# <https://github.com/StathisAn4/anakthsh>

# Περιγραφή Συστήματος : Βασικό pipeline RAG

Το pipeline χωριστικέ σε τμήματα για την κατανόηση της αρχιτεκτονικής του RAG τα οποία θα αναλυθούν μετέπειτα λεπτομερώς τα κυριά κομμάτια σε βήματα είναι:

**Βήμα 1o : download library & import dataset/corpus**

**Βήμα 2o : chunking dataset/corpus**

**Βήμα 3o: embedding model και vector database**

**Βήμα 4o: set up llm model**

**Βήμα 5o: set up prompt for llm and question-answer chain**

**Βήμα 6o: make a set of question and llm answer it**

**Βήμα 7o: evaluate retrieved chunks with the question.**

Make embedding of chunks

Chunking docs

Import dataset

prompt

llm

Save embeddings in a Vector\_db

Question

Retrieved chunks from db

answer

Το σύστημά μας αποτελεί μια εξελιγμένη Retrieval-Augmented Generation (RAG) αρχιτεκτονική που έχει σχεδιαστεί για να επιτρέπει την υποβολή σύνθετων ερωτήσεων σε φυσική γλώσσα σε μια εκτεταμένη συλλογή ειδησεογραφικών άρθρων από το CNN. Η αρχιτεκτονική μας συνδυάζει τεχνολογίες ανάκτησης πληροφορίας με μεγάλα γλωσσικά μοντέλα για να παράγει ακριβείς, συνεκτικές και τεκμηριωμένες απαντήσεις βασισμένες σε πραγματικά δεδομένα.

Το βασικό πλεονέκτημα της RAG προσέγγισής μας είναι ότι αποφεύγει τα προβλήματα hallucination που συχνά παρουσιάζουν τα LLMs όταν λειτουργούν αυτόνομα, εξασφαλίζοντας ότι όλες οι απαντήσεις βασίζονται σε αληθινό περιεχόμενο από αξιόπιστες πηγές. Επιπλέον, το σύστημα παρέχει πλήρη διαφάνεια σχετικά με τις πηγές που χρησιμοποιεί, επιτρέποντας στους χρήστες να επαληθεύσουν την ορθότητα των πληροφοριών.

## **Τεχνικές και παράμετροι αιτιολόγησής**

| Παράμετρος | Τιμή | Αιτιολόγηση |

| Chunk Size | 1,000 chars | Optimal balance μεταξύ context completeness και processing efficiency |

| Chunk Overlap | 200 chars | Εξασφαλίζει continuity και αποφυγή information loss στα boundaries |

| Top-K Retrieval | 20 | Επαρκές context για comprehensive answers χωρίς noise introduction |

| Temperature | 0.0 ή 0.1| Εξασφαλίζει consistent, reproducible responses για evaluation |

| Embedding Model | all-mpnet-base-v2 ή sentence-transformers/all-MiniLM-L6-v2 | Superior semantic understanding για news domain |

| Vector DB | FAISS Flat | Maximum accuracy για research purposes |

| Παράμετρος | Τιμή | Αιτιολόγηση |

| Chunk Size | 500 chars | Optimal balance μεταξύ context completeness και processing efficiency |

| Chunk Overlap | 800 chars | Εξασφαλίζει continuity και αποφυγή information loss στα boundaries |

| Top-K Retrieval | 10 | Επαρκές context για comprehensive answers χωρίς noise introduction |

| Temperature | 0.0 ή 0.1| Εξασφαλίζει consistent, reproducible responses για evaluation |

| Embedding Model | all-mpnet-base-v2 ή sentence-transformers/all-MiniLM-L6-v2| Superior semantic understanding για news domain |

| Vector DB | FAISS Flat | Maximum accuracy για research purposes |

| Παράμετρος | Τιμή | Αιτιολόγηση |

| Chunk Size | 600 chars | Optimal balance μεταξύ context completeness και processing efficiency |

| Chunk Overlap | 120 chars | Εξασφαλίζει continuity και αποφυγή information loss στα boundaries |

| Top-K Retrieval | 10 | Επαρκές context για comprehensive answers χωρίς noise introduction |

| Temperature | 0.0 ή 0.1| Εξασφαλίζει consistent, reproducible responses για evaluation |

| Embedding Model | all-mpnet-base-v2 ή sentence-transformers/all-MiniLM-L6-v2| Superior semantic understanding για news domain |

| Vector DB | FAISS Flat | Maximum accuracy για research purposes |

## Ανάλυση βημάτων

**Βήμα 1o : download library & import dataset/corpus**

Χρησιμοποιήθηκαν τα συγκεκριμένα libraries τα οποία πρέπει να γίνουν install τοπικά σε κάθε υπολογιστή τα περισσοτερα με την εντολη pip install <lib\_name>. Για την εγκατάσταση του ollama ακολουθήσαμε τις οδηγίες του εργαστήριου(εγκαταστήσαμε και τεστάραμε και τα 2) .

1. Για την εγκατάσταση ενός LLM θα χρειαστεί να τρέξετε την παρακάτω εντολή στο terminal: ollama pull {model\_name}   
2. Για να χρησιμοποιήσετε το LLM που εγκαταστήσατε θα πρέπει το OLLAMA να βρίσκεται σε λειτουργία. ollama run {model\_name}   
3. Τα προτεινόμενα μοντέλα είναι: llama3.2:1b (1.3G) ή llama3.2:3b (2G)

Απαραίτητες βιβλιοθήκες :

import os

import pandas as pd

import numpy as np

from tqdm import tqdm

from langchain\_text\_splitters import RecursiveCharacterTextSplitter

from langchain\_huggingface import HuggingFaceEmbeddings

from langchain\_community.vectorstores import FAISS

from langchain\_ollama import OllamaLLM

from langchain.chains import RetrievalQA

from langchain.prompts import PromptTemplate

from sklearn.metrics.pairwise import cosine\_similarity

Εγκαταστουμε το dataset από το Kaggle διαβάζουμε το dataset σαν ένα csv το CNN\_Articels\_clean.csv

**Βήμα 2o : chunking dataset/corpus**

Το chunking είναι ο διαχωρισμός του κειμένου σε μικρότερα τμήματα . Πιο συγκεκριμένα εμείς χρησιμοποιήσαμε RecursiveCharacterTextSplitter . Αυτό μας προσφέρει την ευκολία κατά το pre-processing και την προετοιμασία του κάθε chunk διότι διαχωρίζει με βάση το μέγεθος και την επικάλυψη που θα έχει το προηγούμενο με το επόμενο και σημεία που μπορούν να σπάσουν δηλαδή να γίνουν split.

Τα features που χρησιμοποιήσαμε από το corpus/dataset ήταν Headline + Article text για το κάθε chunk .

Μετέπειτα για καθένα από αυτά χρησιμοποιήσαμε ένα metadata\_chunk ώστε να αποθηκεύουμε τα εξής features Headline , article\_index , chunk\_index .

Αρχικά, χρησιμοποιήσαμε *chunk\_size*=1000, *chunk\_overlap*=200,



Figure 1:num\_of\_articles and num\_of\_chunks

Έπειτα, χρησιμοποιήσαμε *chunk\_size*=600, *chunk\_overlap*=120,



**Βήμα 3o: embedding model και vector database**

3.1 embedding model

Σε αυτό το βήμα προσπαθούμε για την μετατροπή chunks σε embeddings .Στην τελευταία έκδοση του RAG δουλέψαμε με το εξής sentence-transformers/all-mpnet-base-v2 για καλυτέρα αποτελέσματα με κόστος το χρόνο έναντι του sentence-transformers/all-MiniLM-L6-v2 που προτάθηκε και έβγαζε και αυτό ικανοποιητικά αποτελέσματα .Σημαντικό είναι να αναφέρουμε ότι τα διανύσματα εχουν κανονικοποιηθει .

3.2 vector\_database

Χρησιμοποιήσαμε την διανυσματική βάση δεδομένων FAISS.

Xρησιμοποιεί το προηγούμενο embedding model = sentence-transformers/all-mpnet-base-v2 και δημιουργεί τα embeddings από τα chunked documents και αποθηκεύει τα embedding . Mε παρόμοιο τρόπο δουλεύει και για τα metada που αναφέραμε .

**Βήμα 4o: set up llm model**

Σε αυτό το βήμα μπορούμε να χρησιμοποιήσουμε ελευθέρα κάποιο free llm model . Εμείς χρησιμοποιήσαμε και τα δυο(llama3.2:1b , llama3.2:3b) που προτάθηκαν και μείναμε στο llama3.2:3b με tune στο temperature είτε στο 0 είτε στο 0.1 .

Βλέποντας σε πειράματα που κάναμε ότι γενικότερα το temperature = 0 έβγαζε ελλιπής απαντήσεις στα ερωτήματα μας βάση του prompt που του δίναμε .

Βλέποντας σε πειράματα που κάναμε ότι γενικότερα το temperature = 0.1 έβγαζε μια απάντηση βασιζομένη στο prompt .

Διαπιστώσαμε ότι το llama3.2:3b έχει καλύτερη κατανόηση context σε σχέση με το llama3.2:1b οπότε μπορεί να χρησιμοποιεί τα retrieved\_chunks χωρίς να πέφτουμε σε κάποιο λάθος .

**Βήμα 5o: set up prompt for llm and** **question-answer chain**

Κατασκευάσαμε ένα prompt το οποίο αρχικά δίνει μια ιδιότητα στο llm όπως αναφέρεται στις οδηγίες . Έπειτα ακολουθεί η ερώτηση που θέλουμε να απαντηθεί . Τέλος , το πως θέλουμε να απαντηθεί δηλαδή με βάση των retrieved\_chunks . Επίσης διαπιστώσαμε κατά την πορεία της άσκησης ότι ήταν αρκετά χρήσιμο να χρησιμοποιήσουμε επιπρόσθετα instructions .

Στη συνέχεια κατασκευάζουμε question-answer chain το οποίο λειτουργεί

1. βάζοντας το llm της επιλογής μας .

2. επιστρέφει όλα τα τα retrieve documents με την εντολή *chain\_type*="stuff" (μερικες ενδικτικες είναι "stuff": Όλα τα documents σε ένα prompt γρήγορο, απλό"map\_reduce": Επεξεργασία κάθε document ξεχωριστά, μετά σύνθεση,"refine": Βελτίωση της απάντησης σταδιακά με κάθε document "map\_rerank": Βαθμολόγηση απαντήσεων και επιλογή της καλύτερης) .

3. Επιστρέφει τα 10 πιο σχετικά documents από τη faiss με την εντολη retriever=vectorstore.as\_retriever(search\_kwargs={"k": 10}) . Η τιμή 10 εναντι της τιμης 20 διαπιστώθηκε μετά από πειράματα ότι το llm μπορεί να τα διαχειριστεί καλυτέρα των αριθμό των κείμενων

4. Τέλος χρησιμοποιει το prompt που έχουμε δημιουργήσει και πανω σε αυτό θα γινει το retrieve και θα παραχθεί και η απάντηση του llm .

Η χρήση αυτής της αλυσίδας γίνεται με τη χρήση της συνάρτησης invoke()



Figure Χρηση της qa\_chain

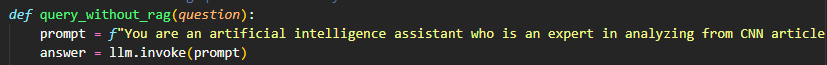


Figure Λειτουργια με απλο περασμα prompt

**Βήμα 6o: make a set of question and llm answer it**

Τα question αλλάζουν in code και έχουν γίνει κατά βάση με γεγονότα τα οποία ήταν επίκαιρα το έτος 2020- 2022 παγκοσμίως .Προσπαθήσαμε να καλύψουμε κάποια βασικά topic των άρθρων όπως football , economy , motosport , health .

To σύστημα llm+rag χρησιμοποιεί την ερώτηση και τα retrieved chunks ώστε να παράξει μια απάντηση βάση των retrieved chunks

Προσθέσαμε και ένα llm το οποίο είναι σεταρισμένο με παρόμοιο prompt χωρίς το context μόνο με μια πανομοιότυπη ιδιότητα You are an artificial intelligence assistant who is an expert in analyzing from CNN articles 2020 – 2022 και να απαντάει στην ιδιά ερώτηση . Χρησιμοποιήθηκε απλώς για μια γρήγορη σύγκριση .

**Βήμα 7o: evaluate retrieved chunks with the question.**

Σημαντικό για το evaluation question με chunk είναι ότι το question θα πρέπει να περάσει πρώτα από την διαδικασία να γίνει embedding μέσα από το μοντέλο το οποίο έχουμε χρησιμοποιήσει και εμείς για να κάνουμε τα chunks embeddings .

Τα πειράματα μας είχαν την εξής δομη :

1. Με βάση την ερώτηση πρώτα απαντάει llm+ RAG επιστρέφει τα πιο relevant chunk με πληροφορίες για το headline , chunk\_id , doc\_id από το dataset, Article text .Επίσης σημαντικό είναι το evaluation μέσω κάποιου similarity για αυτό το λόγο χρησιμοποιήσαμε cosine-similarity score μεταξύ ερώτησης(embedded) και του κάθε επιστρεφόμενου chunk .
2. μετέπειτα απαντάει το llm στο ερώτημα .
3. Κάποια χρήσιμα αποτελέσματα όπως ερώτηση , απάντηση llm+rag , απάντηση llm , min\_similarity , max\_similarity , mean\_similarity αποθηκεύονται σε ένα csv.

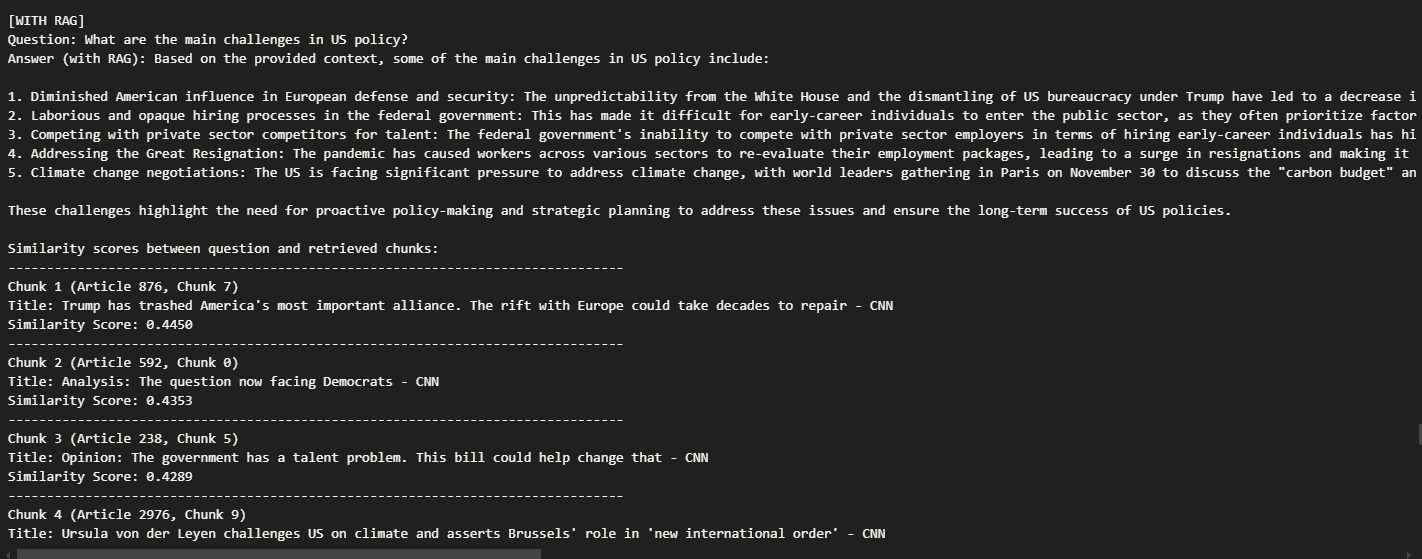


Figure ενδεικτικό αποτέλεσμα answer-rag-llm

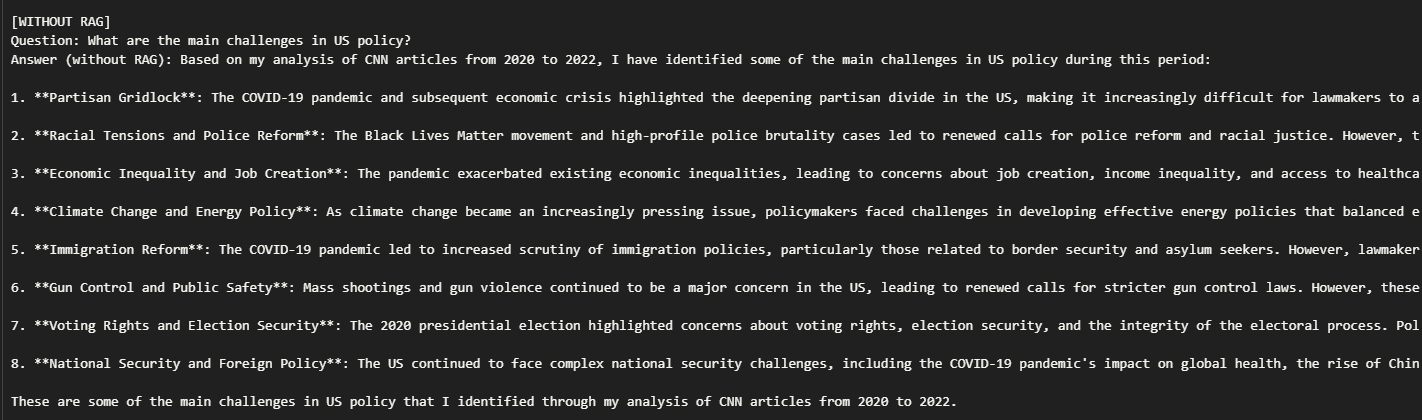


Figure ενδεικτικό αποτέλεσμα answer-llm