Daniel Vazquez

Dr. Osama Alshaykh

EC601

9/29/2024

Project 2

“NRT: Not Reading That ”

Product Description:

I would be developing a multiplatform application called N.R.T., for "Not Reading That". This application’s main function is to use machine learning to summarize text for a user. However, if the text is in the form of an email, text message, or direct message of some kind, the model will summarize the message and provide possible responses to it based on its details. The application will be capable of storing these responses, and the logs will be available for easy viewing. Additionally, the application will have the capability to summarize and respond in various languages.

Possible use cases:

1. Summarizing long emails, and generating responses
2. Generating quick replies for text messages
3. Logging conversations
4. Translating text
5. Automated responses when without wifi/data
6. Simplifying academic reading materials

User Stories:

My own personal “User Story” and inspiration for the application:

As a busy student, I have many family members that will message me paragraphs of (often very random) text multiple times a day, and I simply am not reading that.

ChatGPT can be utilized to create user stories and address the possible use cases, using the product description above as a prompt:

Here are five user stories for your N.R.T. (Not Reading That) application:

1. \*\*Text Summarization for Busy Users:\*\*

- \*As a user\*, I want the app to summarize long articles, documents, or messages so that I can quickly understand the main points without reading everything in full.

2. \*\*Summarization and Response for Communication:\*\*

- \*As a user\*, I want the app to summarize emails, text messages, or direct messages and provide suggested responses based on the content, allowing me to respond efficiently without manually drafting a reply.

3. \*\*Multilingual Summarization and Response:\*\*

- \*As a user\*, I want the app to be able to summarize and provide responses in different languages, so I can communicate effectively with people from different linguistic backgrounds.

4. \*\*Storing Original Messages for Later Review:\*\*

- \*As a user\*, I want the app to store the original messages it summarizes so that I can refer back to the full content whenever I need to, ensuring I don’t lose any important details.

5. \*\*Customization of Response Tones:\*\*

- \*As a user\*, I want the app to offer multiple response tones (e.g., formal, casual, friendly, or professional) when suggesting replies so that I can tailor my communication style to different audiences.

These stories focus on core functionalities that will make N.R.T. user-friendly and versatile.

NRT Example Application:

- Used ChatGPT to help code, and used the ChatGPT API –

1. Copy the code into a python notebook
2. Run the notebook
3. Copy text like the message below and paste it into the top text box
4. Save or load previous responses to a text file

Note: There is different functionality for messages than for regular text. Try both!

Example message (or any other):

Hey honey it's me your mom. I hope your day is going well and all. Anyway, I just read today on Facebook that apparently irradiated chia seeds caused frogs in the amazon to grow human facial features. I know it's so scary... I just thought of you because of that one time you made a smoothie with them 14 years ago. So maybe stay away from eating those unless you’re looking for a new nose LOL. On second thought maybe I'll start buying some chia seeds.

Python Code (I’ll try to include the .ipynb in blackboard):

Preferably change the API KEY if you have one!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

import tkinter as tk

from openai import OpenAI

import os

# Set your OpenAI API key here

OPENAI\_API\_KEY = 'sk-proj-gd6Emxwlh0RgJU\_jbR4tkJVqXE9v1obIz9JVCumYcLQYDdQ4g3nGXDrP\_b44f4tZgakeOClR-kT3BlbkFJu37-MO2Nf\_E\_ZxQxN8tbU09tTu2pDt2VO7Jb6MgYhts32SW5MFtk8yTc3z1giDo5TjdV57p-YA'

client = OpenAI(api\_key=OPENAI\_API\_KEY)

# Function to format messages for the API

def format\_message(role, content):

return {"role": role, "content": content}

# Function to send the message and get the response using the OpenAI API

def get\_response(messages):

completion = client.chat.completions.create(

model='gpt-4-1106-preview', # Latest model

messages=messages,

)

content = completion.choices[0].message.content

return content

# Function to summarize text using the OpenAI GPT API (chat model)

def summarize\_text(text):

try:

# Prepare the instruction with the text for summarization

instructions = """

First, identify if the following text is a message to the user. If the text is a message to the user, summarize it in

the fewest possible sentences such that the summary only includes the main point of the message, and

is at least half the length of the original message. Then list three possible responses to it based on the

details of the message. The possible responses should be formatted such that it just says

"Possible Responses" and beneath it the repsonses ordered by number and spaced out for easy viewing.

If the text is not in the form of a message to the user , simply summarize the following text in the least number of

possible sentences to get the point across, and do not list possible responses. If the text is in a different language, make sure it is translated

into English.

Text to summarize:

""" + text

message = format\_message("system", instructions)

messages = [message]

# Get the summary from the OpenAI API

summary = get\_response(messages)

return summary

except Exception as e:

return f"Error: {str(e)}"

# Function to dynamically create response buttons if "Possible Responses" are detected

def create\_response\_buttons(response\_text):

if "Possible Responses" in response\_text:

# Extract possible responses (assuming they are in the form: "1. Response one", "2. Response two", etc.)

responses = [line.strip() for line in response\_text.split("\n") if line.strip().startswith(("1.", "2.", "3."))]

# Create the buttons if there are exactly 3 possible responses

if len(responses) == 3:

# Clear any existing buttons before adding new ones

for widget in response\_button\_frame.winfo\_children():

widget.destroy()

for i, response in enumerate(responses):

button = tk.Button(response\_button\_frame, text=f"Send Response {i+1}", command=lambda r=response: print(f"Sending: {r}"))

button.pack(pady=5) # Add padding for vertical spacing

# Create the main window

root = tk.Tk()

root.title("NRT: Not Reading That")

root.geometry("400x600")

# Create a Text widget for the input text box with reduced height

text\_box = tk.Text(root, wrap='word', height=10)

text\_box.pack(expand=False, fill='both', padx=10, pady=10)

# Function for the "I am not reading that" button

def button\_action():

# Get the content from the text\_box

input\_text = text\_box.get(1.0, tk.END).strip()

# Use the summarize\_text function to get the summary from OpenAI API

if input\_text:

summary = summarize\_text(input\_text)

# Update the output\_box with the summary

output\_box.config(state=tk.NORMAL) # Enable writing to the output box

output\_box.delete(1.0, tk.END) # Clear the current contents

output\_box.insert(tk.END, summary) # Insert the summary

output\_box.config(state=tk.DISABLED) # Disable the output box to make it read-only

# Check if the response contains "Possible Responses"

create\_response\_buttons(summary)

button = tk.Button(root, text="I am not reading that", command=button\_action)

button.pack(pady=10)

# Create a second text box for display, make it read-only

output\_box = tk.Text(root, wrap='word', height=10)

output\_box.pack(expand=False, fill='both', padx=10, pady=10)

output\_box.config(state=tk.DISABLED) # Set the text box to read-only initially

# Define the function for the Save button

def save\_content():

# Get the content of the output\_box

gpt\_output = output\_box.get(1.0, tk.END).strip()

if gpt\_output: # Only save if there is content

file\_path = "NRT\_Logs.txt" # File name to save the logs

# Open file in append mode, if it doesn't exist, it will create a new one

with open(file\_path, "a") as file:

file.write(f"GPT Output:\n{gpt\_output}\n{'='\*40}\n")

print(f"Content saved to {file\_path}")

# Define the function for the Logs button to open the log file

def open\_logs():

file\_path = "NRT\_Logs.txt"

# Check if the file exists before trying to open it

if os.path.exists(file\_path):

os.startfile(file\_path) # This will open the file with the default application (Notepad on Windows)

print(f"Opening {file\_path}")

else:

print(f"{file\_path} does not exist.")

# Create a frame for Save and Logs buttons

button\_frame = tk.Frame(root)

button\_frame.pack(pady=10)

save\_button = tk.Button(button\_frame, text="Save", command=save\_content)

save\_button.pack(side='left', padx=10)

logs\_button = tk.Button(button\_frame, text="Logs", command=open\_logs)

logs\_button.pack(side='left', padx=10)

# Create a separate frame for the response buttons to appear below Save and Logs

response\_button\_frame = tk.Frame(root)

response\_button\_frame.pack(pady=10)

# Run the Tkinter event loop

root.mainloop()