

Retriever AI

Team Spiller



Purpose

Why Did We Build Retriever AI?

A Natural Language Windows Operating System Interface

We made an application that allows users to access certain elements of the Windows Operating System with **natural language**. We made **Retriever AI** to allow those that are blind or visually impaired to

- Read, Write, and Reply to Emails
- Browse the Web
- Manage File Explorer and Settings

So that **everyone** can access the digital world.

What Makes Us Different?

We are the first market solution that implements some of the latest technology in the AI space such as ElevenLabs' Speech Synthesis and OpenAI's Whisper to make the most human experience possible.



Function

How Does Retriever AI Work?

OpenAI's Whisper

We use OpenAI's latest
speech to text model to
gather auditory input from
the user.

```
whisper.py X
retriever:main > whisper.py >
1 import numpy as np
2 import os
3 import openai
4 from scipy.io import wavfile
5 from audiolib import record_audio
6 from pathlib import Path
7
8 # Get env variables
9 from dotenv import load_dotenv
10 load_dotenv()
11
12 def write_audio_to_file(full_audio: np.ndarray, filename: str, fs: int=44100):
13     # scale the audio data to the range of ints, which is the format used for wav files
14     scaled_audio = np.int16(full_audio / np.max(np.abs(full_audio)) * 32767)
15
16     # write the audio data to a wav file
17     wavfile.write(filename, fs, scaled_audio)
18
19 def call_whisper_api(audio_stream: np.ndarray) -> None:
20     # Call api on audio file path provided
21     write_audio_to_file(audio_stream, r'./prompt.wav')
22     openai.api_key = os.getenv("OPENAI_API_KEY")
23     audio_file= open(r'./prompt.wav', "rb")
24     transcript = openai.Audio.transcribe("whisper-1", audio_file)
25     return transcript
26
27 def transcribe_mic_input():
28     audio = record_audio()
29     transcript = call_whisper_api(audio_stream=audio)
30     print(f"Audio transcript: {transcript}")
31     return transcript
32
33 if __name__ == "__main__":
34     audio = record_audio()
35     transcript = call_whisper_api(audio_stream=audio)
36     print(transcript)
```

```

palm.py X
retiever-main > palm.py > ...
1 """
2 All fns to run a prompt through PaLM
3 """
4 from fastapi import FastAPI
5 from fastapi.middleware.cors import CORSMiddleware
6 from google.auth import credentials
7 from google.oauth2 import service_account
8 import google.cloud.aiplatform as aiplatform
9 from vertexai.preview.language_models import ChatModel, InputOutputTextPair
10 import vertexai
11 import json # add this line
12
13 # load the service account json file
14 # update the values in the json file with your own
15 p_service_acct = './service_account.json'
16 with open(
17     p_service_acct
18 ) as f: # replace 'serviceaccount.json' with the path to your file if necessary
19     service_account_info = json.load(f)
20
21 my_credentials = service_account.Credentials.from_service_account_info(
22     service_account_info
23 )
24
25 # Initialize Google AI Platform with project details and credentials
26 aiplatform.init(
27     credentials=my_credentials,
28 )
29
30 with open(p_service_acct, encoding="utf-8") as f:
31     project_json = json.load(f)
32     project_id = project_json["project_id"]
33
34
35 # Initialize Vertex AI with project and location
36 vertexai.init(project=project_id, location="us-central1")
37
38 # Initialize the FastAPI application
39 app = FastAPI()
40
41 # chat with the model
42 async def handle_chat(human_msg: str):
43     """
44     Endpoint to handle chat.
45     Receives a message from the user, processes it, and returns a response from the model.
46     """
47     chat_model = ChatModel.from_pretrained("chat-bison@001")
48     parameters = {
49         "temperature": 0.8,
50         "max_output_tokens": 1024,
51         "top_p": 0.8,
52         "top_k": 40,
53     }
54     chat = chat_model.start_chat()

```

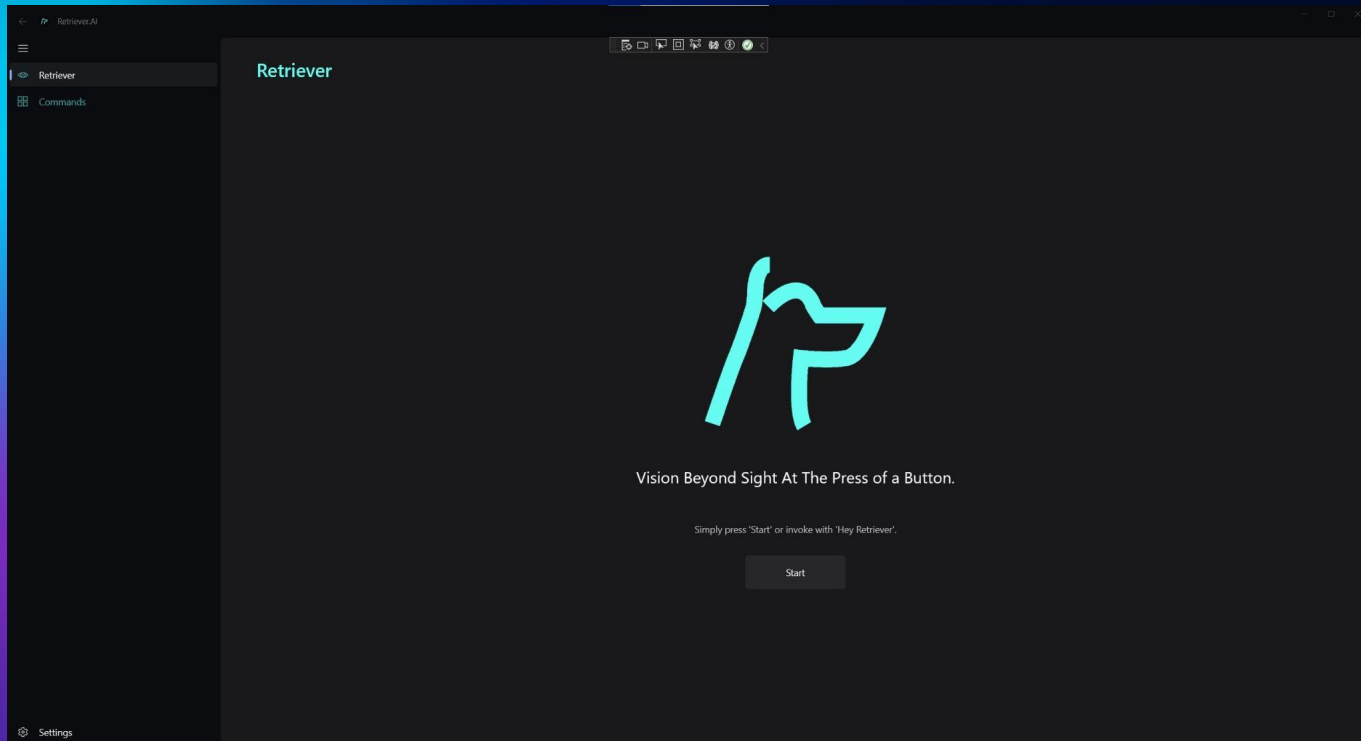
Google's PaLM2

We use Google's PaLM2 LLM to process that data into information our Python scripts and functions can understand.

ElevenLabs' Speech Synthesis

Finally, we use ElevenLabs' latest AI Speech Synthesis models in order to read back output to the end user.

```
retiever:main > ❖ app.py > ...
1 from whisper import transcribe_mic_input
2 from litl import get_user_actions
3 import asyncio
4 import ast
5 from functions.run_app import open_program
6 from elevenlabs import play, generate, set_api_key
7 from browser import run_web_search
8 import os
9
10 from dotenv import load_dotenv
11 load_dotenv()
12
13 set_api_key(os.getenv("ELEVENLABS_API_KEY"))
14
15 def say(message:str) -> None:
16     audio = generate(
17         text=message,
18         voice="Bella",
19         model="eleven_monolingual_v1"
20     )
21
22     play(audio, use_ffmpeg=False)
23
24
25 while True:
26     command = transcribe_mic_input()
27     resp = asyncio.run(get_user_actions(command))
28     resp = resp.replace("'", "")
29     if resp[0] != " ":
30         resp = " "
31     print(resp)
32     resp = ast.literal_eval(resp)
33     print(resp)
34     for tup in resp:
35         print(f"tup[0] {tup[0]}")
36         print(f"tup[1] {tup[1]}")
37         if tup[0] == "run_app":
38             print(f"opening app")
39             open_program(tup[1])
40         elif tup[0] == "SPEAK":
41             say(tup[1])
42         elif tup[0] == "SEARCH_WEB":
43             result = run_web_search(tup[1])
44             say(result)
45         elif tup[0] == "ERROR":
46             say("Sorry, I couldn't get that.")
47             print("ERROR")
48         else:
49             say("Sorry, I can't do that yet.")
50             print("unsupported")
```

Demo

Here's how it works:
Users press **Start** and
continue prompting
Retriever to do certain
tasks until they hit **Stop**.

Thanks!

A lot of effort went into this project, and if you have any feedback, please feel free to reach out at any of the following below:

Jacob Ryabinky || mrjacobry@gmail.com

Dylan Nguyen || dylannguyen373@gmail.com