**Le paso una lista de canales**

from googleapiclient.discovery import build

import re

from datetime import datetime, timedelta

import random

import pytz

import time

from elasticsearch import Elasticsearch

from elasticsearch.helpers import bulk

import redis

import random

from googleapiclient.errors import HttpError

# Conectarse al servidor Redis local

r = redis.Redis(host='192.168.1.191', port=6379, db=0)

**def busqueda\_videos3(tema1):**

#channel\_id='UCx\_qTw9XEzMNcsh2H0S95AQ'

max=3

d=60

es1='ES'

tema, max, d, es1,canalid = tema1.split("#")

print (tema, max, d, es1,canalid)

now = datetime.now(utc)

two\_months\_ago = now - timedelta(days=int(d)) ##########################

# Formatear las fechas en el formato adecuado para la API de YouTube (RFC 3339)

published\_after = two\_months\_ago.isoformat("T")

print (published\_after)

published\_before = now.isoformat("T")

if es1=='ALL' :

request = youtube.search().list(

part='snippet',

q=tema, # Tema de búsqueda

channelId=canalid,

maxResults=int(max), #############################

type='video',

publishedAfter=published\_after,

publishedBefore=published\_before

# regionCode='ES' ################################

)

else :

print ('regioncode ',es1)

request = youtube.search().list(

part='snippet',

q=tema, # Tema de búsqueda

channelId=canalid,

maxResults=int(max), #############################

type='video',

publishedAfter=published\_after,

publishedBefore=published\_before,

regionCode=es1 ################################

)

response = request.execute()

results = response['items']

maxi= int(max)

if 50 < maxi:

while 'nextPageToken' in response and 0 < maxi:

next\_page\_token = response['nextPageToken']

request = youtube.search().list(

part='snippet',

q=tema, # Tema de búsqueda

channelId=canalid,

maxResults=maxi, # Máximo permitido por solicitud

type='video',

publishedAfter=published\_after,

publishedBefore=published\_before,

pageToken=next\_page\_token

)

response = request.execute()

results.extend(response['items'])

maxi=maxi-50

links=[]

# Imprime la información de los videos

canal = 'inferencia2'

for item in results:

title = item['snippet']['title']

video\_id = item['id']['videoId']

video\_url = f'https://www.youtube.com/watch?v={video\_id}'

if search\_articles\_by\_keyword\_url(video\_url,size=1)[0]==0:

print(f'Título: {title}\nURL: {video\_url}')

#print(f'{video\_url}')

links.append(video\_url)

print ('#####',len(links))

for aa in links:

r.rpush(canal, aa)

print (aa)

listaaa=['UCvPiwvp-OlGsQxUyoY1j9lA','UCVSAQp91Q6LRu7ChXDbd8Iw','UCvSeACNsmtlm33iAkao978A']

for canalid in listaaa:

cont=cont+1

print ("#####",canalid)

tema1='#2000#5000#ALL'+'#'+str(canalid)

if cont>1:

api\_key = random.choice(listaapi)

youtube = build('youtube', 'v3', developerKey=api\_key)

cont=0

try:

busqueda\_videos3(tema1)

except HttpError as e:

api\_key = random.choice(listaapi)

youtube = build('youtube', 'v3', developerKey=api\_key)

try:

busqueda\_videos3(tema1)

except HttpError as e:

api\_key = random.choice(listaapi)

youtube = build('youtube', 'v3', developerKey=api\_key)

busqueda\_videos3(tema1)

**Búsqueda por idiomas**

listidioma= (texto\_idiomas( texto)) # llamada al llm

**Búsqueda por tema**

**def busqueda\_videos2(tema1):**

canal1='youtube'

canal\_id1="C06QVUGMMQX"

enviar\_mensaje\_slack ("###terminado tema#n\_videos#dias ultimos#ES/ALL")

max=3

d=60

es1='ES'

tema, max, d, es1 = tema1.split("#")

print (tema, max, d, es1 )

two\_months\_ago = now - timedelta(days=int(d)) ##########################

# Formatear las fechas en el formato adecuado para la API de YouTube (RFC 3339)

published\_after = two\_months\_ago.isoformat("T")

print (published\_after)

published\_before = now.isoformat("T")

if es1=='ALL' :

request = youtube.search().list(

part='snippet',

q=tema, # Tema de búsqueda

#channelId=channel\_id,

maxResults=int(max), #############################

type='video',

publishedAfter=published\_after,

publishedBefore=published\_before

# regionCode='ES' ################################

)

else :

print ('regioncode ',es1)

request = youtube.search().list(

part='snippet',

q=tema, # Tema de búsqueda

maxResults=int(max), #############################

type='video',

publishedAfter=published\_after,

publishedBefore=published\_before,

regionCode=es1 ################################

)

response = request.execute()

results = response['items']

maxi= int(max)

if 50 < maxi:

while 'nextPageToken' in response and 0 < maxi:

next\_page\_token = response['nextPageToken']

request = youtube.search().list(

part='snippet',

q=tema, # Tema de búsqueda

maxResults=maxi, # Máximo permitido por solicitud

type='video',

publishedAfter=published\_after,

publishedBefore=published\_before,

pageToken=next\_page\_token

)

response = request.execute()

results.extend(response['items'])

maxi=maxi-50

links=[]

# Imprime la información de los videos

canal = 'inferencia2'

for item in results:

title = item['snippet']['title']

video\_id = item['id']['videoId']

video\_url = f'https://www.youtube.com/watch?v={video\_id}'

if search\_articles\_by\_keyword\_url(video\_url,size=1)[0]==0:

print(f'Título: {title}\nURL: {video\_url}')

#print(f'{video\_url}')

links.append(video\_url)

r.rpush(canal, video\_url)

print ('#####',len(links))

print ('####',links)

cont=1

resultado=''

n\_videos=len(links)

#with concurrent.futures.ThreadPoolExecutor(max\_workers=5) as executor:

# executor.map(download\_audio, links)

#executor.map(ytvd, links)

#for link in links:

#encontrado=1

# fichero\_descargado=download\_audio( link)

#result2 = pipe(directoriotemp+fichero\_descargado, generate\_kwargs={"language": "spanish"})

#print (result2['text'])

#resultado=resultado+' '+result2['text']

resultado="prueba"

return links

**Configuración ollama**

import ollama

from langchain\_community.llms import Ollama

from datetime import datetime

import random

lollama='http://192.168.1.191:11434'

ollama = Ollama(

#base\_url='http://192.168.1.191:11434',

base\_url=lollama,

model="llama3.1:70b"

#model="mistral-large:latest"

)

lollama2='http://192.168.1.20:11434'

ollama2 = Ollama(

base\_url=lollama2,

#base\_url='http://192.168.1.20:11434',

model="llama3.1:70b"

#model="mistral-large:latest"

)

lollamap='http://192.168.1.144:11434'

ollamap = Ollama(

#base\_url='http://192.168.1.191:11434',

#base\_url='http://192.168.1.20:11434',

base\_url=lollamap,

model="llama3.1"

#model="mistral-large:latest"

)

resultado=(ollama.invoke("eres una analista tecnico, puedes hacerme el resumen largo sin perder detalles de este texto:"+ARTICLE) )

resultado=(ollamap.invoke("en que idioma esta, solo indica el idioma:"+titulo) )

titulot=(ollamap.invoke("traduce al español, solo indica la traduccion:"+titulo) )

**Transcripción**

result2 = pipe(directoriotemp+fichero\_descargado, generate\_kwargs={"language": "spanish"})

from datasets import load\_dataset

from transformers import AutoModelForSpeechSeq2Seq, AutoProcessor

# Set device and data type based on availability of CUDA

device = "cuda:0" if torch.cuda.is\_available() else "cpu"

torch\_dtype = torch.float16 if torch.cuda.is\_available() else torch.float32

# Model ID for the Whisper model

model\_id = "openai/whisper-large-v3"

# Load the model with specified dtype and memory usage optimizations

model = AutoModelForSpeechSeq2Seq.from\_pretrained(

model\_id, torch\_dtype=torch\_dtype, low\_cpu\_mem\_usage=True, use\_safetensors=True

)

# Move the model to the appropriate device

model.to(device)

# Load the processor associated with the model

processor = AutoProcessor.from\_pretrained(model\_id)

pipe = pipeline(

"automatic-speech-recognition",

model=model,

tokenizer=processor.tokenizer,

feature\_extractor=processor.feature\_extractor,

max\_new\_tokens=128,

chunk\_length\_s=30,

**batch\_size=16,**

return\_timestamps=True,

torch\_dtype=torch\_dtype,

device=device,

)

dataset = load\_dataset("distil-whisper/librispeech\_long", "clean", split="validation")

device = torch.device('cuda:0')

#device = torch.device('cpu')

summarizer = pipeline("summarization", model="facebook/bart-large-cnn",device=device)

**Descargar archivos**

def consumidor\_de\_cola(redis\_conn, nombre\_cola):

while True:

# Usamos BRPOP para esperar y desencolar un elemento de la cola

# BRPOP es una operación bloqueante que espera hasta que haya un elemento disponible

# El segundo argumento es el tiempo de espera, 0 significa bloqueo indefinido

\_, mensaje = redis\_conn.brpop(nombre\_cola, 0)

# Procesar el mensaje

# Asumiendo que los mensajes son cadenas, decodificar si es necesario

mensaje = mensaje.decode('utf-8')

print (mensaje)

if search\_articles\_by\_keyword\_url(mensaje,size=1)[0]==0:

print ("###",datetime.now().time(),mensaje)

try:

fichero\_descargado=ytvd( mensaje)

except:

fichero\_descargado=ytvd( mensaje)

canal2 = 'inferencia5'

print ("fichero descargado",fichero\_descargado)

#fichero\_descargado=comprobar\_archivo (fichero\_descargado)

time.sleep(5)

r.rpush(canal2, fichero\_descargado+"|||"+mensaje)

print ("enviado ", fichero\_descargado+"|||"+mensaje)

else:

resumen=search\_articles\_by\_keyword\_url(mensaje,size=1)[1]

enviar\_mensaje\_slack ("-------------------------------------------------\n\n")

hora\_actual = datetime.now()

hora\_formateada = hora\_actual.strftime("%H:%M:%S")

enviar\_mensaje\_slack ("La hora actual es: "+ hora\_formateada)

enviar\_mensaje\_slack (mensaje)

enviar\_mensaje\_slack (resumen)

enviar\_mensaje\_slack ("------

def ytvd(url, output\_path='/sat'):

"""Download only the audio from a YouTube video as an MP3 file."""

global fichero # Ensure global scope for fichero variable

print(output\_path)

def my\_hook(d):

if d['status'] == 'finished':

print(f"Descarga completada, archivo guardado como: {d['filename']}")

global fichero

fichero = d['filename']

ydl\_opts = {

'format': 'bestaudio/best', # Select the best quality audio

'postprocessors': [{

'key': 'FFmpegExtractAudio',

'preferredcodec': 'mp3', # Specify the codec to mp3

'preferredquality': '192', # Specify the quality (bitrate)

}],

'outtmpl': f'{output\_path}/%(title)s.%(ext)s', # Save file to the output\_path with the video title

'noplaylist': True, # Download only the video, not a playlist

'progress\_hooks': [my\_hook], # Add the progress hook

}

try:

with yt\_dlp.YoutubeDL(ydl\_opts) as ydl:

ydl.download([url])

except Exception as e:

try:

with yt\_dlp.YoutubeDL(ydl\_opts) as ydl:

ydl.download([url])

except Exception as e:

print(f"Error downloading video: {e}")

return None

fichero1 = os.path.basename(fichero)

#fichero2=fichero1

#fichero1, extension = os.path.splitext(fichero1)

#print (fichero1, extension)

print ("A0 ",fichero1[-3:])

if fichero1[-3:]=='mp3' or fichero1[-3:]=='m4a':

fichero1 = fichero1[:-4] + ".mp3"

print ("A1 ",fichero1)

else:

fichero1 = fichero1[:-5] + ".mp3"

print ("A2 ",fichero1)

fichero1corto = add\_random\_string(fichero1[:-5][:30]) + ".mp3"

fichero1corto=unicodedata.normalize('NFKD', fichero1corto)

shutil.copy('/sat/'+fichero1, directoriotemp+fichero1corto)

print ("copiamos "+'/sat/'+fichero1+ 'a /mnt/credibyou/'+fichero1corto)

# Busca todos los archivos que empiecen con "pajaro"

patron = os.path.join('/sat/', fichero1[:-5]+"\*")

archivos\_a\_borrar = glob.glob(patron)

# Itera sobre la lista de archivos y los borra

for archivo in archivos\_a\_borrar:

try:

os.remove(archivo)

print(f"Archivo {archivo} borrado con éxito.")

except Exception as e:

print(f"No se pudo borrar {archivo}: {e}")

fichero1corto=fichero1corto

print('output', fichero1corto)

return fichero1corto