Building a Custom Voice-Controlled Assistant on Windows Using Anaconda

Table of Contents

1. Introduction
2. Prerequisites
3. Step 1: Install Required Packages
4. Step 2: Implement Voice Capabilities
5. Step 3: Integrate AI Model
6. Step 4: Implement Device Control
7. Step 5: Combine Everything into a Main Loop

Introduction

Building a custom voice-controlled assistant on Windows can be achieved using Anaconda, which provides an extensive library ecosystem for AI and machine learning applications. This guide will walk you through the process of creating a basic voice-controlled assistant using Anaconda.

Prerequisites

- Anaconda installed on your system
- A microphone for speech-to-text functionality

Step 1: Install Required Packages

To start, install the required packages using conda:

```
conda create -n myenv python=3.9
conda activate myenv
conda install -c anaconda speechrecognition pyttsx3 wikipedia pyautogui
```

This command will create a new environment named `myenv`, activate it, and then install the necessary packages.

Step 2: Implement Voice Capabilities

The following steps implement voice capabilities in your assistant:

Speech-to-Text Functionality

```
import speech_recognition as sr

def get_command():
  r = sr.Recognizer()
  with sr.Microphone() as source:
    print("Listening...")
    r.pause_threshold = 1
    audio = r.listen(source)

  try:
    print("Recognizing...")
    query = r.recognize_google(audio, language='en-us')
    print(f"User said: {query}\n")
    return query.lower()
  except Exception as e:
    print("Sorry, I didn't catch that. Please try again.")
    return "None"
```

Text-to-Speech Functionality

```
import pyttsx3

def speak(text):
  engine = pyttsx3.init()
  engine.say(text)
  engine.runAndWait()
```

Step 3: Integrate AI Model

To integrate an AI model, you can use a local model or a cloud-based API. For this example, we'll use the OpenAI SDK for Anaconda.

```
import openai

openai.api_key = "YOUR_API_KEY"

def chat_with_ai(prompt):
    response = openai.Completion.create(
        engine="davinci",  # or another model
        prompt=prompt,
        max_tokens=50
    )
    return response.choices[0].text
```

Step 4: Implement Device Control

To implement device control, you can use the `subprocess` module to run system commands and `pyautogui` for GUI automation.

```
import subprocess
import pyautogui

def execute_command(command):
    if "open" in command and "notepad" in command:
        speak("Opening Notepad.")
        subprocess.Popen(["notepad.exe"])
    elif "search" in command and "wikipedia" in command:
        # Example using the wikipedia library
        search_query = command.replace("search wikipedia for", "").strip()
        speak(f"Searching Wikipedia for {search_query}.")
        try:
            result = wikipedia.summary(search_query, sentences=2)
            speak(result)
        except wikipedia.exceptions.PageError:
            speak("Sorry, I couldn't find anything on that.")
    elif "press" in command and "enter" in command:
        speak("Pressing the enter key.")
        pyautogui.press('enter')
    else:
        speak("Sorry, I don't know how to do that yet.")
```

Step 5: Combine Everything into a Main Loop

Finally, combine all the steps into a main loop that listens for a wake word and processes the command:

```
import time

def main_assistant_loop():
    while True:
        # Listen for a wake word like "assistant"
        command = get_command()
        if "assistant" in command:
            speak("Yes, how can I help you?")
            action_command = get_command()
            execute_command(action_command)
        # Optional: Add a way to exit the loop
        if "stop" in command:
            speak("Goodbye!")
            break

# Start the assistant
if __name__ == "__main__":
    main_assistant_loop()
```

Conclusion

With this guide, you've successfully created a custom voice-controlled assistant on Windows using Anaconda. This assistant can perform basic tasks such as opening applications, searching for information, and responding to user queries.

Best Practices

- Use the `speechrecognition` package for speech-to-text functionality.
- Utilize the `pyt

Yeah we're gonna keep it true to Ollama llama models so we're not gonna be using Open AI either
edit

In that case, you can use a different library or framework to integrate an AI model into your voice-controlled assistant. Here's an example of how you could use the Hugging Face Transformers library to integrate a language model into your assistant:

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer

# Load pre-trained language model and tokenizer
model_name = "google/bert-base-uncased"
model = AutoModelForCausalLM.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)

def chat_with_ai(prompt):
    # Tokenize the input prompt
    inputs = tokenizer(prompt, return_tensors="pt")

    # Use the model to generate a response
    output = model.generate(**inputs)
    response = tokenizer.decode(output[0], skip_special_tokens=True)

    return response

# Test the function
prompt = "Hello, how are you?"
response = chat_with_ai(prompt)
print(response)
```

This code loads a pre-trained language model and tokenizer using the Hugging Face Transformers library, and then uses the model to generate a response to a given input prompt.

You can replace this with your own custom AI model or integrate it into your existing assistant architecture.

Additionally, you'll need to install the `transformers` library using conda:

```
conda install -c huggingface transformers
```

Please note that this is just an example, and you may need to modify the code to suit your specific use case.