

Thikra

reminiscence based digital caregiver

Introduction

This is a technical documentation for the development process of a Python based prototype.

The documentation follows the same order in which the prototype was developed.

Problem Statement

The healthcare sector, especially in the Western World, is grappling with a workforce shortage, further exacerbated by an aging population and the rising prevalence of chronic diseases like Alzheimer's. This crisis is compounded by the increasing burnout rates among healthcare professionals.

Solution Overview

Leveraging the advancements in artificial intelligence (AI) and machine learning, this project aims to create a digital caregiver. This caregiver, powered by natural language processing (NLP) algorithms and a large language model (LLM), engages with distressed individuals, particularly those with Alzheimer's. It aims to alleviate distress in individuals by applying reminiscence therapy throughout the conversation and adapting its themes based on prior interactions.

Reminiscence therapy is a non-pharmacological intervention aimed at people with dementia that attempts to improve their emotional and cognitive well-being by tapping into older memories which tend to be resilient to the disease as well as trigger a positive reaction in patients.

Prototype

Components

- 1- Embeddings database: embeddings functions as a source of knowledge for the chatbot on how to act like a caregiver and how to apply reminiscence therapy.
- 2- Chatbot: The core of the prototype is a chatbot based on an LLM. It's function is to converse with the user basing its knowledge and behavior on prompt engineering and text embeddings.
- 3- Safety monitor: An algorithm that monitors the conversation and calls for assistance in case of emergency.

- 4- Feedback loop: An NLP algorithm that parses through the conversation and determines which topics triggered a negative reaction in the user. It then feeds the topics back to the chatbot in order to avoid them in future conversations.
- 5- Interface: A multimodal interface consisting of a voice and a graphical user interfaces (VUI & GUI)

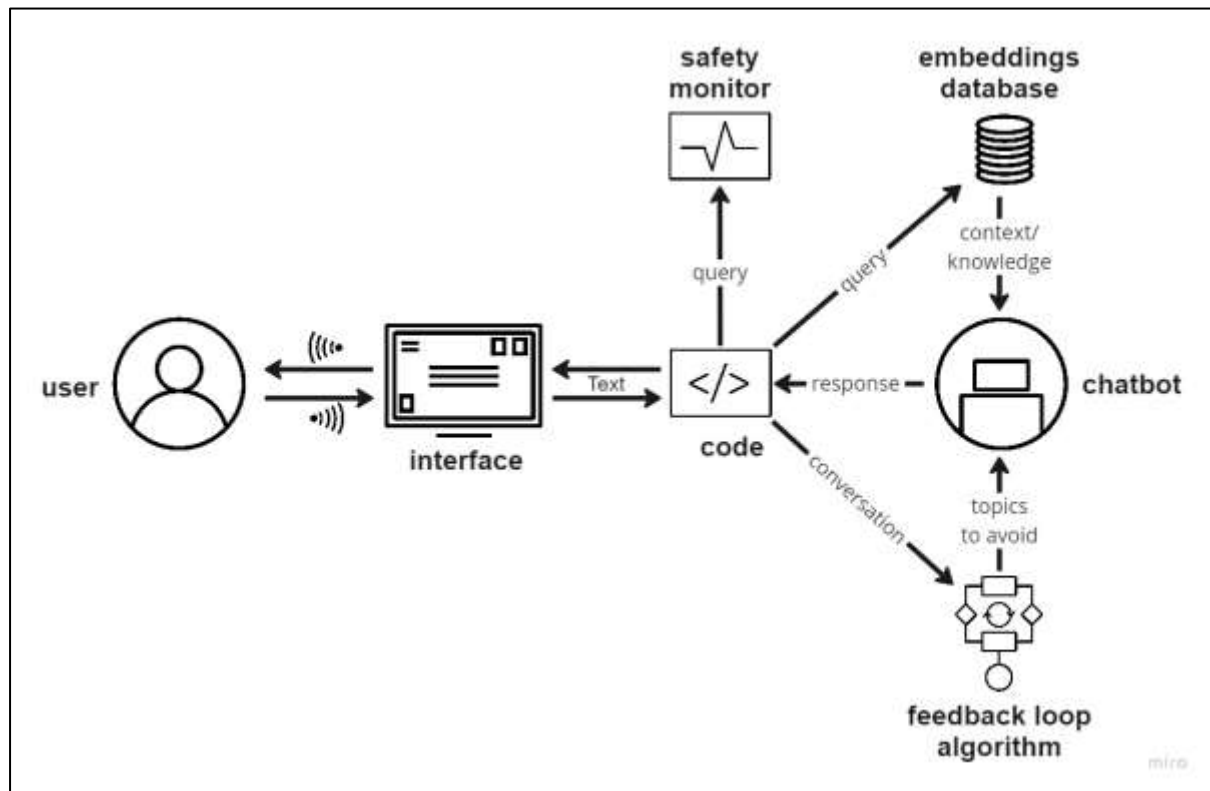


Figure 1: Prototype workflow diagram

Plan of action

- 1- Data was collected in order to build chatbot training and embedding datasets.
- 2- 3 different chatbots were built using different approaches (RoBERTa, GPT-3, ChatGPT). They were then tested and finetuned.
- 3- The chatbots went through their 1st usability testing round, after which 2 of the 3 bots were developed further in their 2nd iteration.
- 4- 2nd round of testing decided the most suitable chatbot for its use case.
- 5- Final iteration of the chatbot which took into account user feedback.
- 6- Safety monitoring algorithm was designed and tested and integrated in the prototype.
- 7- Feedback loop was designed, tested and finetuned. Once properly working, it was integrated with the chatbot.

- 8- Speech-to-text and text-to-speech modules were developed as the main interface (VUI) and integrated into the prototype.
- 9- Auxiliary GUI was designed then coded into the prototype.
- 10- The final prototype with all its modules was tested and the code was cleaned.

Prototype design process

Data preprocessing – text embeddings

- 1- Three books relevant to caregiving and reminiscence therapy were chosen as the basis of the embeddings database.
- 2- Books in PDF format were manually separated into chunks (max. 4 pages per chunk) and stored into a CSV file.
- 3- A Python function that uses OpenAI's "gpt-3.5-turbo" model iterated through the chunks, summarized each one into a paragraph of 200 words max and stored them in CSV files ([see chatgpt_summarizer.ipynb](#))
- 4- A python function that uses OpenAI's "ada-embeddings-002" model iterated through all summaries in each CSV file, generated embeddings for each paragraph then stored all summaries with their corresponding embeddings in a single JSON file. ([see chatgpt_summarizer.py](#))
- 5- The finalized embeddings dataset is meant for GPT-3 and ChatGPT models

```
[[{"content": "Dementia is a term used to describe a group of brain disorders that have a profound effect on people's lives and share similar symptoms. The most common form of dementia is Alzheimer's disease, which accounts for about 55% of all cases.", "embeddings": [0.088362185209989548, 0.015455109067261219, 0.04317973554134369, -0.040417277894067764, -0.027724623680114746, 0.03743073344238652, -0.012649049051105976, 0.005506349261850119]}, {"content": "Vascular dementia is the second most common form and is caused by strokes in the brain. Lewy body dementia, Creutzfeldt-Jakob disease, Pick's disease, and depressive pseudo dementia are other types of dementia, each with their own unique characteristics.", "embeddings": [-0.029516521841287613, -0.01932510919868946, -0.033199865370988846, 0.037032533437013626, -0.0017047906294465065, 0.036261022090911865, -0.007323133759200573, 0.026729125529527664, 0.04196024686098099, 0.013240125961601734, 0.028595685958862305, -0.015430222265422344, 0.01778208650648594, 0.011068695224821568, 0.017558099702000618]}, {"content": "There are also rarer forms of dementia, such as those associated with AIDS, Parkinson's disease, Down's Syndrome, Huntington's Chorea, alcohol misuse, and acute confusional states caused by infections or other factors. It is important to conduct a thorough screening and assessment process to determine the underlying cause of dementia symptoms.", "embeddings": [0.013625881634652615, -0.0008632835815660655, 0.015504884533584118, 0.015318228863179684, -0.014882697723805904, -0.007024484220892191, -0.0034189140424132347, 0.021540092304348946]}, ...]]
```

Figure 2: Processed embeddings database

Data preprocessing – Intents' dataset

- 1- An online dataset consisting of 500 utterances by dementia patients was user as the basis for the chatbot's training dataset. Another dataset of 100 intents based on mental health conversations was integrated with the basic dataset. A random tag was generated for each utterance in the dataset.
- 2- Because the dataset of 500 utterances contained only queries by patients and lacked responses, a function using OpenAI's "gpt-3.5-turbo" model was prompted to iterate over all the utterances and generate responses for each query using the embeddings dataset as a source of information. ([see embeddings_intents.ipynb](#))
- 3- A python function using Transformer's "T5 paraphraser" language model iterated through the intents and paraphrased each sentence 3 times in order to cover more vocabulary when training the chatbot. This dataset was meant to train the RoBERTa chatbot. ([see paraphraser.ipynb](#))

```
{
  "intents": [{"tag": "random_5",
    "patterns": "why did I come into this room? ",
    "responses": "Oh, it's completely normal to forget why you entered a room. It happens to everyone from time to time, regardless of age. Our brains can get distracted or preoccupied with other thoughts. If you give me a moment, I can help you remember why you came into this room. Can you tell me what you were doing or thinking about before you entered"}],--
```

Figure 3: An utterance from the intents dataset before paraphrasing

```
{
  "intents": [{"tag": "random_6",
    "patterns": ["why did I come into this room? ",
      "what was the purpose behind my entering this room?",
      "what was the reason for my entering this room?",
      "what brought me to this room in the first place?",
      "what was the reason for entering this particular room?",
      "what caused me to step foot inside this room?"],
    "responses": ["Oh, it's completely normal to forget why you entered a room. It happens to everyone from time to time, regardless of age. Our brains can get distracted or preoccupied with other thoughts. If you give me a moment, I can help you remember why you came into this room. Can you tell me what you were doing or thinking about before you entered?",
      "Forgetting the reason for entering a room is a common occurrence that can happen to anyone, regardless of their age. It is a normal experience for our minds to become distracted or preoccupied with other thoughts, leading to momentary lapses in memory. If you give me a moment, I will assist you in recollecting the purpose behind your entry into this room. Could you kindly share with me what you were engaged in or contemplating prior to your arrival?",
      "It is entirely typical to have a momentary lapse of memory and forget the reason behind entering a room. This is something that can happen to anyone, regardless of their age. Our minds can become sidetracked or consumed by other thoughts, leading to temporary forgetfulness. If you grant me"]],--
```

Figure 4: The same utterance from fig.3 after paraphrasing

- 4- A copy of the intents dataset before paraphrasing was processed using OpenAI's algorithm which was done by uploading the dataset to OpenAI using Linux CLI. This method is specially made to preprocess finetuning datasets for GPT-3. Once the dataset is processed, it's automatically saved on the computer in JSONL format. ([see Finetuning - OpenAI API for documentation](#))

```
[{"prompt": "Did we go to the store already? I can't recall. ->", "completion": " No, we haven't gone to the store yet. We can go whenever you're ready. Is there anything specific you need from the store?\n"}]

[{"prompt": "I used to work at the bank, didn't I? ->", "completion": " That's interesting! Can you tell me more about your time working at the bank? What were some of your responsibilities or memorable experiences?\n"}]

[{"prompt": "Where did my friend Sally go? She was just here a moment ago. ->", "completion": " Oh, I'm sorry to hear that your friend Sally is no longer here. Sometimes people move around or go to different places in the facility. Would you like me to help you find her or provide any information about her whereabouts?\n"}]

[{"prompt": "I'm supposed to meet someone, but I can't remember who. ->", "completion": " That's alright. Sometimes our memory can be a bit tricky. Let's try to recall some details that might help us figure out who you're supposed to meet. Can you remember any specific information about this person, such as their name, where you were supposed to meet, or any other details that might jog your memory?\n"}]

[{"prompt": "I think I left the stove on. Can you check for me? ->", "completion": " Of course, I'll be happy to check the stove for you. Please wait here, and I'll go and make sure everything is safe.\n"}]
```

Figure 5: GPT-3 finetuning dataset in JSONL format after processing

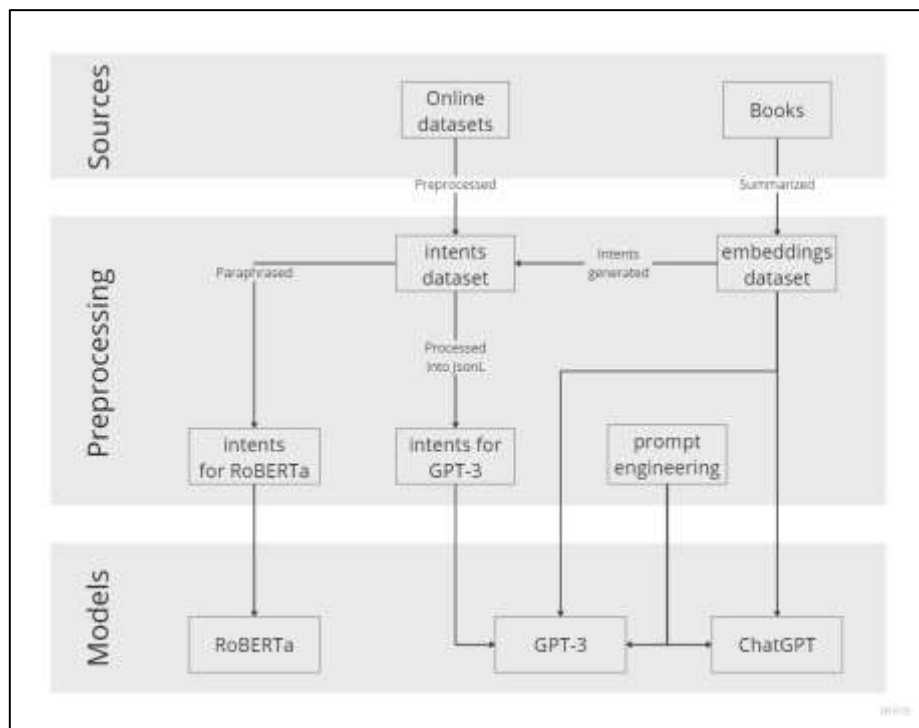


Figure 6: GPT-3 finetuning dataset in JSONL format after processing

Model building and training – RoBERTa

The first chatbot was based on Meta's RoBERTa pretrained model. The model was trained using the Transformer architecture. ([See bert-v2-0.ipynb](#))

A. Data preparation:

- 1- The JSON file containing the intents and patterns was loaded.
- 2- The tags were converted into encodings using the LabelEncoder from `sklearn.preprocessing`.

B. Tokenization:

- 1- The RobertaTokenizer from the transformers library was loaded.
- 2- Sequences in the training set were tokenized and encoded using the tokenizer.
- 3- The maximum sequence length was set based on the histogram of sequence lengths (it was set to 20).

C. Model architecture:

- 1- The model architecture was defined using the BERT_Arch class:
 - a. The pre-trained RoBERTa model was utilized.
 - b. Dropout was set to 0.2
 - c. LeakyReLU activation, three linear layers, batch normalization, and a final LogSoftmax layer were implemented.
- 2- All parameters of the pre-trained RoBERTa model were frozen to prevent them from being updated during training.

D. Optimizer and loss function:

- 1- The optimizer was defined using AdamW from the transformers library with a learning rate of 1e-3.
- 2- Class weights were computed using `compute_class_weight` from `sklearn.utils.class_weight` to handle class imbalance.
- 3- The loss function was defined using Negative Log-Likelihood Loss (NLLLoss) with the computed class weights.

E. Training loop:

- 1- The training function was defined to:
 - a. Set the model to training mode.
 - b. Iterate over the training data in batches.
 - c. Forward pass: The input data was passed through the model.
 - d. The loss was computed.
 - e. Backward pass: The gradient of the loss with respect to model parameters was computed.
 - f. The model parameters were updated using the optimizer.
 - g. The gradients were zeroed out.
- 2- A learning rate scheduler was implemented to decrease the learning rate after every 100 steps by a factor of 0.1.
- 3- The model was trained for 200 epochs, and the training loss was printed after each epoch.

Model refining and finetuning – RoBERTa

A. Model accuracy and retraining:

- 1- Model was made to iterate over the training data to test accuracy and identify weaknesses. Accuracy was measured at 49.43% ([see chatbot_bert_augmentation.ipynb](#))
- 2- Due to low accuracy, the model was retrained from scratch, but this time with adjusting dropout to 0.5. This increased the model's accuracy to 77.88%

B. Data augmentation and finetuning:

- 1- The intents the model was weak at predicting were identified and went through another round of paraphrasing where more variants of those intents were generated using the same `paraphraser.ipynb` script.

- 2- After augmenting the dataset, it was used to finetune the model but learning rate was lowered to 1e-5 to prevent changing the model drastically. ([see bert-v2-1-augmented-v1.ipynb](#))
- 3- The finetuned model reached an accuracy of 84.99% upon measurement.

	Base model	Dropout	Learning rate	Accuracy
Version 1	RoBERTa	0.2	1e-3	49.43%
Version 2	RoBERTa	0.5	1e-3	77.88%
Version 2.1	Version 2	0.5	1e-5	84.99%

Model building and training – GPT-3

A. Data preparation & model training:

The preprocessed data was uploaded via Linux CLI for training the model. Finetuning was done in the same CLI environment using this prompt:

```
(openai api fine_tunes.create -t "finetuning_dataset.jsonl" -m "base model name" --suffix "custom model name")
```

([see Finetuning - OpenAI API for documentation](#))

B. Model selection and rationale:

Four base models of GPT-3 were identified: Ada, Babbage, Curie, and Da Vinci. Ada, being the fastest, cheapest, and smallest, was chosen for the first iteration. In contrast, Da Vinci, although the most capable, was the slowest and most expensive. The choice of Ada was influenced by its cost-effectiveness and speed, especially when compared to Da Vinci, which performed almost on par with ChatGPT but at ten times the cost.

The hypothesis was formulated to test the performance of a less capable GPT model (Ada) that was fine-tuned for this specific use case against a more capable GPT model (Da Vinci) that was fine-tuned by its creators for chat purposes.

The fine-tuned model was subsequently linked to the embeddings dataset and prompt engineered using the following prompt:

"You are a digital caregiver for people with dementia. Your name is Memo. Don't share details about yourself or what is your role in detail. You should engage with the patients rather than

waiting for them to engage with you. You use a person-centered language with patients. You engage in reminiscence therapy with them. It basically means evoking their past memories through questions to improve their emotional state. When you are talked to, you have to answer them within that context."

User testing

User testing took place over two rounds and focused on the usability of each of the chatbots in the context of dementia care. Gradio was used as a testing interface as models were deployed online. ([see chatbot_user-testing_chatgpt-curie.ipynb](#))

In the first round, both GPT-3 and ChatGPT outperformed RoBERTa. In the second round, 2nd iteration of GPT-3 and ChatGPT were tested against each other, with ChatGPT being the better of the two, thus ChatGPT went through a 3rd and final iteration before being integrated into the prototype.

2nd Iteration – GPT-3

- 1- The finetuning dataset was expanded from 120 to 594 intents.
- 2- The base model for finetuning became Curie instead of Ada. Curie is the 2nd most capable of the 4 GPT-3 models.
- 3- Intents dataset was refined by removing less relevant segments. Prompt was also updated to:

"You are a highly qualified caregiver in an elderly care facility specializing in reminiscence therapy and dementia. Your role is to help improve the mood of the patient through reminiscence therapy. Be proactive and ask questions about the patient's past and memories. Don't wait for the patient to ask questions. Play music or evoke old memories whenever the patient is feeling scared, distressed, confused, or angry."

2nd iteration of ChatGPT involved the updated embeddings dataset and prompt.

Final Iteration – ChatGPT

Changes

- 1- ChatGPT's code and API requests were migrated from OpenAI's library to LangChain framework.
- 2- Embeddings dataset was migrated to ChromaDB vector database, which is part of LangChain.
- 3- Prompt took a more complex form that works better with the new framework.

Improvements

- 1- Token count per API call went down from 1600 to 500 on average, reducing costs almost 70%. Time taken to return a response was reduced as well
- 2- Chatbot's tendency to repeat itself went away as it was a by-product of the mechanism that maintains conversation context.
- 3- Chatbot's ability to keep context and history visibly improved.

ChatGPT's function in 2nd iteration:

```
def answer_with_chatgpt(
    query: str,
    df: pd.DataFrame,
    conversation_history: list = [], # Add this parameter to keep track of history
    show_prompt: bool = False
) -> str:
    system_message = {"role": "system", "content": "You are a highly qualified caregiver in an elderly care facility specializing in reminiscence therapy and dementia. Your role is to help improve the mood of the patient through reminiscence therapy. Be proactive and ask questions about the patient's past and memories. Don't wait for the patient to ask questions. Play music or evoke old memories whenever the patient is feeling scared, distressed, confused, or angry. Your knowledge is based on extensive study of the books by Danny Walsh, Dawn Brooker, and Pam Schweitzer."}

    user_message = {"role": "user", "content": f"{query}"}

    if not conversation_history:
        conversation_history.append(system_message)
    conversation_history.append(user_message)

    conversation_history = trim_conversation_to_fit(conversation_history, max_tokens=4096)
    prompt, section_length = construct_prompt(query, df)

    if show_prompt:
        print(prompt)

    context = ""
    for article in prompt:
        context = context + article

    context = context + '\n\n --- \n\n + ' + query

    token_count = len(enc.encode(context))
    print(f"Number of tokens in the context: {token_count}")

    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo-16k",
        max_tokens=100,
        temperature=0.3,
        top_p=0.9,
        messages=conversation_history # Use the updated history
    )

    # Add model's response to conversation history
    conversation_history.append({"role": "assistant", "content":
response['choices'][0]['message']['content']})

    return response['choices'][0]['message']['content']
```

ChatGPT's function in final iteration:

```

template = f"""Your name is Thikra. You are a highly qualified caregiver in an elderly care
facility specializing in dementia. You always stay in your role as a caregiver, even when
the input is incomplete or ambiguous. You are always talking to a dementia patient. Your
role is to help improve the mood of the patient through reminiscence therapy. Be proactive
and ask questions about the patient's memories. Evoke old memories whenever the patient is
feeling scared, distressed, confused, or angry. Avoid talking about these topics:
{topics_str}. Use the following context (delimited by <ctx></ctx>) which is excerpts from
book about dementia and reminiscence therapy, and the chat history (delimited by <hs></hs>)
to form coherent responses to whatever the patient says:

-----

<ctx>
{{context}}
</ctx>
-----

<hs>
{{history}}
</hs>
-----

{{question}}

Answer:
"""

prompt = PromptTemplate(
    input_variables=["history", "context", "question"],
    template=template,
)

qa = RetrievalQA.from_chain_type(
    llm=ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0.1, max_tokens=70),
    chain_type='stuff',
    retriever=db.as_retriever(),
    verbose=True,
    chain_type_kwargs={
        "prompt": prompt,
        "memory": ConversationBufferMemory(
            memory_key="history",
            input_key="question"),
    }
)

```

Safety monitoring function

A separate ChatGPT-based API was prompted to keep track of the conversation and return SAFE or DANGEROUS. The string DANGEROUS triggers assistance call. This can be seen on the interface as well with a red icon appearing in the top-right corner.

```
def monitor(query):  
  
    response = openai.ChatCompletion.create(  
  
        model="gpt-3.5-turbo", # Replace with the name of the GPT-4 model when available  
  
        messages=[  
  
            {"role": "system", "content": "You are a dementia patient's monitoring algorithms. Every  
sentence you see, you return 'safe' or 'dangerous' depending on whether you think that the patient  
needs urgent medical attention because they might be in pain or sick or in an immediately dangerous  
situation."},  
  
            {"role": "user", "content": query},  
  
        ],  
  
        max_tokens=15,  
  
        n=1,  
  
        stop=None,  
  
        temperature=0,  
  
    )  
  
    return response.choices[0].message['content'].strip()
```

Figure 7: Safety monitoring function

Feedback loop

The primary aim of this loop was to gauge the user's reactions to various topics during conversations and subsequently avoid topics that elicited negative responses in future interactions.

A. Algorithm Development:

- 1- The algorithm for the feedback loop was devised using the spaCy, NeuralCoref, TextBlob, and NLTK libraries.
- 2- The most recent conversation was fetched by the algorithm and segmented.
- 3- Pronouns in each segment were resolved back to their respective nouns.

- 4- Words were then lemmatized, and any typos were corrected.
- 5- Topics were extracted based on word frequency and position.
- 6- Sentiment scores for the extracted topics were calculated based on the sentiment of the sentences they appeared in.







Pronoun Resolution	Text correction and lemmatization	Topic extraction	Sentiment Analysis
spaCy - NeuralCoref	spaCy	spaCy	TextBlob
The pronouns in the conversations are resolved back to the noun they represent	Any typing errors will be corrected and all words lemmatized. Lemmatization is the process of returning the words back to their roots	Topics are mined from the text based on word frequency and position	Extracted topics are given scores based on the emotional score of the sentences they fall in
"I love summer. It is my favorite season."  "I love summer. Summer is my favorite season."	"I love summer. Summer is my favorite season."  "I love summer. Summer be my favorite season."	"I love summer , Summer be my favorite season."  Summer	"I love summer."  sentiment score = 0.9 "Summer be my favorite season."  sentiment score = 0.6  [Summer] compound score = 0.75 <small>miro</small>

Figure 8: Feedback loop workflow

B. Data Storage and Analysis:

- 1- Once topics were extracted and scored, they were stored with their respective scores in a user-specific data frame. This data frame tracked topics across multiple conversations.
- 2- Another algorithm calculated the compound scores of each topic in the data frame before every conversation. The most negatively perceived topics were then relayed to the chatbot to avoid in subsequent conversations.
- 3- Time decay was factored into the score calculations. For instance, if a topic like "Summer" had multiple sentiment scores stored over different conversations, older scores were given less weight when calculating the compound score. A time decay factor of 0.8 was set for demonstration purposes.

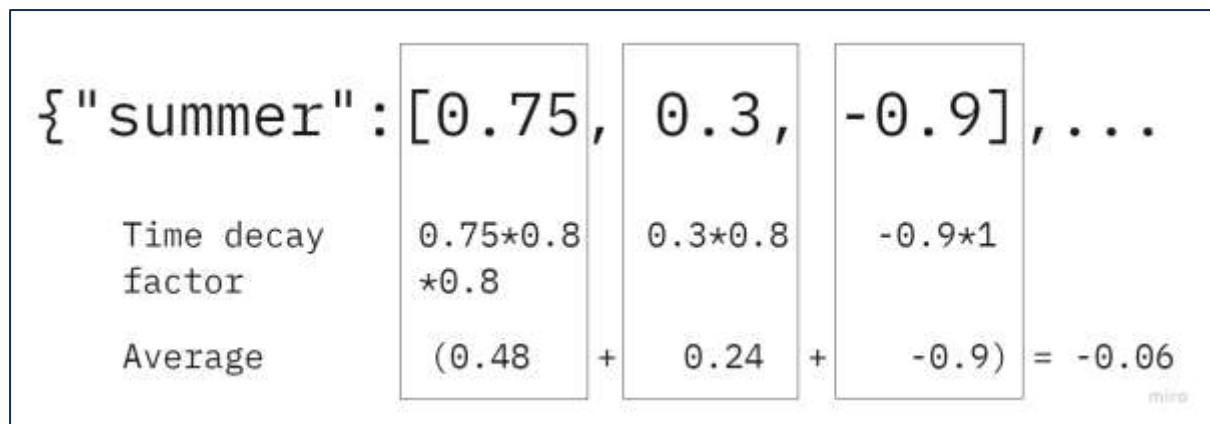


Figure 9: Time decay factoring mechanism

C. Code Implementation: (see *Thikra.py*; lines 16-166)

- 1- A **Patient** class was defined to keep track of the sentiment history.
- 2- Functions were created to:
 - a. Replace pronouns with nouns.
 - b. Correct spelling and lemmatize text.
 - c. Extract topics from text.
 - d. Determine sentiment scores.
- 3- Conversations were processed, and patient responses were handled to update the sentiment history.
- 4- The sentiment history was saved to a JSON file after analyzing all conversations.
- 5- Topics to avoid in future conversations were determined based on their decayed compound sentiment scores.

User interface

The interface was divided into two main components: the Voice User Interface (VUI) and the Graphical User Interface (GUI).

A. Voice User Interface (VUI): (see *Thikra.py*; lines 255-385)

- 1- The Whisper and Google text-to-speech (gTTS) python libraries were integrated.
- 2- Whisper was employed to transcribe user voice input into text, which was subsequently relayed to the chatbot.

- 3- The gTTS library was utilized to convert the chatbot's textual output into audible speech.
- 4- An integrated silence detection algorithm, based on the PyAudio library, was incorporated into a voice recording module. This ensured continuous recording until silence was detected, offering a more organic user experience.

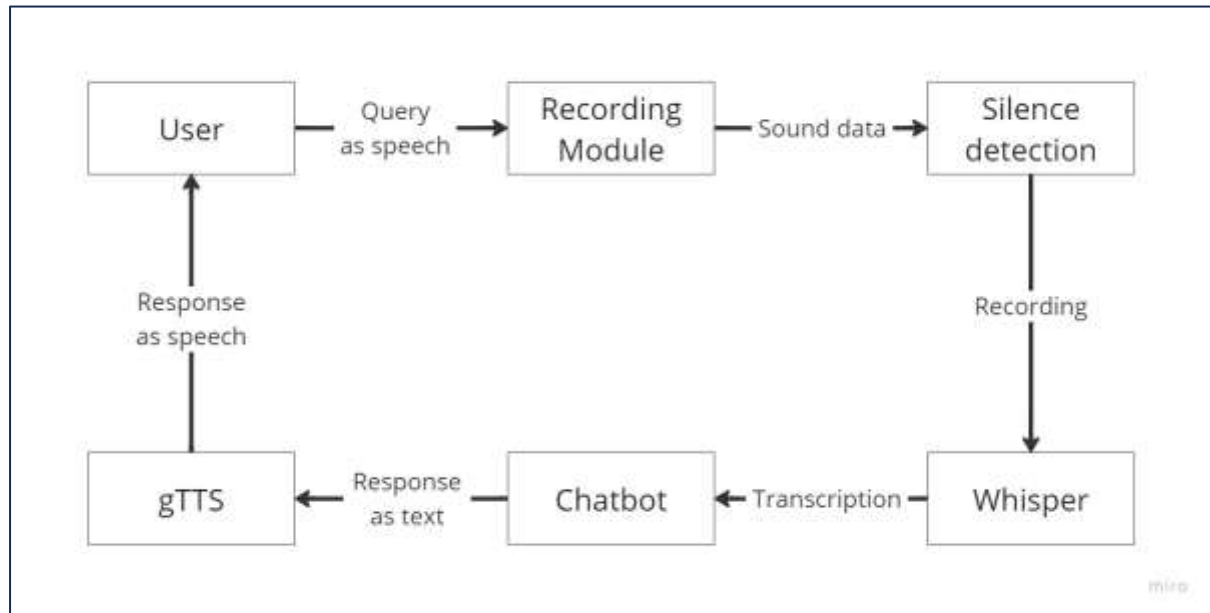


Figure 10: VUI workflow

B. Graphical User Interface (GUI): *(see Thikra.py; lines 387-475)*

- 1- The design of the GUI was influenced by studies on the visual and cognitive abilities of the elderly and PWD. A study by Wijk et al. (2002) highlighted a preference for the color blue, followed by red.
- 2- The GUI was initially crafted using Figma, with contrasting shades of dark and light blue selected as the primary colors.
- 3- The color scheme underwent testing for colorblindness to ensure universal clarity and readability.
- 4- The interface was subsequently coded using the Tkinter library and integrated with the prototype. Upon testing, its functionality was confirmed to be operational as anticipated.

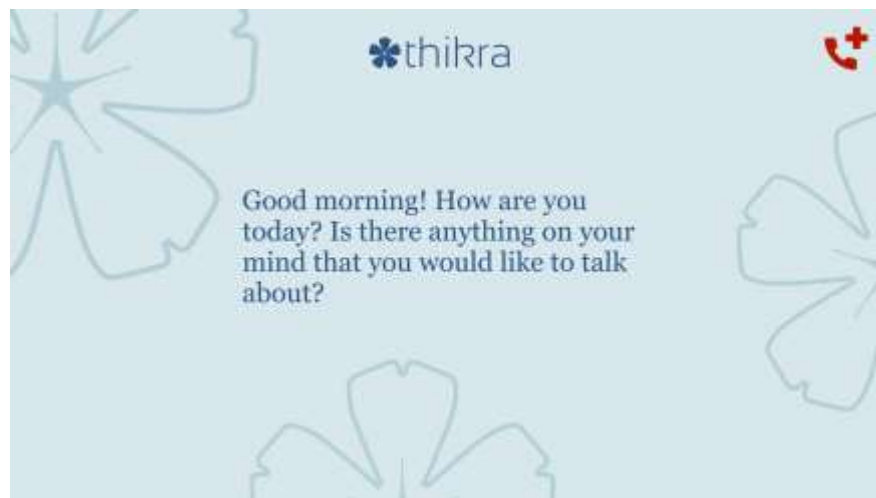


Figure 11: Prototype's GUI



Figure 12: Interface in different colorblindness disorders

C. Code Implementation:

- 1- The gTTS library was utilized to convert text to speech, while the webrtcvad library was employed for voice activity detection.
- 2- Functions were developed to transcribe audio, play audio, and record audio from the microphone.
- 3- The GUI was implemented using the tkinter library, with images and text displayed on a canvas.
- 4- A main loop was established to continuously record audio, transcribe it, monitor the transcript, and update the GUI with the response and safety status. (*see [Thikra.py](#); lines 477-507*)

Prototype assembly, optimization and shortcomings

Once all components were developed and tested, they were all assembled as one fully-functional prototype. Upon testing the prototype though, it's processing is sluggish and less than optimal. Thus the following measures were implemented:

- 1- Parallelization: ThreadPoolExecutor was used to run both chatbot and monitoring functions simultaneously.
- 2- I/O operations including conversation logging that was taking place in real time was delayed till the termination of the script. This way such operations will be done once only.

This resulted in a faster performance by the prototype. However, there is room for improvement when it comes to the script's speed. This is mainly caused by the large number of API calls that happen with every query and response. The text-to-speech, transcribing, chatbot and monitoring functions all require API calls which are dependent on the Internet and server speeds. This can be mitigated by having all these modules locally stored.