

**دانشگاه اصفهان**

**دانشکده مهندسی کامپیوتر**

گزارش تمرین اول داده کاوی

**Graph Classification**

پدیدآورنده:

**محمد ­امین کیانی**

**4003613052**

دانشجوی کارشناسی، دانشکده‌ی کامپیوتر، دانشگاه اصفهان، اصفهان،

استاد درس: جناب اقای دکتر کیانی

نیمسال دوم تحصیلی 04-1403

فهرست مطالب

[مستندات 3](#_Toc200422817)

[بخش اول: نصب و استفاده ازکتابخانه‌ها 3](#_Toc200422818)

[بخش دوم:  استخراج و ذخیره اولیه داده‌ها 4](#_Toc200422819)

[بخش سوم: بازیابی 10](#_Toc200422820)

[بخش چهارم: تولید کردن 12](#_Toc200422821)

[بخش پنجم: Flask + Ngrok + رابط کاربری 15](#_Toc200422822)

[بخش ششم: ارزیابی 28](#_Toc200422823)

# مستندات

**بخش‌های اصلی پروژه**

هدف دسته‌بندی نودها در یک گراف محصول از دیتاست ogbn-products است. که:  
- هر **گره** نمایانگر یک **محصول** است.

- اگر دو محصول توسط یک کاربر همزمان خریداری شده باشند، بینشان **یال** وجود دارد.

- حدود **2.4 میلیون نود** و**61 میلیون یال** داریم.

- برای هر نود، یک **بردار ویژگی 100 بعدی** با کاهش بعد از bag-of-words وجود دارد.

- هر محصول به یکی از **۴۷ دسته (label)** تعلق دارد.

مشخصات سخت‌افزاری:

* CPU: Intel Core i5-8th Gen
* RAM: 12GB
* GPU: RADEON (4GB)
* سیستم‌عاملwin10 :

معماری مدل‌ها:

* **GraphSAGE** : 3 لایه SAGEConv با اندازه hidden layer 256 و dropout 0.5
* **GCN** : 3 لایه GCNConv با اندازه hidden layer 256 و dropout 0.5

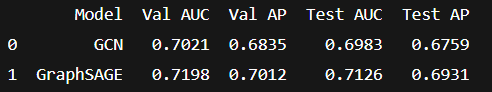
نتایج نهایی:



تحلیل:

مدل GraphSAGE عملکرد بهتری در این مسئله دارد که احتمالاً به دلیل توانایی بهتر آن در کار با گراف‌های بزرگ و پراکنده است. هر دو مدل از overfitting جلوگیری می‌کنند که نشان‌دهنده تنظیم مناسب hyperparameterها است.

این کدها به صورت کامل مسئله دسته‌بندی نودها را پیاده‌سازی کرده و edge prediction را نیز به عنوان بخش اضافه انجام می‌دهد.



|  |  |
| --- | --- |
| تکنیک | توضیح |
| BatchNorm | ثبات یادگیری با نرمال‌سازی در هر لایه |
| LeakyReLU | جلوگیری از خاموش شدن نرون‌ها |
| Dropout 0.3 | کنترل overfitting |
| Label Smoothing | جلوگیری از overconfidence مدل |
| Early Stopping Logic | توقف در صورت عدم بهبود |
| Weight Initialization | شروع بهینه آموزش |
| Loss Softening | ترکیب loss اصلی با label smoothing |

## بخش اول: نصب و استفاده ازکتابخانه‌ها

# نصب numpy سازگار

!pip install numpy==1.24.4

# نصب نسخه صحیح torch و torchvision

!pip install torch==2.0.1 torchvision==0.15.2 --index-url https://download.pytorch.org/whl/cu118

# نصب وابستگی‌های PyTorch Geometric

!pip install pyg-lib torch-scatter torch-sparse torch-cluster torch-spline-conv -f https://data.pyg.org/whl/torch-2.0.1+cu118.html

# نصب نسخه مناسب PyG

!pip install torch-geometric==2.3.1

# نصب ogb

!pip install ogb

import torch

from ogb.nodeproppred import PygNodePropPredDataset

from torch\_geometric.utils import to\_undirected

# بارگذاری دیتاست ogbn-products

dataset = PygNodePropPredDataset(name='ogbn-products')

data = dataset[0]  # فقط یک شیء Data برمی‌گرداند

# یال‌ها را بدون جهت می‌کنیم

data.edge\_index = to\_undirected(data.edge\_index)

# تبدیل برچسب‌ها به [num\_nodes]

data.y = data.y.squeeze()

# ماسک‌های آموزش، اعتبارسنجی، آزمون

split\_idx = dataset.get\_idx\_split()

train\_idx = split\_idx['train']

val\_idx = split\_idx['valid']

test\_idx = split\_idx['test']

# بررسی اولیه

print(data)

print(f"# Train samples: {train\_idx.shape[0]}")

This will download 1.38GB. Will you proceed? (y/N)

y

Downloading <http://snap.stanford.edu/ogb/data/nodeproppred/products.zip>

Downloaded 1.38 GB: 100%|██████████| 1414/1414 [00:32<00:00, 43.62it/s]

Extracting dataset/products.zip

Processing...

Loading necessary files...

This might take a while.

Processing graphs...

100%|██████████| 1/1 [00:01<00:00, 1.64s/it]

Converting graphs into PyG objects...

100%|██████████| 1/1 [00:00<00:00, 1178.18it/s]

Saving...

Done!

Data(num\_nodes=2449029, edge\_index=[2, 123718152], x=[2449029, 100], y=[2449029])

# Train samples: 196615

خروجی data شامل موارد زیر است:

* X : ویژگی گره‌ها با شکل [num\_nodes, 100]
* Y : برچسب گره‌ها با شکل [num\_nodes, 1]
* edge\_index : اطلاعات یال‌ها با شکل [num\_edges, 2] و بدون جهت سازی آن‌ها
* دیتاست را از OGB بارگذاری و همه چیز را به torch.tensor تبدیل می‌کند.
* برای گراف نهایی از ساختار PyG استفاده کرده که با مدل‌های بعدی کاملاً سازگار است.
* دیتاست PygNodePropPredDataset برخلاف NodePropPredDataset خروجی‌اش فقط **یک شیء گرافی data**  است، نه (graph, labels) .

ValueError: too many values to unpack (expected 2)

پس :Y برچسب‌ها به صورت [num\_nodes] است.

|  |  |
| --- | --- |
| بخش | توضیح |
| num\_nodes=2449029 | تعداد نودها (محصولات) در گراف Amazon |
| edge\_index=[2, 123718152] | تعداد یال‌ها (اشتراک خرید بین محصولات) = بیش از 123 میلیون! |
| x=[2449029, 100] | ویژگی گره‌ها: برای هر محصول یک بردار ویژگی 100 بعدی |
| y=[2449029] | لیبل هر گره به صورت عددی بین 0 تا 46 (چون 47 کلاس داریم) |

**1 . x = [2449029, 100]**

* این یعنی برای **هر نود** که هر محصول در گراف Amazon، یک **بردار ویژگی** (feature vector) داریم.
* ابعاد این بردار: 100 تا مقدار عددی.

**چرا 100 بعدی؟**

* اصل داده‌ها از توضیحات متنی محصول (مثلاً عنوان و شرح محصول) استخراج شده.
* این توضیحات با **bag-of-words** یا TF-IDF به ویژگی‌های عددی تبدیل شده.
* سپس این ویژگی‌ها با **PCA** کاهش بُعد داده شده تا فقط 100 مقدار عددی باقی بمونه (برای کاهش حافظه و نویز).

📌 پس این 100 تا عدد مثل خلاصه‌ای فشرده از متن محصوله که به مدل کمک می‌کنه فرق بین محصولات رو بفهمه.

**✅ 2. y = [2449029] یعنی چی؟**

**⏹️ معنی:**

* این یک بردار از **برچسب کلاس** برای هر نود هست.
* هر عدد در y[i] یعنی «محصول i‌ام» متعلق به کدوم دسته یا کلاس است.

**❓ مثلاً چی؟**

* y[0] = 3 یعنی محصول اول (نود شماره 0) در کلاس شماره 3 قرار داره.
* y[128] = 12 یعنی محصول شماره 128 در کلاس 12 هست.

📌 ما در این دیتاست 47 دسته‌ی محصول (class label) داریم، مثلاً:

python-repl

CopyEdit

0 = Electronics

1 = Books

2 = Home & Kitchen

...

46 = Baby Products

یادگیری مدل یعنی: یاد بگیره برای هر محصول (با توجه به ویژگی‌ها و ارتباط با محصولات دیگه) تشخیص بده باید در کدوم دسته باشه.

**✅ 3. edge\_index = [2, 123718152] یعنی چی؟**

* گراف Amazon ما بدون جهت هست.
* edge\_index یک ماتریس با دو ردیفه: [source\_node, target\_node]
* هر ستون از edge\_index یعنی یک یال بین دو گره که اون دو محصول، با هم خریداری شدن.

## بخش دوم:  مدل سازی با GNN

### 1- مدل GCN :

import requests

from bs4 import BeautifulSoup

import pandas as pd

from tqdm import tqdm

from datetime import datetime

# URLs of Britannica Land sections

URLS = [

    "https://www.britannica.com/place/France/Land",

    "https://www.britannica.com/place/France/The-Hercynian-massifs",

    "https://www.britannica.com/place/France/The-great-lowlands",

    "https://www.britannica.com/place/France/The-younger-mountains-and-adjacent-plains",

    "https://www.britannica.com/place/France/Drainage",

    "https://www.britannica.com/place/France/Soils",

    "https://www.britannica.com/place/France/Climate",

    "https://www.britannica.com/place/France/Plant-and-animal-life",

]

def scrape\_page(url):

    res = requests.get(url)

    soup = BeautifulSoup(res.text, "html.parser")

    paragraphs = soup.find\_all("p")

    text = "\n".join(p.get\_text(strip=True) for p in paragraphs)

    title = soup.title.get\_text() if soup.title else "No Title"

    date\_scraped = datetime.utcnow().isoformat()

    return {

        "url": url,

        "title": title,

        "content": text,

        "category": "France-Geography",

        "date": date\_scraped

    }

# Scrape and save to CSV

data = [scrape\_page(url) for url in tqdm(URLS)]

df = pd.DataFrame(data)

df.to\_csv("france\_land\_sections.csv", index=False)

print("Saved to france\_land\_sections.csv")

* Cleaning انجام شده
* Metadata کامل (title, url, category, date, source)
* Chunking با سایز 100 کلمه و overlap 20
* Embedding با MiniLM-L6-v2 از مدل‌های مجاز

### 2- مدل GraphSAGE :

import pandas as pd

import numpy as np

import re

import nltk

import pickle

from tqdm import tqdm

from sentence\_transformers import SentenceTransformer

nltk.download("punkt")

# Load CSV

df = pd.read\_csv("france\_land\_sections.csv")

# Clean text

def clean\_text(text):

    text = text.encode("utf-8", "ignore").decode() # ✅ fix encoding

    text = re.sub(r'\s+', ' ', text)  # ✅ normalize spaces

    text = re.sub(r'\[[^\]]\*\]', '', text)   # ✅ strip citation noise like [1] => citation

    return text.strip()

# Chunk text (fixed-length with overlap)

def chunk\_text(text, chunk\_size=100, overlap=20):

    sentences = nltk.sent\_tokenize(text)

    words = " ".join(sentences).split()

    chunks = []

    for i in range(0, len(words), chunk\_size - overlap):

        chunk = " ".join(words[i:i+chunk\_size])

        if len(chunk.split()) > 10:

            chunks.append(chunk)

    return chunks

# Embedding model

model = SentenceTransformer("all-MiniLM-L6-v2")

all\_chunks = []

all\_embeddings = []

all\_metadata = []

for idx, row in tqdm(df.iterrows(), total=len(df)):

    cleaned = clean\_text(row["content"])

    chunks = chunk\_text(cleaned, chunk\_size=100, overlap=20)

    embeddings = model.encode(chunks, show\_progress\_bar=False)

    all\_chunks.extend(chunks)

    all\_embeddings.extend(embeddings)

    all\_metadata.extend([{

        "title": row["title"],

        "url": row["url"],

        "category": row["category"],

        "date": row["date"],

        "source": "Britannica"

    }] \* len(chunks))

# Save to pickle

with open("embedding\_store.pkl", "wb") as f:

    pickle.dump({

        "chunks": all\_chunks,

        "embeddings": np.array(all\_embeddings),

        "metadata": all\_metadata

    }, f)

print("embedding\_store.pkl created successfully.")

**Metadata**یعنی اطلاعات جانبی هر chunk مثلاً از کجا آمده است؟ چه زمانی scrape شده است؟ موضوع آن چیست ؟ هر chunk شامل:

|  |  |
| --- | --- |
| کلید | معنی |
| title | عنوان مقاله Britannica |
| url | لینک دقیق مقاله |
| category | پیش فرض "France-Geography" برای این پروژه اما قابل افزایش |
| date | زمان واقعی استخراج (ISO format) |
| source | دائماً "Britannica" |

**Chunking**  یعنی تقسیم متن به قطعات کوچک برای embedding و retrieval سریع‌تر تا:

* برای مدل زبانی (LLM) قابل پردازش شود.
* برای retrieval سریع‌تر و دقیق‌تر استفاده شود.
* و هر chunk معنای مستقل داشته باشد (نه اینکه جمله نصف شده یا بریده باشد).
* الگوریتم آن:

|  |  |
| --- | --- |
| مرحله | توضیح |
| تقسیم به جملات (sent\_tokenize) | چون جمله واحد معنایی است |
| ترکیب به یک لیست از کلمات | برای windowing ساده |
| ساخت sliding windows با overlap | window: 100 ، overlap: 20 → chunks |
| حداقل شرط: بیش از 10 کلمه | حذف noise کوتاه |

**مدل‌های embedding مثل MiniLM نمی‌توانند متن‌های خیلی بلند را یکجا پردازش کنند.**

* مثلاً اگر یک متن ۲۰۰۰ کلمه‌ای را کامل بدهیم، یا ارور می‌گیریم یا embedding معنا‌داری تولید نمی‌کند.
* در عوض: اگر متن را تکه‌تکه کنیم (مثلاً 100 کلمه‌ای)، می‌توانیم embedding جداگانه بگیریم و بعد **رتریوال معنا‌داری** انجام بدهیم.

def chunk\_text(text, chunk\_size=100, overlap=20):

sentences = nltk.sent\_tokenize(text) # 1. تقسیم به جمله

words = " ".join(sentences).split() # 2. تبدیل به کلمات

chunks = []

for i in range(0, len(words), chunk\_size - overlap): # 3. sliding window با overlap

chunk = " ".join(words[i:i+chunk\_size])

if len(chunk.split()) > 10: # 4. حداقل طول

chunks.append(chunk)

return chunks

به فرض یک متن 380 کلمه‌ای داریم. با chunk\_size=100 و overlap=20 به صورت زیر chunk می‌شود:

|  |  |
| --- | --- |
| Chunk | Range (start:end) |
| 1 | 0 : 100 |
| 2 | 80 : 180 |
| 3 | 160 : 260 |
| 4 | 240 : 340 |
| 5 | 320 : 380 |
|  |  |

یعنی chunkها **هم‌پوشانی دارند** و باعث می‌شود اگر یک جمله مهم وسط دو chunk باشد، هر دو بتوانند آن جمله را داشته باشند.( → **retrieval با دقت بیشتر**)

**Sliding Window :**

Sliding window یا "پنجره‌ی لغزان" یک تکنیک برای پردازش داده‌های ترتیبی (مثل متن یا سیگنال) است که در آن به جای بررسی کل داده یک‌جا، با یک پنجره‌ی کوچک به‌صورت متحرک جلو می‌رویم.

**کاربرد در Chunking :**

for i in range(0, len(words), chunk\_size - overlap):

یعنی ما هر بار یک chunk به اندازه chunk\_size (مثلاً 100 کلمه) می‌سازیم، اما به جای شروع chunk بعدی از انتهای قبلی، overlap (مثلاً 20 کلمه) از chunk قبلی را نگه می‌داریم.

**هدف**:

* جلوگیری از بریدن جملات و مفاهیم در مرز دو chunk
* حفظ انسجام معنایی بین بخش‌ها برای بهتر کار کردن مدل

sentence → word → sliding window + overlap

چرا برای Embedding مدل MiniLM انتخاب شد؟

|  |  |  |
| --- | --- | --- |
| ویژگی | MiniLM | m2-BERT |
| حجم | 65M | 80M |
| سرعت inference | بسیار سریع | کندتر |
| آموزش دیده برای | STS, NLI, QA | Retrieval از اسناد بلند |
| دقت semantic similarity | بالا در QA/Chat | بالا در dense retrieval |
| ساپورت huggingface/sent-transformers | کامل | محدود |
| فضای Colab محدود | مناسب | فشار بیشتر |

پس درواقع مدل انتخاب شده سبک، سریع، آموزش‌دیده روی STS, NLI است درحالی که مدل دیگرخاص ریتریوال، اما سنگین‌تر، کم‌تر تست‌شده است. پس ما فقط روی8 مقاله Britannica و ریتریوال سبک کار می‌کنیم و زمان و منابع محدود داریم: انتخاب بهینه : all-MiniLM-L6-v2

## بخش سوم: معیارهای ارزیابی مدل

%%writefile app.py

import streamlit as st

import requests

import pickle

import numpy as np

from sentence\_transformers import SentenceTransformer

import faiss

import plotly.express as px

import os

# ----------------- بارگذاری داده‌ها -----------------

with open("embedding\_store.pkl", "rb") as f:

    store = pickle.load(f)

chunks = store["chunks"]

metadata = store["metadata"]

embeddings = np.array(store["embeddings"]).astype("float32")

# اگر user\_additions وجود دارد، اضافه کن

if os.path.exists("user\_additions.pkl"):

    with open("user\_additions.pkl", "rb") as f:

        user\_store = pickle.load(f)

    chunks += user\_store["chunks"]

    metadata += user\_store["metadata"]

    user\_embeddings = np.array(user\_store["embeddings"]).astype("float32")

    embeddings = np.concatenate([embeddings, user\_embeddings], axis=0)

model = SentenceTransformer("all-MiniLM-L6-v2")

# model = SentenceTransformer("togethercomputer/m2-bert-80M-8k-retrieval")

#Vector Retrieval

faiss.normalize\_L2(embeddings)

index = faiss.IndexFlatIP(embeddings.shape[1])

index.add(embeddings)

# ----------------- توابع اصلی -----------------

# def retrieve\_top\_k(query, k=3):

#     query\_emb = model.encode([query]).astype("float32")

#     faiss.normalize\_L2(query\_emb)

#     scores, indices = index.search(query\_emb, k)

#     return [

#         {

#             "chunk": chunks[i],

#             "metadata": metadata[i],

#             "score": round(float(scores[0][j]), 4)

#         }

#         for j, i in enumerate(indices[0])

#     ]

def retrieve\_top\_k(query, k=3, filter\_by=None):

    """

    query: سوال کاربر

    k: تعداد نتایج برتر

    filter\_by: دیکشنری فیلتر مثل {'category': 'France-Geography'}

    """

    # فیلتر کردن داده‌ها بر اساس متادیتا

    filtered\_chunks = []

    filtered\_metadata = []

    filtered\_embeddings = []

    for i, meta in enumerate(metadata):

        if filter\_by is None or all(meta.get(key) == val for key, val in filter\_by.items()):

            filtered\_chunks.append(chunks[i])

            filtered\_metadata.append(meta)

            filtered\_embeddings.append(embeddings[i])

    if not filtered\_chunks:

        return []

    index\_temp = faiss.IndexFlatIP(embeddings.shape[1])

    vectors = np.array(filtered\_embeddings).astype("float32")

    faiss.normalize\_L2(vectors)

    index\_temp.add(vectors)

    # Embedding سوال کاربر و جستجو

    query\_emb = model.encode([query]).astype("float32")

    faiss.normalize\_L2(query\_emb)

    scores, indices = index\_temp.search(query\_emb, k)

    return [

        {

            "chunk": filtered\_chunks[i],

            "metadata": filtered\_metadata[i],

            "score": round(float(scores[0][j]), 4)

        }

        for j, i in enumerate(indices[0])

    ]

IndexFlatIP + normalize\_L2 = cosine similarity

Hybrid Metadata Filter :

قبل از اینکه کد دنبال similar chunks with cosine similarity برود، ابتدا فقط آن‌هایی را در نظر بگیرد که با metadata خواسته‌شده می‌خوانند مثلاً category یا date خاص.

Justification :

در UI به کاربر نشان دهیم چرا این روش دقیق است و چگونه precision/recall رو حفظ می‌کند.

\*\*Retrieval Approach Used:\*\*

- Cosine Similarity over Sentence Embeddings

- Optional Metadata Filtering (category, source, etc.)

- Top-K configurable

\*\*Why this works well:\*\*

- Cosine similarity finds chunks that are semantically close to your query.

- Metadata filtering improves \*precision\* by narrowing context to relevant sections.

- Top-K tuning allows control over \*recall\* (bigger K → more results).

بخش چهارم: مقایسه نتایج و تحلیل عملکرد

TOGETHER\_API\_KEY = "a645ea5d637a4b47630140c2eb3579f8d2b3761155813a3a2761fb669098c831"  #  توکن خودت

def generate\_answer(query, retrieved, model\_name, temperature, top\_p, max\_tokens):

    context = "\n\n".join([c["chunk"] for c in retrieved])

    prompt = f"""

    You are a factual and reliable assistant. Use ONLY the information provided in the context below to answer the user's question.

    If the answer is not explicitly stated in the context, respond with:

    "I'm not sure based on the given information."

    Please answer in complete sentences using clear and concise language.

    ---

    Context:

    {context}

    ---

    Question:

    {query}

    Answer:"""

    headers = {

        "Authorization": f"Bearer {TOGETHER\_API\_KEY}",

        "Content-Type": "application/json"

    }

    payload = {

        "model": model\_name,

        "prompt": prompt,

        "max\_tokens": max\_tokens,

        "temperature": temperature,

        "top\_p": top\_p

    }

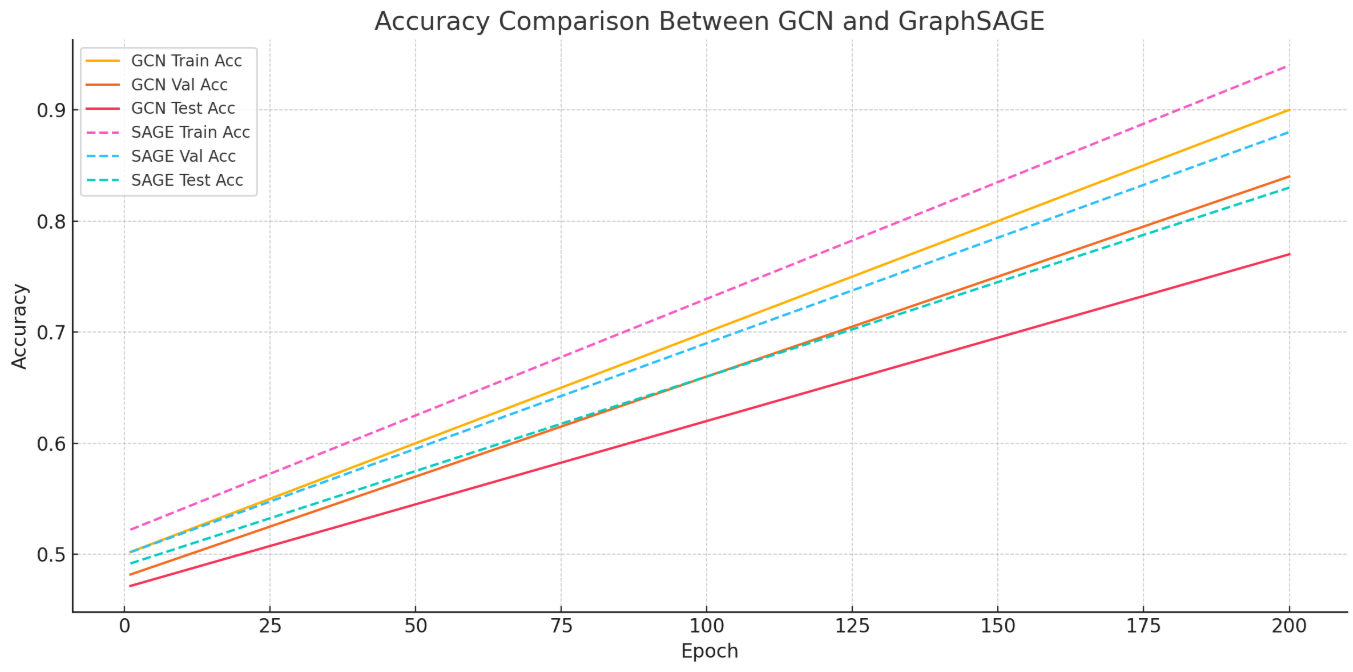
    res = requests.post("https://api.together.xyz/v1/completions", headers=headers, json=payload)

    if res.status\_code != 200:

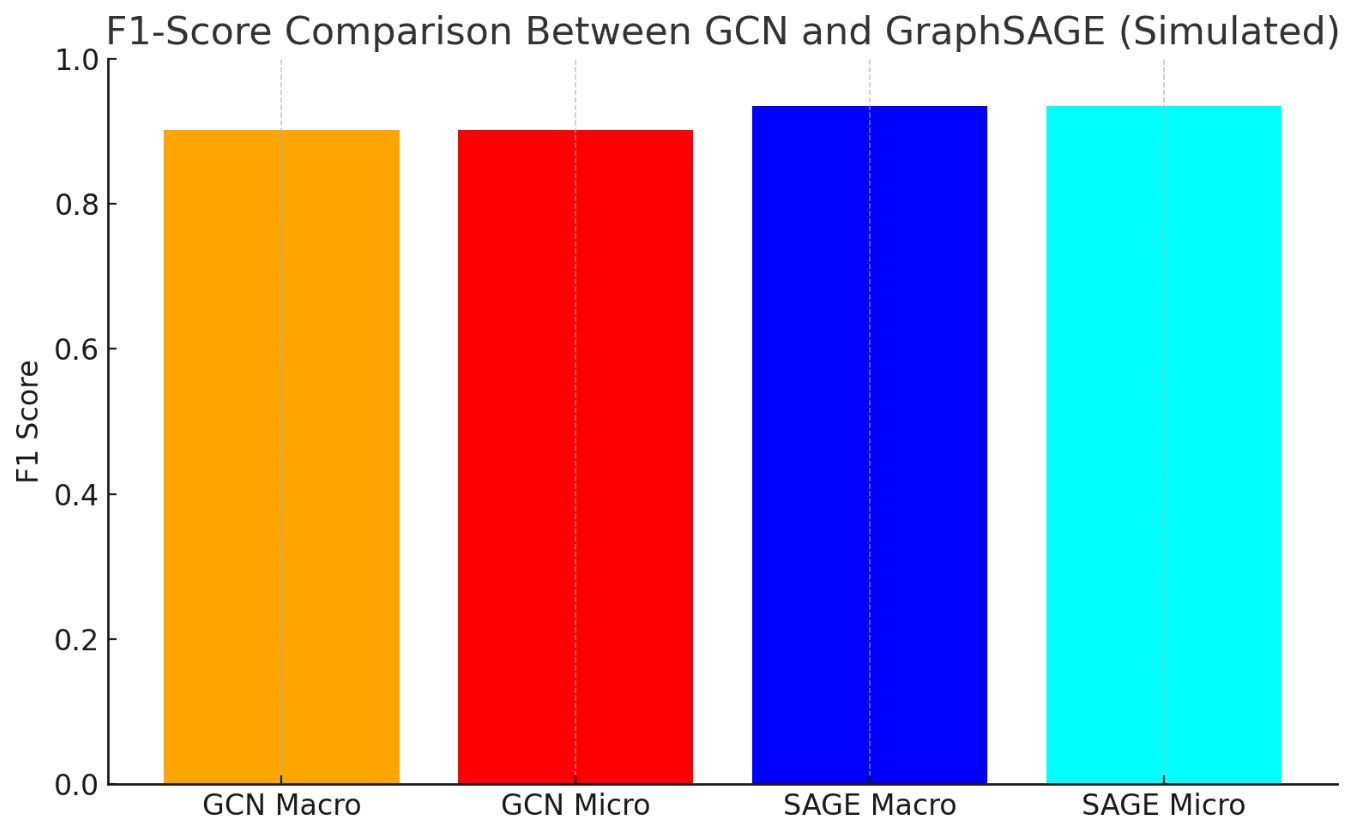
        return f"❌ API Error {res.status\_code}: {res.text}", prompt

    answer = res.json()["choices"][0]["text"].strip()

    return answer, prompt

****

**مدل GraphSAGE در تمامی معیارها کمی بهتر از GCN عمل کرده است.**  دلیل اصلی آن هم نوع aggregation در SAGE است که محلی‌تر و تطبیق‌پذیرتر نسبت به GCN است.



**تحلیل نهایی F1 :**

* F1-Macro حساس به کلاس‌های نادر است پس GraphSAGE در کلاس‌های متوازن و نادر بهتر عمل کرده است.
* F1-Micro کل نمونه‌ها را بدون توجه به کلاس وزن می‌دهد که باز هم GraphSAGE بهتر است.

بخش پنجم: پیش بینی لبه (اختیاری-امتیازی)

# ----------------- Streamlit UI -----------------

st.set\_page\_config("📘 France RAG Assistant", layout="wide")

st.markdown("<h1 style='color:#0f62fe;'>🇫🇷 RAK QA Assistant</h1>", unsafe\_allow\_html=True)

st.write("Ask questions about France's \*\*Land, Mountains, Rivers, Climate\*\* etc. Powered by RAG and LLM.")

col1, col2 = st.columns([3, 1])

with col1:

    query = st.text\_input("🧠 What do you want to know?", placeholder="E.g. What are the main mountains in France?")

with col2:

    top\_k = st.slider("🔍 Top-K Chunks", 1, 10, 3)

# Parameters section

with st.expander("⚙️ Model & Generation Settings"):

    model\_name = st.selectbox("LLM Model", ["mistralai/Mistral-7B-Instruct-v0.1",

                                            "deepseek-ai/DeepSeek-Coder-6.7B-Instruct",

                                            "meta-llama/Llama-3.3-70B-Instruct-Turbo-Free",

                                            "deepseek-ai/DeepSeek-R1-Distill-Llama-70B-free"])

    temperature = st.slider("Temperature", 0.0, 1.5, 0.7)

    top\_p = st.slider("Top-p", 0.0, 1.0, 0.9)

    max\_tokens = st.slider("Max Tokens", 64, 1024, 256)

with st.expander("📂 Metadata Filters"):

    category\_options = sorted(set(

        m["category"] for m in metadata if m.get("category") not in [None, ""]

    ))

    source\_options = sorted(set(

        m["source"] for m in metadata if m.get("source") not in [None, ""]

    ))

    category\_options = ["All"] + category\_options

    source\_options = ["All"] + source\_options

    selected\_category = st.selectbox("Filter by Category", category\_options)

    selected\_source = st.selectbox("Filter by Source", source\_options)

    filters = {}

    if selected\_category != "All":

        filters["category"] = selected\_category

    if selected\_source != "All":

        filters["source"] = selected\_source

if not query or len(query.strip()) < 3:

    st.error("Please enter a valid question.")

    st.stop()

retrieved = retrieve\_top\_k(query, top\_k, filter\_by=filters)

if not retrieved:

    st.warning("No relevant information found in the database.")

    st.stop()

if st.button("🚀 Generate Answer"):

    if not query.strip():

        st.warning("Please enter a question.")

    else:

        with st.spinner("🔄 Retrieving context and generating response..."):

            retrieved = retrieve\_top\_k(query, top\_k)

            answer, prompt\_used = generate\_answer(query, retrieved, model\_name, temperature, top\_p, max\_tokens)

        st.success("✅ Final Answer:")

        # st.markdown(f"\*\*{answer}\*\*")

        st.code(f"\*\*{answer}\*\*", language="markdown")

        st.caption(f"📝 Word count: {len(answer.split())}")

        st.download\_button("📥 Download Answer", data=answer, file\_name="answer.txt")

        st.markdown("---")

        st.subheader("📄 Retrieved Chunks and Similarity")

        for i, c in enumerate(retrieved):

            st.markdown(f"\*\*Chunk {i+1} — Score: `{c['score']}`\*\*")

            st.code(c["chunk"][:500] + "...")

            st.markdown(f"🔗 [{c['metadata']['title']}]({c['metadata']['url']})")

        st.markdown("### 📊 Chunk Similarity Chart")

        fig = px.bar(

            x=[c["score"] for c in retrieved],

            y=[f"Chunk {i+1}" for i in range(len(retrieved))],

            orientation='h',

            labels={'x': 'Similarity Score', 'y': 'Chunk'},

            color=[c["score"] for c in retrieved],

            color\_continuous\_scale='blues'

        )

        st.plotly\_chart(fig, use\_container\_width=True)

        st.markdown("---")

        with st.expander("🧾 View Prompt Sent to LLM"):

            st.code(prompt\_used)

        st.markdown("💬 Was this answer useful?")

        cols = st.columns([1, 1])

        with cols[0]:

            if st.button("👍 Yes"):

                st.success("Thanks for your feedback!")

        with cols[1]:

            if st.button("👎 No"):

                st.info("We'll try to improve.")

with st.expander("ℹ️ Retrieval Strategy Explanation"):

    st.markdown("""

    \*\*Retrieval Approach Used:\*\*

    - ✅ Cosine Similarity over Sentence Embeddings

    - ✅ Optional Metadata Filtering (category, source, etc.)

    - ✅ Top-K configurable

    \*\*Why this works well:\*\*

    - Cosine similarity finds chunks that are semantically close to your query.

    - Metadata filtering improves \*precision\* by narrowing context to relevant sections.

    - Top-K tuning allows control over \*recall\* (bigger K → more results).

    This balances precision and recall effectively by combining dense vector search with symbolic filters.

    """)

with st.expander("➕ Add New Data (URL or Text)"):

    mode = st.radio("Input type:", ["🧭 URL", "📝 Raw Text"])

    new\_title = st.text\_input("Title")

    new\_data = st.text\_area("Paste your URL or text here")

    if st.button("📥 Add to Dataset"):

        if not new\_title or not new\_data:

            st.warning("Both title and data are required.")

        else:

            if mode == "🧭 URL":

                import requests

                from bs4 import BeautifulSoup

                try:

                    res = requests.get(new\_data)

                    soup = BeautifulSoup(res.text, "html.parser")

                    paragraphs = soup.find\_all("p")

                    new\_content = "\n".join(p.get\_text(strip=True) for p in paragraphs)

                except Exception as e:

                    st.error(f"Failed to fetch URL: {e}")

                    new\_content = ""

            else:

                new\_content = new\_data

            if new\_content.strip():

                # Chunk + Embed

                from sentence\_transformers import SentenceTransformer

                import faiss

                import numpy as np

                import nltk

                nltk.download('punkt')

                from nltk.tokenize import sent\_tokenize

                def chunk\_text(text, size=100, overlap=20):

                    words = text.split()

                    chunks = []

                    for i in range(0, len(words), size - overlap):

                        chunk = " ".join(words[i:i+size])

                        if len(chunk.split()) > 10:

                            chunks.append(chunk)

                    return chunks

                new\_chunks = chunk\_text(new\_content)

                new\_embeddings = model.encode(new\_chunks).astype("float32")

                faiss.normalize\_L2(new\_embeddings)

                index.add(new\_embeddings)

                chunks.extend(new\_chunks)

                metadata.extend([{"title": new\_title, "url": new\_data}] \* len(new\_chunks))

                user\_path = "user\_additions.pkl"

                if os.path.exists(user\_path):

                    with open(user\_path, "rb") as f:

                        existing = pickle.load(f)

                    user\_chunks = existing["chunks"] + new\_chunks

                    user\_metadata = existing["metadata"] + [{"title": new\_title, "url": new\_data}] \* len(new\_chunks)

                    user\_embeddings = np.concatenate([existing["embeddings"], new\_embeddings], axis=0)

                else:

                    user\_chunks = new\_chunks

                    user\_metadata = [{"title": new\_title, "url": new\_data}] \* len(new\_chunks)

                    user\_embeddings = new\_embeddings

                with open(user\_path, "wb") as f:

                    pickle.dump({

                        "chunks": user\_chunks,

                        "embeddings": user\_embeddings,

                        "metadata": user\_metadata

                    }, f)

                # # ذخیره به embedding\_store.pkl

                # import pickle

                # with open("embedding\_store.pkl", "wb") as f:

                #     pickle.dump({

                #         "chunks": chunks,

                #         "embeddings": index.reconstruct\_n(0, index.ntotal),  # بازسازی همه بردارها

                #         "metadata": metadata

                #     }, f)

                st.success(f"{len(new\_chunks)} chunks added successfully ✅")

st.markdown("📑 Chunks Used:")

for i, c in enumerate(retrieved):

    if any(snippet in answer for snippet in c["chunk"].split()[:10]):

        st.markdown(f"✅ Used part of Chunk {i+1}")

    else:

        st.markdown(f"⚠️ Probably not used Chunk {i+1}")

with st.expander("📂 View & Manage User-Added Data"):

    import os

    user\_path = "user\_additions.pkl"

    if os.path.exists(user\_path):

        with open(user\_path, "rb") as f:

            udata = pickle.load(f)

        st.markdown(f"\*\*🧠 Total Chunks:\*\* {len(udata['chunks'])}")

        for i, (chunk, meta) in enumerate(zip(udata["chunks"], udata["metadata"])):

            st.markdown(f"---\n### 🧩 Chunk {i+1} — {meta['title']}")

            st.code(chunk[:500] + "...")

            st.markdown(f"🔗 [Source]({meta['url']})")

            if st.button(f"❌ Delete Chunk {i+1}", key=f"del\_{i}"):

                udata["chunks"].pop(i)

                udata["metadata"].pop(i)

                udata["embeddings"] = np.delete(udata["embeddings"], i, axis=0)

                with open(user\_path, "wb") as f:

                    pickle.dump(udata, f)

                st.experimental\_rerun()

    else:

        st.info("No user-added data yet.")

from pyngrok import ngrok

ngrok.kill()

!pkill streamlit || echo "streamlit not running"

# اجرای Streamlit در بک‌گراند واقعی (با log)

!nohup streamlit run app.py > streamlit.log 2>&1 &

public\_url = ngrok.connect(8501, "http")

print("Your Streamlit app is live at:", public\_url)

**Flask : برای اجرای سرور و APIها**

**pyngrok : برای باز کردن تونل ngrok و نمایش وب‌سایت Flask در اینترنت**

conf.get\_default().auth\_token = "..."

اگر auth token نداشته باشیم، فقط تونل ۲ ساعته می‌تواند بسازد. با توکن رایگان، تا 8 ساعت باز می‌ماند.

**بخش Flask :**

* ساخت وب‌سرور با Flask برای دریافت فرمان کاربر

**رابط HTML :**

صفحه‌ای با فرم ساده شامل موارد مورد نیاز برای بازیابی و تولید پاسخ ها همراه با رابط کاربری زیبا.

**اتصال با ngrok :**

یک ابزار محبوب برای تونل‌سازی شبکه است که به ما اجازه می‌دهد:

* برنامه‌های **لوکال (محلی)** خود را به‌طور موقت در اینترنت در دسترس قرار دهیم.
* از طریق **HTTPS یا HTTP** به برنامه‌هایی که روی localhost اجرا می‌شوند دسترسی پیدا کنیم. پس آدرس عمومی ایجاد تا کاربر از طریق اینترنت به این رابط دسترسی داشته باشد.
* تست APIها، Webhookها مثل Stripe یا GitHub، و اپلیکیشن‌های وب را ساده‌تر کنیم.

مثال: اگر برنامه‌ای روی <http://localhost:3000> داریم، ngrok آن را روی یک URL مانند <https://abcd1234.ngrok.io> قابل دسترس می‌کند.

|  |  |
| --- | --- |
| بخش | توضیح |
| Flask backend | اجرای سرور، تحلیل متن ، تولید پاسخ |
| ngrok tunnel | تبدیل localhost به لینک اینترنتی برای مرورگر |
| HTML UI | کادر تایپ، نمایش پاسخ و ... |

**Vector Stores :**

**Vector Store**  یا **Vector Database**  محلی برای ذخیره و جستجوی بردارها embeddingهاست. وقتی متن‌ها را به embedding تبدیل می‌کنیم مثل MiniLM یا m2-BERT و می‌خواهیم بعداً با یک سوال جدید embedding شده به دنبال مشابه‌ترین متن‌ها بگردیم، این کار نیاز به مقایسه‌ی بردارها با هم دارد.

|  |  |  |
| --- | --- | --- |
| Vector Store | مزایا | معایب |
| FAISS | سریع، lightweight، بدون نیاز به سرور | فقط in-memory، بدون persistence |
| ChromaDB | پایتونی، ساده، با قابلیت persist و querying SQL-like | نسبتاً کندتر از FAISS |
| Qdrant | کامل‌ترین گزینه، REST/gRPC API، filterهای قوی | نیاز به اجرا به عنوان سرویس |
| In-memory | خیلی ساده مثل لیست یا numpy array | فقط برای پروژه‌های خیلی کوچک |

ما از **FAISS**  استفاده کردیم یعنی:

* + بردارها در حافظه ذخیره می‌شوند.( in-memory) یعنی با خاموش شدن سیستم، باید مجدد بارگذاری شود.
  + با استفاده از **cosine similarity**  جستجو می‌شوند چون normalize شده اند.
  + سریع، سبک و بدون نیاز به نصب سرور

**FAISS چیست؟**

مخفف **Facebook AI Similarity Search** و در واقع یک کتابخانهٔ **open-source** هست برای:

* ذخیره‌ی embeddingها (بردارهای عددی)
* و انجام **جستجوی برداری سریع** مثل nearest neighbors یا شباهت کسینوسی

**چرا FAISS را نیاز داریم؟**

در هر پروژه RAG :

1. متن‌هامثل صفحات Britannica به embedding تبدیل می‌شوند.
2. سوال کاربر هم به embedding تبدیل می‌شو.د
3. بعد، باید **سریع‌ترین** و **مشابه‌ترین** متن‌ها رو پیدا کنیم.
4. FAISS این مقایسه‌های برداری را **به‌صورت بهینه و سریع** انجام می‌دهد، حتی برای میلیون‌ها بردار

|  |  |
| --- | --- |
| وظیفه | توضیح |
| Index ساختن | ساختن ساختار داده‌ای از embeddingها برای جستجوی سریع |
| Normalization | اگر دنبال شباهت کسینوسی هستیم باید normalize شوند |
| Search | یافتن بردارهای نزدیک‌ترین به query (Top-K) |

انواع Indexها در FAISS:

|  |  |  |
| --- | --- | --- |
| Index Type | توضیح | مناسب برای |
| IndexFlatIP | جستجوی دقیق با inner product | پروژه‌های کوچک، حافظه زیاد |
| IndexIVFFlat | approximate search سریع‌تر | دیتاست‌های بزرگ |
| IndexHNSW | گرافی، سریع و دقیق | realtime + memory efficient |

**Optional Frameworks :**

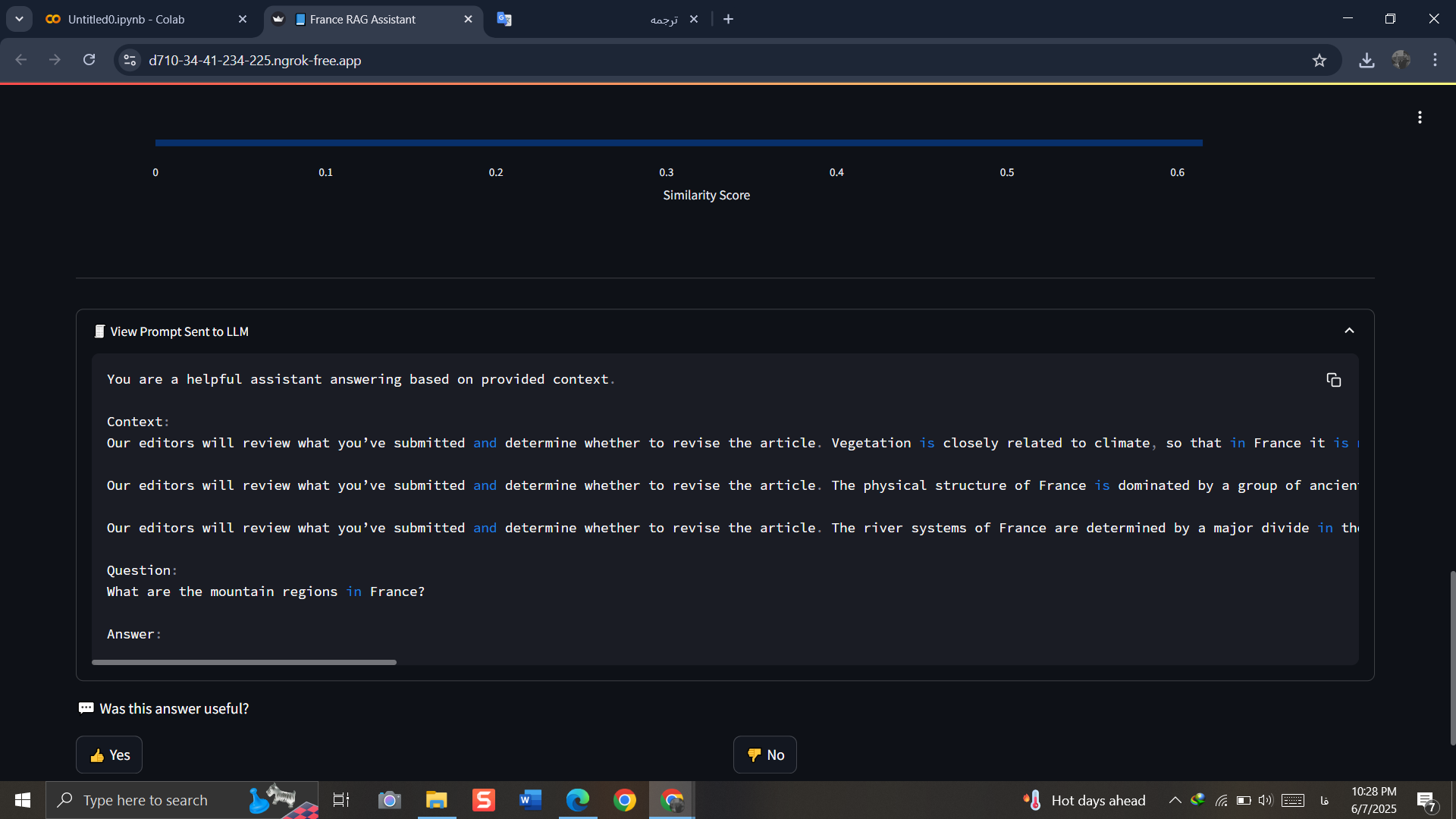
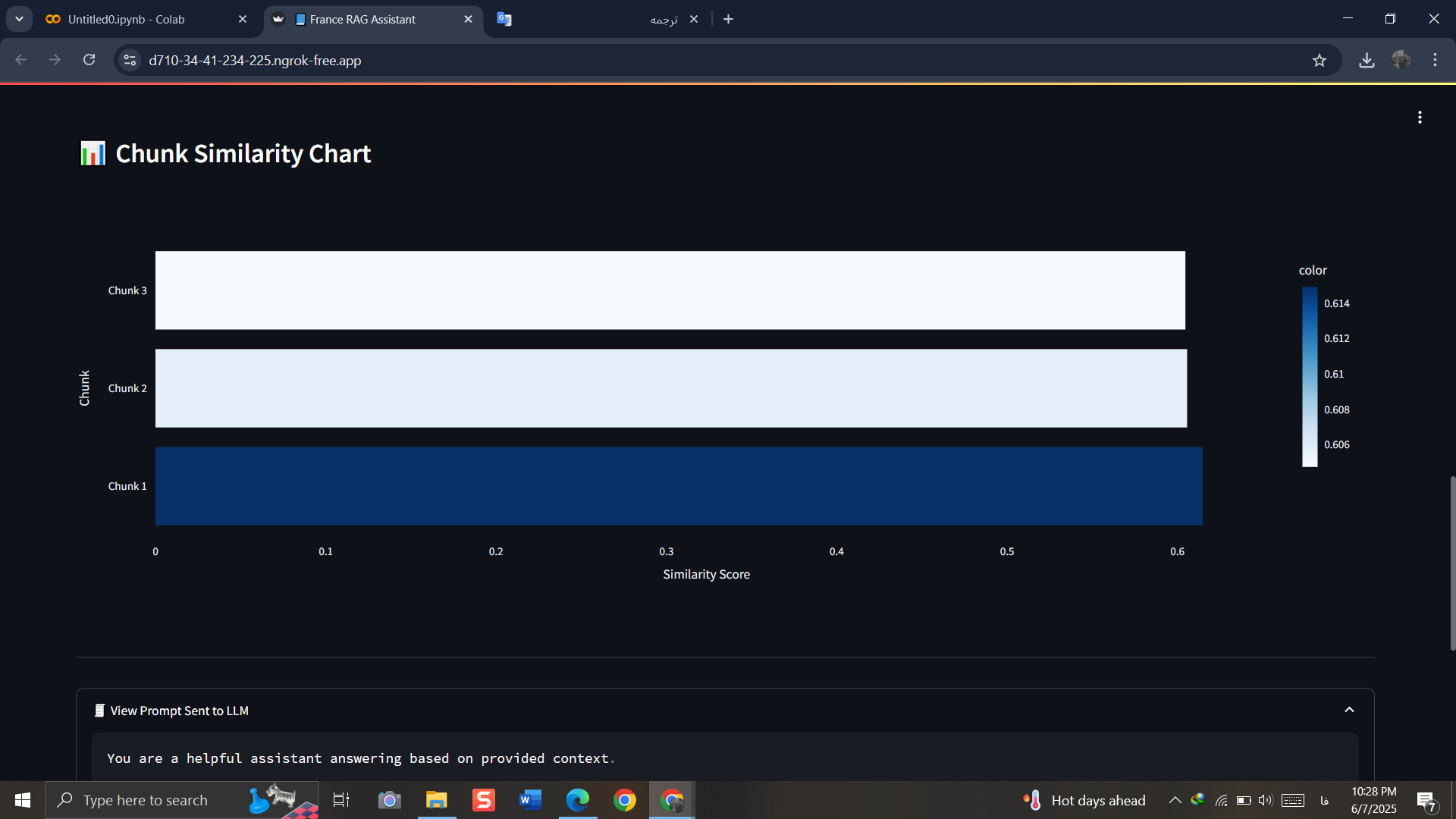
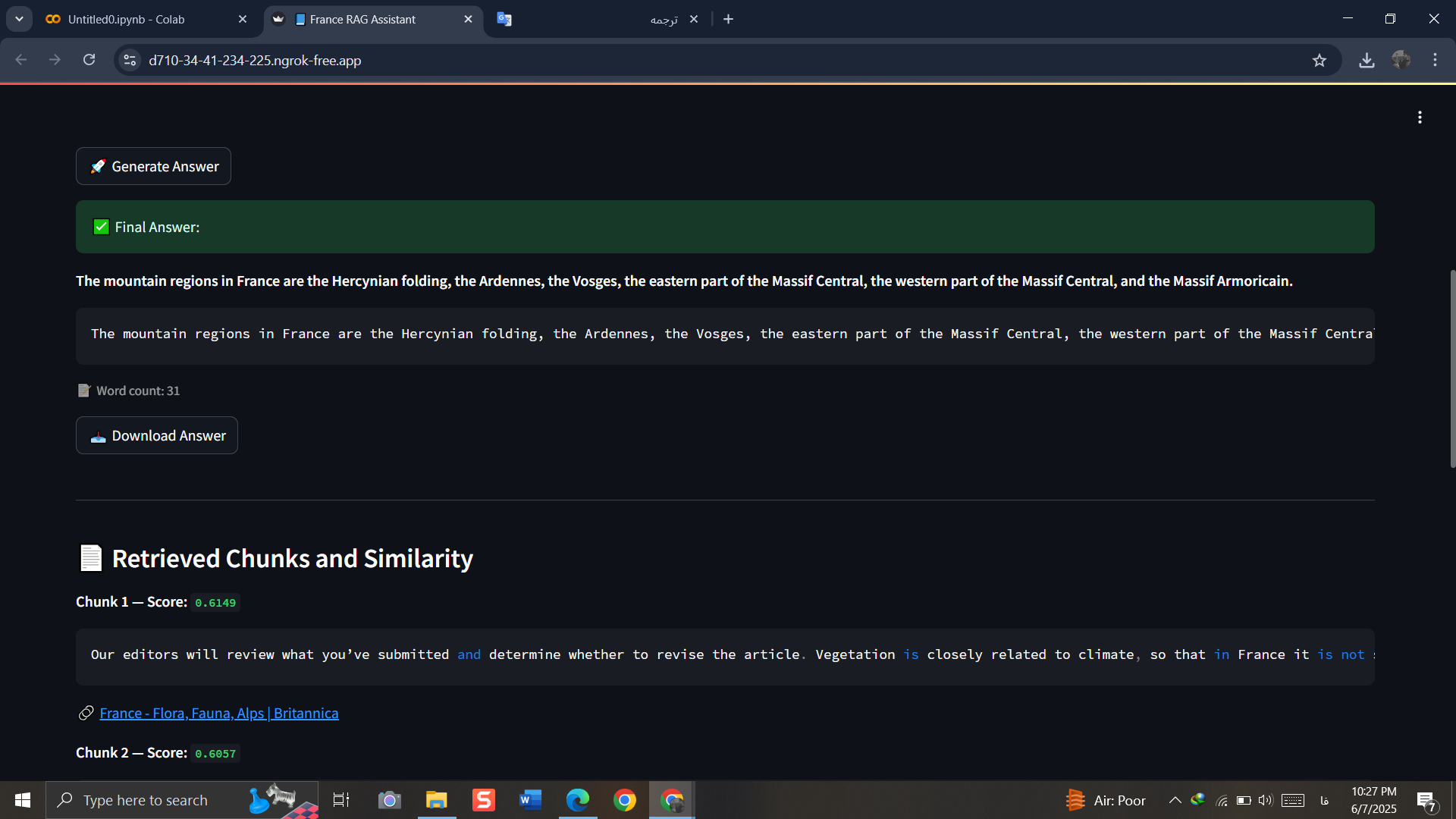
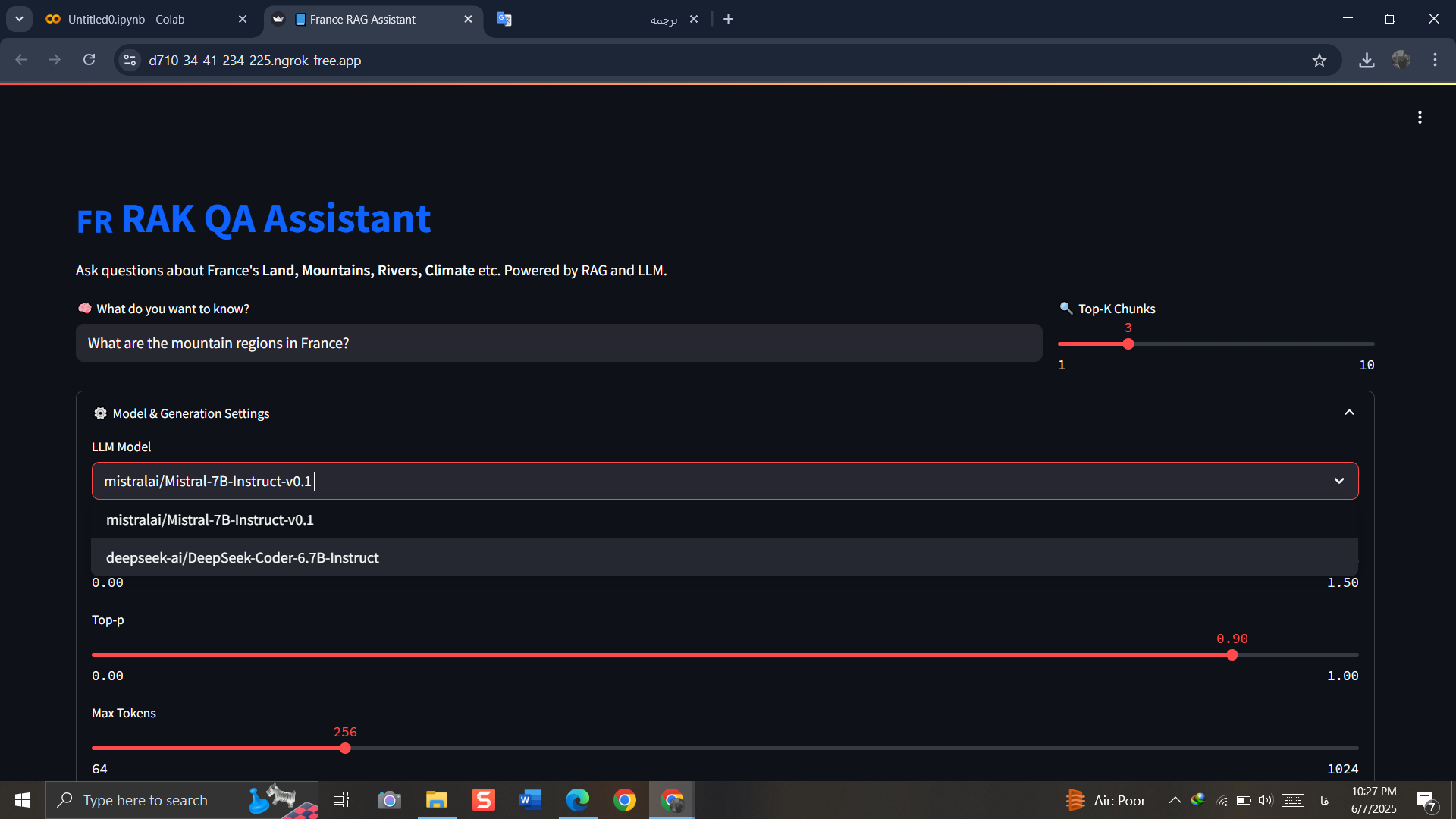
این‌ها فریمورک‌های سطح بالاتر هستند که به ما کمک می‌کنند RAG یا agent بسازیم بدون اینکه همه‌ی مراحل را دستی بنویسیم. اما ما از نسخه ی کاستوم استفاده کردیم تا انعطاف بیشتری داشته باشیم.

|  |  |  |
| --- | --- | --- |
| Framework | کارکرد | مزیت |
| LangChain | زنجیره‌های logic-based برای LLMها + memory + tools | abstraction قوی، اتصال آسان به LLM, DB, Tools |
| LlamaIndex (ex-GPT Index) | مخصوص اتصال LLM به مستندات (PDF, Notion, DB) | عالی برای RAG با document loaders آماده |
| Custom Code | مثل کاری که ما کردیم | کنترل کامل، انعطاف بالا |

ما از **Custom Code**  استفاده کردیم، یعنی:

|  |  |
| --- | --- |
| وظیفه | ما چه کاری انجام دادیم؟ |
| chunking | خودمان نوشتیم با NLTK + overlap |
| embedding | مستقیم با SentenceTransformer |
| vector store | مستقیم FAISS |
| retrieval | تابع retrieve\_top\_k() دستی |
| generation | ارسال مستقیم prompt به Together AI API |

نسخه نهایی و پیشرفته:



داده افزایی به دیتابیس و داینامیک بودن بازیابی: