidualnih veza, pomoću kojih je trening neuronskih mreža jednostavniji zato što mreža neće učiti od početka sve težine, nego će da se fokusira na razlike između izlaza iz transformacije mehanizma samopažnje i vektorskih reprezentacija tokena uvećanih za poziciono enkodiranje. [30] Prikazani grafici na sledećoj slici ilustruju primenu gore pomenutih transformacija.




**Slika 12.** Prikaz višestrukog mehanizma samopažnje [3]

Mehanizam višestruke samopažnje omogućuje da svaki sloj koji ga čini obrađuje različite aspekte ulaza, što je pogodno za bogatije i raznovrsnije predstavljanje ulaznih informacija. Upotrebom samo jednog sloja dolazi do uprosečavanja i gubljenja informacija, dok kod višeslojnog paralelnog pristupa omogućuje se da se različiti aspekti informacija ulazne sekvence zadrže. Kod višestrukog mehanizma samopažnje dimenzije svakog sloja su smanjene ($d_k = d_v = d_{model}/h$, gde je h broj paralelnih slojeva; u originalnom radu korišćeno je osam paralelnih slojeva, što čini dimenzije odgovarajućih $d_k$ i $d_v$ jednakim 64, imajući u vidu da je dimenzija modela 512). To znači da ukupni računski troškovi za ovakav pristup ostaju jednaki troškovima primene jednog sloja sa punom dimenzionalnošću, sa istovremenim povećanjem sposobnosti modela. Formula koja opisuje mehanizam višestruke samopažnje data je sa [3]

$$MultiHead(Q,K,V) = Concat(head_1, \dots, head_h)W^O \tag{31}$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{32}$$

U arhitekturi dekodera kod rečnika tokena prvo se određuju vektorske reprezentacije kojima se dodaju poziciono enkodirane vrednosti, koje su kreirane istim sinusnim i kosinusnim funkcijama kao kod enkodera. Ovako dobijene vrednosti množe se sa odgovarajućim težinama i sabiraju ne bi li se dobile nove vektorske reprezentacije istih dimenzija kao početne, "ključ", "vrednost", "upit" reprezentacije koje se koriste za izračunavanje vrednosti mehanizma samopažnje. Set težinskih koeficijenata dekodera razlikuje se u

odnosu na set težinskih koeficijenata enkodera. Dodavanjem vrednosti samopažnje vektorskim reprezentacijama uvećanim za vrednosti pozicija reči dobijaju se vrednosti rezidualnih veza. Međutim, u arhitekturi dekodera postoji mehanizam koji održava zavisnost izlazne sekvence od ulazne sekvence, koji se naziva Enkoder – Dekoder pažnja.. On podrazumeva izračunavanje sličnosti između jednog tokena (za token dekodera izračunata je nova vektorska reprezentacija koja se naziva "upit" vrednostima) u dekoderu sa svim tokenima u enkoderu (za svaki token enkodera kreirane su nove vektorske reprezentacije koje se nazivaju "ključ" vrednosti). Metrika koja opisuje sličnost "upit" vrednosti sa svim "ključ" vrednostima tokena u enkoderu jeste skalarni proizvod ili neka druga metrika sličnosti dva vektora, nakon čega skalarne vrednosti prolaze transformaciju kroz *softmax* funkciju. Bitno je reći da se "ključ" vrednosti za sve tokene u enkoderu računaju tako što se vrednosti rezidualnih konekcija množe sa novim težinama i sabiraju tako da novi vektor ima iste dimenzije kao vrednosti rezidualnih veza. Vrednosti koje predstavljaju izlaze iz softmax funkcije opisuju u kolikom procentu tokeni u enkoderu učestvuju u određivanju novih vektorskih reprezentacija tokena u dekoderu i na ovaj način održava se zavisnost ulazne sekvence sa izlaznom. Množenjem "vrednost" reprezentacija svih tokena u arhitekturi enkodera sa izlazima iz softmax funkcije dobijaju se vrednosti pažnje između enkodera i dekodera, koje održavaju ove zavisnosti. Set težina koji se koristi za izračunavanje ovih vrednosti je drugačiji od seta težina koji se koristi za određivanje vrednosti samopažnje. Na ove vrednosti dodaju se početne vektorske reprezentacije tokena dekodera uvećane za poziciono enkodriane vrednosti, koje se nazivaju vrednostima rezidualnih veza dekodera. Poslednja vektorska reprezentacija prolazi kroz potpuno povezani sloj kako bi se odredilo kojem tokenu iz rečnika poznatom dekoderu je tekuća vektorska reprezentacija najsličnija. Potpuno povezani sloj ima ulaznu dimenziju jednaku dimenziji poslednje vektorske reprezentacije tekućeg tokena dekodera, a izlaz ima dimenziju jednaku dužini rečnika dekodera. Ovaj sloj predstavlja jednostavnu neuronsku mrežu sa težinskim koeficijentima i vrednostima slobodnog koeficijenta, koje se dodaju sumi proizvoda realnih brojeva vektorske reprezentacije tekućeg tokena i odgovarajućih težina. Izlazi iz ovovg sloja prosleđuju se softmax funkciji, a odabir odgovarajuće reči iz izlaznog rečnika odgovara maksimalnoj izlaznoj vrednosti iz softmax funkcije. Da bi se uspešno enkodirale i dekodirale dugačke sekvence vrši se normalizacija nakon izračunavanja vektorskih reprezentacija uvećanih za poziciono enkodirane vrednosti i nakon izračunavanja vrednosti samopažnje. [30] Neuronske mreže koje se primenjuju nad

vrednostima rezidualnih konekcija mogu biti složenije, dodavanjem skrivenih slojeva kako bi se vrednosti prilagodile kompleksnijim i dužim sekvencama. Najčešća arhitektura neuronske mreže u transformers arhitekturi podrazumeva potpuno povezanu unapred prosleđujuću neuronsku mrežu koja se sastoji od dve linearne transformacije između kojih je primenjena ReLU aktivaciona funkcija. Ova mreža je opisana formulom [3]

$$FFN(x) = \max(0, xW_1 + b_1)\, W_2 + b_2 \tag{33}$$

### 2.1.5 Primena Transformers arhitekture u problemu klasifikacije

Kao što je već rečeno, za kreiranje vektorskih reprezentacija sekvenci teksta pri zadatku klasifikacije da li je tekstualnu sekvencu potrebno izbrisati (označena je kao junk = yes u skupu podataka) ili ne, korišćen je MiniLM-L6-v2 model koji transformiše input u vektorsku reprezentaciju dimenzije 384. To znači da je svaka sekvenca opisana pomoću 384 realna broja koji odgovaraju 384 različita atributa koji ih opisuju u vektorskom prostoru. Implementacija u Python - u prikazana je na sledećim slikama.


**Slika 13.** Inicijalizacija modela u Jupyter notebook – u [27]

**Slika 14.** Kreiranje vektorskih reprezentacija tekstualnih sadržaja [27]

Za realizaciju zadatka klasifikacije korišćen je XGBoost klasifikator, vršena je optimizacija hiperparametara pomoću GridSearch pretrage.

**Slika 15.** Implementacija XGBoost klasifikatora i optimizacija hiperparametara GridSearch metodom [27]

Podela na trening i test vršena je u razmeri 70:30. Rezultati klasifikacije dati su u sledećoj tabeli:

Tabela 2: Rezultati treninga primenom XGBoost klasifikatora

| **Trening** | precision | recall | f1- score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 712 |
| 1 | 1.00 | 1.00 | 1.00 | 390 |
| Accuracy | | | 1.00 | 1102 |
| Macro avg | 1.00 | 1.00 | 1.00 | 1102 |
| Weighted avg | 1.00 | 1.00 | 1.00 | 1102 |

Tabela 3: Rezultati klasifikacije na testnom skupu podataka primenom XGBoost klasifikatora

| **test** | precision | recall | f1- score | support |
|---|---|---|---|---|
| 0 | 0.98 | 0.97 | 0.98 | 293 |
| 1 | 0.96 | 0.97 | 0.96 | 180 |
| Accuracy | | | 0.97 | 473 |
| Macro avg | 0.97 | 0.97 | 0.97 | 473 |
| Weighted avg | 0.97 | 0.97 | 0.97 | 473 |

Za klasifikaciju korišćen je i *bert-base-uncased* model, koji je dino podešen, čija je implementacija data u ***bert_for_text_classification.py*** fajlu.

Kreirana je instance Dataset klase, koja nasleđuje torch.utils.data.Dataset klasu i stoga je potrebno definisati __getitem__ i __len__ metode:

```python
class Dataset(torch.utils.data.Dataset):

    def __init__(self, encodings, labels=None):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key:torch.tensor(val[idx]) for key, val in self.encodings.items()}
        if self.labels:
            item["labels"] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.encodings["input_ids"])
```

**Slika 16.** Implementacija klase Dataset za skup podataka za trening [27]

Za fino podešavanje modela kreirane su instance Trainer - a i TrainingArguments i sačuvan je finalni model:

```python
def train_and_save_model():
    model_name = "bert-base-uncased"
    tokenizer = BertTokenizer.from_pretrained(model_name)
    model = BertForSequenceClassification.from_pretrained(model_name,num_labels=2)


    dataset = pd.read_csv("../data/junk_classification.csv")
    dataset = dataset.fillna('')
    X = list(dataset['text'])
    y = list(dataset['junk'])
    y = [1 if label == 'yes' else 0 for label in y]
    test_size = 0.3

    X_train, X_val, y_train,y_val = train_test_split(X,y,test_size=test_size,stratify=y)
    X_train_tokenized = tokenizer(X_train, padding=True,truncation=True,max_length=512)
    X_val_tokenized = tokenizer(X_val, padding=True,truncation=True,max_length=512)

    train_dataset = Dataset(X_train_tokenized,y_train)
    val_dataset = Dataset(X_val_tokenized, y_val)


    args = TrainingArguments(
        output_dir="output",
        num_train_epochs=1,
        per_device_train_batch_size=8

    )
    trainer = Trainer(
        model=model,
        args=args,
        train_dataset=train_dataset,
        eval_dataset=val_dataset,
        compute_metrics=compute_metrics
    )
    trainer.train()
    trainer.evaluate()

    model_path = "../code/models/bert_model.pth"
    torch.save(model.state_dict(), model_path)
```

**Slika 17.** Implementacija finog podešavanja BERT modela [27]

Dobijeni su sledeći rezultati klasifikacije:

Tabela 4: Evaluacione metrike na trening i test skupu podataka za BERT base model

| **Bert-base-uncased** | loss | accuracy | precision | recall | f1 score |
|---|---|---|---|---|---|
| trening | 0.0059436522 | 1 | 1 | 1 | 1 |
| test | 0.00912 | 0.98 | 0.98 | 0.98 | 0.98 |

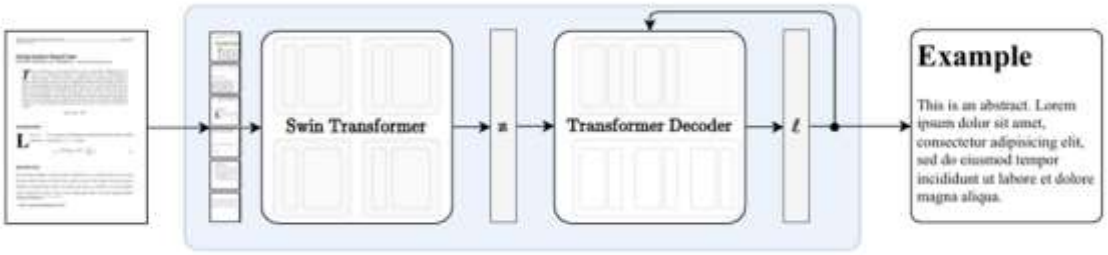
Primena ovog modela korišćena je kao deo logike za eliminaciju pogrešno parsiranih pdf elemenata pre metode za chunk – ovanje teksta u *chunking.ipynb* fajlu:


**Slika 18.** Primena modela u kodu za brisanje stvarnih Header i Footer elemenata [27]

## 2.2 NOUGAT MODEL ZA PARSIRANJE DOKUMENATA

Postojeće OCR tehnologije kao što je Tesseract OCR za optičko prepoznavanje znakova imaju sposobnost klasifikacije različitih simbola ili prepoznavanja šta je napisano na dokumentu, ali ne uspevaju da razumeju odnose između znakova zbog *linija po linija* pristupa obrade na kojem se zasnivaju. Oni tretiraju eksponente i donje indekse kao tekst koji ih okružuje, što je značajan nedostatak kada je reč o parsiranju matematičkih formula, gde postoje razlomci, indeksi, eksponenti, odnosno relacije između karaktera su od značajne važnosti. Postojeće korpusne baze podataka, kao što je S2ORC skup podataka, obuhvataju tekst poreklom iz 12 miliona radova korišćenjem GROBID-a (GeneRation Of Bibliographic Data), ali im nedostaju značajne reprezentacije matematičkih jednačina. GROBID je open – source alat koji se koristi za ekstrakciju i obradu bibliografskih podataka iz naučnih radova i razvijen je kako bi automatizovao proces prepoznavanja i ekstrakcije podataka poput naslova, autora, sažetaka, citata i drugih metapodataka iz PDF dokumenata. On koristi

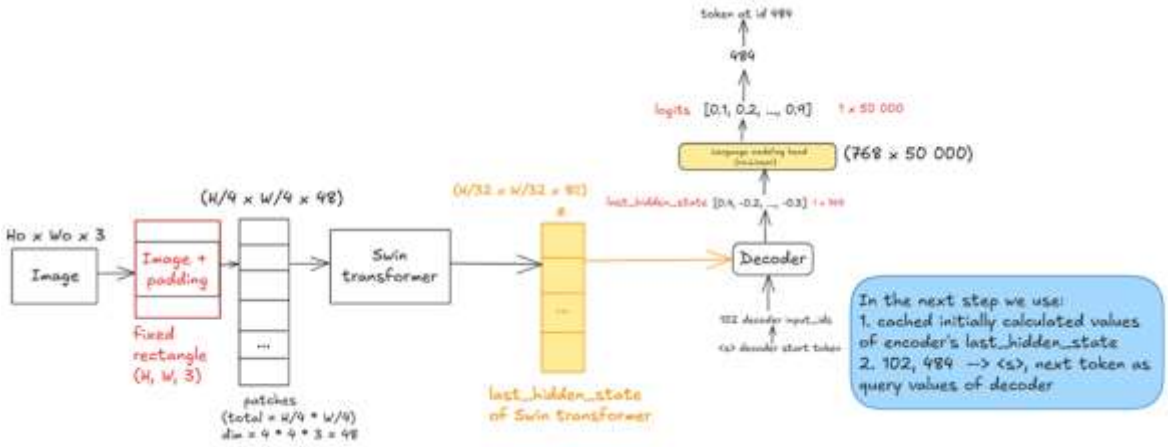
tehnike mašinskog učenja i obrade prirodnog jezika, čime se pomaže istraživačima i institucijama da efikasnije upravjaju bibliografskim informacijama i pomaže pri digitalizaciji i analizi velikog korpusa naučnih radova. Nougat je model koji služi konverziji slika stranica dokumenata u formatirani markup tekst. [19]




**Slika 19.** Arhitektura *Dount* modela [19]

Na slici je prikazana arhitektura koja je zasnovana na Donut – u. Swin Transformer enkoder prihvata sliku dokumenta i pretvara je u "latentne" reprezentacije koje se konvertuju u sekvencu tokena (manjih jedinica teksta) autoregresivno, što omogućuje da se svaki token generiše jedan po jedan, pri čemu svaki novi token zavisi od prethodnog, čime se omogućuje stvaranje koherentnog i smislenog teksta. Swin Transformer je model koji se zasniva na transformer arhitekturi i optimizovan je za obradu slika. Oblast optičkog prepoznavanja znakova (OCR – Optical Character Recognition) je veoma zastupljeno polje istraživanja u kompjuterskoj viziji u kontekstu digitalizacije dokumenata, prepoznavanja rukopisa, prepoznavanja matematičkih izraza i ovi problemi rešavani su upotrebom konvolucionih modela, koji su kasnije zamenjeni RNN dekoder modelima za prepoznavanje rukom pisanih kao i štampanih formula. Kasnije su enkoder i dekoder zamenjeni transformer arhitekturom (attention mehanizam omogućuje bolju obradu složenih struktura kao što su matematički izrazi). VDU (Visual Document Understanding) fokusira se na ekstrakciju relevantnih informacija iz dokumenata različitog tipa tako što modeluje tekst i fizički raspored elemenata koristeći transformer arhitekturu. LayoutLM porodica podela koristi zadatak maskiranog predviđana rasporeda kako bi naučila prostorne odnose između različitih elemenata dokumenta. GROBID, rešenje koje analizira digitalne naučne dokumente u XML fokusirajući se na bibliografske podatke i pdf2htmlEX (konvertuje digitalno stvorene PDF dokumente u HTML zadržavajući raspored i izgled dokumenta) takođe ne mogu da povrate semantičke informacije o matematičkim izrazima. [19]

Kao što je rečeno, Nougat model zasniva se na enkoder – dekoder transformer arhitekturi koja omogućuje end – to – end training proceduru i izgrađen je na Donut arhitekturi. Ovaj model ne zahteva nikakve OCR module ili ulaze, a tekst je implicitno identifikovan pomoću mreže. Vizuelni enkoder prima sliku dokumenta x iz prostora R^(3 x Ho x Wo), obrezuje margine i menja veličinu slike tako da se slika uklopi u pravougaonik fiksne veličine (H, W). Ako je slika manja od pravougaonika, dodaje se "padding" – dodatno popunjavanje, kako bi se osiguralo da slike budu istih dimenzija. Nougat za enkoder koristi Swin transformer, hijerarhijski, vizuelni transformer, koji deli sliku na nepreklapajuće prozore fiksne veličine i primenjuje seriju self – attention slojeva da agregira informacije u tim prozorima. Model proizvodi sekvencu delova z iz prostora $R^{d \times N}$, gde je d latentna dimenzija (dimenzija latentnog sloja), a N broj delova. Enkodirana slika z se dekodira u sekvencu tokena koristeći transformer dekoder arhitekturu sa mehanizmom unakrsne pažnje. Tokeni su generisani u autoregresivnom maniru, koristeći mehanizam samopažnje i mehanizam unakrsne pažnje. Izlaz je projektovan na veličinu rečnika v, kreiran je od logit vrednosti l, l iz $R^{v}$ prostora. [19]




**Slika 20.** Prolazak slike kroz Nougat parser i transformacija do dekodiranih vrednosti

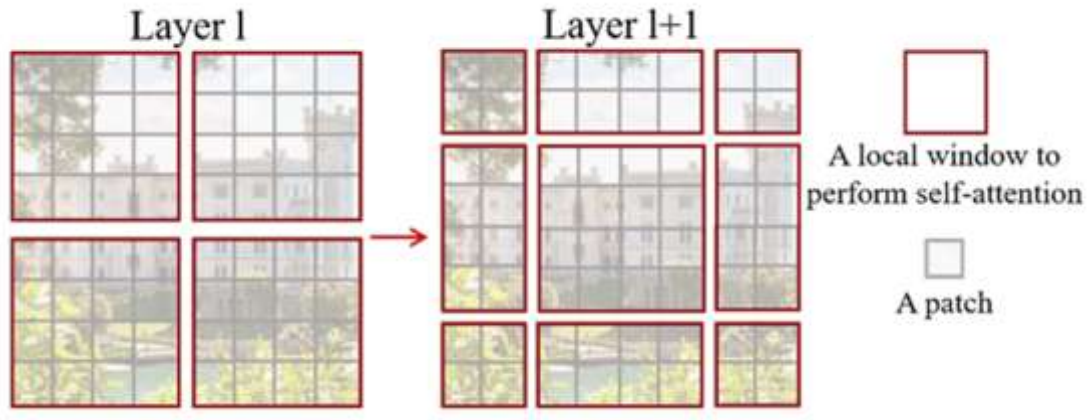
### 2.2.1 Swin transformer

Swin Transformer (Shifted WINdows) zasniva se na hijerarnijskoj obradi slike. Motivacija za implementaciju transformera hijerarhijske strukture polazi od problema neefikasnosti i memorijskog zagušenja prilikom deljenja originalne slike na skupove piksela veoma malih dimenzija. Ako uzmemo za primer jednu sliku 256 x 256 piksela, dekompozicijom na delove dimenzija 16 x 16 piksela postojaće 256 delova koji formiraju sekvencu. Ali ukoliko

za primer uzmemo sliku dimenzija 1920 x 1920 piksela, onda će uz veličinu jednog dela od 16 x 16 piksela originalna slika biti dekomponovana na 14400 vektora koji kreiraju sekvencu, što zahteva veće resursne kapacitete GPU - a. U prethodno opisanim situacijama, korišćenje VIT (VIsion Transformer) modela biće pogodno rešenje. Ali u zadacima u kojima nam je potrebna detaljna analiza slike (zadaci semantičke segmentacije) koja odgovara nivou piksela (kada je potrebno da algoritam na nivou piksela klasifikuje šta se nalazi na delu slike) delovi dimenzija 16 x 16 bili bi suviše veliki. Idealno rešenje u ovakvim situacijama bila bi reprezentacija svakog piksela kao pojedinačnog tokena, ali nakon nekog vremena ono bi postalo neizvodljivo – na slici dimenzija 256 x 256 postojalo bi 65 536 tokena koji bi ju opisivali ne bi li zadatak klasifikacije na nivou piksela bio rešen. FullHD slika dimenzija 1920 x 1080 piksela zahtevala bi dužinu sekvence preko 2 miliona (2073600 tokena). Ukoliko je skup podataka sastavljen od nekoliko hiljada slika, može se zaključiti da ovakav pristup problemu ne bi bio skalabilan i došlo bi do pada efikasnosti (model bi se koncentrisao na učenje obrazaca na nivou piksela i moguće je da ne bi mogao tako lako da nauči obrasce koji se protežu preko više piksela), zagušenja memorije (problemi sa kapacitetom RAM - a i GPU - a). Pri učenju obrazaca, VIT dekomponuje originalnu sliku na delove fiksnih dimenzija koji se ne preklapaju, oni prolaze kroz linearni sloj, dodaju im se poziciono enkoderiane vrednosti i takvi ulazi prosleđuju se transformer arhitekturi. Izlaze čine vektori koji predstavljaju verovatnoće pripadnosti svakoj od klasa koje su opisane rečima iz rečnika. VIT je problematična arhitektura kada je reč o zadacima koji zahtevaju predikcije modela na nivou piksela. [7], [31]

Swin transformer arhitektura pristupa ovom problemu iz drugog ugla. Incijalno, kreiraju se manji delovi originalne slike koji se kasnije spajaju u veće delove u dubljim slojevima transformers arhitekture (poput U-Net – a i konvolucije). Swin transformer početnu sliku dekomponuje na delove dimanzija 3 x 4 x 4 , tako da svaki deo opisuje 3 kanala, 4 piksela u visini i širina od 4 piskela, odnosno ukupno 48 piksela. Za svaki deo 48 piksela prolazi kroz linearnu transformaciju ne bi li se njihova reprezentacija svela na dimenziju C, koja zavisi od odabira. Parametar C (*capacity*) predstavlja veličinu transformer modela (na primer, za bert osnovni model veličina je 768, a za bert veći model je 1024), odnosno broj skrivenih jedinica u potpuno povezanoj mreži transformer arhitekture. Specifično, BERT – veći model ima 24 sloja i koristi skriveni vektor dimenzije 1024, pa svaki token u ulaznom tekstu opisan je kao vektor od 1024 elementa. Za Swin – Tiny *C* parametar ima vrednost

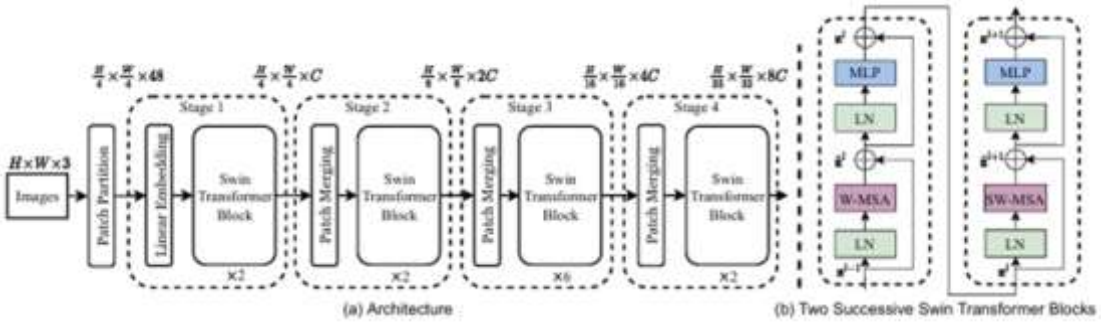
96, dok Swin – Large ima 192 skrivene jedinice u potpuno povezanoj neuronskoj mreži arhitekture transformera. Inicijalni delovi u Swin transformer - u prolaze kroz linearni sloj ne bi li njihova dimenzija bila svedena na dimenziju C. Ovim vrednostima, dodaju se vrednosti pozicionog enkodiranja kada su spremni za ulazak u Swin transformer blok. [32] U ovom sloju, autori predstavljaju koncept mehanizma samopažnje sa pokretnim prozorom – kada model računa vrednosti pažnje između dva dela slike, on obrađuje samo određeni broj delova (M) u svakom trenutku. Umesto da model u obzir uzima celu sliku, on se fokusira na ograničen broj delova unutar prozora čime se omogućuje efikasnija obrada i smanjuju računski troškovi. Umesto složenosti $O(n^2)$ u originalnom pojavljivanju mehanizma samopažnje, kada se u obzir uzimaju svi tokeni, sada je ova složenost redukovana na O(M*n) složenost zato što se pažnja izračunava samo u "prozoru" sastavljenom od M tokena dužine n. Sledeći sloj u hijerarniji naziva se *Sloj spajanja* koji spaja izlaze u vektore u grupama od 2 x 2 grupe delova originalne slike u odnosu na početnu veličinu dela. Od početne dimenzionalnosti C nakon spjanja dolazi se do 4 * C dimenzionalnih vektora koji su prosleđeni linearnom sloju koji redukuje dimenzionalnost sa 4 * C na 2 * C. Pristup sa pomeranjem okvira prilikom izračunavanja vrednosti samopažnje između dva sloja prikazan je na sledećoj slici.




**Slika 21.** *Shifted Window* pristup prilikom izračunavanja *self - attention* vrednosti [32]

Ono što se može uočiti jeste da se promenom granice prozora u dva sukcesivna sloja (Swin transformer bloka) prelazi u prostor drugih prozora definisanim u prethodnom sloju što omogućuje opisivanje veza između nekada nepreklapajućih prozora. Celokupna hijerarhijska procedura se ponavlja sve dok spajanje delova postane nemoguće. Kada je reč

o vrednostima pozicionog enkodrianja, autori napominju da su najbolji rezultati ostvareni upotrebom reprezentacija dobijenih relativnim pozicioniranjem. [32]




**Slika 22.** Arhitektura *tiny* verzije Swin Transformera [7]

Arhitektura "tiny" verzije Swin Transformera (Swin - T) prikazana je na sledećoj slici. Vidi se da je originalna dimenzija slike opisana preko tri kanala boja, visine i širine slike: H x W x 3. Vrši se particionisanje ulazne RGB slike u nepreklapajuće delove pomoću modula za particionisanje delova, kada je svaki deo tretiran kao token a njegovi atributi opisani su kao set RGB vrednosti piksela (svaki token opisan je preko 3 x 4 x 4 = 48 piksela), nakon čega je na ovaj set primenjen linearni sloj ne bi li se dimenzije pojedinačnih delova svele na dimenziju C (*capacity*). Nekoliko blokova transformer arhitekture je primenjeno radi izračunavanja vrednosti pažnje za parove tokena u prozoru M, nakon čega je sačuvan originalni broj tokena, ali je reprezentacija svakog tokena opisana preko C dimenzije. Broj tokena dat je sa (H / 4 * W / 4) i nakon prve faze ostao je isti. [7] Vidi se da slika originalnih dimenzija biva transformisana u delove 4 x 4 piksela, što je manje nego particije 16 x 16 koje koristi VIT. Za uspostavljanje hijerarhijske strukture, slojevi za spajanje delova spajaju početne delove originalne slike u veće delove slike. U drugoj fazi veličina dela postaje veća (8 x 8 piksela) i dolazi do povećanja potpuno povezane neuronske mreže. Svaki token sada je opisan preko vektora dimenzije 2 * C, nakon čega se primenjuje *Swin – transformer Block* za transformaciju originalne reprezentacije tokena u onu reprezentaciju koja u obzir uzima relacije između tokena u granicama prozora "M". Kroz blok za spajanje delova u fazi 3 delovi dimenzija 8 x 8 povezuju se u delove dimenzije 16 x 16, a dubina skrivenog sloja potpuno povezane neuronske mreže postaje 4 * C, nakon čega ovi delovi prolaze kroz *Swin Transformer* blok čiji izlazi imaju iste dimenzije. Ista procedura ponavlja se i u fazi 4. Na slici označenoj sa b prikazana je arhitektura *Swin Transformer* bloka. Kao što je već rečeno, standardna transformer arhitektura podrazumeva izračunavanje vrednosti pomoću

mehanizma samopažnje za svaka dva tokena. Međutim, ovaj pristup dovodi do kvadratne kompleksnosti u odnosu na ukupan broj tokena, O($n^2$) pri izračunavanju ovih vrednosti, što je nepogodno za mnoge vizuelne probleme u kojima je slika predstavljena visokom rezolucijom. Zato se vrednosti samopažnje izračunavaju u okviru lokalnih prozora. Ako svaki prozor ima M x M delova, računska kompleksnost MSA modula i prozorno ograničenog modula (W – MSA) na slici dimenzija h x w date su sa [7]

$$\Omega(\text{MSA}) = 4hwC^2 + 2(hw)^2C \tag{34}$$

$$\Omega(\text{W} - \text{MSA}) = 4hwC^2 + 2M^2hwC \tag{35}$$

Može se reći da je kompleksnost računanja linearna u odnosu na veličinu slike kada je M nepromenljiva vrednost. Izračunavanje reprezentacija mehanizmom samopažnje globalno je neskalabilno i nepristupačno kada je reč o vizuelnim problemima. [31] Među sukcesivnim transformer blokovima za održavanje povezanosti između različitih prozora koristi se mehanizam particionisanja sa blokom koji se pomera, koji menja način particionisanja između dva sukcesivna transformer bloka. Na slici 2 vidi se da je originalna slika dimenzija 8 x 8 podeljena na prozore dimenzija 4 x 4, što je ukupno 4 prozora. Ovo podrazumeva regularnu deobu originalne slike. U sledećem transformer sloju vidi se da se prozori pomeraju u odnosu na prethodni raspored za (M/2, M/2) piksela. Upotrebom pristupa pomeranja prozora, izlaze uzastopnih Swin transformer blokova možemo izračunati na sledeći način [7]

$$\hat{z}^l = W - MSA\left(LN(\hat{z}^{l-1})\right) + \hat{z}^{l-1} \tag{36}$$

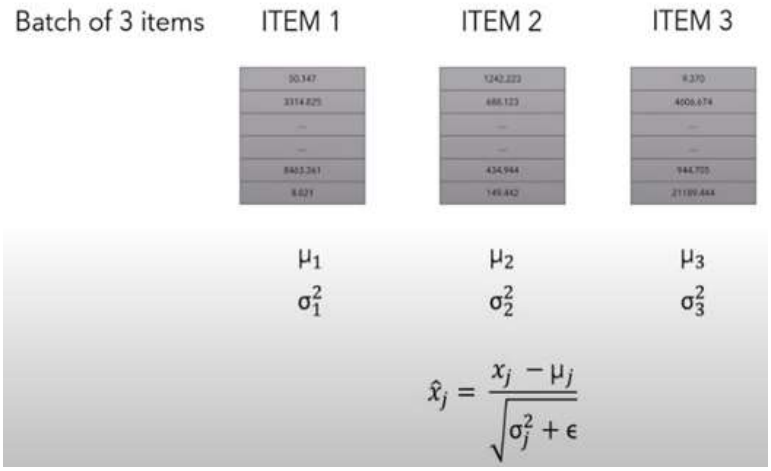
$$z^l = MLP\left(LN(\hat{z}^l)\right) + \hat{z}^l \tag{37}$$

$$\hat{z}^{l+1} = SW - MSA\left(LN(z^l)\right) + z^l \tag{38}$$

$$z^{l+1} = MLP\left(LN(\hat{z}^{l+1})\right) + \hat{z}^{l+1} \tag{39}$$

gde prve dve jednačine opisuju izlazne karakteristike (S)W – MSA (*Shifted window multi self – attention layer*) modula i MLP modul za blok l, respektivno. [7]

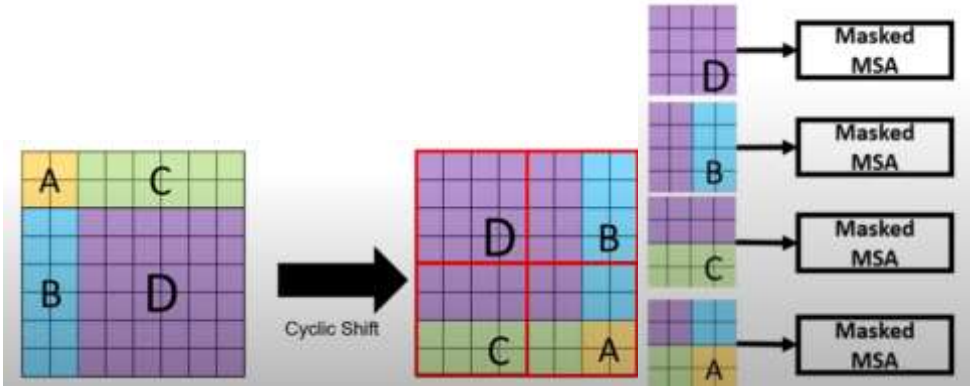
LN (Layer Normalization) blok podrazumeva normalizaciju takvu da se na nivou jednog batch – a za svaki patch (deo) obračunava srednja vrednost i standardna devijacija, nakon čega se ovako dobijena vrednost množi gama parametrom (multiplikativ) i dodaje joj se beta parametar (aditiv) koji uvodi fluktuaciju u podatke, zato što bi kompresovanje vrednosti u opseg od 0 do 1 bilo suviše restriktivno. Mreža prilagođava ova dva parametra ne bi li se uvela fluktuacija u podacima (neuronska mreža uči ove parametre). [32]




**Slika 23.** *Layer* normalizacija [32]

W – MSA (*Window based multi – assess attention layer*) i SW – MSA označavaju *window – based multihead self – attention* koristeći konfiguracije particionisanja za regularni i pomereni prozor. Zašto izračunavamo vrednosti samopažnje za delove u okviru jednog prozora možemo intuitivno objasniti preko primera da nije potrebno računati vrednosti pažnje gornjeg levog piksela sa donjim desnim pikselom početne slike zato što oni nemaju nikakvu semantičku povezanost. Međutim, ukoliko pristupimo računanju vrednosti samopažnje u okviru jednog prozora, pojavljuje se problem semantičke sličnosti između više prozora. (Na primer, ako Swin model pokušava da identifikuje da je na slici mačka, u jednom prozoru je njena glava, a u drugom prozoru je njen rep potrebno je uspostaviti relaciju između prozora). Zato se uvodi koncept *Shifted – window Multi self - attention* prema kom je potrebno "pomeriti" granicu prozora za polovinu njegove visine i širine. Na

primer, ako je originalna veličina prozora M = 4 i slika je podeljena na 4 prozora, onda će pomeranje biti izvršeno 2 piksela po dužini i 2 piksela po širini. Posledica ovog pristupa je mnogo piksela sa nula vrednostima i povećanje računske kompleksnosti. Zato se umesto prozora pomeraju delovi slike. Ova operacija pomeranja može se izvršiti roll metodom iz biblioteke torch, pa se nakon pomeranja delova određuju granice prozora jednake originalnim veličinama prozora. Na prvoj slici prikazan je primer originalnih delova, dok je metodom roll dobijen raspored sa druge slike, tzv. *shifted patches,* nad kojima se primenjuje postavljanje prozora veličine M = 4. [32]




**Slika 24.** *Cyclic shift* mehanizam i maskiranje [32]

Na drugom prozoru vidimo da postoje dva dela (ljubičasti i plavi delovi) koji, posmatrano sa originalne slike, ne bi trebalo da imaju nikakvu relaciju. Da bi se ovo izbeglo, uvodi se mehanizam maskiranja, nakon čega se opet obezbeđuje povratak na originalni raspored delova. Mask mehanizam u imeplemntaciji podrazumeva proveru vrednosti shift_size varijable; ako je vrednost ove varijable pozitivna, onda se setuje maska slike na tenzor popunjen nula vrednostima dimenzija (1, H, W, 1), gde poslednja jedinica predstavlja broj kanala. Takođe, definisane su i h_slices i w_slices vrednosti – to su delovi početne maske. Primer kako funkcioniše maskiranje opisaćemo na dimenziji H = W = 8, window_size = 4 i shift_size = 2. [7], [32]

**Slika 25.** Implementacija metode koja definiše masku [32]

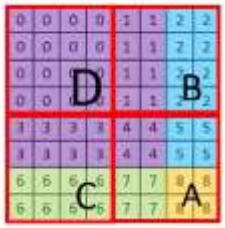
Jednostavnim prolaskom kroz for petlje koda popunjavaju se odgovarajući delovi (*slices*) originalne maske vrednostima cnt promenljive. Za gore pomenute vrednosti maska bi izgledala ovako:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 2 | 2 |
| 0 | 0 | 0 | 0 | 1 | 1 | 2 | 2 |
| 0 | 0 | 0 | 0 | 1 | 1 | 2 | 2 |
| 0 | 0 | 0 | 0 | 1 | 1 | 2 | 2 |
| 3 | 3 | 3 | 3 | 4 | 4 | 5 | 5 |
| 3 | 3 | 3 | 3 | 4 | 4 | 5 | 5 |
| 6 | 6 | 6 | 6 | 7 | 7 | 8 | 8 |
| 6 | 6 | 6 | 6 | 7 | 7 | 8 | 8 |

**Slika 26.** Izgled rezultujuće maske [32]

Podelom maske na prozore veličine 4, možemo primetiti da postoje prozori koji sadrže više fragmenata originalnih delova. Na primer, u gornjem desnom prozoru postoje delovi ljubičastog i delovi plavog originalnog segmenta o kojima je bilo reči da ne treba da imaju nikakvu relaciju jer se nalaze na suprotnim ivicama slike. U kodu ovo je obezbeđeno računanjem razlike između svaka dva elementa jednog prozora; ako je njihova razlika nula, onda oni pripadaju istom delu, vrednosti maske popunjavaju se nulama, a u suprotnom ako ovo nije slučaj, vrednosti maske popunjavaju se vrednošću -100.00, kako bi se kasnijom primenom *softmax* funkcije -100 vrednost transformisala u 0, a nula u 1. Ukratko, primenom

maske ostvaruje se rezutat 0 onda kada u okviru istog prozora imamo različite delove originalne slike i za te vrednosti neće se računati vrednosti pažnje. [32]




**Slika 27.** Vizuelni prikaz izračunate maske, *patches* delova slike i prozora [32]

Još jedan od koncepata koji je uveden hijerarhijskim pristupom jeste spajanje delova – kada dolazi do spajanja originalnih delova u veće delove pri čemu se povećava i broj kanala koji opisuje sliku. Početni delovi veličine su 4 x4 piksela, nakon čega se svaka dimenzija povećava za još četiri dodatna piksela (8 x 8) i kao posledica povećava se i broj kanala 4 puta u odnosu na originalni broj kanala. Nakon spjanja delova dodat je linearni sloj koji redukuje broj kanala (sa 4C na 2C tako da delovi imaju veličine 8 x 8). Swin model sastoji se od 4 faze u kojima se spajanje delova ponavlja dok delovi ne ostvare veličinu od 32 x 32. [32]

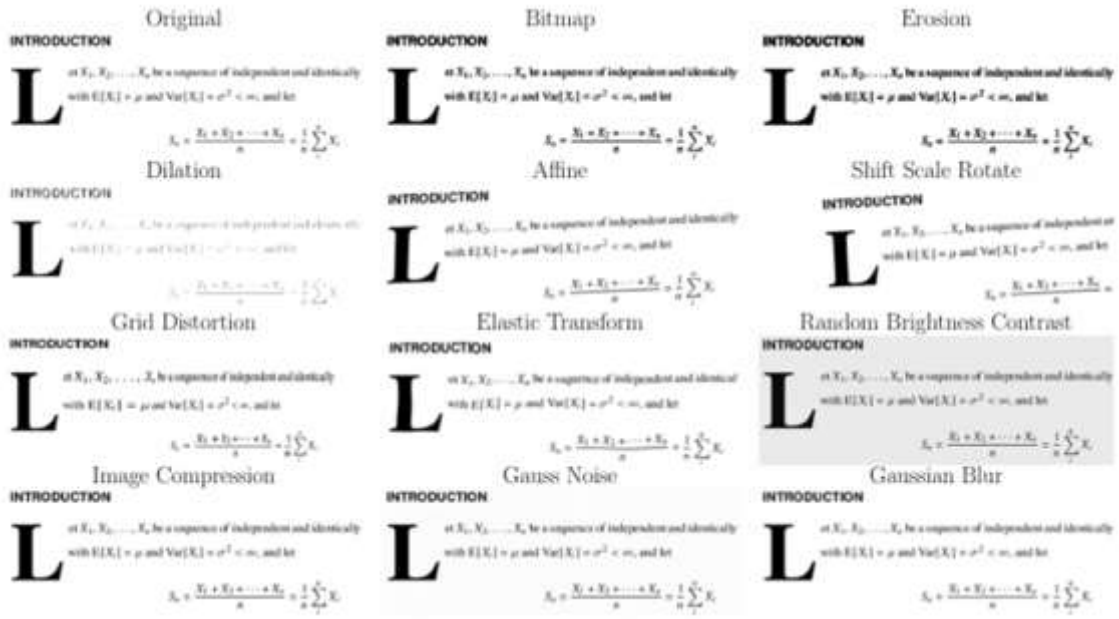
Rezultati pokazuju da Swin transformer ima bolje performanse od ViT i DeiT modela na klasifikaciji slika, detekciji objekata i zadatku semantičkog segmentiranja. Očekuju se bolje performanse na detekciji malih objekata. Njegove performanse nadmašuju prethodni *state – of – the – art* model za +2.7 box AP kada je reč o zadacima detekcije objekata, + 2.6 mask AP na COCO skupu podataka, kao i + 3.2 mloU na ADE20K skupu podataka.[7]

Dekoder u Donut arhitekturi odgovara arhitekturi dekodera transformer arhitekture. Ova arhitektura prima "ključ" i "vrednost" vektore koje je generisao Swin transformer, dok query vrednosti generiše kao izlaz, autoregresivno. Kao što je bilo reči, izlaz iz dekodera predstavlja ulaz u linearni sloj koji mapira izlazni token na dimenziju koja odgovara veličini rečnika (za transformers arhitekturu, predstavlja rečnik od 50 000 tokena) i opisuje verovatnoće da je sledeći token onaj čija je verovatnoća maksimalna, metoda *argmax* za maksimalnu verovatnoću vraća indeks tog tokena u rečniku. Facebook istraživački tim je za veličinu ulazne slike odabrao sledeće dimenzije (H, W) = (869, 672), što odgovara : 3 odnosu stranica, Broj tokena koji dekoder prima je 4096, što je veliki broj, vodeći računa o

sadržaju poput tabela u naučnim radovima koje su opisane kao “token intenzivne” strukture podataka. BART dekoder koji je deo Donut arhitekture je transformer koji ima samo dekoder koji ima 10 slojeva. Celokupna arhitektura sastoji se od 350 miliona parametara. [7]

### 2.2.2 Trening Nougat modela

Tokom treninga Nougat modela, korišćene su tehnike augmentacije podataka kako bi se poboljšala generalizacija, imajući u vidu da je model treniran na digitalnim naučnim radovima, pa je potrebno da se uvedu nesavršenosti u rezoluciji dokumenta koji je skenrian. Transformacije koje su vršene nad podacima su: erosion, dilatacija, Gausov šum, Gausov blur, bitmap konverzija, kompresija slike, grid distortion i elastična transformacija. [19]



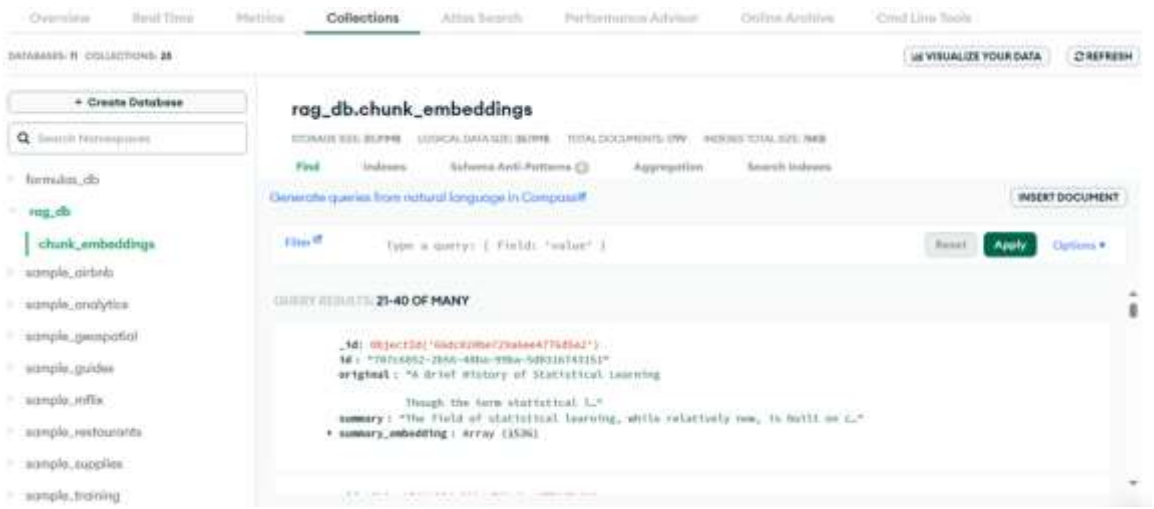

**Slika 28.** Augmentacija podataka prilikom treninga [19]

Tokom treninga korišćen je AdamW optimizator, trening je trajao 3 epohe, sa vrednošću parametra batch_size = 192. Originalna stopa učenja bila je $5 * 10^{-5}$ , smanjivajući se za 0.9996 na svakih 15 ažuriranja dok ne dostigne vrednost $7.5 * 10^{-6}$. [19]

# 3 SKLADIŠTENJE TEKSTUALNIH PODATAKA

## 3.1 MONGO ATLAS BAZA PODATAKA I INDEKS

MONGO DB Atlas je servis baze podataka na cloud – u koji pojednostavljuje operacije sa bazama podataka za programere i omogućuje im da se fokusiraju na razvoj aplikacija. Pomoću ovog servisa moguće je kreirati klaster bez obaveze za upravljanjem bazom podataka; ovu obavezu preuzima MongoDB, kao i mnoge slične funkcionalnosti – skaliranje, kreiranje rezervnih kopija, sigurnost. Ove funkcionalnosti ponuđene su pod nazivima *globalni klasteri, napredna sigurnost, praćenje u realnom vremenu, za moderne* i sl. Takođe, postoji mogućnost da se konfiguriše lista dozvoljenih IP adresa koje mogu da pristupe podacima koji su skladišteni u bazi. [33]




**Slika 29.** Prikaz klastera na kontrolnoj tabli Mongo DB Atlas servisa

Jednostavni prikaz klastera prikazan je na slici 27. U njemu se nalazi baza podataka *rag_db* i kolekcija *chunk_embeddings* u kojoj su skladišteni tekstualni i tabelarni podaci, tekstualni opisi formula kao i tekstualni opisi slika. *Pymongo* biblioteka omogućuje kreiranje *MongoClient* instance, povezivanje preko konekcionog stringa sa klasterom, ekstrakciju baze podataka iz klastera i konačno pristup kolekciji dokumenata.

```
client = pymongo.MongoClient("mongodb+srv://zmahovac:gUwtDkUSbglrGxep@cluster0.ictm8.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0",connect=False)
db = client.rag_db
collection = db.chunk_embeddings
```

**Slika 30.** Povezivanje sa klasterom iz Python fajla [27]

Kolekcija je napunjena podacima najbolje evaluiranih veličina chunkova u tekstualnom obliku, njihovim sažecima, id – jevima kao i vektorskim reprezentacijama koje su generisani *OpenAIEmbeddings* modelom za kreiranje vektorskih reprezentacija *langchain_openai* biblioteke.

```
def insert_txt_data():

    txt_summaries = get_data('results/1_txt_summaries.pkl')
    txt_originals = get_data('results/raw_txt_elements.pkl')
    text_ids = [str(uuid.uuid4()) for _ in txt_originals]
    for idx,summary in enumerate(txt_summaries):
        document = {
            "id":text_ids[idx],
            "original":txt_originals[idx].text,
            "summary":summary,
            "summary_embedding":generate_embeddings(summary)
        }
        collection.insert_one(document)
```

```
def insert_table_data():
    tbl_summaries = get_data('results/1_table_summaries.pkl')
    tbl_originals = get_data('results/raw_table_elements.pkl')
    tbl_ids = [str(uuid.uuid4()) for _ in tbl_summaries]
    for idx, tbl_summary in enumerate(tbl_summaries):
        document = {
            "id":tbl_ids[idx],
            "original":tbl_originals[idx].text,
            "summary":tbl_summary,
            "summary_embedding":generate_embeddings(tbl_summary)
        }
        collection.insert_one(document)
```

```
def insert_img_data():
    img_summaries = get_data("results/1_image_summaries.pkl")
    counter = 0
    for element in img_summaries:
        print(counter)
        document = {
            "id":element[0],
            "original":element[1],
            "summary":element[1],
            "summary_embedding":generate_embeddings(element[1])
        }
        collection.insert_one(document)
        counter += 1
```

```
def insert_formulas_data():
    formula_descriptions = get_data('results/formula_descriptions.pkl')
    counter = 0
    for element in formula_descriptions:
        print(counter)
        document = {
            "id":element[0],
            "original":element[1],
            "summary":element[1],
            "summary_embedding":generate_embeddings(element[1])
        }
        collection.insert_one(document)
        counter += 1
```

**Slika 31.** Implementacija koda za unos podataka u kolekciju [27]

```
{
  "mappings": {
    "dynamic": true,
    "fields": {
      "summary_embedding": {
        "dimensions": 1536,
        "similarity": "dotProduct",
        "type": "knnVector"
      }
    }
  }
}
```

**Slika 32.** Kreiranje indeksa iz Mongo DB Atlas servisa

Na slici 29 nalazi se JSON koji prikazuje definiciju indeksa za polje *summary_embedding* u MongoDB Atlas, čiji su parametari:

(1) **dynamic** - kada je postavljeno na *true*, ovo polje omogućava automatsko prepoznavanje i dodavanje novih polja u indeks bez potrebe za prethodnim definisanjem. Ovo podešavanje je korisno kada se očekuje da će se struktura podataka često menjati; (2) **fields** - ovaj JSON element definiše specifična polja koja će biti deo indeksa. U ovom slučaju, definisano je samo jedno polje, *summary_embedding*; (3) **summary_embedding** - ovo je ime polja koje se indeksira. (3.1) **dimensions** - ovaj parametar postavlja broj dimenzija za vektorsko polje. U ovom slučaju, *summary_embedding* ima 1536 elemenata, što sugeriše da se radi o vektoru sa 1536 komponenti; (3.2) **similarity** - ovaj parametar određuje način na koji će se izračunavati sličnosti između vektora. U ovom primeru, postavljeno je na *dotProduct,* što znači da će se sličnost između vektora meriti kao skalarni proizvod; (3.3) **type** - ovaj parametar definiše tip indeksa. *Knn Vector* označava da se radi o indeksu koji omogućava brzo pretraživanje "k najbližih suseda" (KNN) koristeći vektore, što je korisno za efikasnu pretragu sličnosti u velikim skupovima podataka. [34]

```
def semantic_search(query:str):
    results = collection.aggregate([
    {"$vectorSearch":{
        "queryVector":generate_embeddings(query),
        "path":"summary_embedding",
        "numCandidates":1531,
        "limit":5,
        "index":"ChunkSemanticSearch"
        }
    }
    ]);
    documents = []
    for document in results:
        documents.append(document['original'])
    return documents
```

**Slika 33.** Implementacija semantičke pretrage u Python - u Mongo DB Atlas indeksom [27]

Metoda *semantic_search* implementira proces pretrage semantičke sličnosti u kolekciji dokumenata koristeći vektorske reprezentacije. Ova metoda uzima jedan argument, *query,* koji predstavlja korisnički upit, i vraća listu dokumenata koji su najbliži tom upitu prema definisanoj metrici sličnosti. Unutar funkcije, prvi korak uključuje izvođenje agregacione operacije na kolekciji podataka. Koristi se MongoDB Atlas funkcionalnost $vectorSearch, koja omogućava pretraživanje vektora na osnovu sličnosti. Ovaj operator zahteva nekoliko ključnih parametara: (1) **queryVector** - parametar se dobija pozivanjem funkcije

*generate_embeddings(query),* koja konvertuje korisnički upit u vektorsku reprezentaciju; (2) **path** - parametar ukazuje na putanju do polja u dokumentima koje sadrži vektorsku reprezentaciju, u ovom slučaju *summary_embedding* polje; (3) **numCandidates** - parametar određuje maksimalan broj vektora koji će biti uzeti u obzir pri pretraživanju i postavljen je na 1531. Ovaj pristup omogućava efikasno filtriranje kroz veliku količinu podataka pre nego što se izaberu konačni rezultati; (4) **limit** - parametar je postavljen na 5 i definiše maksimalan broj rezultata koji će biti vraćeni korisniku. Ovo osigurava da se samo najrelevantniji dokumenti prikažu kao ishod pretrage. Nakon izvršavanja agregacione operacije, rezultati se procesiraju u petlji. Svaki dokument u rezultatu se dodaje u listu documents, pri čemu se izdvaja originalni sadržaj iz svakog dokumenta. Na kraju, metoda vraća listu dokumenata koji su najbliži korisničkom upitu, čime omogućuje pristup relevantnim informacijama na osnovu semantičke sličnosti. [34]

## 3.2 VEKTORSKA SKLADIŠTA PODATAKA

*Chroma vectorstore* je vektorska baza podataka za razvoj aplikacija pomoću velikih jezičkih modela. Ona omogućuje skladištenje vektorskih reprezentacija dokumenata i njihovih metapodataka, kao i pretragu na osnovu sličnosti između vektorskih reprezentacija dokumenta i korisničkog upita. [35] Chroma skladište podataka u arhitekturi apliakcije korišćeno je sa jedne strane kao vektorsko skladište instance klase *MultiVectorRetriever*, kao i implementaciju instance jednostavnog pretraživača iz skladišta podataka sa druge strane.

*FAISS* (*Facebook AI Similarity Search*) je još jedan od metoda efikasne pretrage sličnih tekstova koji je razvio AI istraživački tim Mete. Dostupan u istoimenoj biblioteci, pogodan je i za zadatke klasterovanja sličnih vektora. Sadrži algotitme koji pretražuju setove vektora različitih dimenzija, pri čemu je u okviru ove biblioteke implementiran kod za evaluaciju i prilagođavanje hiperparametara. Za dati set vektora oblika $x_i$ i dimenzije d efikasno izvodi sledeću operaciju:

$$j = argmin_i ||x - x_i|| \tag{40}$$

U kojoj je ||.|| Euklidova distanca. Za faiss bazu podataka, relevantna struktura je index u koji se skladište ovi vektori, pa postoji metoda *add* za dodavanje vektora $x_i$. Još neke od

dodatnih mogućnosti Faiss baze podataka su: (1) Vraćanje ne samo najbližeg suseda, već i 2. najbližeg, 3. najbližeg, …, k. najbližeg suseda; (2) Pretragu više vektora istovremeno umesto jednog (serijska obrada). Za mnoge tipove indeksa, ovo je brže od pretrage jednog vektora za drugim; (3) Balansiranje između preciznosti i brzine, tj. može dati netačan rezultat 10% vremena koristeći metodu koja je 10x brža ili koristi 10x manje memorije; (4) Izvođenje pretrage maksimalnog skalarnog proizvoda umesto minimalne Euklidske pretrage. Takođe postoji ograničena podrška za druge distance ($L_1$, $L_{inf}$, itd.); (5) Vraćanje svih elemenata koji su unutar datog radijusa od tačke upita (pretraga u opsegu); (6) Skladištenje indeksa na disku umesto u RAM-u; (7) Indeksiranje binarnih vektora umesto vektora sa pokretnim tačkama; (8) Ignorisanje podskupa indeksa vektora prema uslovu na identifikatorima vektora. Implementacija u Python programskom jeziku podrazumevala bi zamenu pređašnje kreiranog Chroma vektorskog skladišta za FAISS vektorsko skladište, dok je ostatak implementacije identičan (implementacija biće data uz upotrebu *MultiVectorRetriever* - a). [36]

**Slika 34.** Implementacija FAISS baze podataka uz upotrebu instance *MultiVectorRetriever* klase [27]

Kako bi se evaluiralo vreme koje je potrebno faiss i chroma retrieverima da vrate relevantne dokumente za korisničke upite iz evaluacionog seta podataka, implementirana je metoda *measure_execution_time.* Prosečno vreme odziva na korisničke upite za chroma bazu iznosi 0.212888993714951 sekundi, dok za faiss bazu podataka iznosi 0.2816542700717324 sekundi. Kako je Chroma baza brža od Faiss skladišta podataka za 0.068765 sekundi kada

je reč o vremenu odgovora na korisničke upite, u okviru instanci pretraživača biće implementirana vektorska skadišta podataka pomoću Chroma baze podataka.

```
import time
import numpy as np

def measure_execution_time(retriever):
  times = list()
  for query in questions:
    start_time = time.time()
    results = retriever.invoke(query)
    end_time = time.time()
    times.append(end_time-start_time)
  return np.mean(times)

time_chroma = measure_execution_time(chroma_retriever)
time_faiss = measure_execution_time(faiss_retriever)
```

**Slika 35.** Implementacija metode za merenje vremena odziva na korisničke upite uz upotrebu retriever - a sa Chroma i Faiss skladištima [27]

## 3.3 TIPOVI PRETRAŽIVAČA

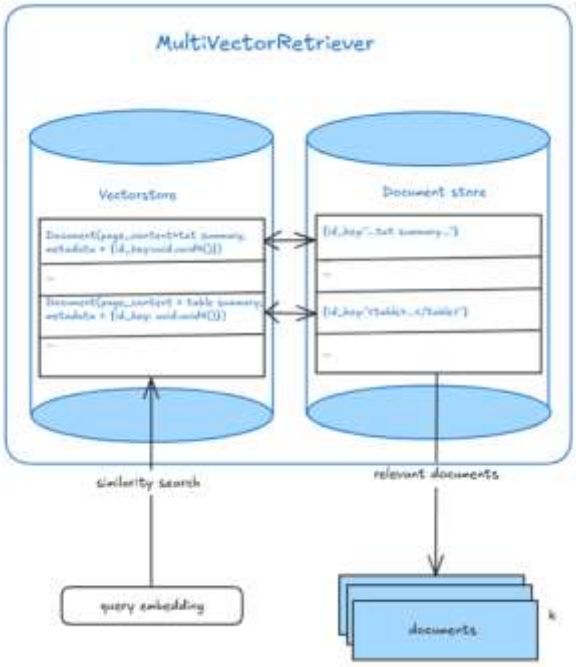
### 3.3.1 Multi Vector Retriever

*Multi Vector Retriever* omogućuje skladištenje više vektorskih reprezentacija za jedan dokument. Postoji nekoliko metoda da se kreira više vektorskih reprezentacija originalnog dokumenta:

1. Podela dokumenta u manje chunk – ove i kreiranje njihovih vektorskih reprezentacija – tzv. *ParentDocumentRetriever;*

2. Kreiranje sažetaka tekstualnih reprezentacija originalnih dokumenata za svaki dokument zajedno sa (ili umesto) vektorske reprezentacije originala;

3. Kreiranje hipotetičkih pitanja na koje bi originalni dokument mogao da odgovori i kreiranje vektorskih reprezentacija ovih pitanja zajedno sa (ili umesto) vektorskim reprezentacijama originalnog dokumenta. Ovakav način manuelnog dodavanja vektorskih reprezentacija pitanja odgovarajućem dokumentu obezbeđuje više kontrole u razvoju aplikacije pomažući da manuelno dodata pitanja “vode” do originala dokumenta. [16]




**Slika 36.** Vizuelni prikaz instance *Multi Vector Retriever* klase koji u svom skladištu podataka sadrži sažetke originalnih dokumenata

U implementaciji aplikacije kreirana je instanca klase *MultiVectorRetriever* pomoću kreiranja sažetaka originalnog dokumenta, budući da je praksa pokazala da sažeci mogu preciznije da opišu sadržaj originala. Ovi sažeci su skladišteni u vektorsko skladište podataka, *Chroma vectorstore,* oni su povezani sa originalima dokumenata, skladištenim u instanci *InMemoryByteStore,* gde su byte – string reprezentacije sadržaja dokumenata. Sadržaji *Chroma* skladišta vektora i *InMemoryByteStore* skladišta dokumenata povezani su preko *id_key* parametra, tako da se pretraga vrši upoređivanjem vektorskih reprezentacija sažetka dokumenta i korisničkog upita. Metrike koje se koriste opisane su preko parametra *type* koji prima string vrednosti ili instance tipa *SearchType* i imaju vrednosti *mmr, similarity, similarity score threshold.* Kao što je rečeno, poređenje se vrši između vektorskih reprezentacija sažetka i korisničkog upita, dok iz faze vraćanja relevantnih dokumenata izlaze originalni dokumenti kojima se pristupa preko *id_key* vrednosti. Pomoću parametra *k* moguće je podešavati koliko dokumenata mora biti vraćeno iz ove faze. Parametar *similarity score threshold* opisuje tip pretrage koji se takođe fokusira na sličnost, ali uključuje prag koji rezultati moraju da dostignu da bi bili uključeni u ishod pretrage. *MMR (Maximal Marginal Relevance)* tip ptretrage koristi tehniku marginalne relevantnosti za

ponovno rangiranje rezultata sličnosti. MMR je metoda koja pomaže u smanjenju redundantnosti u rezultatima pretrage, osiguravajući da su vraćeni dokumenti što raznovrsniji i relevantniji. [16]

```python
def create_chroma_vectorstore(table_originals,text_originals,table_summaries,text_summaries,image_summaries,formula_descriptions,search_type:str,k:int):

    vectorstore = Chroma(
        collection_name="summaries",embedding_function=OpenAIEmbeddings(openai_api_key=OPENAI_API_KEY)
    )
    #2. store za originalne dokumente
    store = InMemoryStore()
    id_key="doc_id"

    if search_type=="similarity":
        s_type = SearchType.similarity
    else:
        s_type=SearchType.mmr

    #3. MultiVectorRetriever
    retriever = MultiVectorRetriever(
        vectorstore=vectorstore,
        docstore=store,
        id_key=id_key,
        search_type=s_type, # similarity ili mmr
        search_kwargs={"k":k} # 15
    )

    table_ids = [str(uuid.uuid4()) for _ in table_originals] #ids originalnih tabela
    summary_tables = [Document(page_content=s, metadata={id_key:table_ids[i]}) for i, s in enumerate(table_summaries)] #ids se proslede summaryjima i od njih se naprave documen
    retriever.vectorstore.add_documents(summary_tables)
    retriever.docstore.mset(list(zip(table_ids,table_originals))) # originali u obliku ebcre27381:'html notacija za original tabelu u stringu'

    # tekstovi: text_originals, text_summaries
    text_ids = [str(uuid.uuid4()) for _ in text_originals]
    summary_texts = [Document(page_content=s, metadata={id_key:text_ids[i]}) for i,s in enumerate(text_summaries)]
    retriever.vectorstore.add_documents(summary_texts) # dokumenti svih summaryja sa idjevima koji odgovaraju originalima
    retriever.docstore.mset(list(zip(text_ids,text_originals)))

    print(f"{len(retriever.vectorstore.get()['documents'])} documents stored.")

    return retriever
```

**Slika 37.** Kreiranje instance *Multi Vector Retriever - a* sa *Chroma* skladištem podataka [27]

```python
def load_text_summarize_chain():

    def parse(ai_message: AIMessage) -> str:
        """Parse the AI message."""
        return ai_message.content

    prompt_text = """ You are an assistant who's task is to summarize tables and chunks made of text. \
    Retrieve the summary response in the following format: ''
    Give precise and informative summary of table or text. Table or text element \
    is provided to you from following content: {element}
    """
    prompt = ChatPromptTemplate.from_template(prompt_text)
    summarize_chain = {"element":lambda x: x} | prompt | chat | parse
    return summarize_chain
```

**Slika 38.** Lanac za sumiranje sadržaja tekstualnih chunk – ova [27]


**Slika 39.** Metoda za kreiranje sažetaka tekstualnih sadržaja [27]

Metoda *create_chroma_vectorstore* ima za cilj da kreira vektorsku bazu podataka koristeći Chroma skladište podataka, koje omogućava efikasno skladištenje i pretragu dokumenata na osnovu njihovih vektorskih reprezentacija. Prvo se inicijalizuje instanca *Chroma*, koja koristi funkciju za generisanje vektorskih reprezentacija putem OpenAI API ključa. Ova vektorska baza podataka služi kao centralno mesto za skladištenje vektora dokumenata, omogućavajući brzu i efikasnu pretragu na osnovu sličnosti. Zatim, kreira se instanca *InMemoryStore* koja funkcioniše kao lokalni skladišni sistem za originalne dokumente, gde se koristi *ide_key* kao ključ identifikacije dokumenata. Na osnovu tipa pretrage koji korisnik definiše (bilo da se radi o pretrazi sličnosti ili MMR - maksimalnoj marginalnoj relevantnosti), metoda određuje odgovarajući tip pretrage pomoću enumeracije *SearchType.* U sledećem koraku, instancira se *MultiVectorRetriever*, koji integriše vektorsku bazu podataka (*vectorstore*) i skladište originalnih dokumenata (*store*). Ovaj retriver omogućava pretragu dokumenata koristeći definisani tip pretrage i argumente pretrage, pri čemu se k koristi za određivanje broja vraćenih rezultata. Nakon postavljanja retrivera, kreiraju se jedinstveni identifikatori za originalne tabele i tekstove koristeći UUID. Ovi identifikatori se povezuju sa sažecima dokumenata, koji se zatim dodaju u vektorsku bazu podataka. Dokumenti se kreiraju sa sadržajem sažetaka i pripadajućim metapodacima, čime se obezbeđuje efikasno prepoznavanje i pretraga. Na samom kraju, metoda ispisuje broj dokumenata koji su uspešno skladišteni u vektorsku bazu podataka i vraća instancu retrivera, koja omogućava dalju interakciju s skladištenim dokumentima.

### 3.3.2 Parent Document Retriever

Kada se vrši deoba PDF dokumenta često postoji konflikt između želja: (1) dekomponovanje na male dokumente tako da vektorske reprezentacije što tačnije reflektuju njihovo značenje - ako su sadržaji dokumenata predugački, onda vektorske reprezentacije mogu da izgube pravo značenje; (2) dovoljno dugački dokumenti ne bi li se održalo

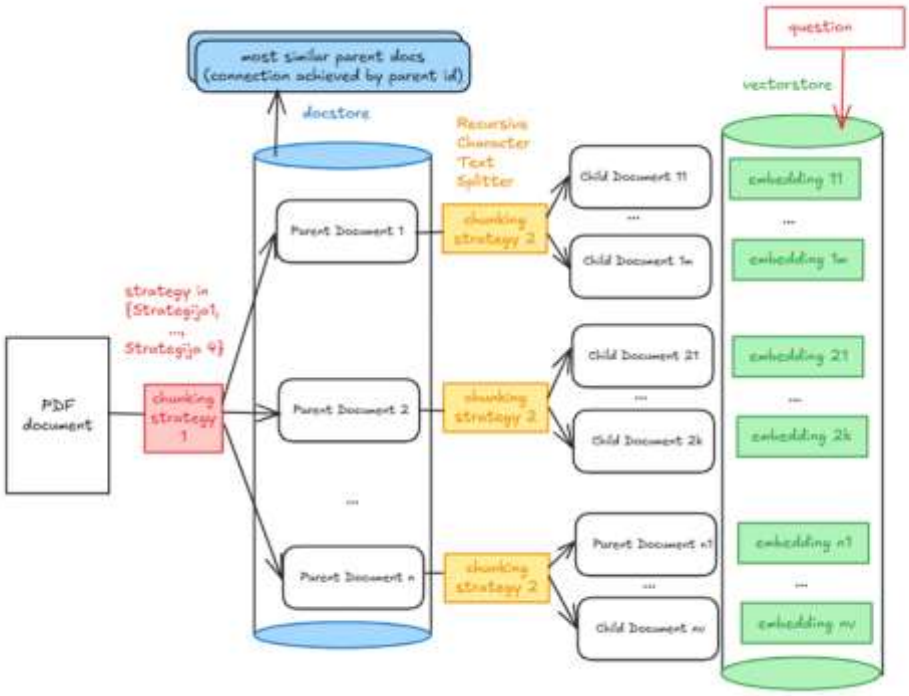
originalno značenje tekstualnog sadržaja svakog dela teksta. *Parent Document Retriever* održava balans tako što deli i čuva male skupove podataka, a tokom faze pretrage prvo vrši poređenje sa malim tekstualnim fragmentima, zatim pronalazi "roditelj" dokument preko *parent_id* parametra i vraća originalni, veći tekstualni isečak. Roditeljski dokument je dokument iz kog je manji dokument potekao. To može biti celokupan dokument ili veći tekstualni segment. [37] Implementacija instance *ParentDocumentRetriever* klase data je na slici broj 37.

```
def create_parent_retriever(txt_chunks_path, tbl_elements_path,k_parameter): # txt i tabele
    """
    txt_chunks_path = "./data_dir/chunked_elements_1.pkl", tbl_elements_path = "./data_dir/raw_table_elements.pkl"
    """
    txt_chunks = load_pickle_file(txt_chunks_path)
    text_elements = [Element(type="text",text=element.text,page_no=dict(element.metadata.fields)['page_number']) for element in txt_chunks]
    tbl_elements = load_pickle_file(tbl_elements_path)

    # documents
    txt_docs = [Document(page_content=element.text, metadata={"page_no":element.page_no}) for element in text_elements]
    tbl_docs = [Document(page_content=element.text, metadata={"page_no":element.page_no}) for element in tbl_elements]

    all_docs = txt_docs + tbl_docs

    child_splitter = RecursiveCharacterTextSplitter(chunk_size=400)
    vectorstore = Chroma(
        collection_name="full_documents_1", embedding_function=OpenAIEmbeddings(api_key=os.getenv("OPENAI_API_KEY"))
    ) # za children, ovde su sacuvani subdocs!!!
    store = InMemoryStore() # za parent document
    retriever = ParentDocumentRetriever(
        vectorstore=vectorstore,
        docstore=store,
        child_splitter=child_splitter,
    )
    retriever.add_documents(all_docs, ids=None)
    retriever.search_kwargs = {"k":k_parameter,}
    return retriever
```

**Slika 40.** Implementacija *Parent Document Retriever* klase [27]

*Child_splitter* se koristi za deobu "roditelja" dokumenta, u kodu je korišena strategija primenom klase *RecursiveCharacterTextSplitter,* koja pokušava da izvrši deobu originalnog dokumenta dok "deca" dokumenti ne postanu dovoljno mali (korišćena je preporučena veličina od 400 karaktera za ove potrebe). Podrazumevana lista separatora data je sa ["\n\n", "\n", " ", ""], pri čemu je "\n\n" paragraf separator, "\n" rečenični separator, " " separator na nivou reči, a "" separator na nivou karaktera. Kako paragrafi i rečenice podrazumevaju najjače semantičke veze reči u tekstu, ova deoba ima za efekat da zadrži pasuse i rečenice zajedno dokle god je to moguće. Tekstualni sadržaj podeljen je u liste karaktera, a veličina tekstualnog segmenta merena je preko broja karaktera. Prvi separator koji se proverava je na nivou paragrafa, tekst se deli pomoću ovog separatora i vrši se provera da li je segment nastao deobom manji od predefinisane veličine *chunk_size.* Ako je chunk manji od *chunk_size* parametra po broju karaktera, on se zadržava. U suprotnom, on se deli na

rečenice pomoću rečeničnog separatora "\n". Parametrom *chunk_size* data je maksimalna veličina chunk – a, *chunk_overlap* parametar omogućuje preklapanje susednih tekstualnih segmenata ne bi li se očuvala njihova povezanost pri deobi, *length_function* je funkcija koja određuje veličinu segmenta teksta. Neki jezici nemaju granice između reči (kineski, japanski, tajlandski), pa podrazumevana lista separatora može da izazove deobu samih reči između dva tekstualna segmenta. Da bi se to preveniralo, može se definisati lista separatora. Na primer, separator "." označava ASCII tačku, separator ". " – Unicode tačku korišćenu u kineskom jeziku, separator "。" koristi se u japanskom i kineskom jeziku (u listu separatora za deobu dokumenta može se uvrstiti na sledeći način: "\uff0e"). [15] Vizuelni prikaz *ParentDocumentRetriever – a* opisan je na sledećoj slici:




**Slika 41.** Vizuelni prikaz *Parent Document Retriever* - a

Primer adekvatnog slučaja kada je pogodno primeniti ovaj tip instance retriever – a je situacija u kojoj postoji dokument koji opisuje zarade od različitih proizvoda, kada će više različitih opisa biti ponuđeno retriever – u, dok će vektorske reprezentacije specifičnih opisa biti generalne (na primer o zaradama uopšteno, a ne u vezi sa iznosima koji opisuju zaradu od konkretnog proizvoda za koji je korisnik postavio upit). U ovakvim situacijama bitno je da veliki jezički model izvrši poređenje između različitih kategorija zarada (na primer, veliki

jezički model uporediće podatke o zardama od prodaje proizvoda A i zaradama od prodaje proizvoda B ne bi li formirao odgovor koji se odnosi na proizvod iz korisničkog upita). [38]

Prilikom implementacije instance *ParentDocumentRetriever* instance polju istoimene klase *child_metadata_fileds* nije dodeljena vrednost, pa su u metapodacima "dete" dokumenta ostavljeni metapodaci odnosnog "roditelj" dokumenta. [37]

```
child_metadata_fields: Optional[Sequence[str]] = None
"""Metadata fields to leave in child documents. If None, leave all parent document
    metadata.
"""
```

**Slika 42.** Implementacija *Parent Document Retriever* klase u kojoj je naznačeno da su za vrednost parametra *None* prepisani svi podaci iz dokumenta roditelja u dokument dete. [37]

U implementaciji *ParentDocumentRetriever – a* može se primetiti da istoimena klasa nasleđuje *MultiVectorRetriever* klasu, odnosno da je prvi retriever podvrsta drugog, pa nasleđuje odgovarajuće attribute *MultiVectorRetriever* klase – *vectorstore: VectorStore, docstore : BaseStore[str, Document], byte_store : Optional[ByteStore], id_key : str, search_kwargs : dict, search_type : SearchType* i dodatno implementira *child_splitter : TextSplitter, parent_splitter: Optional[TextSplitter]* i *child_metadata_fields : Optional[Sequence[str]]*. [37]

### 3.3.3 Contextual Compression Retriever

Jedan od izazova u sistemima za pretragu i povratak informacija je što često nije poznato koje će tačne upite sistem za čuvanje dokumenata primiti kada se podaci unesu. To znači da se najrelevantnije informacije za određeni upit mogu nalaziti duboko u dokumentu koji sadrži mnogo irelevantnog teksta. Slanje celog dokumenta kroz aplikaciju može dovesti do skupljih poziva velikih jezičkih modela i slabijih odgovora. Kontekstualna kompresija je osmišljena da reši ovaj problem. Ideja je jednostavna: umesto da se odmah vraćaju dokumenti kakvi jesu, oni se mogu „kompresovati" u skladu sa kontekstom datog upita, tako da se vrate samo relevantne informacije. Ovde „kompresovanje" podrazumeva i skraćivanje sadržaja pojedinačnog dokumenta i uklanjanje dokumenata koji nisu relevantni. Za korišćenje klase Contextual Compression Retriever potrebni su: (1) osnovni retriever (mehanizam za pretragu dokumenata); (2) Kompresor dokumenata. *Contextual*

*Compression Retriever* prosleđuje upite osnovnom pretraživaču, koji vraća početne dokumente. Zatim prosleđuje te dokumente kompresoru dokumenata. Kompresor dokumenata uzima listu dokumenata i skraćuje ih smanjujući sadržaj dokumenata ili potpuno uklanjajući irelevantne dokumente. Ovaj proces omogućava da se samo najrelevantniji i koncizni sadržaji vraćaju kao rezultat upita, što smanjuje troškove i poboljšava kvalitet odgovora. **Error! Reference source not found.** Implementacija *Contextual Compression Retriever* klase opisana je na slici.

```
def create_retriever_with_cc(txt_chunks_path, tbl_elements_path,k_parameter): # txt i tabele
    """
    txt_chunks_path = "./data_dir/chunked_elements_1.pkl", tbl_elements_path = "./data_dir/raw_table_elements.pkl"
    """
    txt_chunks = load_pickle_file(txt_chunks_path)
    text_elements = [Element(type="text",text=element.text,page_no=dict(element.metadata.fields)['page_number']) for element in txt_chunks]
    tbl_elements = load_pickle_file(tbl_elements_path)

    # documents
    txt_docs = [Document(page_content=element.text, metadata={"page_no":element.page_no}) for element in text_elements]
    tbl_docs = [Document(page_content=element.text, metadata={"page_no":element.page_no}) for element in tbl_elements]

    all_docs = txt_docs + tbl_docs

    retriever = Chroma.from_documents(
      all_docs,OpenAIEmbeddings(api_key=os.getenv("OPENAI_API_KEY"))
    ).as_retriever()


    # sada ce iz dokumenata koje je vratio base retriever compresor da izvuce relevantne info
    llm = OpenAI(temperature=0, api_key=os.getenv("OPENAI_API_KEY"))
    compressor = LLMChainExtractor.from_llm(llm)
    compression_retriever = ContextualCompressionRetriever(
        base_compressor=compressor, base_retriever=retriever
    )
    return compression_retriever
```

**Slika 43.** Implementacija klase *Contextual Compression Retriever* [27]

*ContextualCompressionRetriever* je klasa pogodna za upotrebu kada korisničko pitanje zahteva nekoliko specifičnih činjenica iz različitih tekstualnih segmenata, koje je potrebno sintetisati zajedno. Početna ideja zasniva se na situaciji u kojoj korisnik zahteva koncizan odgovor od svega nekoliko reči, a instanca pretraživača vraća tekstualni segment pun dodatnih informacija. Cilj je da do modela u fazi generisanja stignu samo neophodne informacije, zbog njegovih limitirajućih sposobnosti kada je reč o veličini. [39]

### 3.3.4 Ensemble Retriever

*Ensemble Retriever* koristi listu različitih pretraživača kao ulaz i kombinuje rezultate njihovih metoda *get_relevant_documents(),* zatim rangira rezultate koristeći algoritam za fuziju recipročnog ranga. Kombinovanjem prednosti različitih algoritama, instance *EnsembleRetriever* klase može postići bolje performanse od bilo kog pojedinačnog

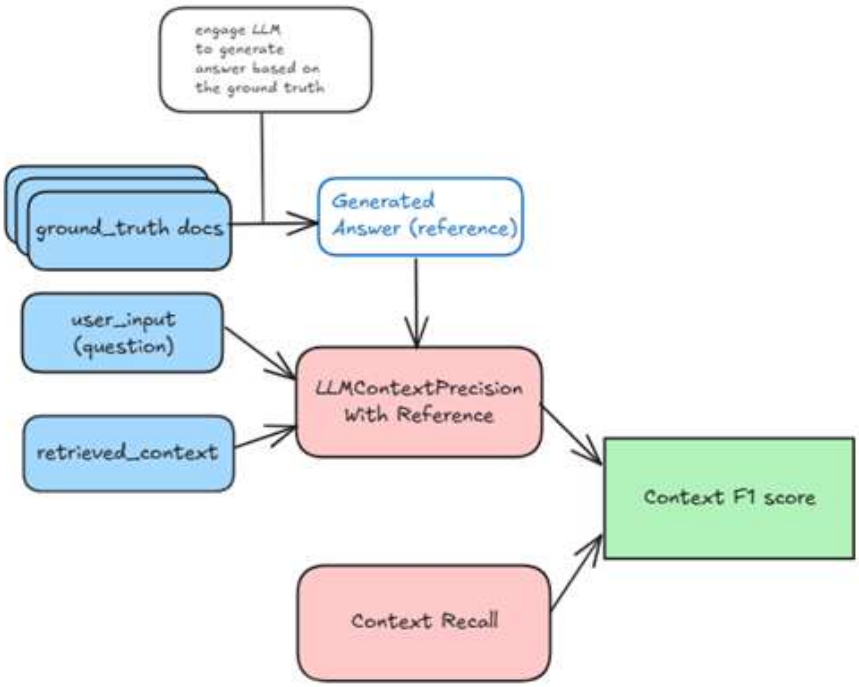
algoritma. Najčešći obrazac je kombinovanje sparsnog pretraživača (kao što je BM25) sa densnim pretraživačem (kao što je sličnost vektorskih reprezentacija), jer su njihove snage komplementarne. Ovaj pristup se takođe naziva hibridna pretraga**.** Sparsni pretraživač je dobar u pronalaženju relevantnih dokumenata na osnovu ključnih reči, dok je densni pretraživač dobar u pronalaženju relevantnih dokumenata na osnovu semantičke sličnosti. [40] Implementacija instance *EnsembleRetriever* klase data je na slici.

**Slika 44.** Implementacija instance klase *Ensemble Retriever* [27]

## 3.4 EVALUACIJA INSTANCI PRETRAŽIVAČA

Kada je reč o evaluaciji faze pretrage u RAG lancu, u obzir ćemo uzeti sledeće opcije: (1) Chroma baza podataka kao šretraživač; (2) Instanca klase MultiVectorRetriever sa Chroma skladištem podatakaza vektorske reprezentacije sažetaka  i InMemoryStore skladištem originalnih dokumenata; (3) Instanca klase ParentDocumentRetriever; (4) Pretraživač sa omotačem za kontekstualnu kompresiju; (5) Instanca klase EnsembleRetriever. Ove instance pretraživača biće testirane na dva načina parsiranja originalnog PDF dokumenta – upotrebom Nougat parsera, kao i upotrebom unstructured parsera. Kako je evaluacioni set podataka već napravljen, a uz to i setovi traženih dokumenata koje je potrebno da pretraživač vrati kao relevantne, promena strategije podele originalnog dokumenta na tekstualne segmente podrazumevala bi kreiranje ciljanih vrednosti za svaku strategiju. Međutim, ovaj problem može se rešiti odabirom evaluacionih metrika ove faze koje bi bile nezavisne u odnosu na veličinu tekstualnog segmenta i strategiju podele. Naša ideja bila je

da tražene dokumente asistiranjem sa velikim jezičkim modelima generišemo u finalne odgovore i da se na osnovu toga primeni (1) metoda *LLMContextPrecisionWithReference* za izračunavanje preciznosti; (2) metrika *context_recall* koja u obzir ne uzima originalne dokumente već tvrdnje koje su generisane asistencijom velikih jezičkih modela iz kontekstnog skupa dokumenata i stvarnog, ciljnog skupa dokumenata. Na kraju, obe metrike biće uzete u obzir izračunavanjem *F1 score* metrike i tako ćemo odrediti koja instanca pretraživača će biti u finalnoj arhitekturi aplikacije. U sledećem kodu opisan je način generisanja odgovora, kao i implementacija evaluacionih metrika.




**Slika 45.** Vizuelni prikaz ideje o evaluaciji *Retriever* - a

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_openai import ChatOpenAI
import os
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")

template = """
Your task is to generate complete answer based on the provided document or concatenated documents.
You need to use complete context provided to you in order to make informative answers.
You are not allowed to use external knowledge in the answer generation process.
This is your context: {context}
This is input question: {question}
"""
prompt = ChatPromptTemplate.from_template(template)

chat = ChatOpenAI(temperature = 0, api_key = OPENAI_API_KEY)

chain = prompt | chat | StrOutputParser()

answers = [chain.invoke({"context":gt, "question":question}) for question, gt in zip(questions, ground_truths) ]
```

**Slika 46.** Lanac za generisanje finalnih odgovora baziran na ciljnim dokumentima [27]

```
def prepare_docs(docs:list[Document]):
  if isinstance(docs[0], str):
    return docs
  return [doc.page_content for doc in docs]

def evaluate_one_retrieval_precision(question:str, retrieved_context:list, answer:str):
  evaluator_llm = LangchainLLMWrapper(ChatOpenAI(model="gpt-4o-mini",temperature=0,api_key=os.getenv("OPENAI_API_KEY")))
  context_precision = LLMContextPrecisionWithReference(llm=evaluator_llm)
  sample = SingleTurnSample(
      user_input = question,
      reference = answer,
      retrieved_contexts = prepare_docs(retrieved_context)
  )
  return context_precision.single_turn_score(sample)   # fali return provari kad stigne token!!!

def evaluate_one_retrieval_recall(question:str,retrieved_context:list, answer:str):
  evaluator_llm = LangchainLLMWrapper(ChatOpenAI(model="gpt-4o-mini",temperature=0,api_key=os.getenv("OPENAI_API_KEY")))
  context_recall = LLMContextRecall(llm=evaluator_llm)
  sample = SingleTurnSample(
      user_input = question,
      reference = answer,
      retrieved_contexts = prepare_docs(retrieved_context)
  )
  return context_recall.single_turn_score(sample)


def calculate_f1_score(precision, recall):
  return (2*precision*recall)/(precision + recall)

def evaluate_all_retrievals(retriever):
  """ovaj retriever treba da bude napunjen dokumentima koji su chunkovani strategijom Strategija i mora da prima pitanje u obliku {question:question_txt}"""
  precisions = []
  recalls = []
  for question, answer in zip(questions, answers):
    precisions.append(evaluate_one_retrieval_precision(question, retriever.invoke(question), answer))
    recalls.append(evaluate_one_retrieval_recall(question, retriever.invoke(question), answer))
  return np.mean(precisions), np.mean(recalls), calculate_f1_score(np.mean(precisions), np.mean(recalls))
```

**Slika 47.** Metode za evaluaciju pretraživača pomoću *ragas* biblioteke [27], [11]

Kada je reč o strategijama podele dokumenta na tekstualne segmente, anotacija i opis parametara svake od njih biće opisana u sledećoj tabeli, pri čemu važi da je combine_text_under_n_chars < new_after_n_chars < max_characters:

Tabela 5: Strategije podele PDF dokumenta u tekstualne segmente:

| | metod | Combine_text_under_n_chars | max_characters | multipage_sections | new_after_n_chars | overlap |
|---|---|---|---|---|---|---|
| **Strategija 1** | Chunk_by_title | 1000 | 1600 | True | 1200 | True |
| **Strategija 2** | Chunk_by_title | 2000 | 2500 | True | 2200 | True |
| **Strategija 3** | Chunk_by_title | 500 | 1500 | True | 1000 | True |
| **Strategija 4** | Chunk_by_title | 2500 | 4000 | True | 3800 | True |

Rezultati evaluacije prikazani su u sledećim tabelama.

Tabela 6: Rezultati evaluacije pretraživača za strategije 1 i 2

| **unstructured parser** | **Strategija 1** | | | **Strategija 2** | | |
|---|---|---|---|---|---|---|
| | **P** | **R** | **F1** | **P** | **R** | **F1** |
| ***ParentDocumentRetriever*** | 0.893 | 0.786 | 0.836 | 0.936 | 0.779 | 0.852 |
| ***ContextualCompressionRetriever*** | 0.841 | 0.443 | 0.581 | 0.833 | 0.426 | 0.564 |
| ***EnsembleRetriever*** | 0.805 | 0.794 | 0.799 | 0.826 | 0.837 | 0.832 |
| ***MultiVectorRetriever*** | 0.934 | 0.782 | 0.911 | 0.963 | 0.740 | 0.837 |

Tabela 7: Rezultati evaluacije pretraživača za strategije 3 i 4

| **unstructured parser** | **Strategija 3** | | | **Strategija 4** | | |
|---|---|---|---|---|---|---|
| | **P** | **R** | **F1** | **P** | **R** | **F1** |
| ***ParentDocumentRetriever*** | 0.816 | 0.786 | 0.801 | 0.911 | 0.851 | 0.880 |
| ***ContextualCompressionRetriever*** | 0.817 | 0.464 | 0.592 | 0.688 | 0.487 | 0.570 |
| ***EnsembleRetriever*** | 0.746 | 0.796 | 0.770 | 0.808 | 0.878 | 0.841 |
| ***MultiVectorRetriever*** | 0.865 | 0.744 | 0.800 | 0.930 | 0.889 | 0.909 |

Nad najboljom instancom pretraživača evaluirane su performanse u pretrazi pre klasifikacije *Header, Footer, ListItem* elemenata u junk = yes i junk = no klase i zašažena su značajna poboljšanja u vrednostima evaluacionih metrika.

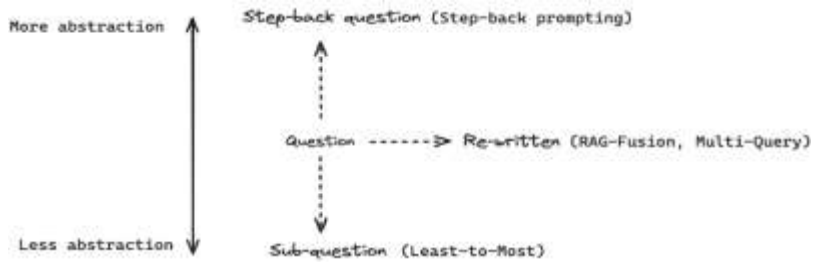
Tabela 8: Rezultati evaluacije insance klase *MultiVectorRetriever* pre i nakon klasifikacije elemenata nakon parsiranja PDF fajla u klase junk = {yes, no}

| ***MultiVectorRetriever*** | **Strategija 1** | **Strategija 2** | **Strategija 3** | **Strategija 4** |
|---|---|---|---|---|
| **F1 score pre** | 0.889 | 0.810 | 0.779 | 0.889 |
| **F1 score posle** | 0.911 | 0.837 | 0.800 | 0.909 |

Ono što se da zaključiti iz rezultata jeste da sa povećanjem veličine tekstualnog segmenta dolazi do povećanja preciznosti i smanjenja odziva. Najveća vrednost metrike F1 – score zabeležena je za strategiju 1 upotrebom *MultiVectorRetriever* instance. Iznenađujuće niske vrednosti odziva zabeležene su za instancu *ContextualCompressionRetriever* klase, za koju je istovremeno i vreme evaluacije bilo najduže (oko 13 minuta po strategiji).

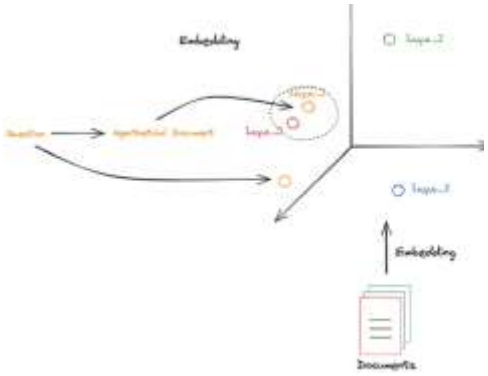
# 4 TRANSFORMACIJE KORISNIČKOG UPITA

Prva faza u naprednom RAG sistemu jeste *prevođenje upita* – preformulisanje korisničkog upita koje za cilj ima unapređenje pretrage skladišta podataka tako da najrelevantniji dokumenti stvarno konstituišu vrh rezultata pretrage. Na primer, ako je korisnički upit veoma dugačak i bogat semantičkim sadržajem, veliki broj različitih dokumenata biće vraćen pretragom vektorskog skladišta, pa će veliki jezički model halucinirati; zato je potreno obraditi korisnički upit tako da se pretragom zaista obezbede najrelevantniji dokumenti. Postoji nekoliko pristupa u rešavanju ovog problema i ovi pristupi mogu se primenjivati u skladu sa sadržajem korisničkog upita, a osnovna razlika između pristupa jeste nivo apstrakcije sa kog se polazi kada se preformuliše korisnički upit. [41]




**Slika 48.** Pristupi u transformaciji korisničkog upita prema nivou apstrakcije [8]

Nekada originalni korisnički upit u vektorskom prostoru nije najsličniji vektorskoj reprezentaciji dokumenta koji želimo da bude na vrhu rezultata pretrage; drugačija formulacija omogućuje da odgovarajući dokument obezbedi svoju poziciju u samom vrhu pretrage. Ono što se želi postići fazom prevođenja upita može se opisati sledećom slikom. Radi pojednostavljenja problema, vektorske reprezentacije dokumenata prikazane su u 3 – dimenzionalnom prostoru tačkama crvene, plave i zelene boje. Postavljanjem korisničkog upita zahteva se rangiranje crvenog dokumenta na vrh pretrage. Međutim, vektorska reprezentacija originalnog korisničkog pitanja najbliža je vektorskoj reprezentaciji drugog dokumenta; zato se tehnikom prevođenja upita (u ovom slučaju primenjena je tehnika *HyDE* o kojoj će reči biti kasnije) obezbeđuje odgovarajući dokument. [41]

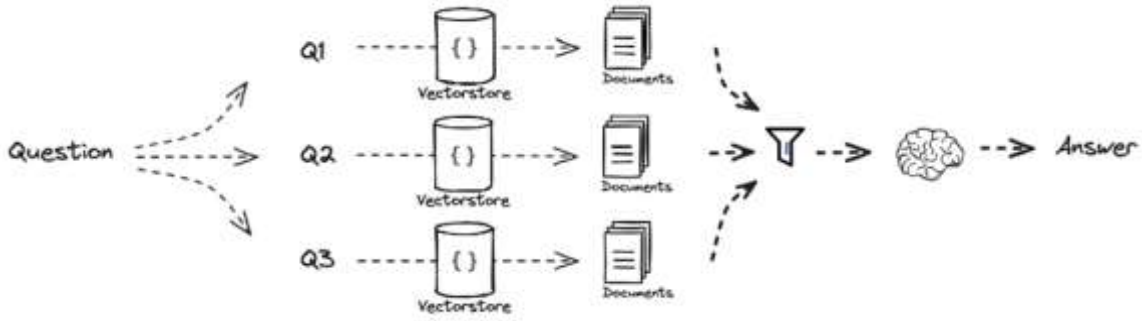


**Slika 49.** Zadatak transformacije korisničkog upita na primeru HyDE transformacije [8]

## 4.1 MULTIQUERY TRANSFORMACIJA KORISNIČKOG UPITA

Prvi pristup u rešavanju problema je preformulisanje korisničkog upita tako da nivo apstrakcije ne bude promenjen. On podrazumeva kreiranje različitih perspektiva (formulacija) početnog korisničkog upita čiji je rezultat skup različitih perspektiva originalnog korisničkog upita nad kojima se vrši pojedinačna pretraga skladišta podataka.

Rezultat ovog procesa je lista čiji su elementi liste dokumenata koji su vraćeni kao odgovor na pojedinačna pitanja, perspektive originalnog korisničkog upita. [41]




**Slika 50.** Vizuelni prikaz ideje *MultiQuery* pristupa [8]

Ovaj pristup najčešće se kombinuje sa rangiranjem rezultujućih dokumenta mehanizmom *RAG* fuzije. Nad listom rezultata pojedinačnih modifikovanih korisničkih upita generiše se set dokumenata sa numeričkim rezultatima koji opisuju njihovu relevantnost. Metrika koja se koristi je RRF (*Reciprocal Rank Fusion*), metoda za agregaciju rezultata pretrage iz više različitih izvora rangiranja sa ciljem povećanja tačnosti pronalaženja relevantnih dokumenata. [41] Formula za RRF data je sa

$$RRF\ score = \sum \frac{1}{k + rank} \tag{41}$$

gde je rank pozicija dokumenta u listi rangiranih rezultata, počevši od nule, a k je konstanta koja smanjuje uticaj dokumenata sa višim rangom tako što se za svaki dokument u rangiranoj listi računa recipročna vrednost ranga uz dodatak ove konstante . Implementacija ovog pristupa u Python – u data je na sledečoj slici.

```
def reciprocal_rank_fusion(results: list[list], k=60):
    """This method ranks documents based on the formula for fused scores and their individual rank from each question and optional parameter k"""
    fused_scores = {}
    for documents in results:
        for rank, document in enumerate(documents):
            document_string = dumps(document)
            if document_string not in fused_scores:
                fused_scores[document_string] = 0
            previous_score = fused_scores[document_string]
            fused_scores[document_string] += 1/(rank + k)

    reranked_results = [(loads(doc), score) for doc, score in sorted(fused_scores.items(), key=lambda el: el[1], reverse=True)]
    return reranked_results
```

**Slika 51.** Implementacija metode za izračunavanje recipročne rank fuzije u Python – u [27]

Za implementaciju liste različitih perspektiva originalnog korisničkog upita kreirana je *MultiQuery* klasa za strukturiranje odgovora *gpt-4o-mini* modela koja nasleđuje *BaseModel* klasu biblioteke *pydantic*, što omogućuje da različite perspektive originalnog korisničkog upita budu dostupne kao lista pitanja tipa string, bez dodatnih informacija koje bi model mogao da uvrsti u odgovor (na primer: *Here is your answer; Sure!*).

```
class MultiQuery(BaseModel):
    """Make a list of different perspectives of initial query."""
    queries: list[str] = Field(
        description="Given a user question make five different perspectives of initial query."
    )
```

**Slika 52.** Model podataka za rezultat lanca koji generiše pitanja *Multi query* strategijom [27]

Radi generisanja skupa različitih perspektiva korisničkog upita, kreiran je *langchain* lanac koji se sastoji iz šablona prompta prosleđenog *gpt-4o-mini* modelu sa strukturiranim odgovorom:

```
system = """You are an AI language model assistant. Your task is to generate five
different versions of the given user question to retrieve relevant documents from a vector
database. By generating multiple perspectives on the user question, your goal is to help
the user overcome some of the limitations of the distance-based similarity search.
Provide these alternative questions separated by newlines. Original question: {question}
"""

fusion_template = ChatPromptTemplate.from_messages([
    ("system",system),
    ("human","{question}")
])

rag_fusion_llm = chat.with_structured_output(MultiQuery)
query_generation_chain = (
    fusion_template
    | rag_fusion_llm
)
```

**Slika 53.** Implementacija *Multi Query* lanca u Python – u [27]

Radi provere da li je vraćeni dokument relevantan za korisnički upit kreiran je još jedan model podataka za strukturisani izlaz iz provere i ova logika primenjuje se nakon završetka algoritma *RRF*.

```python
class GradeDocument(BaseModel):
    """Checks if document is relevant for asked question."""
    binary_score: str = Field(
        description="Documents are relevant to the question, 'yes' or 'no'"
    )

structured_llm_grader = chat.with_structured_output(GradeDocument)
system_grader = """
You are a grader assessing relevance of a retrieved document to a user question. \n
    If the document contains keyword(s) or semantic meaning related to the user question, grade it as relevant. \n
    It does not need to be a stringent test. The goal is to filter out erroneous retrievals. \n
    Give a binary score 'yes' or 'no' score to indicate whether the document is relevant to the question
"""
grade_prompt = ChatPromptTemplate.from_messages(
    [
        ("system", system_grader),
        ("human","Retrieved document: \n\n {document} \n\n User question: {question}"),
    ],
)
retrieval_grader = grade_prompt | structured_llm_grader
```

**Slika 54.** Implementacija lanca za ocenjivanje relevantnosti dokumenata [27], [8]

Celokupna logika implementirana je u okviru *final_chain_template* lanca u okviru kog se pojedinačnim instancama pretraživača prosleđuje po jedna perspektiva originalnog korisničkog upita, a kao rezultat se dobija lista dokumenata. Sve liste predstavljaju ulaz u metodu za rangiranje dokumenata *reciprocal_rank_fusion*, nakon čega se vrši dodatna validacija pomoću *retrieval_grader* lanca. Bitno je napomenuti da je neophodno pri integraciji korisnički definisane metode u langchain lanac transformisati odgovarajuću metodu u *RunnableLambda* instancu i ulaz u takve metode mora biti izlaz iz prethodne karike lanca. Detaljnim opisima u sistemskim porukama za model obezbeđeno je da model svoje odgovore u finalnom lancu konstituiše samo na osnovu dokumenata koji su rezultat faze pretrage vektorskog skladišta, a ne na osnovu celokupnih podataka nad kojima je model treniran. Na kraju finalnog lanca korišćena je instanca klase *StrOutputParser* kako bi se obezbedio odgovor string tipa podataka, sadržaj atributa *content*, koji je jedan od atributa poslednjeg odgovora *gpt-4o-mini* modela.

```
def multiquery_retrieval_chain(retriever):
    return (query_generation_chain
                |(lambda x: x.queries)
                | retriever.map()
                | reciprocal_rank_fusion
                )

final_chain_template = """
You don't have any knowledge about statistics and machine learning.
You are only allowed to answer the questions based on the data from provided context.
Also, you are not allowed to fix answers based on your knowledge which is not provided from the context.
Your answer must be completely created from the retrieved context.
If answer is not contained in your context, you will tell to the user that you don't know the answer and sugest to the user to be more specific.
\n
Allowed context for answering the user question:
\n
\n
{context_documents}
\n
\n
Question: {question}
"""
final_prompt = ChatPromptTemplate.from_template(final_chain_template)

def final_chain(retriever):
    return (
        {"context_documents":multiquery_retrieval_chain(retriever),
        "question":itemgetter("question")}
        | RunnableLambda(grade_retrieved_documents)
        | final_prompt
        | chat
        | StrOutputParser()
    )
```

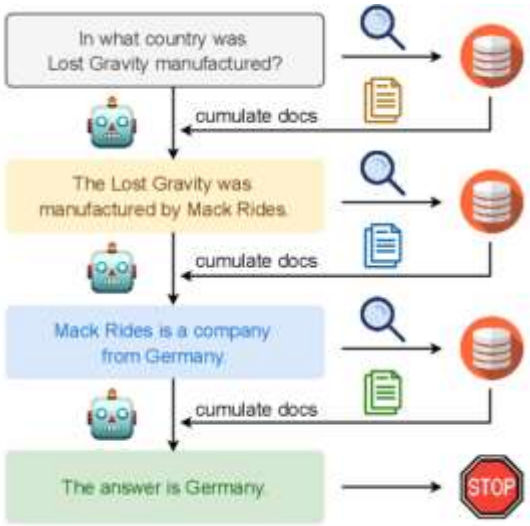
**Slika 55.** Implementacija finalnog lanca koji integriše *Multi query* transformaciju sa ocenjivačem relevantnosti [8], [27]

## 4.2 IDEJA O DEKOMPOZICIJI KORISNIČKOG UPITA

Dekompozicija je tehnika za transformaciju korisničkog upita u set potpitanja na koje se traže izolovani odgovori. Ovaj princip podrazumeva (1) rekurzivno odgovaranje na potpitanja tako da odgovor na jedno potpitanje postaje deo prompta prosleđenog instanci pretraživača koja je zadužena da da odgovor na sledeće potpitanje ili (2) pojedinačno nezavisno odgovaranje na potpitanja, kada se na svako potpitanje odgovara individualno, a potom u finalnom lancu pojedinačni odgovori se agregiraju. [41]

Ideja o dekompoziciji korisničkog pitanja na potpitanja kao polazište uzima *Od jednostavnog do složenijeg upita* pristup pomoću kojeg jezički modeli najpre rešavaju jednostavnije, manje zahtevne zadatke, pre nego što se pređe na teže i složenije zadatke – na taj način, model se postepeno priprema za rešavanje kompleksnih problema. *Razmišljanje kroz lanac mišljenja (CoT)* predstavlja tehniku u kojoj svaki model prolazi kroz niz logičkih koraka kako bi stigao do rešenja, razmišljajući "korak po korak", poput lančanog rezonovanja. Kada je reč o velikim jezičkim modelima, za dekompoziciju korisničkog upita takođe može da se koristi ovaj pristup, pomoću tzv. *inženjering upita sa*

*nekoliko primera*. Suštinski, model se podučava pomoću nekoliko primera kako da razlaže kompleksni korisnički upit u jednostavnije probleme. [41]




**Slika 56.** *Interleave retrieval with Chain of Thought (IR - CoT)* [41]

*Od jednostavnijeg do složenijeg upita* pristup rešavanju problema opisan je na zadatku povezivanja poslednjih karaktera nekog tekstualnog inputa koji je prosleđen modelu. U ovom zadatku, svaki input je lista reči iz kojih se izdvajaju poslednji karakteri reči i povezuju u jednu sekvencu. *CoT* pristup pri rešavanju ovog problema daje odlične rezultate kada je ulazna lista iste dužine kao i primeri prosleđeni promptu modela. Međutim, performanse pristupa dosta su lošije kada su liste za testiranje mnogo duže nego primeri zadati putem prompta. *Od jednostavnijeg do složenijeg upita* pristup prevazilazi ova ograničenja i daje bolje performanse na različitim dužinama ulaznih listi reči. Zadatak je opisan na sledećoj slici. [41] Ovaj pristup ulaznu sekvencu dekomponuje na subprobleme na sledeći način:

Q: "think, machine, learning"
A: "think", "think, machine", "think, machine, learning"

**Slika 57.** Dekompozicija problema na više subproblema **Error! Reference source not found.**

Na osnovu uvida u rezultat prethodnog subproblema, ovaj pristup rešava kompleksnije probleme nad proizvoljnim dužinama ulazne liste.[41]

Q: “think, machine”
A: The last letter of “think” is “k”. The last letter of “machine” is “e”. Concatenating “k”, “e” leads to “ke”. So, “think, machine” outputs “ke”.

Q: “think, machine, learning”
A: “think, machine” outputs “ke”. The last letter of “learning” is “g”. Concatenating “ke”, “g” leads to “keg”. So, “think, machine, learning” outputs “keg”.

**Slika 58.** Odgovori velikog jezičkog modela na subprobleme **Error! Reference source not found.**

*CoT* pristupom ovaj zadatak rešava se nešto drugačije – umesto da koristi rezultat prethodne obrade, CoT pristup kreira finalno rešenje problema ispočetka. **Error! Reference source not found.**

Q: “think, machine”
A: The last letter of “think” is “k”. The last letter of “machine” is “e”. Concatenating “k”, “e” leads to “ke”. So, “think, machine” outputs “ke”.

Q: “think, machine, learning”
A: The last letter of “think” is “k”. The last letter of “machine” is “e”. The last letter of “learning” is “g”. Concatenating “k”, “e”, “g” leads to “keg”. So, “think, machine, learning” outputs “keg”.

**Slika 59.** Pristup problem pomoću CoT [42]

Implementacija lanca za dekompoziciju korisničkog upita u Python – u podrazumeva definisanje šablona sistemske poruke u kojoj je opisan zadatak dekompozicije osnovnog pitanja, ova poruka prosleđena je promptu modela i invokacijom lanca komandi dobija se set dekomponovanih pitanja.

```
template = """You are a helpful assistant that generates multiple sub-questions related to an input question. \n
The goal is to break down the input into a set of sub-problems / sub-questions that can be answered in isolation. \n
Generate multiple search queries related to: {question} \n
Output (3 queries):"""

prompt_decomposition = ChatPromptTemplate.from_template(template)

generate_queries_decomposition = ( prompt_decomposition | chat | StrOutputParser() | (lambda x: x.split("\n")))

def decompose_query(question):
    questions = generate_queries_decomposition.invoke({"question":question})
    return questions
```

**Slika 60.** Implementacija lanca za dekompoziciju korisničkog upita [27], [8]

Tokom svake iteracije (postoji onoliko iteracija koliko i generisanih potpitanja) potrebno je par novog subpitanja i odgovora na subpitanje dodati u string koji čuva ove podatke tako da svaki sledeći generisani prompt modela dobije dopunjeni skup subpitanja i odgovora. Tokom pretrage, u rečniku sa odgovarajućim ključevima dostupni su podaci o kontekstu –

dokumentima koje je tekući retriever vratio kao relevantne na osnovu subpitanja korisničkog upita; subpitanje – tekuće subpitanje na koje se zahteva odgovor; parovi pitanja i odgovora – string koji sadrži istoriju prethodno postavljenih subpitanja i odnosnih odgovora.

```
def query_decomposition_recursive_answering_chain(question, retriever):
    questions = decompose_query(question)
    q_a_pairs = ""
    for q in questions:

        rag_chain = (
            {"context": itemgetter("question") | retriever,
            "question": itemgetter("question"),
            "q_a_pairs": itemgetter("q_a_pairs")}
            | decomposition_prompt
            | chat
            | StrOutputParser())

        answer = rag_chain.invoke({"question":q,"q_a_pairs":q_a_pairs})
        q_a_pair = format_qa_pair(q,answer)
        q_a_pairs = q_a_pairs + "\n---\n"+  q_a_pair
    return answer,q_a_pairs
```

**Slika 61.** Implementacija lanca za rekurzivnu dekompoziciju i lanca za generisanje finalnog odgovora [8], [41], [27]

```
# Prompt
template = """
You have no knowledge of statistics, mathematical concepts, or machine learning, except for the content provided to you.

Here is the question you need to answer:

\n --- \n {question} \n --- \n

Here are any available background question + answer pairs:

\n --- \n {q_a_pairs} \n --- \n

Here is additional context relevant to the question:

\n --- \n {context} \n --- \n

Use only the above context and any background question + answer pairs to answer the question: \n {question}
If there is no suitable answer in the provided context, simply say that you don't have that information.
"""

decomposition_prompt = ChatPromptTemplate.from_template(template)

def format_qa_pair(question, answer):
    """Format Q and A pair"""
    formatted_string = ""
    formatted_string += f"Question: {question}\nAnswer: {answer}\n\n"
    return formatted_string.strip()
```

**Slika 62.** Implementacija metode za formatiranje pitanja i odgovora kod rekurzivne dekompozicije [8], [27]

Pored rekurzivnog pristupa u kom svako sledeće potpitanje ima informaciju o skupu prethodnih subpitanja i odgovora, postoji još jedan pristup koji podrazumeva individualne

odgovore na subpitanja bez uvida u odgovore dobijene na osnovu drugih subpitanja. Kod ovog pristupa konačni skup odgovora na set subpitanja prosleđuje se modelu, kada se odgovori agregiraju.

```
prompt_rag = hub.pull("rlm/rag-prompt")

template = """You are a helpful assistant that generates multiple sub-questions related to the input question. \n
The goal is to break down the input into a set of sub-problems / sub-questions that can be answered in isolation. \n
Generate multiple search queries related to: {question} \n
Output (3 queries):"""

prompt_decomposition = ChatPromptTemplate.from_template(template)

def retrieve_and_rag(question,prompt_rag,sub_question_generator_chain,retriever):
    sub_questions = sub_question_generator_chain.invoke({"question":question})
    rag_results = []

    for sub_question in sub_questions:
        retrieved_docs = retriever.get_relevant_documents(sub_question)

        answer = (prompt_rag | chat | StrOutputParser()).invoke({"context": retrieved_docs,
                                                                 "question": sub_question})
        rag_results.append(answer)

    return rag_results,sub_questions


def format_qa_pairs(questions, answers):
    """Format Q and A pairs"""

    formatted_string = ""
    for i, (question, answer) in enumerate(zip(questions, answers), start=1):
        formatted_string += f"Question {i}: {question}\nAnswer {i}: {answer}\n\n"
    return formatted_string.strip()
```

**Slika 63.** Implementacija dekompozicije sa individualnim odgovorima [8], [27]

U okviru konteksta finalnog lanca prosleđena je lista subpitanja i odgovora na ta subpitanja u okviru string promenljive.

```
template = """Here is a set of Q+A pairs:

{context}

Use these to synthesize an answer to the question: {question}
"""

prompt = ChatPromptTemplate.from_template(template)

final_rag_chain = (
    prompt
    | chat
    | StrOutputParser()
)
```

**Slika 64.** Implementacija finalnog lanca u Python - u [8], [27]

```
generate_queries_decomposition = ( prompt_decomposition | chat | StrOutputParser() | (lambda x: x.split("\n")))
answers, questions = retrieve_and_rag(question, prompt_rag, generate_queries_decomposition,retriever)
context = format_qa_pairs(questions, answers)
```

**Slika 65.** Sadržaj *main* metode pri dekompoziciji korisničkog upita sa individualnim odgovorima [8], [27]

## 4.3 STEP BACK TRANSFORMACIJA KORISNIČKOG UPITA

*Korak unazad* je jedna od tehnika preformulisanja upita koja se zasniva na ideji poznatoj u okviru koncepta *Upitivanje korak unazad* u kojoj veliki jezički model pokušava da definiše opštije, apstraktnije pitanje u odnosu na početni korisnički upit. Ovaj metod koristi se kada postoji teorijsko znanje koje čini osnovu pitanja koje je korisnik postavio. Implementacija podrazumeva upotrebu pristupa *inženjering kroz nekoliko primera pitanja* u kojem se promptu modela daje nekoliko primera generalizacije korisničkog upita na osnovu kojih on generalizuje tekući upit. [41]

```
examples = [
    {
        "input": "Could the members of The Police perform lawful arrests?",
        "output": "what can the members of The Police do?",
    },
    {
        "input": "Jan Sindel's was born in what country?",
        "output": "what is Jan Sindel's personal history?",
    },
]
# We now transform these to example messages
example_prompt = ChatPromptTemplate.from_messages(
    [
        ("human", "{input}"),
        ("ai", "{output}"),
    ]
)
few_shot_prompt = FewShotChatMessagePromptTemplate(
    example_prompt=example_prompt,
    examples=examples,
)
prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            """You are an expert at world knowledge. Your task is to step back and paraphrase a question to a more generic step-back question,
            which is easier to answer. Here are a few examples:""",
        ),
        # Few shot examples
        few_shot_prompt,
        # New question
        ("user", "{question}"),
    ]
)
```

**Slika 66.** Implementacija *Nekoliko primera pitanja* pristupa za potrebe *Korak unazad* transformacije korisničkog upita [8], [27]

U okviru finalnog lanca prosleđuje se kontekst originalnog pitanjapretraživaču, kao i kontekst generalizovanog pitanja. Kreiran je šablon finalnog lanca koji pri odgovoru u obzir uzima kontekst koji je rezultat pretrage pri postavljanju originalnog pitanja, odgovor na generalizovano pitanje, kao i tekst originalnog pitanja. [41]

```
# Response prompt
response_prompt_template = """
You are an expert of world knowledge. I am going to ask you a question.
Your response should be comprehensive and not contradicted with the following context if they are relevant.
Otherwise, ignore them if they are not relevant.

# {normal_context}
# {step_back_context}

# Original Question: {question}
# Answer:"""
response_prompt = ChatPromptTemplate.from_template(response_prompt_template)

def step_back_chain(question,retriever):
    chain = (
        {
            "normal_context": RunnableLambda(lambda x: x["question"]) | retriever,
            "step_back_context": prompt | chat | StrOutputParser() | retriever,
            "question": lambda x: x["question"],
        }
        | response_prompt
        | chat
        | StrOutputParser()
    )
    return chain.invoke({"question": question})
```

**Slika 67.** Implementacija *Korak unazad* transformacije korisničkog upita [8], [27]

## 4.4 HYDE TRANSFORMACIJA KORISNIČKOG UPITA

*Hipotetički Dokument* je pristup u skupu metoda za transformaciju upita koji korisnički upit prevodi u oblik dokumenta, odnosno vrši pretragu u vektorskom prostoru dokumenata, u kojem se izračunava sličnost između dva dokumenta. Polazi od pretpostavke da je 1536 kao dimenzija vektorskog prostora dokumenata prevelik prostor za jedan korisnički upit i od njega pravi hipotetički dokument, u obliku naučnog rada. [41]

```
template = """Please write a scientific paper passage to answer the question
Question: {question}
Passage:"""
prompt_hyde = ChatPromptTemplate.from_template(template)

chunks_path = "results/chunked_elements_1.pkl"
txt_summaries_path = "results/1_txt_summaries.pkl"
tbl_summaries_path = "results/1_table_summaries.pkl"
retriever = create_retriever(chunks_path,txt_summaries_path,tbl_summaries_path,"similarity",5)

retrieval_chain = (prompt_hyde | chat | StrOutputParser() ) | retriever

template = """Answer the following question based on this context:

{context}

Question: {question}
"""

prompt = ChatPromptTemplate.from_template(template)

final_rag_chain = (
    prompt
    | chat
    | StrOutputParser()
)
```

**Slika 68.** Implementacija *HyDE* transformacije korisničkog upita [8], [27]

## 4.5 EVALUACIJA TRANSFORMACIJA KORISNIČKOG UPITA

Za evaluaciju različitih mehanizama u transformaciji korisničkog upita korišćena je ragas biblioteka. Pristupi su testirani na skupu raznovrsnih pitanja, međusobno različitih po svojoj **složenosti** (neka pitanja zahtevaju odgovore koji obuhvataju više zadataka), **sadržaju koji je očekivan u odgovoru** (evaluacioni set pitanja zahteva kao odgovore tekstualni sadržaj ili tabele podataka koji su sačuvani u okviru html tagova, imajući u vidu da je *gpt-4o-mini* treniran na ovakvim podacima), **dimenziji odgovora** (neki odgovori zahtevaju podatke koji se nalaze u više različitih chunkova, oslikavajući krakteristike nekog koncepta nabrojane u simbolima za nabrajanje, dok neki zahtevaju odgovor koji se sastoji od nekoliko reči i sadržaj je specifične tekstualne sekvence). Metode za transformaciju korisničkog upita korišćene su samo na tekstualnom sadržaju i tabelama. [43]


**Slika 69.** Evaluacione metrike *ragas* biblioteke [43]

Metrike za evaluaciju faze pretrage i generisanja koje ragas biblioteka implementira su:

1. **Preciznost konteksta –** evaluaciona metrika koja meri udeo relevantnih dokumenata u okviru skupa dokumenata koji je vraćen iz vektorskog skladišta. Izračunava se kao srednja vrednost preciznosti za svaki dokument u okviru vraćenih dokumenata. **Precizion@k** je racio ukupnog broja relevantnih dokumenata do pozicije k i svih dokumenata do te pozicije. Ona pokazuje koliko su prvih k dokumenata zaista relevantni u odnosu na sve dokumente do te pozicije. Vk predstavlja indikator relevntnosti na poziciji k – ako je dokument na ovoj poziciji relevantan, indikator uzima vrednost jedan, u suprotnom on je jednak nuli. Sabiranjem proizvoda preciznosti i indikatora relevntnosti za svaku poziciju od 1 do K uzima se u obzir samo rang na kojem je dokument relevantan. True positives@k označava broj relevantnih dokumenata u prvih k rezutata, koji su tačno identifikovani kao relevantni,

dok false positive@k predstavlja broj irelevantnih delova u prvih k rezultata, koji su pogrešno identifikovani kao relevantni. [44]

$$Context\ Precision@K = \frac{\sum_{k=1}^{K}(Precision@k * v_k)}{total\ no\ of\ relevant\ items\ in\ the\ top\ K\ results} \quad (42)$$

$$Precision@k = \frac{true\ positives@k}{(true\ positives@k + false\ positives@k)} \quad (43)$$

2. **Odziv konteksta** – evaluaciona metrika meri koliko relevantnih dokumenata je uspešno vraćeno iz vektorskog sladišta. Za izračunavanje ove metrike potrebno je definisati set relevantnih dokumenata manuelno, za svako pitanje evaluacionog seta pitanja. Set ovih dokumenata naziva se *ground truth,* set dokumenata koji je vraćen pretragom naziva se *context.* Metrika uzima vrednosti od nula do jedan, gde veće vrednosti ukazuju na pokriće odgovora u dokumentima sadržanim u tačnim. [45] Ovaj set dokumenata dekomponuje se na individualne iskaze i proverava se da li je vraćeni sadržaj iz konteksta sadržan u setu iskaza. Formula koja opisuje ovu metriku data je sa:

$$context\ recall = \frac{|GT\ claims\ that\ can\ be\ attributed\ to\ context|}{|Number\ of\ claims\ in\ GT|} \quad (44)$$

3. **Vernost odgovora –** evaluaciona metrika meri konzistentnost generisanog odgovora sa kontekstom sačinjenim od dokumenata vraćenih iz vektorskog skladišta. Izračunava se na osnovu finalnog odgovora koji je rezultat faze generisanja i seta dokumenata obezbeđenog kontekstom. Odgovor je skaliran na vrednosti opsega (0, 1). Generisani odgovor smatra se "vernim" ako sve tvrdnje koje su generisane iz odgovora mogu biti izvedene iz obezbeđenog konteksta. [46]

$$Faithfulness\ score = \frac{|No\ of\ claims\ in\ the\ generated\ answer\ that\ can\ be\ inferred\ from\ given\ context|}{|total\ no\ of\ claims\ in\ the\ generated\ answer|} \quad (45)$$

4. **Relevantnost odgovora –** metrika koja procenjuje koliko je generisani odgvor značajan u odnosu na postavljeno pitanje. Niži rezultat odnosi se na rezultate koji podrazumevaju nekompletne odgovore i redudansu u podacima. Matematički, metrika je

definisana preko kosinusne sličnosti originalnog pitanja sa setom "veštačkih" pitanja koji su generisani na osnovu odgovora. $Eg_i$ je vektorska reprezentacija generisanog pitanja na indeksu i, Eo označava vektorsku reprezentaciju originalnog pitanja, dok N predstavlja broj generisanih pitanja. [47]

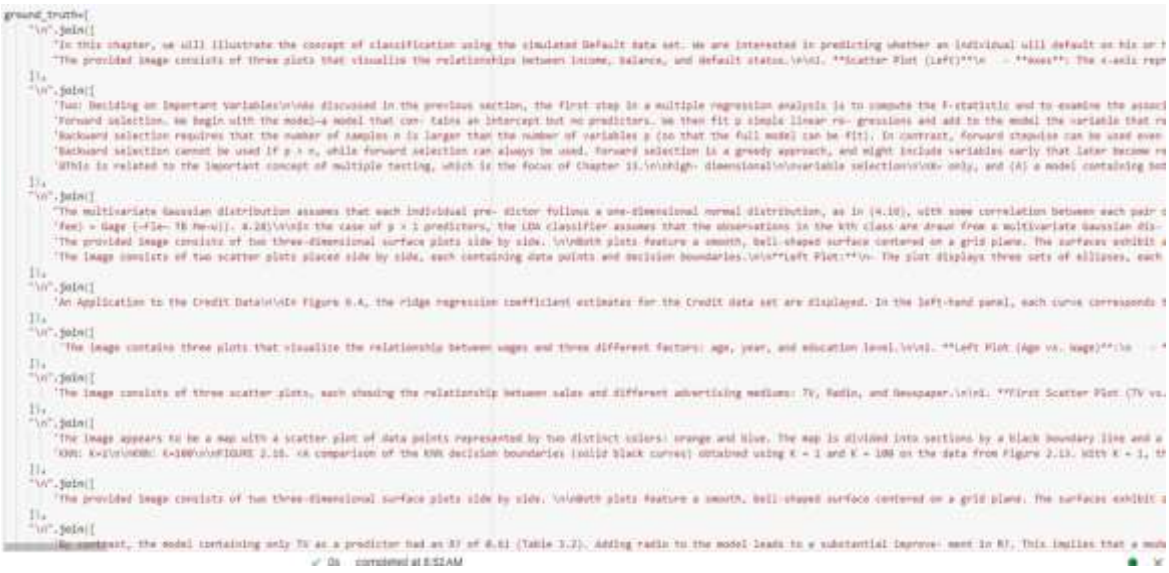
$$answer\ relevancy = \frac{1}{N}\sum_{i=1}^{N} \cos\left(E_{g_i}, E_o\right) \tag{46}$$

$$answer\ relevancy = \frac{1}{N}\sum_{i=1}^{N} \frac{E_{g_i} * E_o}{||E_{g_i}|| * ||E_o||} \tag{47}$$

Implementirana je klasa *Evaluation* u evaluation.py fajlu u kojoj su pobrojana evaluaciona pitanja, kao i ground_truth lista dokumenata za odnosno pitanje, atribut koji čuva instancu retriever – a, langchain lanac čije se performanse evaluiraju, lista za čuvanje dokumenata koji se vraćaju iz pretrage i lista konačnih odgovora.

**Slika 70.** Prikaz skupa podataka i klase za evaluaciju [27]


**Slika 71.** Prikaz atributa klase *Evaluation [27]*



**Slika 72.** Prikaz *ground truth* vrednosti [27]

**Slika 73.** Prikaz istinitih vrednosti, odgovora i liste kontekstnih dokumenata [27]

Pozivom metode za evaluaciju svaki od testnih upita prolazi kroz lanac poslat na testiranje i rezultati se evidentiraju u direktorijumu results/.

**Slika 74.** Implementacija metode za evaluaciju transformacija korisničkog upita [27]

Kompletna evaluacija gore pomenutih metoda obarde implementirana je u *evaluation_of_text_data.ipynb* fajlu. Rezultati evaluacije različitih lanaca koji implementiraju logiku za obrade početnog korisničkog upita dati su u sledećoj tabeli:

Tabela 9: Evaluacione metrike za tekstualne i tabelarne podatke upotrebom MultiVectorRetriever - a i različitih metoda transformacije korisničkih upita

| MultiVectorRetriever | Context precision | Context recall | Faithfulness | Relevancy |
|---|---|---|---|---|
| **Multiquery chain** | 0.925 | 0.740 | 0.735 | 0.779 |
| **Decomposition with recursive answering** | 0.955 | 0.684 | 0.479 | 0.678 |
| **Decomposition with individual answering** | 0.967 | 0.716 | 0.754 | 0.906 |
| **Step – back method** | **0.943** | **0.712** | **0.861** | **0.925** |
| **HyDE method** | 0.944 | 0.728 | 0.608 | 0.886 |

Drugi retriever koji je evaluiran je vektorsko skladište podataka transformisano u retriever objekat, čiji su podaci nastali parsiranjem PDF dokumenta pomoću Nougat parsera, o kojem je bilo reči. Korišćena je metoda *split_mmd_by_headers* u fajlu *chroma_vectorstore_as_retriever_with_mmd_file.py,* čiji su naslovi i podnaslovi obeleženi sa ## - Header1, ### - Header2, #### - Header3. Kako je ranije bilo reči, neke naslove poglavlja Nougat parser nije mogao da prepozna kao Header1 zato što je on treniran na tekstualnim i tabelarnim podacima iz sledećih skupova podataka: arXiv, PMC, IDL, odnosno nije obučen za prepoznavanje slika (svi nazivi poglavlja u PDF dokumentu predstavljaju sike, a ne tekstualni sadržaj), pa je neke slike parser prepoznao kao naslov veličine 1, dok je neke izostavio. Prva kombinacija parametara koja je evluirana za ovaj retriever jeste *chunk_size* = 1500*, chunk_overlap* = 30.

```python
from langchain.text_splitter import MarkdownTextSplitter, MarkdownHeaderTextSplitter
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_openai.embeddings import OpenAIEmbeddings
from langchain_chroma import Chroma
import os
from dotenv import load_dotenv

load_dotenv()

OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
def create_splits(split_chunks, chunk_size=1500,chunk_overlap=30):
  text_splitter = RecursiveCharacterTextSplitter(
      chunk_size = chunk_size, chunk_overlap = chunk_overlap
  )
  return text_splitter.split_documents(split_chunks)


def create_retriever(splits):
  vectorstore = Chroma.from_documents(documents=splits, embedding=OpenAIEmbeddings(api_key=OPENAI_API_KEY))
  retriever = vectorstore.as_retriever()
  return retriever


def split_mmd_by_headers(fpath, chunk_size=1500, chunk_overlap=30):
  if fpath == None:
    print(os.getcwd())
    mmd_file_path = "./data_dir/ISLP.mmd"
  else:
    mmd_file_path = fpath
  with open(mmd_file_path, "r", encoding="utf-8") as file:
      ISLP_mmd = file.read()
  headers_to_split_on = [
    ('##', 'Header1'),
    ('###', 'Header2'),
    ('####','Header3'),
  ]
  mmd_header_splitter = MarkdownHeaderTextSplitter(headers_to_split_on)
  split_chunks = mmd_header_splitter.split_text(ISLP_mmd)
  splits = create_splits(split_chunks, chunk_size, chunk_overlap)
  retriever = create_retriever(splits)
  return retriever

```

**Slika 75.** Implementacija pretraživača za .mmd dokument koji je parsiran nougat parserom [27]

Tabela 10: Vrednosti evaluacionih metrika za vektorsko skladište podataka kao retriever sa chunk_size = 1500 i chunk_overlap = 30

| Vektorsko skladište kao retriever; chunk_size = 1500 chunk_overlap = 30 | Context precision | Context recall | Faithfulness | Relevancy |
|---|---|---|---|---|
| **Multiquery chain** | 0.895 | 0.732 | 0.700 | 0.732 |
| **Decomposition with recursive answering** | 0.906 | 0.760 | 0.437 | 0.495 |
| **Decomposition with individual answering** | 0.901 | 0.698 | 0.694 | 0.906 |
| **Step – back method** | **0.901** | **0.738** | **0.849** | **0.878** |
| **HyDE method** | 0.907 | 0.736 | 0.581 | 0.880 |

Tabela 11: Vrednosti evaluacionih metrika za vektorsko skladište podataka kao retriever sa chunk_size = 2000 i chunk_overlap = 40

| Vektorsko skladište kao retriever; chunk_size = 2000; chunk_overlap = 40 | Context precision | Context recall | Faithfulness | Relevancy |
|---|---|---|---|---|
| **Multiquery chain** | 0.900 | 0.751 | 0.731 | 0.715 |
| **Decomposition with recursive answering** | 0.914 | 0.787 | 0.568 | 0.656 |
| **Decomposition with individual answering** | 0.917 | 0.792 | 0.783 | 0.928 |
| **Step – back method** | **0.923** | **0.803** | **0.796** | **0.919** |
| **HyDE method** | 0.919 | 0.794 | 0.634 | 0.930 |

Na osnovu rezultata pri povećanju veličine jednog tekstualnog segmenta evaluacione metrike odziv konteksta, vernost odgovora i relevantnost odgovora su se povećale u odnosu

na pređašnje vrednosti, pa možemo zaključiti da u većim tekstualnim segmentima ima više tvrdnji koje su relevantne za korisničke upite u evaluacionom skupu podataka.

# 5 FINO PODEŠAVANJE COLBERT MODELA ZA PREPOZNAVANJE SLIKA I FORMULA

Modeli koji dokument i korisnički upit predstavljaju kao jedan vektor u vektorskom prostoru jesu efikasni za pretragu, ali imaju limitirajuće performanse. ColBERT je jedan od modela koji tekst predstavlja pomoću više vektora. Literatura je pokazala da su ovakvi modeli mnogo bolji u pretrazi kada je reč o problemima specifičnih domena.[4]

Pošto su dokumenti nekada veoma sadržajni, nekada jedna vektorska reprezentacija nije dovoljna da bi obuhvatila odgovore na potencijalno širok spektar korisničkih upita – potrebno je da jedna vektorska reprezentacija dokumenta bude dovoljno blizu svakom od pitanja iz širokog spektra potencijalnih pitanja u sistemima pretrage. [4]

Tokom treninga model balansira između toga da približi vektore dokumenata različitim upitima i da očuva semantičke razlike između tih upita. To znači da model ne sme postaviti sve upite i dokumente na istu poziciju u vektorskom prostoru, jer bi to umanjilo kvalitet pretrage odnosno model ne bi bio u stanju da razlikuje sve semantičke aspekte upita.[4] Sa ovom situacijom bilo je susreta tokom treniranja Sentence-BERT modela za problem klasifikacije da li formula pripada paragrafu – model je na setu podataka tokom treninga pokazivao zadovoljavajuće performanse kada je reč o metrikama za klasifikaciju, međutim kada je dobio zadatak da predvidi da li pojedinačna formula pripada ostalim paragrafima koji su van trening seta podataka, model ih je pogrešno klasifikovao predviđajući da su pojedinačne formule pripadale većini modelu nepoznatih paragrafa.

Postoji nekoliko paradigmi kada je reč o sličnosti između korisničkog upita i dokumenta:

(1) ***Sličnost bazirana na reprezentacijama*** – podrazumevaju nezavisno računanje vektorskih reprezentacija za korisnički upit i dokument, potom određivanje sličnosti pomoću raspoloživih metrika.

(2) ***Sličnost koja se zasniva na interakciji korisničkog upita i dokumenta*** - podrazumeva nezavisno izračunavanje vektorskih reprezentacija tokena korisničkih

upita i dokumenata. Najčešće se kreira matrica interakcije između svakog tokena upita i dokumenta koja se prosleđuje u neuronsku mrežu (na primer CNN, koja identifikuje lokalne obrasce u matrici interakcije kao što su podudaranje fraza ili zavisnosti između bliskih reči).

(3) ***Interakcija svih sa svima*** **–** podrazumevaju modelovanje interakcije između tokena unutar i kroz upit i dokumente u isto vreme, kao u BERT transformer arhitekturi. Primer ovakvih modela je BERT.

(4) ***Kasna interakcija*** – kombinovanje modela koji izvršavaju intrakciju korisničkih upita i dokumenata sa izračunavanjem reprezentacija dokumenata pre upoređivanja sličnosti tako da se sama interakcija između pomenutih odloži. Primer ovakvih modela je ColBERT. [4]

ColBERT model pripada grupi modela čiji je deo arhitekture implementiran pomoću koncepta kasne interakcije. Odlaganjem interakcije između upita i dokumenta može se postići jeftinije rangiranje dokumenata na osnovu slaganja sa korisničkim upitom tako što će reprezentacije dokumenata biti ranije izračunate, a ponovno rangiranje koristiće se samo za kasnije faze, omogućavajući korisnicima brze odgovore uz visok kvalitet rezultata. [4]

## 5.1 ARHITEKTURA COLBERT MODELA

Arhitektura ColBERT modela sačinjena je od: enkodera upita, enkodera dokumenta i mehanizma kasne interakcije. Za korisnički upit q enkoder korisničkog upita kreira "vreću" vektorskih reprezentacija, dok enkoder dokumenta d kreira "vreću" vektorskih reprezentacija tokena svih dokumenata. Pri izračunavanju vektorskih reprezentacija enkoderi bazirani na BERT modelu uzimaju u obzir kontekst pri njihovom izračunavanju (na primer, reč *stock* imaće drugačiju vektorsku reprezentaciju u sintagmi *stock prices* i *stock the shelves*). Za dostupne vektorske reprezentacije ColBERT izračunava metriku relevantnosti tako što za svaki vektor tokena korisničkog upita pronalazi njemu najsličniju vektorsku reprezentaciju u skupu vektorskih reprezentacija tokena dokumenta, odnosno maksimalnu kosinusnu sličnost između dva vektora tokena, koja je u literaturi poznata još kao *MaxSim* operator. Pored kosinusne, poznata je i Euklidska distanca kao metrika sličnosti između dva vektora. Kosinusna sličnost zanemaruje dužinu vektora i fokusira se samo na njegov pravac, dok L2 distanca meri direktnu geometrijsku udaljenost između dva vektora,

uzimajući u obzir i pravac i dužinu vektora. Kada su izračunate sve maksimalne sličnosti između vektora tokena korisničkog upita i dokumenta, relevantnost dokumenta za korisnički upit izračunava se kao zbir maksimalnih sličnosti za sve vektore tokena korisničkog upita. Postoji nekoliko prednosti u upotrebi ovog mehanizma interakcije upita sa dokumentom u odnosu na sofisticiranije mehanizme koji su dostupni (poznatiji kao modeli fokusirani na interakciju): pristup kasne interakcije je izuzetno efikasan po pitanju broja operacija (FLOPs) i, s obzirom na jednostavnost, on zahteva manje računarskih resursa u poređenju sa sofisticiranijim modelima sa složenim interakcijama; zbog svoje strukture ovaj pristup omogućava efikasno uklanjanje dokumenata koji nisu relevantni u početnoj fazi pretrage, bez potrebe za izračunavanjem celokupne interakcione matrice, kako je to slučaj sa sofisticiranijim mehanizmima, pa sistem može brzo odlučiti koje dokumente neće razmatrati dalje, fokusirajući se na potencijalno najrelevantnije (top k dokumenata).[4]

Kako bi se enkodirali korisnički upit i dokumenti, koristi se zajednički BERT-based enkoder, ali razlikovanje sekevence koja odgovara upitu i dokumentu, postiže se pomoću dodavanja specijalnih tokena [Q] za korisnički upit i [D] za dokument. Za dati tekstualni upit, tokenizacija se vrši pomoću tehnike WordPiece. Specijalni token [Q] dodaje se nakon [CLS] tokena koji označava sveukupnu reprezentaciju ulaza. Ako upit ima manje tokena nego što je definisano, predefinisana dužina obezbeđena je dodavanjem [PAD] tokena kao maske. Ukoliko korisnički upit ima više tokena nego što je ograničeno predefinisanom dužinom, sekvenca tokena biće skraćena do predefinisane dužine. Konačno, prolaskom kroz BERT transformer tokeni postaju vektorske reprezentacije, s posebnom pažnjom na kontekstualizaciju. Proširenje upita maskiranim tokenima omogućava BERT-u (i ColBERT-u) da ne samo analizira upit na nivou reči koje su direktno prisutne, već i da proširi upit i automatski doda relevantne termine koji mogu poboljšati rezultate pretrage. Ova tehnika omogućava modelu da dinamički prilagođava upite kako bi postigao bolju preciznost u podudaranju upita sa dokumentima i poznata je kao augmentacija upita. Kada se obezbede vektorske reprezentacije za svaki token, kontekstualizovani output se prosleđuje na obradu kroz linearni sloj. Ovaj sloj omogućuje kontrolu dimenzionalnosti vektora, koja je obično manja nego BERT fiksna dimenzija skrivenog sloja. Maksimalna dimenzija skrivenog sloja u BERT-u je 768 za osnovni model. Manja dimenzionalnost ima pozitivan uticaj na efikasnost kodiranja upita, značajno utiče na vreme izvršavanja upita posebno kada je reč o transportu vektorskih reprezentacija sa CPU na GPU, što može biti

najskuplji korak u rangiranju top k dokumenata pomoću ColBERT-a. Na samom kraju, izlazne vektorske reprezentacije se normalizuju pomoću L2 norme. Na ovaj način, izračunavanjem skalarnog proizvoda dva vektora rezultat će biti jednak njihovoj kosinusnoj sličnosti. Kada je reč o enkodiranju dokumenta, prvo se vrši segmentacija dokumenta u tokene, pri čemu sekvenca tokena počinje [CLS] tokenom, koji je ispraćen [D] tokenom (indicira da je u pitanju sekvenca tokena koja se odnosi na dokument). Za razliku od upita, tokom enkodiranja dokumenata ne dodaje se [mask] token. Nakon prolaska sekvence tokena kroz BERT transformer arhitekturu i linearni sloj, enkoder filtrira vektore koji odgovaraju simbolima interpunkcije, koji su određen unapred definsanom listom. Ovo filtriranje je neophodno da bi se smanjio broj vektorskih reprezentacija po dokumentu, jer je osnovna pretpostavka da vektorske reprezentacije interpunkcijskih znakova limitiraju efikasnost (čak i kontekstualizovane). [4]

$$E_q := Normalize(CNN(BERT("[Q]_{q_o,q_1\#\#\ldots\#}"))) \tag{48}$$

$$E_d := Filter(Normalize(CNN(BERT([\mathrm{D}]_{\mathrm{d_o d_1 \#\#} \ldots d_n \#})))) \tag{49}$$

Mehanizam kasne interakcije koristi se kako bi se odredio skor relevantnosti za svaki dokument. Definisan je kao zbir maksimalnih sličnosti kontekstualizovanih vektorskih reprezentacija svih tokena jednog upita sa vektorskim reprezentacijama za dokument. Kao metrike sličnosti koriste se već pomenute kosinusna sličnost ili L2 (Euklidska) distanca. Kosinusna sličnost iračunava se kao

$$Cosine\ Similarity = \frac{a \cdot \mathrm{b}}{||a||||b||} \tag{50}$$

gde je a·b skalarni proizvod dva vektora, ||a||, ||b|| - L2 norme vektora a i b. [4]

Kada su vektori normalizovani, L2 norme vektora a i b su 1, pa je kosinusna sličnost jednaka skalarnom proizvodu dva vektora. [4] Formula za sličnost između dokumenta i korisničkog upita može se prikazati kao

$$S_{q,d} := \sum_{i \in ||E_q||} \max_{j \in ||E_d||} E_{q_i} \cdot E_{d_j}^T \tag{51}$$

gde je $S_{q,d}$ rezultat sličnosti, a $E_q$, $E_d$ vektorske reprezentacije tokena upita i dokumenta, respektivno, pri čemu i opisuje indeks tokena upita, a j indeks tokena dokumenta.

## 5.2 OPEN - DOMAIN QUESTION ANSWERING SISTEMI

Strategija treninga ColBERT-QA podrazumeva upotrebu postojećeg modela za pretragu i vraćanje dokumenata sa ciljem prikupljanja top k dokumenata za svako pitanje trening skupa podataka, zatim, pomoću jednostavne heuristike sortiranje tih dokumenata u pozitivne ($ve^+$) i negativne ($ve^-$) primere koji se koriste za treniranje novog efektivnijeg retrievera. [4]

Sistemi za OpenQA zavise od dve komponente: *model za pretragu,* sa zadatkom da pronađe relevantne dokumente u odnosu na korisničko pitanje; *model za čitanje,* sa zadatkom da pronađe odgovor na pitanje u skupu relevantnih dokumenata. Ako model zadužen za vraćanje relevantnih tekstualnih pasaža ne pronađe relevantne dokumente, MRC (*machine reading comperhension*) model neće uvideti najbolji pasaž i sistem će imati loše performanse. Mnogi pristupi OpenQA sistema oslanjaju se na klasične modele za pretragu informacija (na primer, BM25 model) koji nisu prilagođeni za odgovaranje na pitanja, odnosno klasični modeli ocenjuju relevantnost dokumenata prema ključnim rečima, ali ne uzimaju u obzir specifične karakteristike upita. Takođe, postojeći OpenQA modeli pretrage imaju dve glavne limitacije: reprezentacije koje je naučio ovaj model grube su i neprecizne – svaki dokument predstavljen je kao jedan vektor velikih dimenzija, a kao metrika sličnosti koristi se vektorski proizvod između pitanja i vektorske reprezentacije dokumenta; takođe, postojeći sistemi očekuju pozitivne pasuse kojih često nema ili, koristeći jednostavne modele kao što je BM25 za uzorkovanje pozitivnih i negativnih primera za trening, model ne može da kreira dovoljno izazovne negativne primere ili kreira slabe pozitivne primere, što dovodi do zaključka da metode nadzora treba da budu fleksibilnije i skalabilnije. [5]

Kao rešenje ovih problema, nudi se *slabo nadgledano učenje vođeno relevantnošću* (RGS) kao efikasna strategija slabo nadgledanog učenja, koja dozvoljava modelu za pretragu da vodi svoj trening počinjući od BM25 modela koji obezbeđuje top k dokumenata relevantnih

za korisnički upit i koristi heuristiku da razvrsta pozitivne i negativne pasaže. Ovi primeri korišćeni su da se trenira efektivniji model pretrage, a ovaj proces ponavljan je 2 - 3 puta. Kod velikih korpusa dokumenata, ključno je to što RGS zahteva ponovno indeksiranje jednom ili dva puta tokom treninga, pa shodno tome, vraća pozitivne i negativne primere u 2- 3 velike serije. Na taj način, RGS omogućuje fino podešavanje enkodera dokumenata tokom treninga, puštajući ga da se zajedno prilagođava sa enkoderom upita prema složenosti zadatka tokom treninga. [5]

Mašinsko razumevanje teksta (MRC) odnosi se na familiju zadataka koja se bavi odgovaranjem na pitanja u vezi sa tekstualnim pasažima. Označičemo pasaž sa p, pitanje sa q i nazvati ih inputima u MRC zadatku; tačan odgovor obeležičemo kao $\hat{a}$, kao tekstualni deo pasaža p. U opštem obliku, OpenQA zadatak definisan je na sledeći način: za dati korpus dokumenata C i pitanje q cilj je odrediti kratak tekstualni odgovor $\hat{a}$. OpenQA je relaksirana varijacija standardnog MRC zadatka zato što pasaž p nije zadat, mora biti pronađen u korpusu, dok MRC komponenta traži odgovor u vraćenom skupu pasaža bez garancije da će odgovor biti pronađen u bilo kom od njih. [5]

U literaturi je poznat i *Inverzni zadatak zatvaranja* (ICT), koji predstavlja samonadzirani (*self - supervised*) zadatak ne bi li se trenirao enkoder modela za pretragu. Enkoder upita dobija korisničko pitanje q, a enkoder dokumenta dobija skup pasaža p, među kojima jedan sadrži odgovor na postavljeno pitanje q, a model uči da identifikuje koji je to pasaž. Kao proširenje ovog zadatka javlja se i *retrieval – augmented language modeling* (REALM), gde su enkoderi optimizovani da određuju relevantne pasaže koji mogu pomoći u zadatku maskiranog modelovanja jezika (Mask Language Modeling task), koji se koristi za učenje BERT modela, gde model pokušava da predvidi maskirane reči u rečenici na osnovu konteksta. Nakon treniranja modela poput OpenQA i REALM, enkoder dokumenta i indeks sa enkodiranim dokumentima se "zamrzavaju", a enkoder korisničkog upita fino se podešava tako da može da pretraži pasuse koji pomažu čitaču da pronađe tačan odgovor u tekstu. Kod REALMa, trening se sprovodi u 200 000 koraka, gde se ažuriranje indeksa korpusa vrši na svakih 500 koraka. Tokom treninga, enkoderi dokumenta i korisničkog upita se zajedno treniraju i adaptiraju na zadatak, a vektorske reprezentacije koje proizvode enkoderi se menjaju. Ako se indeks ne ažurira redovno, pretraga dokumenata postaće manje tačna jer dokumenti u indeksu neće odgovarati trenutnom stanju enkodera. Redovno

ažuriranje indeksa omogućuje modelu da koristi najsvežije vektorske reprezentacije dokumenata, što poboljšava preciznost pretrage. Veoma učestalo indeksiranje moglo bi postati neefikasno za trening modela (trening bi se odvijao sporije). S druge strane, kada bi se ažuriranje odvijalo veoma retko, vektorske reprezentacije dokumenta bi postale zastarele i to bi dovelo do loših rezultata pretrage, pa je interval od 500 koraka kompromis između tačnosti i efikasnosti koji su predložili autori REALM-a. Još jedan od pristupa predstavljen je kao "gusti" pretraživač dokumenata (DPA) koji direktno trenira arhitekturu sličnosti zasnovane na reprezentacijama ne bi li sortiro dokumente u grupe pozitivnih i negativnih. Za svako pitanje iz trening skupa podataka, prikupljani su ručno obeleženi pozitivni pasaži, koji služe kao dokazi, ukoliko su oni dostupni (u suprotnom koristi BM25 model kao pristup slabo nadgledanog učenja), dok se negativni pasaži uzorkuju iz rezultata naivnog modela za pretragu, prema BM25 algoritmu. Takođe, ovaj pristup jedinstvenim čini i upotreba negativnih primera unutar jedne serije podataka, gde svaki pozitivni pasus iz serije podataka trening seta postaje tretiran kao negativan za sva ostala pitanja osim za izvorno pitanje. Ovaj pristup omogućuje DPR – u da značajno nadmaši OpenQA i REALM i postaje novi standard za ekstraktivni OpenQA. [5]

## 5.3 TRENING COLBERT MODELA

Glavni pristupi treniranju i evaluaciji OpenQA modela su: ***(1) Pristup nadgledanog učenja sa "zlatnim pasažima"*** – neki OpenQA skupovi podataka sadrže dokazne ili ručno odabrane kontekstne pasaže iz kojih se mogu izvesti potpuno tačni odgovori. Ovi pasaži, poznatiji kao zlatni dokazi mogu da služe kao vodič za treniranje modela za pretragu koji je zadužen za pronalaženje relevantnih pasaža, što odgovara pristupu nadgledanog učenja. Međutim, ovaj pristup nije moguć u skupovima podataka koji sadrže samo parove pitanja i odgovora bez pasaža iz kojih se odgovori izvode. Takođe, čak i kad su pasaži ručno označeni, oni možda ne prikazuju u potpunosti sve relevantne informacije ili različite pasaže koji bi mogli biti dobri odgovori na postavljeno pitanje zbog pristrasnosti koja je načinjena kada su ovi pasaži označeni; ***(2) Pristup slabo nadgledanog učenja –*** ovaj pristup obezbeđuje svoje dokazne pasaže tokom treninga, tako što za svako pitanje q kreira kratki odgovor, niz karaktera $\hat{a}$ iz pasaža, a ovakav pasaž je tretiran kao potencijalni kandidat za pozitivne pasaže. Da bi se poboljšala tačnost, koriste se dodatne strategije ***(a) pretraga i filtriranje –*** upotreba postojećeg sistema za pretragu zajedno sa jednostavnom heuristikom da bi se

pronašli relevantni pasusi (ovo može biti pretraga pasaža rangiranih kao top k pomoću BM25 modela); ***(b) pretraga u unutrašnjoj petlji*** **–** tokom obuke modela, pasaži se pretražuju pomoću fino podešenog modela za pretragu (na primer, kod OpenQA, REALM, RAG sistema pristupajući potpunoj obuci modela za pretragu). Ova metoda zahteva značajne računarske resurse zato što model mora da izračuna predikcije tokom prolaska kroz mrežu unapred za svaku seriju trening podataka, a zatim izračuna gradijente tokom prolaska unazad ne bi li ažurirao svoje težine. Da bi se prevazišao ovaj problem, OpenQA, REALM i RAG "zamrzavaju" svoj enkoder dokumenata kada fino podešavaju svoj enkoder dokumenata modela za OpenQA, što ograničava prilagodljivost modela zadatku. Tokom pretreniranja, REALM dozvoljava ažuriranje težina enkodera dokumenata tokom obuke, što omogućuje modelu da se prilagođava podacima koje koristi, ali to dovodi do frekventnog reindeksiranja korpusa pasaža. RGS nudi skalabilno i efektivno rešenje koje dozvoljava da model za pretragu samostalno prikuplja primere za obuku dok fino podešava enkoder dokumenta i vrši indeksiranje dodatno jednom ili dva puta. [5]

ColBERT QA sistem koristi ***strategiju nadgledanja pomoću relevantnosti*** koja koristi model pretraživač treniran na heuristici za prikupljanje trening podataka u nekoliko rundi. Ovaj sistem koristi RGS da fino podesi ColBERT modele u 3 faze – ColBERT - $QA_1$, ColBERT-$QA_2$, ColBERT-$QA_3$. Za postavljeno pitanje q, vrši se tokenizacija pomoću tokenizatora BERT modela, nakon čega sekvenca tokena može biti skraćena ukoliko nadrasta maksimalne dimenzije, ili označena [MASK] tokenima ako je kraća od maksimalne dimenzije, nakon čega se vrši metoda augmentacije korisničkog upita. Isti postupak vrši se nad dokumentima, izuzev procesa augmentacije. Kako bi se kontrolisala dimenzija izlaza, primenjuje se linearni sloj na vektorske reprezentacije pitanja i upita. Za svaki dokument u ondosu na korisnički upit ColBERT računa skor sličnosti, o kojem je već bilo reči. Naivni pretraživač (poput BM25) vraća set od top k pasaža u odnosu na korisničko pitanje, a ColBERT je treniran da da pozitivnim pasažima viši skor sličnosti nego negativnim pasažima. On koristi trojke $<q, d^+, d^->$ , gde je q korisnički upit, $d^+$ pozitivni pasaž, a $d^-$ negativni pasaž. Za ocenu sličnosti pozitivnih i negativnih pasaža sa korisničkim upitom koristi se već pomenuti skor sličnosti i ovakav problem se tretira kao problem binarne klasifikacije gde se optimizacija parametara vrši pomoću funkcije gubitka sa unakrsnom entropijom. Negativni pasaži u IR aplikacijama mogu biti obezbeđeni iz top k pasaža koji su nastali primenom naivnog modela pretrage, dok pozitivni pasaži u većini

slučajeva zahtevaju označene pozitivne podatke (na primer, kao što je učinjeno u MS MARCO Ranking skupu podataka). [5]

Algoritam RGS (***Relevance-Guided Supervision***) koristi kao input set trening pitanja Q, korpus pasaža C i inicijalni pretraživač model (na primer BM25). RGS pretpostavlja da heuristika H, koja se koristi za slabo nadgledani zadatak, može da identifikuje da li pasaž sadrži kratki odgovor $\hat{a}$ i u skladu sa tim svrstava ga u set pozitivnih ili negativnih pasaža. RGS se odvija u n diskretnih rundi, gde se najčešće koristi n = 3 runde. Svaka runda primenjuje *mehanizam pretrage i filtriranja*. Linija 3 algoritma prikazuje određivanje top k dokumenata naivnog pretraživača, nakon čega heuristika određuje članice setova pozitivnih i negativnih dokumenata. Autori određuju za t parameatar, koji označava broj vraćenih relevantnih / irelevantnih dokumenata, vrednost tri, a oni su dobijeni kao podskup top k pasaža za koje je zadovoljen uslov, rezultat heuristike H. Novi model pretrage trenira se pomoću ovako kreiranih trojki. Početna hipoteza autora je da će efektivniji pretraživač vratiti tačnije i različitije pozitivne rezultate, kao i izazovnije i realnije negativne primere. Da bi se izbegao problem preteranog učenja, koristi se deterministička podela trening skupa na dva dela tako da nijedan model ne vidi isto pitanje tokom učenja i faze pretrage pozitivnih i negativnih primera. [5]

**Algorithm 1:** Relevance-guided Supervision, given a weak heuristic $H$

**Input:** Training questions $Q$ and corpus $C$
**Input:** Supervision heuristic $H$
**Input:** Retrieval model $R_0$, number of rounds $n$
**Output:** Stronger retriever $R_n$

1 **for** round $t \leftarrow 1$ ***to*** $n$ **do**
2  Index corpus $C$ for retrieval with $R_t$.
3  $Rank_t \leftarrow \{R_i.retrieve(q) : q \in Q\}$
4  $P_t = \{H.getPositives(r[0:k^+])[0:t]$
5  $: r \in Rank_t\}$
6  $N_t = \{H.getNegatives(r[0:k^-])$
7  $: r \in Rank_t\}$
8  $T_t = \{\langle q, p, n\rangle : q \in Q,$
9  $p \in P_t[q], n \in N_t[q]\}$
10  Train a new retriever $R_{t+1}$ using triples $T_t$.
11 **end**
12 **return** *Final retriever* $R_n$

**Slika 76.** Relevance - guided Supervision algoritam sa slabom heuristikom [5]

Neposredno pre startovanja RGS sistema kreiraju se jednaki podskupovi celokupnog trening skupa podataka i svaki ColBERT-QA model trenira se pomoću 50% trening pitanja. Prva

faza treninga je početna obuka, gde se polazi od osnovnog modela BM25, čiji trening rezultuje ColBERT-$QA_1$ modelom. Primenom RGS pristupa, postojeći pretraživač se poboljšava iterativno, RGS se primenjuje u dva kruga što rezultuje novim verzijama modela, ColBERT-$QA_2$ i ColBERT-$QA_3$. [5]

ColBERT-QA koristi BERT enkoder kao čitač OpenQA sistema. Ovaj model uzima kao input [CLS] q [SEP] d [SEP] sekvencu, za svaki dokument d i top k set. Reader model ocenjuje svaki isečak relevantnog dokumenta d, a zatim se modeluju verovatnoće date sa P(s | d) za svaki span dokumenta, a ova verovatnoća može se iraziti kao

$$P(s|d) \propto \text{MLP}(h_{start}(\text{s}); h_{end}(\text{s})) \tag{52}$$

gde je MLP – višeslojna perceptronska mreža; $h_{start}$(s) – početak span-a unutar dokumenta; $h_{end}$(s) – kraj span-a unutar dokumenta. [6]

Kao i model pretraživača, model čitač zahteva trening nad trojkama za svako pitanje trening skupa podataka, gde se ove trojke konstituišu pomoću heuristike. Za podskup isečaka $\hat{S}$ koji odgovara zlatnim odgovorima u $d^{+,}$ minimizira se gubitak dat sa

$$log \sum_{s \in \hat{S}} P(s|d^{+}) \text{ [6]} \tag{53}$$

## 5.4 FINO PODEŠAVANJE COLBERT MODELA ZA ZADATAK PREPOZNAVANJA SLIKA

Kada je reč o implementaciji, korišćen je *ragatouille* okvir, koji daje mogućnost finog podešavanja ColBERT modela i pogodan je za integraciju u RAG lanac. Ragatouille okvir poseduje klasu RAGTrainer, koja može da prima trening podatke u različitim oblicima, automatski ih transformiše u trojke <pitanje, pozitivni dokument, negativni dokument> , briše duplikate i pronalazi negativne primere dokumenata koje je teško razlikovati od pozitivnih, a koji su zbog toga korisni za treniranje. Kreiranjem instance RAGTrainer klase, učitava se osnovni model; u pitanju je već trenirani ColBERT model HuggingFace transformers biblioteke za pretragu dokumenata, *colbert-ir/colbertv2.0*. Konstruktor Ragatouille klase takođe prima kao jedan od parametara *model_name*, kao naziv modela koji je potrebno fino podesiti za zadatak prepoznavanja opisa slika, *n_usable_gpu* kao broj

GPU jedinica koji se koristi tokom finog podešavanja modela, čija je podrazumevana vrednost -1 (koristiće se sve dostupne GPU jedinice). Postoji nekoliko načina za čuvanje podataka za trening u okviru Ragatouille okvira – parovi [pitanje, pozitivni_dokument], [pitanje, dokument, oznaka_dokumenta], [pitanje, pozitivni_dokumenti, negativni_dokumenti]. Prilikom generisanja seta podataka za trening, ovaj okvir čuva podatke na disku kada generiše trojke za trening. Metoda *prepare_training_data* već pomenute klase priprema podatke sa ulaza u oblik pogodan za trening ColBERT modela, a kao ulazne parametre prima *raw_data* – listu parova [pitanje, pozitivni_dokument] ili trojke; *all_documents* – listu dokumenata koja će biti korišćena za pretragu i generisanje izazovnih negativnih primera; *data_out_path* – direktorijum koji čuva listu upita, korpus dokumenata i trojke uređene za trening modela; *num_new_negatives* – broj negativnih dokumenata koji će biti kreirani za svaki korisnički upit (podrazumevana vrednost je 10), *pairs_with_labels* – Boolean indikator kojim se precizira način na koji su podaci poslati u RAGTrainer instancu, kao i *positive_label* i *negative_label*, labele za obeležavanje pozitivnih i negativnih parova (najčešće 1 i 0). Ukratko, metoda *prepare_training_data* smešta kolekciju dokumenata kao vrednost atributa *collection* RAGTrainer objekta, set upita kao vrednost *queries* atributa, kreira *hard_negatives* primere ukoliko je to specificirano. Ako se parametar *mine_hard_negatives* podesi na True, pomoću klase SimpleMiner kreiraju se ovakvi primeri. Ideja podrazumeva odabir manjeg jezičkog modela u odnosu na izvorni jezik na kom su dokumenti napisani (svi jezici i odnosni modeli pobrojani su u enumeratoru DenseModels u paketu ragatouille.negative_miners). Za sve dokumente kolekcije kreira se indeks u kojem se čuvaju njihove vektorske reprezentacije, određuje se *min_rank* (minimalni rang rezultata koji se uzima u obzir i koristi se da bi se izbegli najrelevantniji dokumenti, jer se oni obično smatraju pozitivnim primerima i želimo da budu izostavljeni iz *hard negative* primera) i *max_rank* (maksimalni rang rezultata koji se uzima u obzir, gornja granica do koje se dokumenti uzimaju u obzir pri generisanje *hard negative* primera). Sličnost upita sa dokumentom pri rangiranju rezultata meri se metrikom kosinusne sličnosti.

```
trainer = RAGTrainer(model_name="images-fine-tune-colbert-bs6",
                     pretrained_model_name="colbert-ir/colbertv2.0")

trainer.prepare_training_data(raw_data=my_data,
                              data_out_path="./data/",
                              all_documents = full_corpus)
```

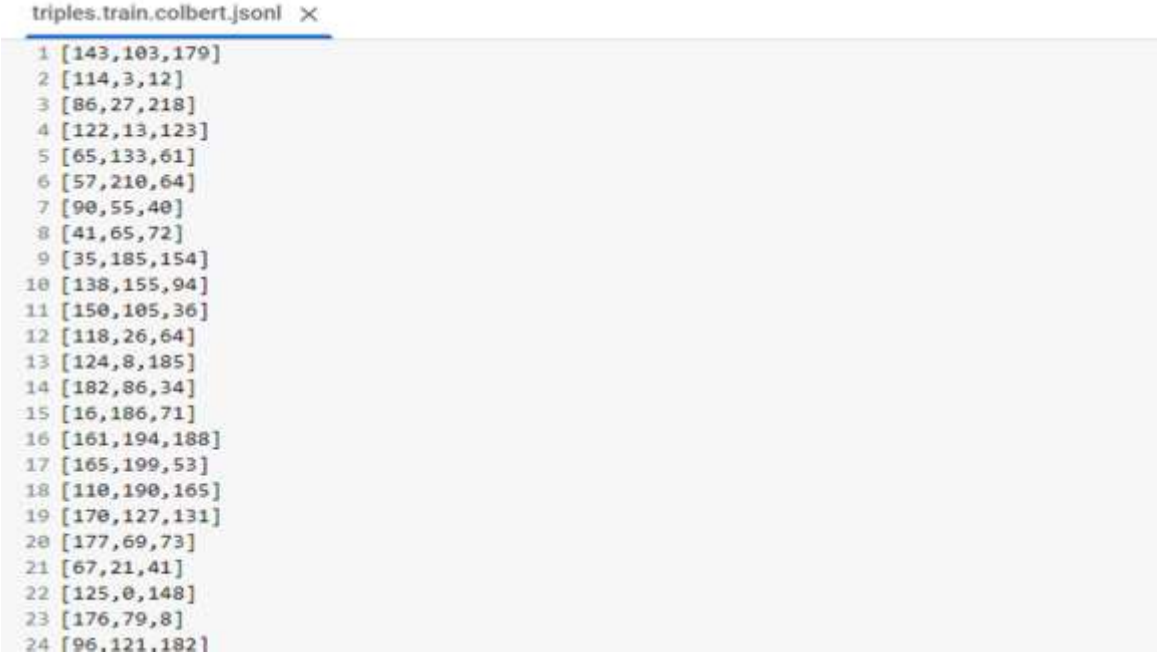
**Slika 77.** Kreiranje RAG trainer instance [27]

Konačno, rezultat metode *prepare_training_data* je putanja do direktorjuma gde su sačuvani obrađeni podaci pripremljeni za fino podešavanje ColBERT modela. Direktorijum sadrži *corpus.train.colbert.tsv* dokument (lista indeksa dokumenata i tekstualnih opisa slika u tab separated values formatu), *queries.train.colbert.tsv* dokument (lista indeksa upita i upita koji su pripremljeni za treniranje ColBERT modela, nastalih pomoću opisa grafika u PDF dokumentu) i *triples.train.colbert.jsonl* fajl, koji sadrži trojke sa indeksima upita, pozitivnog dokumenta i negativnog dokumenta.


**Slika 78.** direktorijum data/ u kojem su generisani fajlovi prepare_train_data metode


**Slika 79.** Generisani korpus corpus.train.colbert.tsv fajl

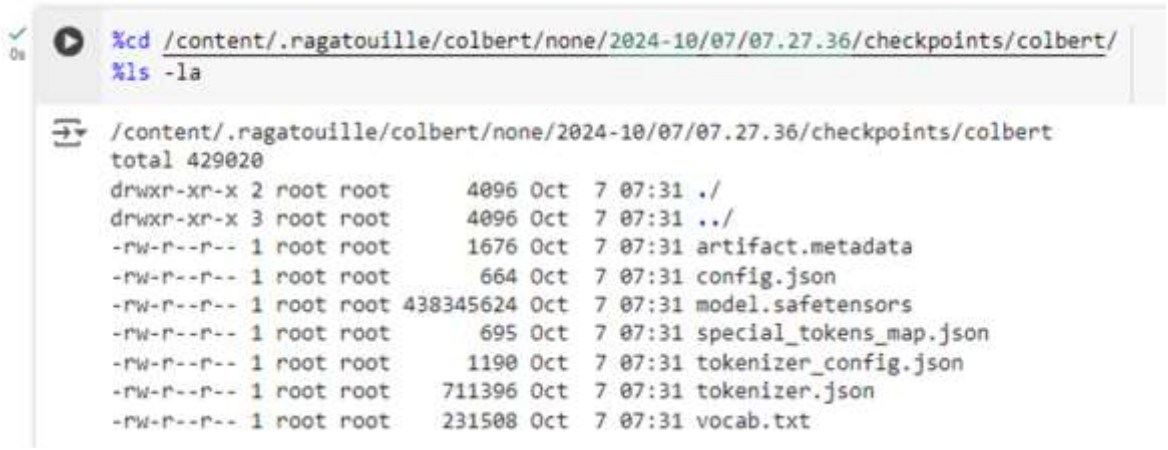

**Slika 80.** generisani queries.train.colbert.tsv fajl




**Slika 81.** generisani triplets.train.colbert.jsonl fajl

Nakon pripreme podataka, pomoću kreirane RAGTrainer instance poziva se *train* metoda koja pokreće trening ili fino podešavanje ColBERT modela. Kao parametre ova metoda prima: (1) *batch_size*, broj trojki koje se obrađuju u jednoj iteraciji tokom treninga modela (umesto da model ažurira svoje težine nakon svake pojedinačne trojke, što je veoma neefikasno, on obrađuje više trojki odjednom, nakon čega koristi informacije iz tih trojki da ažurira svoje težine). Tokom treninga ukupni *batch_size* podeljen je između više dostupnih

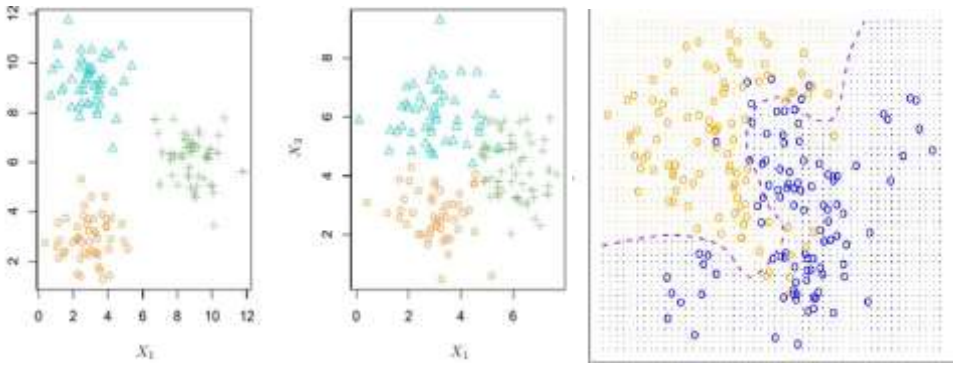
GPU – ova (na primer, ako je *batch_size* 32, a broj dostupnih GPU jedinica četiri, svaka od njih dobiće po osam trojki); (2) *nbits* – broj bita koji se koriste za kompresiju vektorskih reprezentacija dokumenata i upita, odnosno broj bita koji se koristi za reprezentaciju svake dimenzije vektora. Podrazumevano, ovaj parametar ima vrednost dva bita, što znatno smanjuje količinu memorije potrebno za skaldištenje vektorskih reprezentacija; (3) *max_steps* – parametar koji zaustavlja trening ukoliko on prevaziđe maksimalni broj koraka, odnosno maskimalan broj ažuriranja koeficijenata modela tokom treninga. Podrazumevano ovaj parametar ima vrednost 500 000 koraka; (4) *learning_rate* – stopa učenja, parametar koji određuje brzinu promena težina tokom izračunavanja funkcije gubitka tokom gradijentnog spusta; (5) *dim* – dimenzionalnost vektorske reprezentacije jednog tokena (ColBERT nakon tokenizacije pasaža kreira vektorsku reprezentaciju za svaki token - na primer ukoliko je dužina pasaža 320 tokena, dimenzionalnost tenzora koji opisuje numerički ovaj pasaž biće 320x128, ukoliko je dimenzija jednog tokena 128, što je podrazumevana vrednost); (6) *doc_maxlen* – maksimalan broj tokena pasaža (ukoliko je on duži, sadržaj koji prevazilazi ovu dužinu biće zanemaren, a ukoliko je kraći sadržaj će biti popunjen [PAD] tokenima do maksimalne dužine); (7) *warmup_steps* – procenat od ukupnog broja koraka koji se koristi za postepeno povećavanje stope učenja do definisane vrednosti (podrazumevana vrednost je 10% što znači da će u prvih 50 000 koraka, od maksimalnih 500 000 koraka, vrednost stope učenja postepeno da se povećava od 0 do 5e-6 nakon čega će trening da se nastavi upotrebom definisane stope učenja); (8) *accumsteps* – broj koraka tokom kojih će gradijenti koji su obračunati biti akumulirani, a težine modela biti ažurirane tek nakon ovog broja koraka (na primer, ako se trening izvršava na serijama podataka od 32 trojke i accumsteps parametar bude postavljen na četiri, model će efektivno raditi sa veličinom serije od 128 trojki (32 x 4) bez povećanja memorijskog opterećenja na GPU jedinicama jer će se gradijenti akumulirati tokom četiri koraka pre nego što se izvrši ažuriranje težina). Rezultat treninga je skriveni direktorijum sa checkpoint – ima treniranog modela.




**Slika 82.** Direktorijum sa fajlovima fino podešenog ColBERT modela [27]

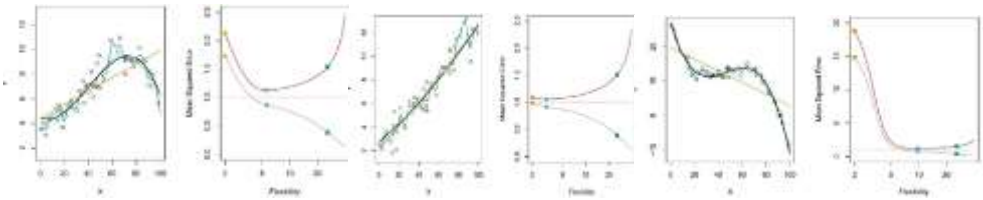
Direktorijum konstituišu (1) *artifact.metadata* fajl u kojem su skladišteni metapodaci o modelu (verzija, datum kada je model sačuvan, dodatne informacije relevantne za identifikaciju i reprodukciju modela); (2) *config.json* fajl koji sadrži konfiguracione postavke modela (hiperparametri, dimenzije modela i slično); (3) *model.safetensors* fajl koji sadrži težine i arhitekturu modela; (4) *special_tokens_map.json* – fajl koji sadrži mapu specijalnih tokena koji se koriste u tokenizaciji, kao i tokene za maskiranje i razdvajanje rečenica; (5) *tokenizer_config.json* – fajl koji sadrži konfiguracione postavke tokenizatora (struktura tokenizatora, pravila tokenizacije); (6) *tokenizer.json* – fajl za implementaciju tokenizatora koji sadrži informacije o rečniku za konverziju između reči i njihovih numeričkih predstavnika; (7) *vocab.txt* – fajl koji sadrži rečnik za tokenizaciju u kojem je svaki token povezan sa jedinstvenim identifikatorom (ID) koji se koristi pri konverziji između reči i njihovih numeričkih predstavnika tokom obrade podataka. Ragatouille okvir koristi *train* metodu definsanu u colbert.training paketu u training.py fajlu, koja prima trenirani ColBERT model za zadatke IR tipa koji je prosleđen RAGTRainer instanci zajedno sa parametrima prosledjenim metodi train RAGTrainer klase. Definiše se instanca AdamW optimizatora, koja predstavlja varijantu Adam algoritma sa komponentom koja reguliše težine, *weight decay*. Za funkciju gubitka uzeta je *CrossEntropyLoss* funkcija i prolaskom kroz trening petlju računa se kosinusna sličnost između upita i dokumenata (pasaža) koji pripadaju jednoj seriji podataka za trening, a rezultati obračunati ovom metrikom sličnosti prosleđuju se instanci CrossEntropyLoss klase PyTorch biblioteke ne bi li se izračunao gubitak za seriju podataka.

Za zadatak prepoznavanja slika na osnovu korisničkog upita bilo je potrebno fino podesiti ColBERT model. Iako je korišćen gpt-4o model za prepoznavanje sadržaja na slikama i njegovu transformaciju u tekstualnu reprezentaciju pogodnu za primenu pretraživača, na osnovu sadržaja većine korisničkih upita pretraživač nije mogao da vrati odgovarajuću sliku zato što sadržaj slike nije sadržao informacije koje su relevantne za korisnički upit, pa samim tim tekstualni opisi slika nisu bili sadržani u top k dokumenata. Primeri ovakvih slika dati su u nastavku:



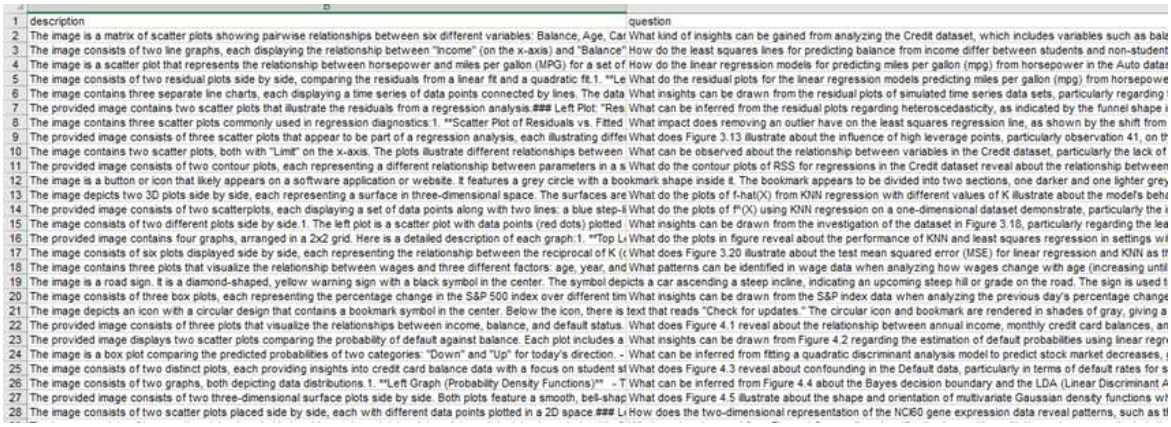
**Slika 83.** Primeri slika koje retriever nije mogao da prepozna kao odgovor na korisnički upit kada su skladišteni opisi slika [28]

Tekstualni opis druge prikazane slike bio bi: *The image displays a scatter plot with two distinct groups of data points. One group consists of orange circles and the other of blue circles. The plot has a background grid, and a purple dashed line curves through the middle, indicating a boundary that separates the two groups. The orange circles are mostly located on the upper left side of the plot, while the blue circles are concentrated on the lower right side. The boundary appears to be non-linear.*, dok bi upit koji korisnik potencijalno može da postavi, a odnosi se na kontekst slike imao bi sledeći sadržaj: *Show me figure which displays Bayes decision boundary for the simulated data set consisting of 100 observations.* Drugi tip problema koji postoji u skupu slika bio je još kompleksniji, a odnosio se na slike koje nisu sadržale relevantne podatke sa kojima njihov tekstualni opis može biti povezan sa korisničkim upitom, a koje su, pri tome, veoma slične:



**Slika 84.** Veoma slične slike koje prikazuju na levim grafikonima procene modela linearne regresije,dve spline fits funkcije i različitie stvarne vrednosti izlazne varijable i na desnim grafikonima trening i test MSE krive. [28]

Prilikom finog podešavanja ColBERT modela, kao skup dokumenata korišćen je set opisa slika koje je opisao gpt-4o model u obliku tekstualnih reprezentacija, a pitanja koja se postavljaju u cilju vraćanja ovih slika kreirana su na osnovu opisa slika iz knjige, sa ciljem da budu što opširnija i obuhvate što više detalja ne bi li model mogao da vrati odgovarajuću sliku na širok broj varijanti korisničkih upita. Lista ovakvih pitanja zajedno sa odnosnim opisima slika skladištena je u *images_all_data.csv* fajlu.



| | description | question |
|---|---|---|
| 1 | description | question |
| 2 | The image is a matrix of scatter plots showing pairwise relationships between six different variables: Balance, Age, Car | What kind of insights can be gained from analyzing the Credit dataset, which includes variables such as bal |
| 3 | The image consists of two line graphs, each displaying the relationship between "Income" (on the x-axis) and "Balance" | How do the least squares lines for predicting balance from income differ between students and non-student |
| 4 | The image is a scatter plot that represents the relationship between horsepower and miles per gallon (MPG) for a set of | How do the linear regression models for predicting miles per gallon (mpg) from horsepower in the Auto data |
| 5 | The image consists of two residual plots side by side, comparing the residuals from a linear fit and a quadratic fit.1. **Le | What do the residual plots for the linear regression models predicting miles per gallon (mpg) from horsepowe |
| 6 | The image contains three separate line charts, each displaying a time series of data points connected by lines. The data | What insights can be drawn from the residual plots of simulated time series data sets, particularly regarding |
| 7 | The provided image contains two scatter plots that illustrate the residuals from a regression analysis.### Left Plot: "Res | What can be inferred from the residual plots regarding heteroscedasticity, as indicated by the funnel shape |
| 8 | The image contains three scatter plots commonly used in regression diagnostics:1. **Scatter Plot of Residuals vs. Fitted | What impact does removing an outlier have on the least squares regression line, as shown by the shift from |
| 9 | The provided image consists of three scatter plots that appear to be part of a regression analysis, each illustrating differ | What does Figure 3.13 illustrate about the influence of high leverage points, particularly observation 41, on th |
| 10 | The image contains two scatter plots, both with "Limit" on the x-axis. The plots illustrate different relationships between | What can be observed about the relationship between variables in the Credit dataset, particularly the lack of |
| 11 | The provided image consists of two contour plots, each representing a different relationship between parameters in a s | What do the contour plots of RSS for regressions in the Credit dataset reveal about the relationship between |
| 12 | The image is a button or icon that likely appears on a software application or website. It features a grey circle with a bookmark shape inside it. The bookmark appears to be divided into two sections, one darker and one lighter grey | |
| 13 | The image depicts two 3D plots side by side, each representing a surface in three-dimensional space. The surfaces are | What do the plots of f-hat(X) from KNN regression with different values of K illustrate about the model's beh |
| 14 | The provided image consists of two scatterplots, each displaying a set of data points along with two lines: a blue step-li | What do the plots of f^(X) using KNN regression on a one-dimensional dataset demonstrate, particularly the |
| 15 | The image consists of two different plots side by side.1. The left plot is a scatter plot with data points (red dots) plotted | What insights can be drawn from the investigation of the dataset in Figure 3.18, particularly regarding the lea |
| 16 | The provided image contains four graphs, arranged in a 2x2 grid. Here is a detailed description of each graph:1. **Top L | What do the plots in figure reveal about the performance of KNN and least squares regression in settings wi |
| 17 | The image consists of six plots displayed side by side, each representing the relationship between the reciprocal of K (1 | What does Figure 3.20 illustrate about the test mean squared error (MSE) for linear regression and KNN as t |
| 18 | The image contains three plots that visualize the relationship between wages and three different factors: age, year, and | What patterns can be identified in wage data when analyzing how wages change with age (increasing until |
| 19 | The image is a road sign. It is a diamond-shaped, yellow warning sign with a black symbol in the center. The symbol depicts a car ascending a steep incline, indicating an upcoming steep hill or grade on the road. The sign is used t | |
| 20 | The image consists of three box plots, each representing the percentage change in the S&P 500 index over different tim | What insights can be drawn from the S&P index data when analyzing the previous day's percentage change |
| 21 | The image depicts an icon with a circular design that contains a bookmark symbol in the center. Below the icon, there is text that reads "Check for updates." The circular icon and bookmark are rendered in shades of gray, giving a | |
| 22 | The provided image consists of three plots that visualize the relationships between income, balance, and default status. | What does Figure 4.1 reveal about the relationship between annual income, monthly credit card balances, an |
| 23 | The provided image displays two scatter plots comparing the probability of default against balance. Each plot includes a | What insights can be drawn from Figure 4.2 regarding the estimation of default probabilities using linear regr |
| 24 | The image is a box plot comparing the predicted probabilities of two categories: "Down" and "Up" for today's direction. - | What can be inferred from fitting a quadratic discriminant analysis model to predict stock market decreases, |
| 25 | The image consists of two distinct plots, each providing insights into credit card balance data with a focus on student st | What does Figure 4.3 reveal about confounding in the Default data, particularly in terms of default rates for s |
| 26 | The image consists of two graphs, both depicting data distributions.1. **Left Graph (Probability Density Functions)** - T | What can be inferred from Figure 4.4 about the Bayes decision boundary and the LDA (Linear Discriminant A |
| 27 | The provided image consists of two three-dimensional surface plots side by side. Both plots feature a smooth, bell-shap | What does Figure 4.5 illustrate about the shape and orientation of multivariate Gaussian density functions wi |
| 28 | The image consists of two scatter plots placed side by side, each with different data points plotted in a 2D space.### Le | How does the two-dimensional representation of the NCI60 gene expression data reveal patterns, such as t |

**Slika 85.** images_all_data.csv fajl

Tokom evaluacije podataka korišćen je beir (Benchmarking Information Retrieval) paket sa sledećim metrikama:

1. **NDCG (Normalized Discounted Cumulative Gain) –** metrika meri kvalitet rangiranja rezultata pretrage tako što uzima u obzir ne samo relevantnost ponuđenih dokumenata, nego i njihov rang tj. poziciju u rezultatima pretrage. Ova metrika naglašava veći značaj visokorangiranih dokumenata. **DCG – Discounted Cumulative Gain** je metrika po kojoj se relevantnost dokumenta diskontuje zavisno od pozicije na način opisan formulom gde je

$$DCG_k = \sum_{i=1}^{k} \frac{rel_i}{log_2(i+1)} \tag{54}$$

**k** – broj dokumenata koji se uzima u razmatranje; **rel$_i$** – *relevance score* dokumenta na poziciji i; **log$_2$(i+1)** – diskontovanje relevantnosti u zavisnosti od pozicije (što je dokument niže rangiran, "diskont" ima veću vrednost tj. kreće se po logaritamskoj funkciji). Zatim, potrebno je izračunati **IDCG (Ideal Discounted Cumulative Gain)**, odnosno najveći mogući DCG potignut pri idealnom rangiranju, kada su svi relevantni dokumenti savršeno rangirani. Računa se na isti način kao i DCG, ali za idealno rangiranje:

$$IDCG_k = \sum_{i=1}^{k} \frac{rel_{ideal,i}}{log_2(i+1)} \tag{55}$$

gde je rel $_{ideal,i}$ – relevantnost dokumenta u idealnom slučaju. Konačno, NDCG metrika se dobija primenom sledeće formule

$$NDCG_k = \frac{DCG_k}{IDCG_k} \tag{56}$$

2. **Recall (odziv) –** metrika koja meri udeo pronađenih relevantnih dokumenata u ukupnom broju relevantnih dokumenata, čija je formula data sa:

$$Recall = \frac{broj\ relevantnih\ pronađenih\ dokumenata}{ukupan\ broj\ relevantnih\ dokumenata} \tag{57}$$

3. **Precision (preciznost) –** metrika koja meri broj relevantnih dokumenata u odnosu na ukupan broj pronađenih (vraćenih) dokumenata. Formula za preciznost data je sa:

$$Precision = \frac{broj\ relevantnih\ pronađenih\ dokumenata}{ukupan\ broj\ pronađenih\ dokumenata} \tag{58}$$

U nastavku će biti objasnjeno zašto preciznost nije adekvatna metrika za merenje performansi sistema u ovom zadatku.

4. **MAP (Mean average precision)** - metrika koja se dobija kao prosek prosečnih preciznosti svih korisničkih upita, u kojoj je AP (average precision) dat sa:

$$AP = \frac{\sum_{k=1}^{n} Precision(k) * rel(k)}{broj\ relevantih\ dokumenata\ za\ taj\ upit} \quad (59)$$

pri čemu je n - broj vraćenih dokumenata, Precision(k) – preciznost na poziciji k, data sa

$$\frac{broj\ relevantnih\ dokumenata\ među\ prvih\ k\ rezultata}{k} \quad (60)$$

Kada uvrstimo ovu formulu u celokupni skup korisničkih upita za trening i evaluaciju, dobija se MAP

$$MAP = \frac{1}{|Q|}\sum_{q=1}^{|Q|} AP(q)\ [22] \quad (61)$$

Metrika precision nije pogodna za evaluaciju našeg skupa podataka, zato što sa promenom parametra k preciznost postaje sve manja, imajući u vidu da za svako pitanje postoji samo jedan relevantan dokument, odnosno jedan tekstualni opis slike kojoj odgovara korisnički upit. Na primer, ako je u skupu relevantnih dokumenata za k = 1 pronađen odgovarajući, onda izračunavanje preciznosti za k = 3, 5, 10 doneće manje vrednosti preciznosti jer su ostalih k – 1 dokumenata označeni kao irelevantni u skupu podataka, pa će samim tim preciznost za k > 1 biti manja. Zato ćemo za evaluaciju skupa podataka koristiti metrike recall@k i NDCG@k.

Za evaluaciju modela kreiran je skup pitanja koji je koncizniji u odnosu na originalni skup pitanja i on je konstituisan pomoću drugačijeg vokabulara. Skup pitanja dobijen je sledećim kodom:

```
from langchain_openai.chat_models import ChatOpenAI
from langchain.prompts import ChatPromptTemplate

template = """Your task is to make concise question from the provided descriptive question using different vocabulary. \
Here is your question: {question} \
You need to provide answer that only consists of question, with no additional information.
"""
prompt = ChatPromptTemplate.from_template(template)

llm = ChatOpenAI(model="gpt-4o-mini",temperature=0)
chain = prompt | llm
test_data = [(chain.invoke({"question":elem[0]}).content,elem[1]) for elem in my_data]
```

**Slika 86.** Implementacija koda kojim je dobijen skup pitanja [27]

Za izračunavanje evaluacionih metrika, beir biblioteka zahteva odgovarajući format podataka:

(1) korpus dokumenata – rečnik čiji su ključevi id-jevi dokumenata, a vrednosti rečnici koji sadrže ključeve *text* – sadržaj dokumenta (tekstualni opis slike), *title* – naslov dokumenta (nije obavezan, pa je u našem slučaju prosleđen prazan string). Instanca RAGTrainer klase tokom pripreme podataka dobija atribut *pid_docid_map* koji transformiše indeks dokumenta u id dokumenta iz FAISS skladišta vektora.

```
corpus_dict = dict()

for key, val in RAG.model.__dict__['pid_docid_map'].items():
  pid = key
  corpus_dict[str(val)] = {"text":full_corpus[pid], "title":""}
```

**Slika 87.** Priprema korpusa za evaluaciju [27]

(2) lista korisničkih upita – rečnik čiji su ključevi indeksi pitanja, a vrednosti pitanja.

```
queries = {str(ind):q for ind, q in enumerate(filtered_data["question"]) }
```

**Slika 88.** Priprema upita za evaluaciju [27]

(3) rečnik sa odgovarajućim očekivanim vrednostima metrike relevantnosti za svaki korisnički upit – opisi slika koji odgovaraju korisničkom upitu za skor relevantnosti dobiće vrednost 1, u suprotnom biće jednaki nuli. Zato je prilikom evaluacije potrebno normalizovati dobijene vrednosti metrike relevantnosti na opseg [0,1] upotrebom min-max normalizacije.

```
qrels_dict = dict()

qrels_dict = {str(question_id):{str(document_id):1} for question_id, document_id in zip(list(queries.keys()),answer_doc_ids)}
```

**Slika 89.** Priprema rečnika koji spaja pitanja i dokumente [27]

Za evaluaciju potrebno je proslediti listu pitanja RAGTrainer instanci, a kao rezultat evaluacije dobija se lista dokumenata za svaki korisnički upit, sa vrednostima skora relevantnosti za svaki dokument:

```
search_results = RAG.search(list(queries.values()),k=10)
```

**Slika 90.** Prolazak pitanja trening skupa kroz search metodu RagTrainer instance [27]

Rezultate *search* metode potrebno je prilagoditi fomatu pogodnom za primenu metode za izračunavanje evaluacionih metrika pomoću beir paketa; potrebno je da metrika relevantnosti bude u opsegu [0,1], a statičkoj metodi *evaluate* klase EvaluateRetrieval potrebno je proslediti rečnik čiji su ključevi id-jevi upita, a vrednosti rečnici sa id-jevima dokumenata koji su relevantni i njihove vrednosti skora relevantnosti.

```
def normalize_data(search_results):
  normalized_scores = dict()
  max_score = search_results[0]["score"]
  min_score = search_results[-1]["score"]
  for result in search_results:
    norm_score = (result["score"] - min_score) / (max_score - min_score)
    normalized_scores[str(result["document_id"])] = norm_score
  return normalized_scores
scores = normalize_data(search_results[0])
```

```
results_of_retrieval = dict()
for query_id, query_results in zip(list(queries.keys()), search_results):
  results_of_retrieval[str(query_id)] = normalize_data(query_results)
```

**Slike 91, 92.** Normalizacija rezultata pretrage najrelevantnijih dokumenata [27]

Primena metode za izračunavanje evaluacionih metrika prikazana je na sledećoj slici:

```
from beir.retrieval.evaluation import EvaluateRetrieval
ndcg, _map, recall, precision = EvaluateRetrieval.evaluate(qrels_dict, results_of_retrieval, k_values=[1, 3, 5, 10])
```

**Slika 93.** Implementacija evaluacionih metrika *beir* biblioteke [27]

Batch_size parametar ima vrednost 2, learning rate 2.1e-5. Cilj je odrediti onaj skup hiperparametara tako da je vrednost evaluacionih metrika na validacionom setu podataka najmanji, tokom promene vrednosti hiperparametra maxsteps.

Tabele 12, 13: Vrednosti evaluacionih metrika recall i NDCG za promene parametra max_steps

| recall@k | Trening | | | | test | | | |
|---|---|---|---|---|---|---|---|---|
| | k=1 | k=3 | k=5 | k=10 | k=1 | k=3 | k=5 | k=10 |
| 510 000 | 0.68834 | 0.72812 | 0.77891 | 0.79972 | 0.58873 | 0.68234 | 0.76202 | 0.78832 |
| 525 000 | 0.75 | 0.90816 | 0.95408 | 0.96939 | 0.64796 | 0.86735 | 0.92347 | 0.95918 |
| **530 000** | **0.7551** | **0.90816** | **0.96429** | **0.98469** | **0.67857** | **0.89796** | **0.94388** | **0.96939** |
| 540 000 | 0.73469 | 0.92347 | 0.93878 | 0.96939 | 0.67857 | 0.86735 | 0.88776 | 0.95408 |
| 550 000 | 0.71939 | 0.86735 | 0.93367 | 0.95918 | 0.63776 | 0.83673 | 0.89796 | 0.93367 |
| 600 000 | 0.70918 | 0.84694 | 0.91327 | 0.96429 | 0.62755 | 0.82143 | 0.87755 | 0.92857 |

| NDCG@k | Trening | | | | Test | | | |
|---|---|---|---|---|---|---|---|---|
| | k=1 | k=3 | k=5 | k=10 | k=1 | k=3 | k=5 | k=10 |
| 510 000 | 0.69848 | 0.77938 | 0.79483 | 0.80929 | 0.59834 | 0.70374 | 0.72936 | 0.75111 |
| 525 000 | 0.75 | 0.843378 | 0.86735 | 0.8679 | 0.64796 | 0.77703 | 0.8003 | 0.81175 |
| **530 000** | **0.7551** | **0.84633** | **0.86938** | **0.8761** | **0.67857** | **0.81098** | **0.82986** | **0.83809** |
| 540 000 | 0.73469 | 0.84645 | 0.85282 | 0.86349 | 0.67857 | 0.78966 | 0.79823 | 0.82039 |
| 550 000 | 0.71939 | 0.80673 | 0.83395 | 0.84239 | 0.63776 | 0.75595 | 0.78075 | 0.79241 |
| 600 000 | 0.70918 | 0.79476 | 0.82176 | 0.83812 | 0.62755 | 0.74253 | 0.76513 | 0.78189 |

Fiksiraćemo maxsteps parametar na maxsteps = 530 000 i ispitati evaluacione metrike za različite vrednosti parametra learning rate. Batch_size ostaće fiksiran na vrednost 2.

Tabele 13, 14: Vrednosti evaluacionih metrika recall i NDCG za promene parametra learning_rate:

| NDCG@k | Trening | | | | Test | | | |
|---|---|---|---|---|---|---|---|---|
| | k=1 | k=3 | k=5 | k=10 | k=1 | k=3 | k=5 | k=10 |
| 5e-6 | 0.58673 | 0.73378 | 0.7632 | 0.77856 | 0.55612 | 0.70383 | 0.73083 | 0.75537 |
| 1e-5 | 0.70408 | 0.82294 | 0.84622 | 0.85146 | 0.63776 | 0.77126 | 0.79848 | 0.80885 |
| 2e-5 | 0.66837 | 0.79555 | 0.80806 | 0.82309 | 0.58673 | 0.69472 | 0.73074 | 0.7546 |
| 2.2e-5 | 0.68878 | 0.81541 | 0.83604 | 0.83913 | 0.60714 | 0.72843 | 0.76445 | 0.78263 |
| 2.6e-5 | 0.64286 | 0.7817 | 0.79883 | 0.81235 | 0.61224 | 0.73044 | 0.74318 | 0.76202 |

| recall@k | Trening | | | | Test | | | |
|---|---|---|---|---|---|---|---|---|
| | k=1 | k=3 | k=5 | k=10 | k=1 | k=3 | k=5 | k=10 |
| 5e-6 | 0.58673 | 0.83673 | 0.90816 | 0.95408 | 0.55612 | 0.80612 | 0.87245 | 0.94898 |
| 1e-5 | 0.70408 | 0.90306 | 0.95918 | 0.97449 | 0.63776 | 0.86735 | 0.93367 | 0.96429 |
| 2e-5 | 0.66837 | 0.88265 | 0.91327 | 0.95918 | 0.58673 | 0.76531 | 0.85204 | 0.92347 |
| 2.2e-5 | 0.68878 | 0.89796 | 0.94898 | 0.95918 | 0.60714 | 0.81633 | 0.90306 | 0.95918 |
| 2.6e-5 | 0.64286 | 0.87245 | 0.91327 | 0.95408 | 0.61224 | 0.81122 | 0.84184 | 0.89796 |

Nakon promene learning rate parametra, zaključujemo da je najbolja vrednost evaluacionih metrika na validacionom setu podataka ostala za vrednost stope učenja od 2.1e-5. Sada će biti ispitana promena parametra batch_size za learning_rate = 2.1e-5, maxsteps = 530 000. Najveće vrednosti validacionih metrika dobijaju se za batch_size = 2, dok za veće vrednosti {4, 6, 16} razlika između metrika na trening i validacionom setu postaje veća za sve vrednosti k = 1.

Tabele 14, 15: Vrednosti evaluacionih metrika recall i NDCG za promene parametra batch_size:

| NDCG@k | Trening | | | | Test | | | |
|---|---|---|---|---|---|---|---|---|
| | k=1 | k=3 | k=5 | k=10 | k=1 | k=3 | k=5 | k=10 |
| 4 | 0.7398 | 0.83102 | 0.84596 | 0.85465 | 0.64286 | 0.75917 | 0.78684 | 0.8003 |
| 6 | 0.67347 | 0.80387 | 0.82562 | 0.83236 | 0.60714 | 0.7452 | 0.77309 | 0.78846 |
| 16 | 0.66837 | 0.7769 | 0.80919 | 0.81982 | 0.60714 | 0.74909 | 0.77609 | 0.78917 |

| recall@k | Trening | | | | Test | | | |
|---|---|---|---|---|---|---|---|---|
| | k=1 | k=3 | k=5 | k=10 | k=1 | k=3 | k=5 | k=10 |
| 4 | 0.7398 | 0.89286 | 0.92857 | 0.95408 | 0.64286 | 0.83673 | 0.90306 | 0.94388 |
| 6 | 0.67347 | 0.89286 | 0.94388 | 0.96429 | 0.60714 | 0.84184 | 0.90816 | 0.95408 |
| 16 | 0.66837 | 0.75085 | 0.76922 | 0.77404 | 0.60714 | 0.84694 | 0.72993 | 0.95408 |

## 5.5 FINO PODEŠAVANJE COLBERT MODELA ZA ZADATAK PREPOZNAVANJA FORMULA

Arhitektura ColBERT modela korišćena je i za zadatak pretrage formula. Celokupni PDF dokument parsiran je pomoću nougat parsera. Ono što je bio problem pri parsiranju jeste što su naslovi poglavlja predstavljeni kao slike, a ne kao tekst, pa je na nekim poglavljima ovaj parser prepoznao naslove, a negde slike, pa su takvi naslovi preskočeni. Još jedan od problema bio je i izmeštanje sadržaja opisa slika u odnosu na mesto teksta opisa slika u originalnom dokumentu; često se dešavalo da je za sliku čiji je opis na početku stranice, taj opis u .mmd fajlu, koji je rezultat parsiranja PDF - a ovim parserom, na kraju originalne stranice. Ovaj parer je odlično poslužio za prepoznavanje matematičkih formula, imajući u vidu da je nougat treniran na naučnim podacima. Upotreba ovog modela je prilično jednostavna i opisana je na sledećoj slici:

```
!nougat /content/ISLP.pdf -o /content/results/ --no-markdown --recompute --no-skipping
```

```
2024-10-16 13:46:42.467484: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on.
```

**Slika 94.** Parsiranje PDF fajla pomoću nougat parsera [27]

Napisan je kod koji prepoznaje formule u okviru celokupnog .mmd fajla i prikazan je na sledećoj slici.

```
with open("/content/results/ISLP.mmd","r",encoding="utf-8") as f:
  content = f.read()
```

```
formulas = re.findall(r'($.*?$|$$.*?$$|\

                                          .*?

)', content)
```

```
import pickle

with open("results/latex_formulas.pkl", "wb") as file:
  pickle.dump(formulas, file)
```

```
import pickle
```

**Slika 95.** Ekstrakcija formula iz .mmd fajla [27]

Nakon parsiranja, formule su povezane ručno sa odgovarajućim pasusima iz knjige iz kojih su potekle i za nekolicinu formula kreirano je pitanje čiji odgovor bi trebalo da predstavlja parsirana formula. Da bi se kreirala pitanja koja odgovaraju ostalim formulama, korišćen je *FewShotChatMessagesPromptTemplate* koji na osnovu sadržaja tekstualnog pasusa iz kog je potekla formula i do sada ručno kreiranih pitanja pomaže u kreiranju ostatka pitanja.

**Slika 96.** Kreiranje pitanja za trening ColBERT modela za prepoznavanje formula [27]

Kreiran je još jedan *FewShotChatMessagesPromptTemplate* koji povezuje pitanje i odgovor u potpuniju verziju odgovora koja sadrži u sebi pun odgovor koji sadrži reči iz pitanja, kako bi model lakše trenirao.

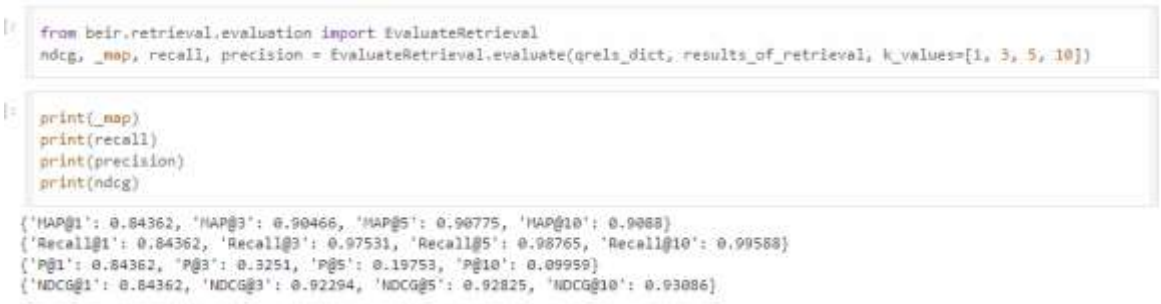

**Slika 97.** Kreiranje punih odgovora povezivanjem originalnih pitanja sa formulama [27]

Finalni skup podataka prikazan je na sledećoj slici.

| | question | answer |
|---|---|---|
| **0** | What is the formula for the support vector cla... | The formula for the support vector classifier ... |
| **1** | What is the formula for calculating the Bayes ... | The formula for calculating the Bayes error ra... |
| **2** | What is the purpose of the tuning parameter \(... | The purpose of the tuning parameter \( \lambda... |
| **3** | How do you calculate the posterior probability... | The posterior probability that a given observa... |
| **4** | What is the function F(X) derived in the conte... | The function F(X) derived in the context of us... |
| **...** | ... | ... |
| **197** | What is the formula for calculating the probab... | The formula for calculating the probability of... |
| **198** | What is the formulation of the loss function i... | The relationship between the loss function L(X... |
| **199** | How is the variance of a dataset calculated in... | The Proportion of Variance Explained (PVE) in ... |
| **200** | What is the formula for approximating the func... | The significance of the formula for the approx... |
| **201** | What is the formula for calculating the cross-... | The formula for calculating the coefficient of... |

**Slika 98.** Skup podataka nad kojim je treniran ColBERT model [27]

Identičnim treningom kao za model koji prepoznaje slike i grafike, treniran je i ColBERT za prepoznavanje formula. Korišćena je sledeća kombinacija hiperparametara: *batch_size = 2, dim = 128, doc_maxlen = 256, use_relu = False, learning_rate = 2.3e-5, nbits = 2, masteps = 540 000.* Evaluacijom pomoću *beir* biblioteke dobijeni su sledeći rezultati:



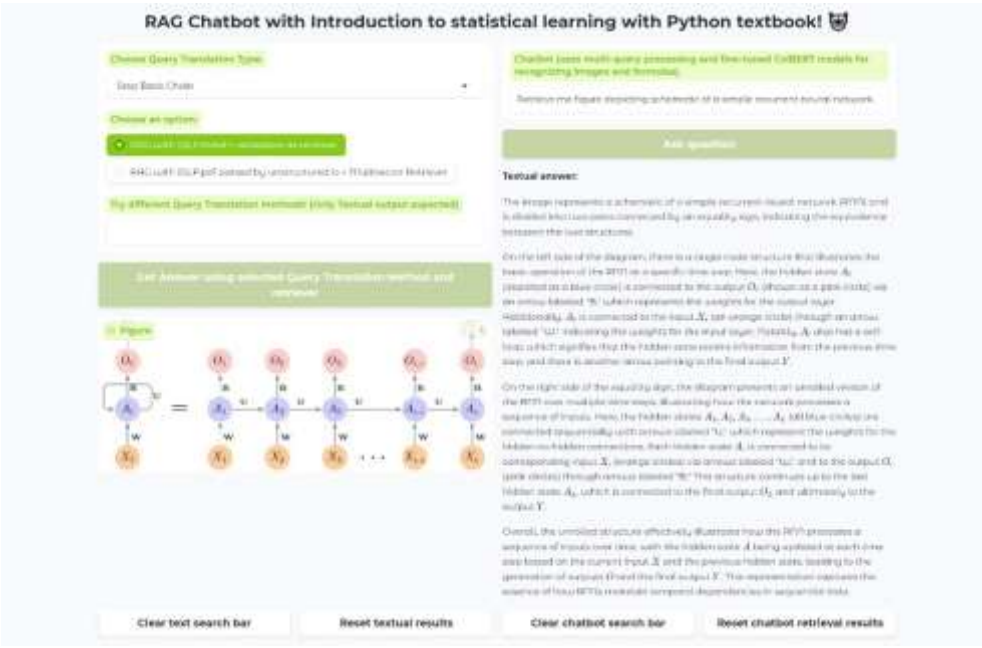

**Slika 99.** Vrednosti evaluacionih metrika za zadatak pretrage formula [27]

# 6 ARHITEKTURA APLIKACIJE

Arhitekturu aplikacije čini *langgraph* koji je implementiran radi orkestracije svih tokova u jednu celinu. On započinje čvorom koji podrazumeva izvršavanje metode *classify_intent* koja klasifikuje korisnički zahtev u jedan od tri moguća zahteva za odgovorom; (1) ukoliko je reč o odgovorima koji zahtevaju tekstualne i tabelarne informacije, onda je u pitanju *chroma index* čvor (čvor koji podrazumeva izvršenje metode *text_retriever*); (2) ukoliko je korisnik zahtevao formulu, tada se izvršni tok grafa prebacuje na *formulas index* čvor koji implementira metodu za pretragu indeksa povezanog sa fine - tune - ovanim ColBERT modelom za potrebe pretrage formula, nakon čega se izvršni tok prebacuje na čvor *formula_context* koji pronalazi tekstualni kontekst povezan sa vraćenom formulom; (3) ukoliko je korisnik zahtevao sliku, tada se izvršni tok grafa prebacuje na *images index* i izvršavanje metode *images_retriever*, nakon čega se tok izvršenja grafa prebacuje na pronalaženje tekstualnog konteksta metodom *search_for_image_context,* kao i prikazivanje slike pomoću metode *show_image* koja prikazuje odgovarajuću sliku čiji je naziv sa *bucket* - a sačuvan u metapodacima dokumenata. Za vraćanje tekstuanih podataka kreiran je *Multi Vector Retriever* sa sažecima originalnih podataka. U polju za pretragu prikazanom u levoj koloni korisnik može da bira način transformacije korisničkog upita ne bi li dobio tačniji odgovor i dokument je u ovom delu aplikacije parsiran *Nouga* ili *YOLOX* modelom, u zavisnosti od izbora korisnika. Korisnički interfejs kreiran je upotrebom *gradio* biblioteke. Detaljnija implementacija pojedinačnih metoda i korisničkog interfjesa mogu se pronaći u fajlovima *app.py* i *ui/ui.py,* respektivno. Link ka GitHub repozitorijumu u kojem se nalazi aplikacija dat je u nastavku: https://github.com/zm20200109/ISLP_CHATBOT/tree/main .

```
graph_builder = StateGraph(State)

graph_builder.add_node("intent_classifier", classify_intent)
graph_builder.add_node("formulas_index", formulas_retriever)
graph_builder.add_node("images_index",images_retriever)
graph_builder.add_node("chroma_index",text_retriever)
graph_builder.add_node("formula_context",search_for_formula_context)
graph_builder.add_node("image_context_search",search_for_image_context)
graph_builder.add_node("show_image",show_image)

graph_builder.add_edge(START, "intent_classifier")
graph_builder.add_conditional_edges(
    "intent_classifier",
    decide_next_intent,
    {
        "formulas_index":"formulas_index",
        "images_index":"images_index",
        "chroma_index":"chroma_index",
    },
)
graph_builder.add_edge("formulas_index", "formula_context")
graph_builder.add_edge("formula_context",END)
graph_builder.add_edge("images_index","image_context_search")
graph_builder.add_edge("image_context_search","show_image")
graph_builder.add_edge("show_image",END)
graph_builder.add_edge("chroma_index",END)

graph = graph_builder.compile()
```

**Slika 100.** Implementacija *langgraph* strukture [27]




**Slika 101.** Korisnički interfejs sa primerom pretrage slika [27]

# 7 ZAKLJUČAK

Tokom merenja prosečne brzine odziva na set upita evaluacionog skupa podataka zabeleženo je da je *Chroma* baza podataka brža za 0.068765 sekundi u odnosu na *FAISS,* čija je prosečna brzina 0.28165427007 sekundi po upitu. Pri evaluaciji pretraživača utvrđeno je da je merenjem kontekstnog odziva i *LLMContextPrecisionWithReference* evaluacionih metrika najbolja vrednost F1 metrike zabeležena upotrebom instance klase *MultiVectorRetriever* sa sažecima originalnih dokumenata, pri čemu je kao najbolja strategija separatisanja PDF fajla zabeležena Strategija 1, čiji je metod *chunk_by_title* i parametric dati sa *combine_text_under_n_chars* = 1000, *max_characters* = 1600, *multipage_sections* = True, *new_after_n_chars* = 1200, *overlap* = True. Najlošije vrednosti na evaluacionom setu zabeležene su upotrebom instance *ContextualCompressionRetriever* (za 0.33 manja vrednost F1 metrike od najboljeg pretraživača). Ovaj tip pretraživača odlikuje veoma nizak odziv na evaluacionom skupu podataka. Utvrđena su poboljšanja nakon klasifikacije *Header, Footer* i *ListItem* elemenata (F1 metrika uvećana je za 0.022 uz upotrebu *MultiVectorRetriever* klase). Evaluacije transformacije korisničkog upita vršene su za *MultiVectorRetriever* klasu nad podacima koji su parsirani *unstructured* parserom, kao i *Nougat* modelom (korišćene su dve kombinacije parametara *RecursiveCharacterTextSplitter* klasom), pri čemu su oba slučaja najbolje vrednosti postignute *Korak Unazad* transformacijom (*vernost odgovora* = 0.796 i *relevantnost odgovora* = 0.919 za postavke *chunk_size* = 2000 i *chunk_overlap* = 40). Za *MultiVectorRetriever* instancu, kao najbolji pretraživač čiji su segmenti dobijeni *unstructured* parserom i parametrima Strategije 1 dobijene su vrednosti *vernost odgovora* = 0.861, *relevantnost odgovora* = 0.925 pomoću *Korak Unazad* transformacije.

Za potrebe zadatka pretrage formula izračunati su Recall@1 = 0.844, Recall@3 = 0.975, Recall@5 = 0.98765, Recall@10 = 0.99588 i NDCG@1 = 0.84362,NDCG@3 = 0.923, NDCG@5 = 0.928, NDCG@10 = 0.931. Rezultati za Recall@1, Recall@3, Recall@5 i Recall@10 ukazuju na izuzetnu sposobnost sistema da vraća relevantne formule, sa posebno visokim vrednostima u nižim rangovima (Recall@1 = 0.844) i stabilnim poboljšanjem u višim rangovima (Recall@5 = 0.98765, Recall@10 = 0.99588). To znači da sistem često nalazi odgovarajuće rezultate na samom početku liste, ali i dalje uspešno proširuje pretragu kako bi uključio relevantne formule, čak i kada se proširuje broj kandidata za pretragu.

Slično tome, NDCG metričke vrednosti (NDCG@1 = 0.84362, NDCG@10 = 0.931) pokazuju visok kvalitet rangiranja, pri čemu je kvalitet rezultata stabilan kroz rangove. Visoke vrednosti za NDCG sugerišu da sistem ne samo da pronalazi relevantne odgovore, već i pravilno rangira formule prema njihovoj relevantnosti. Za potrebe prepoznavanja slika u odnosu na korisničke upite treningom ColBERT modela dobijene su nešto niže vrednosti evaluacionih metrika na testnom skupu pitanja u samom vrhu k dokumenata pretrage, ali je do k = 5 odziv 0.944, a NDCG 0.830 na testnom skupu pitanja.

## LITERATURA


[1] Ge, Z., Liu, S., Wang, F., Li, Z., Sun, J. (2021). YOLOX: Exceeding YOLO series in 2021. *arXiv preprint arXiv:2107.08430*. https://arxiv.org/pdf/2107.08430

[2] Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2016). You only look once: Unified, real-time object detection. *arXiv preprint arXiv:1506.02640*. https://arxiv.org/pdf/1506.02640

[3] Vaswani, A., Jones, L., Shazeer, N., Parmar, N., Gomez, A. N., Uszkoreit, J., Kaiser, Ł., & Polosukhin, I. (2023). Attention is all you need. *arXiv preprint arXiv:1706.03762v7*. https://arxiv.org/pdf/1706.03762

[4] Khašab, O., & Zaharia, M. (2020). ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. *arXiv preprint arXiv:2004.12832*. https://arxiv.org/pdf/2004.12832

[5] Khattab, O., Potts, C., Zaharia, M. (2020). Relevance-guided Supervision for OpenQA with ColBERT. *arXiv preprint arXiv: 2007.00814*. https://arxiv.org/pdf/2007.00814

[6] Lee, K., Chang, M. - W., Toutanova, K. (2019). Latent Retrieval for Weakly Supervised Open Domain Question Answering. *arXic preprint arXiv: 1906.00300*. https://arxiv.org/pdf/1906.00300

[7] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., & Guo, B. (2021). Swin Transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*. https://arxiv.org/pdf/2103.14030

[8] langchain-ai. (2023). rag-from-scratch GitHub. https://github.com/langchain-ai/rag-from-scratch

[9] stanford-futuredata. (2020). ColBERT [Izvorni kod]. GitHub. https://github.com/stanford-futuredata/ColBERT

[10] AnswerDotAI. (2023). *RAGatouille* [Izvorni kod]. GitHub. https://github.com/AnswerDotAI/RAGatouille

[11] explodinggradients. (2023). *ragas* [Izvorni kod]. GitHub. https://github.com/explodinggradients/ragas

[12] LangChain. (2023). *Retrievers* [Dokumentacija]. LangChain. https://python.langchain.com/v0.1/docs/modules/data_connection/retrievers/

[13] LangChain. (2023). *MultiQueryRetriever* [API referenca]. LangChain. https://python.langchain.com/api_reference/langchain/retrievers/langchain.retrievers.multi_query.MultiQueryRetriever.html

[14] LangChain. (2023). *Retrievers* [Dokumentacija]. LangChain. https://python.langchain.com/v0.1/docs/modules/data_connection/retrievers/

[15] LangChain. (2023). *Character text splitter* [Dokumentacija]. LangChain. https://python.langchain.com/v0.1/docs/modules/data_connection/document_transformers/character_text_splitter/

[16] LangChain. (2023). *Multi vector retriever* [Dokumentacija]. LangChain. https://python.langchain.com/v0.1/docs/modules/data_connection/retrievers/multi_vector/

[17] Unstructured. (2023.). *Models* [Dokumentacija]. Unstructured. https://docs.unstructured.io/open-source/concepts/models#models

[18] Unstructured-IO. (2023.). *Unstructured* [Izvorni kod]. GitHub. https://github.com/Unstructured-IO/unstructured

[19] Blecher, L., Cucurull, G., Scialom, T., & Stojnic, R. (2023). Nougat: Neural Optical Understanding for Academic Documents. *arXiv*. https://arxiv.org/pdf/2308.13418

[20] Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv*. https://arxiv.org/pdf/1502.03167

[21] BEIR. (2021). BEIR: A Benchmark for Evaluating Information Retrieval. GitHub repository. https://github.com/beir-cellar/beir

[22] Thakur, N., Reimers, N., Rücklé, A., Srivastava, A., & Gurevych, I. (2021). BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. *Ubiquitous Knowledge Processing Lab (UKP Lab), Department of Computer Science, Technical University of Darmstadt*. https://openreview.net/pdf?id=wCu6T5xFjeJ

[23] Patel, S. (2022, February 21). YOLOX Object Detector Paper Explanation and Custom Training. *LearnOpenCV*. https://learnopencv.com/yolox-object-detector-paper-explanation-and-custom-training/

[24] Bing Videos. (2022). YOLO Model Paper [Video]. *Bing*. https://www.bing.com/videos/riverview/relatedvideo?&q=yolo+model+paper+&&mid=C57329F4255A188683C0C57329F4255A188683C0&mmscn=mtsc&aps=32&FORM=VRDGAR

[25] Chandrasekhar, S. (2020, May 6). Final Layers and Loss Functions of Single Stage Detectors: Part 1. *Medium*. https://medium.com/oracledevs/final-layers-and-loss-functions-of-single-stage-detectors-part-1-4abbfa9aa71c

[26] Liu, S. (2018, May 14). Review: FPN (Feature Pyramid Network) for Object Detection. *Towards Data Science*. https://towardsdatascience.com/review-fpn-feature-pyramid-network-object-detection-262fc7482610

[27] zm20200109. (2024). *ISLP_CHATBOT* [GitHub repository]. GitHub. https://github.com/zm20200109/ISLP_CHATBOT/tree/main

[28] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning with Python*. Springer.

[29] Reimers, N., & Gurevych, I. (2020). *all-MiniLM-L6-v2* [Model]. Hugging Face. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[30] StatQuest. (2023). *Transformers video explanation* [Video]. Bing. https://www.bing.com/videos/riverview/relatedvideo?&q=transformers+video+explanation+statquest&&mid=4159B0610A315CBB12B24159B0610A315CBB12B2&mmscn=mtsc&aps=0&FORM=VRDGAR

[31] AI Coffee Break with Letitia. (2021). * Swin transformer paper animated and explained*[Video].Bing.https://www.bing.com/videos/riverview/relatedvideo?&q=swin+transformer+paper+&&mid=977CB67DE4564F68B4E3977CB67DE4564F68B4E3&mmscn=mtsc&aps=0&FORM=VRDGAR

[32] Mehraban, S. (2023). *Swin transformer – Paper Explained* [Video]. Bing. https://www.bing.com/videos/riverview/relatedvideo?&q=swin+transformer+paper+&&mid=CFEF2202BBEFC240A1D1CFEF2202BBEFC240A1D1&mmscn=mtsc&aps=0&FORM=VRDGAR

[33] MongoDB. (n.d.). Atlas documentation. MongoDB. https://www.mongodb.com/docs/atlas/?msockid=06037b531edf63ac18b26f9e1fc2621e

[34] Free Code Camp. (2021). *RAG tutorial* [Video]. Bing. https://www.bing.com/videos/riverview/relatedvideo?q=rag+free+code+camp+tutorial&&view=riverview&mmscn=mtsc&mid=EB02F11D8FC3BECEAA38EB02F11D8FC3BECEAA38&&aps=0&FORM=VMSOVR

[35] LangChain. (n.d.). Chroma integration documentation. LangChain. https://python.langchain.com/docs/integrations/vectorstores/chroma/

[36] Facebook AI Research. (n.d.). Faiss documentation. Faiss. https://faiss.ai/

[37] LangChain. (n.d.). *Parent document retriever guide*. LangChain. https://python.langchain.com/docs/how_to/parent_document_retriever/

[38] Bing. (2023). *Advanced RAG 02 - Parent document retriever*. Bing. https://www.bing.com/videos/riverview/relatedvideo?&q=parent+document+retriever&&mid=0DE9D46219682C56D88C0DE9D46219682C56D88C&&FORM=VRDGAR

[39] LangChain. (n.d.). *How to do retrieval with contextual compression*. LangChain. https://python.langchain.com/docs/how_to/contextual_compression/

[40] LangChain. (n.d.). *EnsembleRetriever*. LangChain Documentation. Retrieved November 17, 2024, from https://api.python.langchain.com/en/latest/langchain/retrievers/langchain.retrievers.ensemble.EnsembleRetriever.html

[41] Bing Videos. (2024). Learn RAG From Scratch – Python AI Tutorial from a LangChain Engineer. Bing. https://www.bing.com/videos/riverview/relatedvideo?q=rag+from+scratch+tutorial&mid=49436CE0F3B1A080BE4249436CE0F3B1A080BE42&FORM=VIRE

[42] Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q., & Chi, E. (2023). *Least-to-most prompting in large language models* (arXiv:2205.10625v3). https://arxiv.org/pdf/2205.10625

[43] Siddhant, A. (2021, November 10). *Evaluating RAG applications with RAGAs*. Towards Data Science. https://towardsdatascience.com/evaluating-rag-applications-with-ragas-81d67b0ee31a

[44] RAGAs Team. (n.d.). *Context Precision - RAGAs documentation*. RAGAs. https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/context_precision/

[45] RAGAs Team. (n.d.). *Context Recall - RAGAs documentation*. RAGAs. https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/context_recall/
[46] RAGAs Team. (n.d.). *Faithfulness - RAGAs documentation*. RAGAs. https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/faithfulness/
[47] RAGAs Team. (n.d.). *Answer relevance - RAGAs documentation*. RAGAs. https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/answer_relevance/
[48] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., & Wang, H. (2023). *Retrieval-Augmented Generation for Large Language Models: A Survey*. arXiv. https://arxiv.org/pdf/2312.10997
[49] Unstructured-IO. (n.d.). *Unstructured Inference* [Computer software]. GitHub. https://github.com/Unstructured-IO/unstructured-inference
[50] X Engineer. (n.d.). *Bilinear interpolation*. from https://x-engineer.org/bilinear-interpolation/