Creating a Useful Voice-Activated Fully Local RAG System

This article will explore initiating the RAG system and making it fully voice-activated.

By **Cornellius Yudha Wijaya**, KDnuggets Technical Content Specialist on February 11, 2025 in **Language Models**

Image by Author | Ideogram.ai

RAG, or retrieval augmented generation, is a technique for using external knowledge for additional context that passes into the large language model (LLM) to improve the model's accuracy and relevance. It's a much more reliable way to improve the generative AI result than constantly fine-tuning the model.

Traditionally, RAG systems rely on user text queries to search the vector database. The relevant documents retrieved are then used as context input for the generative AI, which

Latest Posts

5 LLM Prompting Techniques Every Developer Should Know

Top 5 Freelancer Websites Better Than Fiverr and Upwork

Creating a Useful Voice-Activated Fully Local RAG System

10 Little-Known Python Libraries That Will Make You Feel Like a Data Wizard

Beginner's Guide to Subqueries in SQL

Data Science Showdown: Which Tools Will Gain Ground in 2025

Top Posts

produces the result in text format. However, we can extend them even more so that they can accept and produce output in voice form.

This article will explore initiating the RAG system and making it fully voice-activated.

Building a Fully Voice-Activated RAG System

In this article, I will assume the reader has some knowledge about the LLM and RAG systems, so I will not explain them further.

To build a RAG system with full voice features, we'll structure it around three key components:

1. Voice Receiver and Transcription
2. Knowledge Base
3. Audio File Response Generation

Overall, the project workflow will follow the image below.

10 Little-Known Python Libraries That Will Make You Feel Like a Data Wizard

Using Llama 3.2-Vision Locally: A Step-by-Step Guide

Beginner's Guide to Subqueries in SQL

Python Oddities That Might Surprise You

Building AI Application with Gemini 2.0

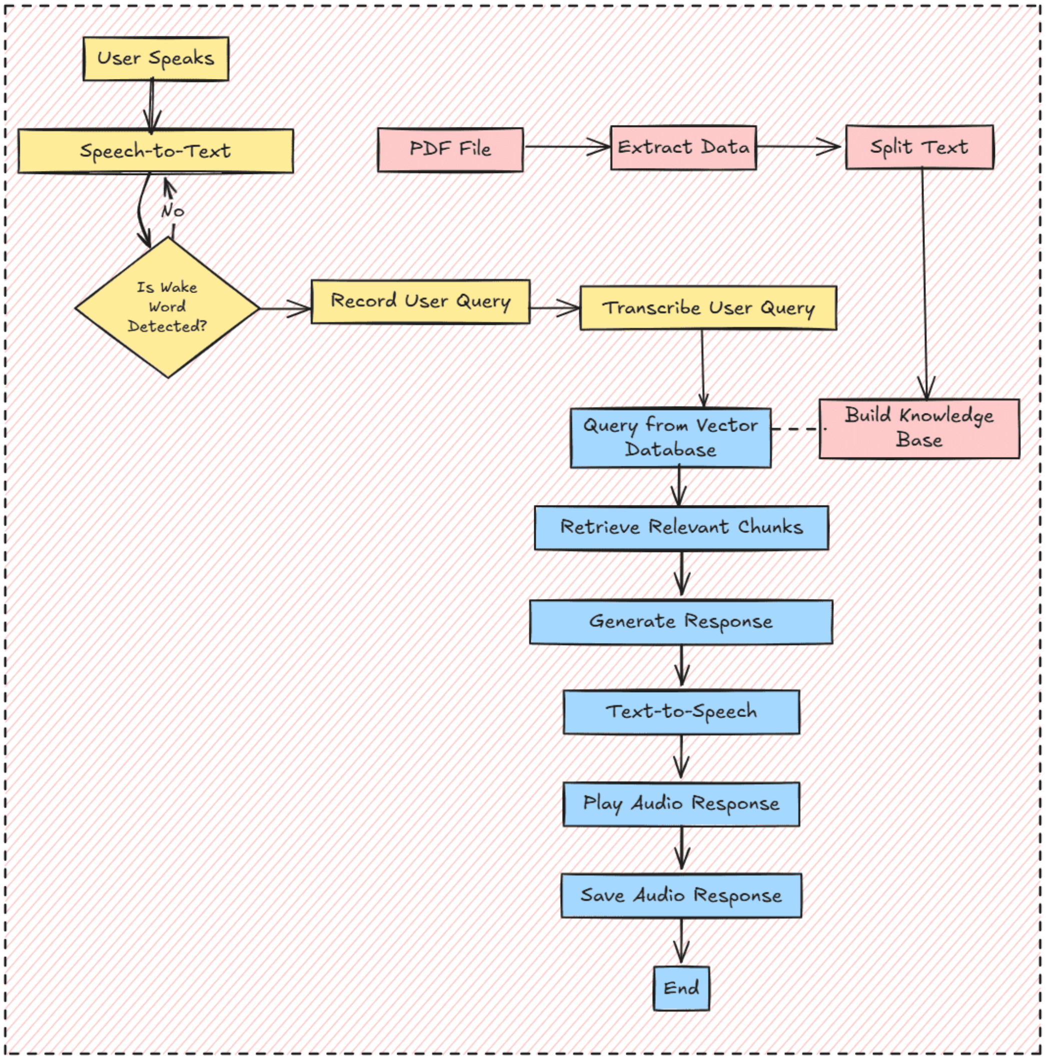
How to Fine-Tune DeepSeek-R1 for Your Custom Dataset (Step-by-Step)

Building Multilingual Applications with Hugging Face Transformers: A Beginner's Guide

Using Gemini 2.0 Pro Locally

5 AI Agent Frameworks Compared

Data Science Showdown: Which Tools Will Gain Ground in 2025



If you are ready, let's get started to prepare all we need for this project to succeed.

First, we will not use the Notebook IDE for this project, as we want the RAG system to work like things in production.
Therefore, a standard programming language IDE such as Visual Studio Code (VS Code) should be prepared.

Next, we also want to create a virtual environment for our project. You can use any method, such as Python or Conda.

```
python -m venv rag-env-audio
```

With your virtual environment ready, we must install all the necessary libraries for this tutorial.

```
pip install openai-whisper chromadb sentence-transformers sounddevice numpy sc
```

If you have GPU access, you can also download the GPU version of the PyTorch library.

```
pip install torch torchaudio --index-url https://download.pytorch.org/whl/cu11
```

With everything ready, we will start to build our voice-activated RAG system. As a note, the project repository containing all the code and dataset is in this repository.

We will start by importing all the necessary libraries and the environmental variables with the following code.

```
import os
import whisper
import chromadb
from sentence_transformers import SentenceTransformer
import sounddevice as sd
import numpy as np
from scipy.io.wavfile import write
from sklearn.metrics.pairwise import cosine_similarity
from transformers import AutoModelForCausalLM, AutoTokenizer
from langchain_text_splitters import RecursiveCharacterTextSplitter
import torch

AUDIO_FILE = "user_input.wav"
RESPONSE_AUDIO_FILE = "response.wav"
PDF_FILE = "Insurance_Handbook_20103.pdf"
SAMPLE_RATE = 16000
WAKE_WORD = "Hi"
SIMILARITY_THRESHOLD = 0.4
MAX_ATTEMPTS = 5
```

All the variables will be explained when it’s used in their respective code. For now, let’s just keep it as it is.

After we import all the necessary libraries, we will set up all the functions necessary for the RAG systems. I will break it down individually so you can understand what happened in our project.

The first step is to create a feature to record our input voice and transcribe the voice into text data. We will use the sound device library for audio recording, and for audio transcribing, we will use OpenAI Whisper.

```
# For recording audio input.
def record_audio(filename, duration=5, samplerate=SAMPLE_RATE):
    print("Listening... Speak now!")
    audio = sd.rec(int(duration * samplerate), samplerate=samplerate, channels
    sd.wait()
    print("Recording finished.")
    write(filename, samplerate, (audio * 32767).astype(np.int16))

# Transcribe the Input audio into text
def transcribe_audio(filename):
    print("Transcribing audio...")
    model = whisper.load_model("base.en")
    result = model.transcribe(filename)
    return result["text"].strip().lower()
```

The above functions will become the foundation for accepting and returning our voice as text data. We will use them multiple times during the project, so keep it in mind.

We will create an entrance feature for our RAG system with the functions to accept audio ready. In the next code, we create a voice-activation function before accessing the system using `WAKE_WORD`. The word can be anything; you can set it up as required.

The idea behind the voice activation above is that the RAG system is activated if our recorded transcribed voice matches the Wake Word. However, it will not be feasible if the transcription needs to match Wake Word exactly, as the possibility of the transcription system producing the text result in a different format is high. We can standardize the transcribed output for that. However, for now, I have an idea to use embedding similarities so the system is still activated even with slightly different word compositions.

```
# Detecting Wake Word to activate the RAG System
def detect_wake_word(max_attempts=MAX_ATTEMPTS):

    print("Waiting for wake word...")
    text_embedding_model = SentenceTransformer('all-MiniLM-L6-v2')
    wake_word_embedding = text_embedding_model.encode(WAKE_WORD).reshape(1, -1

    attempts = 0
    while attempts = SIMILARITY_THRESHOLD:
            print(f"Wake word detected: {WAKE_WORD}")
            return True
        attempts += 1
        print(f"Attempt {attempts}/{max_attempts}. Please try again.")
    print("Wake word not detected. Exiting.")
    return False
```

By combining the `WAKE_WORD` and `SIMILARITY_THRESHOLD` variables, we will end up with the voice activation feature.
Next, let's build our knowledge base using our PDF file. To do that, we will prepare a function that extracts text from the file and splits it into chunks.

```
def load_and_chunk_pdf(pdf_file):
    from PyPDF2 import PdfReader
    print("Loading and chunking PDF...")
    reader = PdfReader(pdf_file)
    all_text = ""
    for page in reader.pages:
        text = page.extract_text()
        if text:
            all_text += text + "\n"

    # Split the text into chunks
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=250,  # Size of each chunk
        chunk_overlap=50,  # Overlap between chunks to maintain context
        separators=["\n\n", "\n", " ", ""]
     )
    chunks = text_splitter.split_text(all_text)
    return chunks
```

You can replace the chunk size with your intended. There are no exact numbers to use, so experiment with them to see which is the best parameter.

The chunks from the function above will then be passed into the vector database. We will use the ChromaDB vector database and SenteceTransformer to access the embedding model.

```
 setup_chromadb(chunks):
 print("Setting up ChromaDB...")
 client = chromadb.PersistentClient(path="chroma_db")
 text_embedding_model = SentenceTransformer('all-MiniLM-L6-v2')

 # Delete existing collection (if needed)
 try:
     client.delete_collection(name="knowledge_base")
     print("Deleted existing collection: knowledge_base")
 except Exception as e:
     print(f"Collection does not exist or could not be deleted: {e}")

 collection = client.create_collection(name="knowledge_base")

 for i, chunk in enumerate(chunks):
     embedding = text_embedding_model.encode(chunk).tolist()
     collection.add(
         ids=[f"chunk_{i}"],
         embeddings=[embedding],
         metadatas=[{"source": "pdf", "chunk_id": i}],
         documents=[chunk]
     )
 print("Text chunks and embeddings stored in ChromaDB.")
 return collection
litionally, we will prepare the function for retrieval with the text query to Ch
 query_chromadb(collection, query, top_k=3):
 """Query ChromaDB for relevant chunks."""
 text_embedding_model = SentenceTransformer('all-MiniLM-L6-v2')
 query_embedding = text_embedding_model.encode(query).tolist()
 results = collection.query(
     query_embeddings=[query_embedding],
     n_results=top_k
 )

 relevant_chunks = [chunk for sublist in results["documents"] for chunk in subl
 return relevant_chunks
```

Then, we need to prepare the generation feature to complete the RAG system. In this case, I will use the Qwen-1.5.-0.5B-Chat model hosted in the HuggingFace. You can tweak the prompt and the generation model as you please.

```
def generate_response(query, context_chunks):
    device = "cuda" if torch.cuda.is_available() else "cpu"
    model_name = "Qwen/Qwen1.5-0.5B-Chat"
    model = AutoModelForCausalLM.from_pretrained(
        model_name,
        torch_dtype="auto",
        device_map="auto"
    )
    tokenizer = AutoTokenizer.from_pretrained(model_name)

    # Format the prompt with the query and context
    context = "\n".join(context_chunks)
    messages = [
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": f"Use the following context to answer the 
    ]

    text = tokenizer.apply_chat_template(
        messages,
        tokenize=False,
        add_generation_prompt=True
    )

    model_inputs = tokenizer(
        [text],
        return_tensors="pt",
        padding=True,
        truncation=True
    ).to(device)

    # Generate the response
    generated_ids = model.generate(
        model_inputs.input_ids,
        attention_mask=model_inputs.attention_mask,
        max_new_tokens=512,
        pad_token_id=tokenizer.eos_token_id
    )
    generated_ids = [
        output_ids[len(input_ids):] for input_ids, output_ids in zip(model_inpu
    ]
    response = tokenizer.batch_decode(generated_ids, skip_special_tokens=True)
    return response
```

Lastly, the exciting part is transforming the generated response into an audio file with the text-to-speech model. For our example, we will use the Suno Bark model hosted in the HuggingFace. After generating the audio, we will play the audio response to complete the pipeline.

```
def text_to_speech(text, output_file):
    from transformers import AutoProcessor, BarkModel
    print("Generating speech...")

    processor = AutoProcessor.from_pretrained("suno/bark-small")
    model = BarkModel.from_pretrained("suno/bark-small")

    inputs = processor(text, return_tensors="pt")

    audio_array = model.generate(**inputs)
    audio = audio_array.cpu().numpy().squeeze()

    # Save the audio to a file
    write(output_file, 22050, (audio * 32767).astype(np.int16))
    print(f"Audio response saved to {output_file}")
    return audio
```

```
def play_audio(audio, samplerate=22050):
    print("Playing response...")
    sd.play(audio, samplerate=samplerate)
    sd.wait()
```

That's all the features we need to complete the fully voice-activated RAG pipeline. Let's combine all together to make a cohesive structure.

```
def main():
    # Step 1: Load and chunk the PDF
    chunks = load_and_chunk_pdf(PDF_FILE)

    # Step 2: Set up ChromaDB
    collection = setup_chromadb(chunks)

    # Step 3: Detect wake word with embedding similarity
    if not detect_wake_word():
        return  # Exit if wake word is not detected

    # Step 4: Record and transcribe user input
    record_audio(AUDIO_FILE, duration=5)
    user_input = transcribe_audio(AUDIO_FILE)
    print(f"User Input: {user_input}")

    # Step 5: Query ChromaDB for relevant chunks
    relevant_chunks = query_chromadb(collection, user_input)
    print(f"Relevant Chunks: {relevant_chunks}")

    # Step 6: Generate response using a Hugging Face model
    response = generate_response(user_input, relevant_chunks)
    print(f"Generated Response: {response}")

    # Step 7: Convert response to speech, save it, and play it
    audio = text_to_speech(response, RESPONSE_AUDIO_FILE)
    play_audio(audio)

    # Clean up
    os.remove(AUDIO_FILE)  # Delete the temporary audio file

if __name__ == "__main__":
    main()
```

I have saved the whole code in a script called app.py, and we can activate the system using the following code.

```
python app.py
```

Try it yourself, and you will acquire the response audio file that you can use to review.

That's all you need to build the local RAG system with voice activation. You can evolve the project even further by building an application for the system and deploying it into production.

Conclusion

Building an RAG system with voice activation involves a series of advanced techniques and multiple models that work together as one. By utilizing retrieval and generative functions to build the RAG system, this project adds another layer by embedding audio capability in several steps. With the foundation we have built, we can evolve the project even further depending on your needs.

Cornellius Yudha Wijaya is a data science assistant manager and data writer. While working full-time at Allianz Indonesia, he loves to share Python and data tips via social media and writing media. Cornellius writes on a variety of AI and machine learning topics.

More On This Topic

- Llama, Llama, Llama: 3 Simple Steps to Local RAG with Your Content
- Baby AGI: The Birth of a Fully Autonomous AI
- How to Implement a Basic Reranking System in RAG
- GPT4All is the Local ChatGPT for your Documents and it is Free!
- LangChain + Streamlit + Llama: Bringing Conversational AI to Your...
- 7 Steps to Running a Small Language Model on a Local CPU

What do you think?

2 Responses

Upvote Funny Love Surprised Angry Sad

2 Comments 1 Login ▼

G Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

♡ • Share Best Newest Oldest

Vaibhav Joshi

3 hours ago

I have a similar but less complicated system and with all free(but require online connection) system https://huggingface.co/spaces/vsj0702/voice_ai

0 0 Reply Share ›

Prescienced Data

a day ago

Nice content! thanks for sharing information. Prescience Decision Solutions:- https://prescienceds.com/artificial-intelligence-and-data-science-boosting-sales-and-customer-engagement-in-e-commerce/

0 0 Reply Share ›

Subscribe Privacy Do Not Sell My Data DISQUS

<= Previous post Next post =>