**# extract 5000 random sample from “squad” dataset**

extract\_5000\_sample\_squad\_dataset.py

from datasets import load\_dataset

import random

import pandas as pd

# Load the SQuAD v1.1 training dataset

dataset = load\_dataset("squad", split="train")

# Convert to list of dictionaries

data = [{"context": item["context"], "question": item["question"], "answer": item["answers"]["text"][0]} for item in dataset]

# Sample 5,000 randomly

sampled\_data = random.sample(data, 5000)

# Save as DataFrame and export to CSV for inspection

df = pd.DataFrame(sampled\_data)

df.to\_csv("sampled\_squad\_5000.csv", index=False)

print("✅ Sampled 5,000 examples and saved to sampled\_squad\_5000.csv")

**# Generate synthetic answer for 1000 random sample from “squad” dataset**

**1**

Generate\_synthetic\_answers.py

import pandas as pd

from openai import OpenAI

from dotenv import load\_dotenv

import os

import time

# Load environment variables

load\_dotenv()

client = OpenAI()  # Auto-reads your OPENAI\_API\_KEY from .env

# Load the 5,000 sampled SQuAD rows

df = pd.read\_csv("sampled\_squad\_5000.csv").head(1000)

# Define function to call GPT-4o using new SDK

def ask\_gpt4o(context, question):

    prompt = f"""Use ONLY the following context to answer the question.

If you don’t know the answer from the context, say "I don’t know".

Context:

{context}

Question:

{question}

Answer:"""

    try:

        response = client.chat.completions.create(

            model="gpt-4o",

            messages=[

                {"role": "user", "content": prompt}

            ],

            temperature=0.2,

        )

        return response.choices[0].message.content.strip()

    except Exception as e:

        print("❌ Error:", e)

        return "ERROR"

# Generate synthetic answers

gpt\_answers = []

for i, row in df.iterrows():

    print(f"Processing {i+1}/1000")

    answer = ask\_gpt4o(row['context'], row['question'])

    gpt\_answers.append(answer)

    time.sleep(1)  # Avoid OpenAI rate limits

# Save output

df["gpt4o\_answer"] = gpt\_answers

df.to\_csv("squad\_gpt4o\_answers.csv", index=False)

print("✅ Finished! Saved to squad\_gpt4o\_answers.csv")

**# Calculate Hallucination**

**2**

hallucination\_check.py

from transformers import pipeline

import pandas as pd

# Load your first 1000 answers

df = pd.read\_csv("squad\_gpt4o\_answers.csv").head(1000)

# Load NLI (Natural Language Inference) model pipeline

classifier = pipeline("zero-shot-classification", model="facebook/bart-large-mnli")

# Define hallucination detection logic

def detect\_hallucination(context, answer):

    try:

        result = classifier(

            sequences=answer,

            candidate\_labels=["correct based on context", "hallucination", "unrelated"],

            hypothesis\_template="This answer is {}."

        )

        prediction = result["labels"][0]

        return prediction

    except Exception as e:

        print("❌ Error:", e)

        return "error"

# Run classification

hallucination\_labels = []

for i, row in df.iterrows():

    print(f"Checking {i+1}/1000")

    prediction = detect\_hallucination(row["context"], row["gpt4o\_answer"])

    hallucination\_labels.append(prediction)

# Save result

df["hallucination\_class"] = hallucination\_labels

df.to\_csv("hallucination\_scored\_1000.csv", index=False)

print("✅ Done. Saved results to hallucination\_scored\_1000.csv")

**#Draw Hallucination rate chart**

**3**

hallucination\_rate\_chart.py

import pandas as pd

import pandas as pd

import matplotlib.pyplot as plt

# Load scored results

df = pd.read\_csv("hallucination\_scored\_1000.csv")

# Count total and hallucinated

total = len(df)

hallucinated = (df["hallucination\_class"] == "unrelated").sum()

grounded = (df["hallucination\_class"] == "correct based on context").sum()

# Calculate rate

rate = (hallucinated / total) \* 100

# Print stats

print("📊 Total Samples:", total)

print("✅ Grounded Answers:", grounded)

print("❌ Hallucinated Answers:", hallucinated)

print(f"🤯 Hallucination Rate: {rate:.2f}%")

# Load hallucination-labeled results

df = pd.read\_csv("hallucination\_scored\_1000.csv")  # Make sure this file exists in your folder

# Count label frequencies

counts = df["hallucination\_class"].value\_counts()

# Prepare pie chart

labels = counts.index

sizes = counts.values

colors = ['#66b3ff', '#ff6666']  # You can change these

explode = [0.05] \* len(labels)   # Slightly separate slices

# Create the pie chart

plt.figure(figsize=(6, 6))

plt.pie(

    sizes,

    labels=labels,

    colors=colors,

    explode=explode,

    autopct='%1.1f%%',

    startangle=140

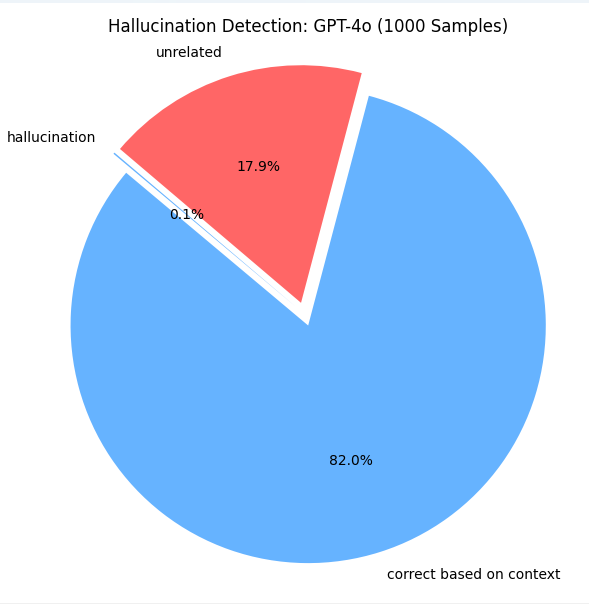
)

plt.title("Hallucination Detection: GPT-4o (1000 Samples)")

plt.axis('equal')  # Ensures it's a perfect circle

plt.tight\_layout()

plt.show()



Using RAG:

**#Indexing using FAISS**

prepare\_faiss.py

import pandas as pd

from langchain.vectorstores import FAISS

from langchain\_community.embeddings import OpenAIEmbeddings

from langchain.schema import Document

from langchain.text\_splitter import CharacterTextSplitter

# Load SQuAD data

df = pd.read\_csv("sampled\_squad\_5000.csv").head(1000)  # Use same 1000 rows

import os

from dotenv import load\_dotenv

load\_dotenv()

openai\_api\_key = os.getenv("OPENAI\_API\_KEY")

embeddings = OpenAIEmbeddings(openai\_api\_key=openai\_api\_key)

# Turn contexts into Document objects

documents = [Document(page\_content=row["context"]) for \_, row in df.iterrows()]

# Optional: Chunking (helps FAISS indexing)

splitter = CharacterTextSplitter(chunk\_size=500, chunk\_overlap=50)

chunks = splitter.split\_documents(documents)

# Embed using OpenAI

embeddings = OpenAIEmbeddings()

# Create FAISS vector store

vectorstore = FAISS.from\_documents(chunks, embeddings)

# Save for later

vectorstore.save\_local("faiss\_index")

print("✅ FAISS vectorstore created and saved.")

**#Indexing load FAISS +** **retriever + LLM**

generate\_rag\_answers.py

import pandas as pd

from langchain\_community.vectorstores import FAISS

from langchain\_community.embeddings import OpenAIEmbeddings

from langchain.chat\_models import ChatOpenAI

from langchain.chains import RetrievalQA

from dotenv import load\_dotenv

import os

import time

# Load API key

load\_dotenv()

api\_key = os.getenv("OPENAI\_API\_KEY")

# Load FAISS

embeddings = OpenAIEmbeddings(openai\_api\_key=api\_key)

vectorstore = FAISS.load\_local("faiss\_index", embeddings, allow\_dangerous\_deserialization=True)

# Create retriever + LLM

retriever = vectorstore.as\_retriever()

llm = ChatOpenAI(model="gpt-4o", temperature=0.2, openai\_api\_key=api\_key)

qa\_chain = RetrievalQA.from\_chain\_type(llm=llm, retriever=retriever)

# Load 1000 SQuAD samples

df = pd.read\_csv("sampled\_squad\_5000.csv").head(1000)

# Generate answers

rag\_answers = []

for i, row in df.iterrows():

    print(f"Processing {i+1}/1000")

    result = qa\_chain.run(row["question"])

    rag\_answers.append(result)

    time.sleep(1)  # Respect rate limits

# Save results

df["rag\_answer"] = rag\_answers

df.to\_csv("rag\_answers\_1000.csv", index=False)

print("✅ Done! RAG answers saved to rag\_answers\_1000.csv")

**#Detect Hallucinations in RAG Answers**

detect\_rag\_hallucinations.py

from transformers import pipeline

import pandas as pd

# Load answers

df = pd.read\_csv("rag\_answers\_1000.csv")

# Load classifier

classifier = pipeline("zero-shot-classification", model="facebook/bart-large-mnli")

# Detection function

def is\_hallucinated(context, answer):

    try:

        result = classifier(

            sequences=answer,

            candidate\_labels=["based on the context", "not based on the context"],

            hypothesis\_template="This answer is {}."

        )

        return result['labels'][0]

    except Exception as e:

        print("❌", e)

        return "error"

# Run on all RAG answers

labels = []

for i, row in df.iterrows():

    print(f"Checking {i+1}/1000")

    label = is\_hallucinated(row["context"], row["rag\_answer"])

    labels.append(label)

# Save with hallucination labels

df["hallucination\_label"] = labels

df.to\_csv("rag\_hallucination\_scored\_1000.csv", index=False)

print("✅ RAG hallucination results saved to rag\_hallucination\_scored\_1000.csv")

**#Draw Hallucination rate chart (RAG)**

RAG\_piecharts.py

import pandas as pd

import matplotlib.pyplot as plt

# Load RAG results only

df\_rag = pd.read\_csv("rag\_hallucination\_scored\_1000.csv")

# Count hallucination labels

rag\_counts = df\_rag["hallucination\_label"].value\_counts()

# Get values

total = len(df\_rag)

grounded = rag\_counts.get("based on the context", 0)

hallucinated = rag\_counts.get("not based on the context", 0)

rate = (hallucinated / total) \* 100

# Print stats with emojis

print("📊 Total Samples:", total)

print("✅ Grounded Answers:", grounded)

print("❌ Hallucinated Answers:", hallucinated)

print(f"🤯 Hallucination Rate: {rate:.2f}%")

# Setup for pie chart

labels = ["based on the context", "not based on the context"]

colors = ["#66b3ff", "#ff6666"]

explode = [0.05, 0.05]

# Plot pie chart

plt.figure(figsize=(6, 6))

plt.pie(

    [grounded, hallucinated],

    labels=labels,

    autopct='%1.1f%%',

    startangle=140,

    colors=colors,

    explode=explode

)

plt.title("🤖 RAG Answer Hallucination Rate", fontsize=14)

plt.tight\_layout()

plt.show()