



TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY

FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY
Session 202309

BMCS2123 NATURAL LANGUAGE PROCESSING



ONG WENG KAI
22WMR03309

1. Speech to Text
2. Accept different file types
3. Handling unrelated questions
4. Handle File-Related Questions
5. User Interface

RYAN KHO YUEN THIAN
22WMR04097

1. Handling Spelling mistakes
2. Document Similarity
3. Answering General Question
4. Handle File-Related Questions
5. User Interface

THONG CHENG HOW
22WMR03154

1. Text to Speech
2. Translation into different language
3. Summarisation of file related content
4. Handle File-Related Questions
5. User Interface

Title	:	Interactive Chat Application for Conversations with PDF or Document Content Using a Language Model (LLM)
Tutorial Class	:	RDS2S2G2
Tutor Name	:	Ms. FATIN IZZATI BINTI RAMLI
Submission Date	:	31/12/2023

Table of Contents

Introduction.....	3
Research Background.....	5
a) What is a File Chatbot?.....	5
b) Existing File Chatbots in the Market.....	5
c) General Methodology of a Conversational Agent that is Text-Based.....	6
d) LangChain.....	6
e) Language Model (LLM).....	7
f) Projects related to File Chatbot.....	8
g) Relevant Concepts & Techniques.....	17
• Technique/Concept 1: Text-Summarisation.....	17
• Technique/Concept 2: CountVectorizer.....	18
• Technique/Concept 3: TF-IDF Vectorizer.....	19
• Technique/Concept 4: Cosine Similarity.....	19
• Technique/Concept 5: Jaccard Similarity.....	20
• Technique/Concept 6: Speech-To-Text.....	21
• Technique/Concept 7: Text-To-Speech.....	22
• Technique/Concept 8: Translation.....	23
Methodology.....	25
Everyone.....	27
1. Answering File Related Questions.....	27
RYAN KHO YUEN THIAN.....	31
1. Handling Spelling Mistakes or Typos (RYAN KHO YUEN THIAN).....	31
2. Document Similarity (RYAN KHO YUEN THIAN).....	34
3. Answering General Questions with/without Uploading a File (RYAN KHO YUEN THIAN).....	35
THONG CHENG HOW.....	39
1. Text to speech (THONG CHENG HOW).....	39
2. Translates text into different languages (THONG CHENG HOW).....	42
3. Summarisation (THONG CHENG HOW).....	44
ONG WENG KAI.....	47
1. Speech to Text (ONG WENG KAI).....	47
2. Accept PDF/DOCX/TXT (ONG WENG KAI).....	48
3. Handling unrelated question (ONG WENG KAI).....	50
Results.....	52
a) User Interface.....	52
b) Accepting Multiple File Types (PDF, DOCX or TXT).....	53
c) Validation of Responses to File-Related Questions (includes summarisation & translation).....	54
d) Answering General or Unrelated Questions with/without Uploading a File.....	69
e) Document Similarity Check.....	72

f) Speech-to-Text and Text-to-Speech.....	73
g) Handling Spelling Mistakes or Typos.....	74
Discussion and Conclusion.....	75
1. Achievement of Objectives.....	75
2. The Significance of Result.....	76
3. Limitations.....	77
4. Future improvements.....	78
References.....	79

Introduction

In the contemporary era driven by data, individuals, whether students aiming to derive insights from academic papers or data analysts in pursuit of answers within datasets, face a deluge of information stored in a range of file formats with files in all shapes and sizes. From PDFs containing research papers to DOCX reports and plain text documents (TXT), the demand for efficiently accessing and extracting information from these diverse document types continues to grow also with technology constantly evolving.

In the field of Academic Research and Education, students and researchers may need to extract key information from research papers in PDF format to answer certain questions that they may have in mind or to write a literature review or summary of the research paper. In schools and universities, students may need to answer questions related to their textbooks. For professionals in Business Intelligence and Data Analytics, it may not be simple and convenient for them to retrieve data from business reports formatted in DOCX to access insights and key metrics. Within the legal domain, it may be challenging for lawyers to extract vital information from lengthy legal documents, including contracts, case briefs, agreement documents and enterprises to navigate intricate regulatory documents to stay compliant with the regulations and laws. In the healthcare industry, medical professionals often need to extract important information from patient records to facilitate diagnosis and treatment decisions for the patient. As for medical researchers, they may need to study medical research papers in order to extract findings that are pertinent for their studies. In customer support systems, businesses may need to manage inquiries and give information sourced from documents like FAQs, guides and manuals (K, 2023).

A File/Document Chatbot can address all the issues mentioned above. This is a flexible tool designed to streamline the extraction and analysis of information stored in PDFs, DOCX files and TXT documents file type. This would help users to save time and avoid information overload from lengthy passages and pages. Users no longer need to comb through pages of text as a File Chatbot can help to extract information from the article, which would undoubtedly change how we process and interact with information. Users can read through the summary generated by the Chatbot and then decide whether or not to spend their time reading the full-length article. Not only that, users could also generate the summary contents in point form to easily link the points together for easier understanding. This innovation would help change how we learn and access written content, making learning more effective and efficient across various sectors (K, 2023).

In order to create a chatbot with the capability of allowing the user to upload PDF, DOCX and TXT files, we used **Streamlit** for the user interface. Because **Streamlit** is considered as one of the top open source Python web frameworks for Data Science and Machine Learning projects, most professionals use this framework to create creative, innovative and responsive user interfaces. Apart from the user interface, we used the **Spyder** IDE to write our code and **Anaconda prompt** to run the source code. We chose **Python** as the programming language. As for the **Large Language Model** (LLM), we decided not to use OpenAI's LLM because it's a paid service. Since there are other alternative free LLMs, we chose the **google/flan-t5-xxl** LLM from **HuggingFace**, which is a Data Science and Machine Learning platform that is open source, serving as a central hub for AI professionals and enthusiasts, functioning akin to a GitHub specifically designed for AI. As for creating the embeddings, we will use **sentence-transformers/all-MiniLM-L6-v2**.

The Objectives of this project are:

- To automate the process of extracting information from documents, which are PDF, DOCX and TXT files.
- To provide assistance to users in navigating and understanding documents, making it easier for them to find the information they need in a short amount of time.
- To enhance user engagement by providing a user-friendly interface for extracting information from documents with a click of a button.
- To enrich the capabilities of natural language understanding to interpret and respond to user queries related or not related to the content of the uploaded documents.
- To allow users to make queries via text or voice and to receive responses via text or voice.
- To allow users to translate the contents of the documents uploaded to another language, which is either Bahasa Malaysia or Mandarin.
- To provide a fast and efficient way to summarise the document, possibly within seconds, depending on the size of the document.
- To allow the user to check for document similarity by comparing two files (the file types of the 2 files do not have to be the same).
- To correct spelling mistakes made by the user while names and special terms remain unchanged.
- To use pre-processing (lemmatisation, stop words removal) before performing Cosine Similarity.

The Research Questions to be answered in this project are:

- How can the File Chatbot be designed to ensure a user-friendly experience?
- What factors contribute to user satisfaction and efficiency when interacting with a file chatbot?
- How accurately does the chatbot understand and respond to user queries related to file content?
- How robust is the chatbot in handling errors, unexpected inputs or ambiguous queries related to file interactions?

The Research Design:

We chose the “Natural Language Processing (NLP) Evaluation” research design, which refers to the assessment of the File Chatbot’s language understanding capabilities using established NLP evaluation metrics. This also involves assessing the chatbot’s ability to accurately interpret and understand user queries related to files and identifying instances where the chatbot misinterprets or fails to understand certain file-related queries.

Research Background

a) What is a File Chatbot?

A File Chatbot represents a cutting-edge software application driven by Artificial Intelligence (AI) and Natural Language Processing (NLP) capabilities. Its purpose is to examine and retrieve information from various file types, such as PDF, DOCX and TXT files. In contrast to conventional chatbots that mainly engage in text-based conversations, a File Chatbot concentrates on comprehending and answering queries related to the information contained within these files (K, 2023).

b) Existing File Chatbots in the Market

There are several File Chatbot products that exist today, such as PDFgear Chatbot, Monica ChatPDF, ChatPDF and ChainDesk (MonicaChatPDF, n.d.;Chaindesk, 2023;ChatPDF, n.d.;PDFGear, 2023). Integrated with the latest ChatGPT API, the PDFgear Chatbot represents an innovative technology designed to swiftly retrieve information from extensive PDF files, including books, essays, textbooks, research papers and legal contracts. It was awarded as the best AI PDF summarizer and the best AI summarizer (PDFGear, 2023). ChatPDF is an AI tool provided by Monica, facilitating comprehension and interactive communication with PDF documents. It can be used for document summary, Q&A, content comparison and document translation (ChatPDF, n.d.). Chaindesk empowers you to create a personalised AI chatbot, trained on your custom data, within a matter of seconds. Some of its capabilities include chatting about any Word document, obtaining summaries, searching for information and querying (Chaindesk, 2023). **Overall**, these examples of file chatbots in the market have introduced additional, interesting concepts and functionalities, such as text summarisation, content comparison and document translation, that can improve the overall usefulness of the chatbot.

c) General Methodology of a Conversational Agent that is Text-Based

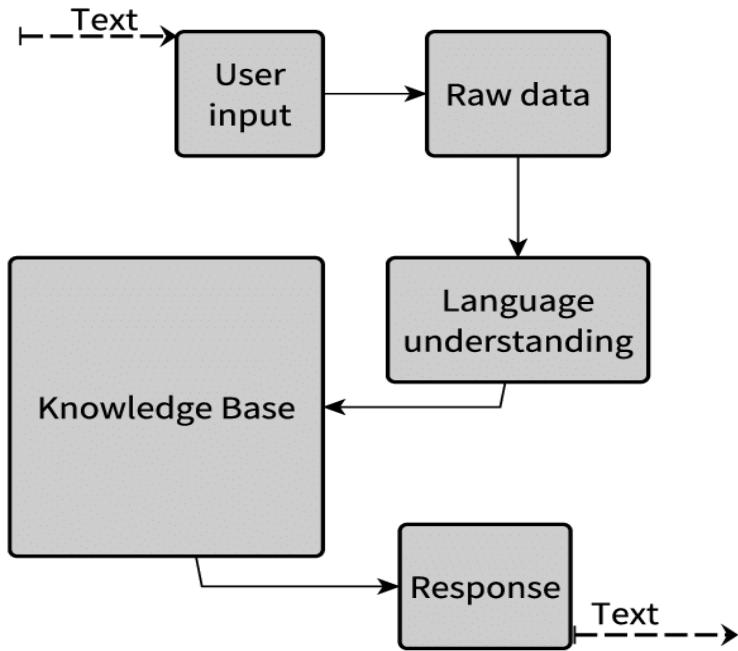


Figure c.1: How a Text-based Conversational Agent Works? (Garrido, 2018)

Modern mobile devices are now equipped with uncomplicated chatbots that manage basic tasks, such as initiating phone calls and setting alarms. Moreover, there are now automation tools designed for crafting more intricate chatbots, readily accessible for commercial applications. The pervasive integration of machine learning into everyday activities has elevated the process of 'thinking' beyond mere pattern matching, with notable instances being readily available from major tech entities like Facebook, Google and Amazon. Despite variations in implementation details and limitations among frameworks, the fundamental concept remains consistent—interpret raw data, imbue it with meaning, and respond appropriately based on a knowledge base (refer to Figure c.1 above for the general methodology of a conversational agent that is text-based). In the realm of natural language processing, chatbots heavily rely on pattern recognition and bigram identification, typically facilitated by conversation frameworks. Nevertheless, it falls upon chatbot designers to curate the knowledge base and furnish the learning engine with relevant examples (Sánchez-Díaz et al., 2018).

d) LangChain

It serves as a framework for building applications driven by language models. It facilitates applications that possess contextual awareness (linking a language model to contextual sources such as prompt instructions, few-shot examples, or relevant content to anchor its responses) and engage in reasoning (depending on a language model for reasoning, determining how to respond based on the provided context and deciding on appropriate actions) (LangChain, 2023).

e) Language Model (LLM)

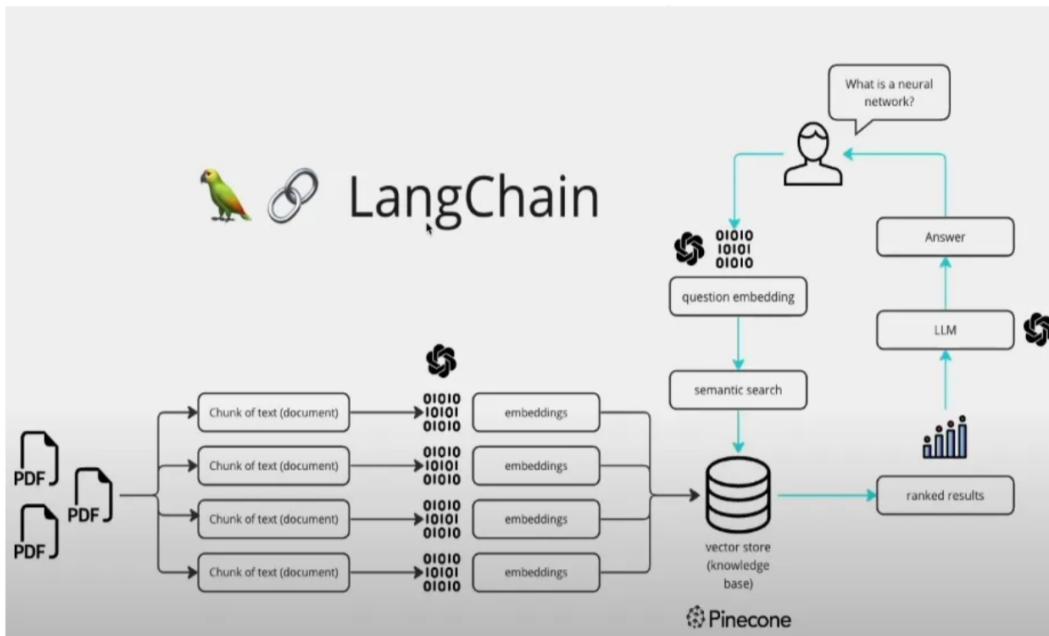
A **language model** employs machine learning to generate a probability distribution across words, predicting the most probable subsequent word in a sentence based on the preceding input. These models learn from textual data and can be utilised for generating new text, anticipating the next word in a given text, as well as for tasks such as speech recognition, optical character recognition, and handwriting recognition (Kapronczay, 2022). The training regimen for a Large Language Model (LLM) entails immersing the model in extensive datasets, typically encompassing billions or even trillions of words. These datasets may be sourced from a variety of outlets, including books, articles, websites, and other textual reservoirs. The LLM undergoes training through unsupervised learning, wherein it predicts the subsequent word in a provided context. Through iterative cycles and exposure to a wide range of textual content, the model gains proficiency in grasping grammar, semantics, and the wealth of worldly knowledge embedded in the training data. An exemplary instance of a large language language model is the GPT (Generative Pre-trained Transformer) series developed by OpenAI, including models like GPT-3/GPT-4 (Kanade, 2023).

Large Language Models (LLMs) have found **application** across a diverse spectrum of uses. They can be specialised for particular tasks through fine-tuning with additional supervised training data, enabling them to excel in activities like sentiment analysis, named entity recognition, or even strategic games like chess. LLMs are also deployable as content creators, language translation systems, chatbots and virtual assistants. **However**, it's important to acknowledge that LLMs have the potential to produce misleading or biased content due to their learning from biases inherent in the training data. Initiatives are underway to address and minimise these biases, ensuring responsible LLM use. Notably, influential figures in the tech industry, such as Elon Musk, and academic researchers have endorsed a plea to AI labs, urging a temporary pause in the training of powerful AI systems to prevent unforeseen societal consequences like the dissemination of misinformation (Kanade, 2023).

f) Projects related to File Chatbot

Example 1: Building a PDF Chatbot using a Langchain Architecture

Below is the **architectural representation** of the file chatbot explored by Wahid (2023), which consists of 7 components.



○ Vector Store

- Enables information to be retrieved efficiently from the PDF files
- Functions as a storage facility for housing vectorized representations of text segments extracted from PDFs (These vector representations are created through the utilisation of language models and embeddings)

○ Embeddings

- They capture the meaning and context of the text segments, allowing for significant comparisons and retrieval. Embeddings are produced by inputting the text chunks into pre-existing language models or embedding models, like those offered by OpenAI or Hugging Face. These models transform the textual information into compact vector representations (Wahid, 2023).

○ Large Language Models (LLM)

- Play a pivotal role in the chatbot framework, offering sophisticated language generation and comprehension capabilities.
- These pre-trained models have acquired the ability to comprehend and produce text resembling human language. Through fine-tuning on particular tasks or domains, like conversational retrieval, the chatbot can participate in more dynamic and context-aware conversations

- **Conversational Retrieval**

- Retrieves pertinent information by considering user queries and contextual comprehension, entailing the comparison of the vector representation of the user's query with the vectors stored in the Vector Store.
- Pinpoints the most applicable text segments containing the information necessary to respond to the user's inquiry.

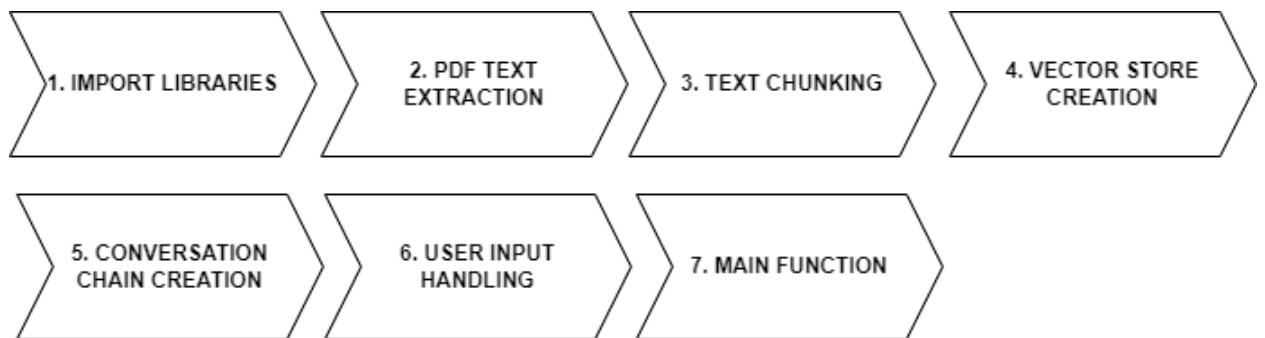
- **Chat History and Context**

- Preserves the continuity of the conversation
- A chatbot usually monitors the chat history, encompassing both the user's queries and the chatbot's responses, in order to deliver coherent and pertinent answers, drawing on the context of the ongoing conversation.

- **User Interface**

- Offers a smooth interaction platform, allowing users to input questions and receive pertinent answers.
- Typically involves a text input field where users can enter their queries.
- One of the ways to create the user interface is by using Streamlit. Because Streamlit is considered as one of the top choices for data science and machine learning projects, most professionals use this application to create creative, innovative and responsive designs of user interfaces for their client to interact with.

The **implementation** of this chatbot is illustrated below:



1. **Import libraries**

2. **PDF Text Extraction**

- a. Extract text content from PDF files uploaded

3. **Text Chunking**

- a. Divides the obtained text into smaller segments utilising the CharacterTextSplitter class from the Langchain library, enhancing retrieval efficiency and offering more accurate responses.

4. **Vector Store Creation**

- a. Accepts the text chunks as input and employs either OpenAIEmbeddings or HuggingFaceInstructEmbeddings to produce embeddings for each text chunk. Subsequently, these embeddings are applied to construct a vector store with the FAISS library.

5. Conversation Chain Creation

- a. Establishes a chat model, choosing between ChatOpenAI or HuggingFaceHub, and initiates a memory buffer for the conversation. The chat model, along with the vector store retriever and memory, is subsequently integrated to constitute the conversation chain.

6. User Input Handling

- a. Receives a user inquiry as input and leverages the conversation chain to produce a response. The chat history is retained in the session state to preserve the context of the ongoing conversation.

7. Main Function (Program Entry Point)

- a. Configures the Streamlit user interface, manages user input, and oversees the uploading of PDF files. Additionally, it invokes essential functions for text extraction, vector store creation, and the establishment of the conversation chain.

Although the Langchain Chatbot investigated by Wahid (2023) signifies a notable progression in the fields of natural language processing and information retrieval, it clearly has several **limitations**.

1. Token Limit Imposed by OpenAI

- o The OpenAI token limit, capped at 4,096 tokens, poses a constraint on information retrieval. This restriction may lead to incomplete data extraction, especially evident when dealing with numerous documents. To overcome this limitation, exploring alternative LLMs with higher token limits is crucial. The current trend in LLM development emphasises models with larger token capacities, offering potential improvements in accuracy and comprehensiveness, particularly for extensive document collections (Wahid, 2023).

2. Lack of Support for Other Document Formats

- o Beyond PDFs, the Chatbot can be improved to handle various document formats like Word documents or web pages. This broader compatibility would enable users to extract information from a more extensive array of sources, augmenting the Chatbot's versatility (Wahid, 2023).

3. Limited Error Handling and Feedback from User

- o Improve error handling to provide clearer guidance on incorrect queries, thereby enhancing the overall user experience, and integrate feedback mechanisms for continuous learning and improvement based on user interactions (Wahid, 2023).

4. Did not use More Sophisticated Language Models and Embeddings

- o For enhanced conversational proficiency and precision, exploring the integration of advanced language models and embeddings is a potential avenue. This may entail using cutting-edge models and embeddings from evolving sources like OpenAI or HuggingFace, providing improved language understanding and generation capabilities (Wahid, 2023).

Limitations aside, based on the current chatbot functionality, Wahid (2023) recommends the following to effectively use it:

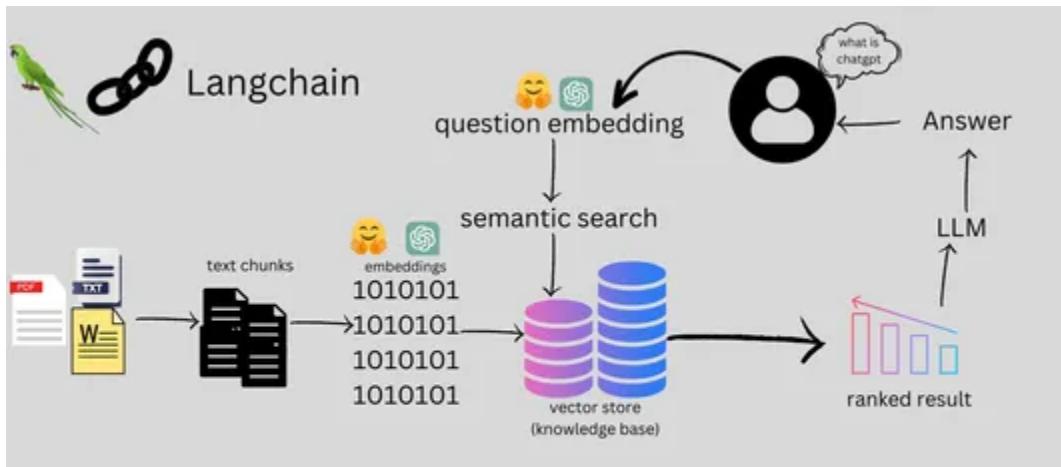
1. Initiating the sequence involves **uploading PDF documents** through the application's sidebar. This step grants users the flexibility to integrate diverse sources, as elucidated by Wahid (2023). The selection and upload of preferred PDF files are seamlessly executed through the provided interface.
2. Upon successful upload, users transition to the main **chat interface**, where interaction with the chatbot commences. Queries, posed in natural language, are directed towards the content of the uploaded PDFs, aligning with Wahid's (2023) insights. The chatbot, employing advanced natural language processing and machine learning techniques, exhibits a capacity for accurate comprehension and interpretation.
3. Post-query submission, the chatbot expeditiously processes information extracted from the uploaded PDFs, as described by Wahid (2023). Utilising conversational retrieval techniques and the efficacy of language models, it crafts **responses** marked by contextual relevance, accuracy, and informativeness. This process, as elucidated by Wahid (2023), contributes significantly to enriching the user's information-seeking experience.

Evaluation/Discussion of Chatbot System (Example 1):

This project has laid out the essential components needed for a file chatbot, ranging from vector store and LLM to the user interface and chat history. However, there are several notable issues with this project. Firstly, it is not very versatile in terms of the files that it can work with since it can only accept PDFs. In today's world, people usually work with several kinds of files, such as DOCX, TXT, CSV, etc so it may not be convenient or possible for the users to always convert their files into PDF just to interact with the File Chatbot. Secondly, the chatbot's conversational functionality is limited. This means that the user is only able to interact with the chatbot when a file has been uploaded. Therefore, if the user tried to interact with the chatbot without uploading any file, it would result in an error because at that point in time, the knowledge base/vector store is not initialised so the chatbot will not be able to answer the user's questions. Moreover, even if a file is uploaded, if the user tries to ask the chatbot questions that are not related to the file, the chatbot may either produce an error, give responses reflecting uncertainty like "I don't know" or give unusual or inappropriate responses. Last but not least, the chatbot is only restricted to text input and output. It did not incorporate the ability to take voice input and produce voice output.

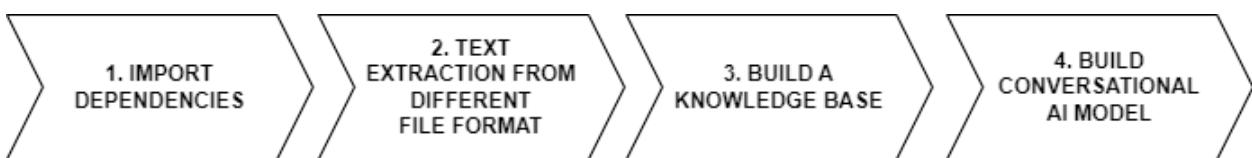
Example 2: MultiFileChatbot

The workflow of the Multi-File Chatbot involves several key steps, from user interaction to file processing and answering questions (K, 2023). Below is the **overview** of the workflow :



- Users engage with the Multi-File Chatbot through a web or chat platform.
- Users submit queries for the chatbot's information retrieval.
- Users have the option to upload specific files, including PDFs, DOCX, TXT, and CSV formats.
- The chatbot processes text from the uploaded files, incorporating cleaning and segmentation.
- Efficient indexing and storage of processed text are carried out by the chatbot.
- The chatbot employs Natural Language Processing (NLP) for understanding user queries.
- Relevant information is retrieved, and answers are generated by the chatbot.
- Responses from the chatbot are provided in natural language.
- Users receive responses and can continue the interaction.
- The conversation progresses with additional queries.
- The conversation concludes at the user's discretion.

The **implementation** of this chatbot is illustrated below:



1. Import Dependencies
2. Extract Text from Different File Format
 - Involves processing PDF, DOCX, TXT and CSV files
3. Build a Knowledge Base
 - Information gathered from various files is consolidated and divided into manageable chunks.

- b. These chunks are subsequently employed to construct a sophisticated knowledge base for the chatbot.
- c. Advanced NLP techniques are used to enhance the comprehension of the content.
- d. For creating the vector store, K (2023) combined Hugging Face models with LangChain to achieve peak performance.

4. Build a Conversational AI Model

- a. In order for the chatbot to deliver meaningful responses, a conversational AI model is required. Within this project, K (2023) implemented a model sourced from Hugging Face's model hub.

5. Answer User Queries

- a. The chatbot leverages its knowledge base and NLP models to furnish timely and pertinent answers.

6. Deploy Chatbot with Streamlit

- a. K (2023) deployed the chatbot using Streamlit where users have the capability to upload their documents and pose questions, with the chatbot generating responses based on the document content.

Although this chatbot is clearly better than the project that precedes it, there are still **rooms for improvement**.

1. Use More Sophisticated NLP

- a. Expand the language proficiency of the chatbot to cater to users in various languages (Multi-Language Support)
- b. Assess user emotions and customise responses accordingly, enhancing the personalised nature of the experience (Sentiment Analysis).

2. Improve File Handling

- a. Support extra file formats, such as Excel spreadsheets or PowerPoint presentations
- b. Integrate OCR technology to extract text from scanned documents and images

3. Integrate Machine Learning

- a. Integrate active learning methodologies to consistently enhance the chatbot's effectiveness.
- b. Collect user feedback and employ it to refine models, improving the accuracy of responses.
- c. Train NLP models customised to enhance understanding and provide responses that are contextually aware.

4. Provide a Feedback System

- a. Prompt users to share their feedback on chatbot interactions, enabling the identification of areas for enhancement.
- b. Integrate automated feedback analysis to gain insights into user satisfaction and pinpoint areas that require attention.

Evaluation/Discussion of Chatbot System (Example 2):

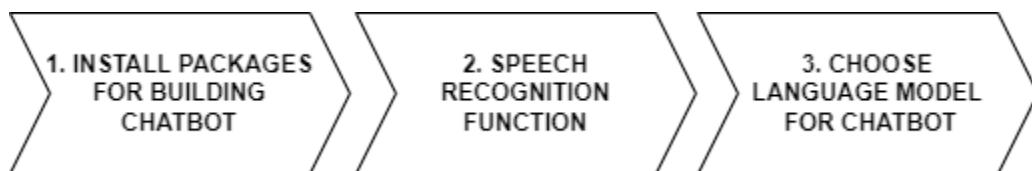
The architecture of the chatbot system in example 2 is more or less the same as the one in Example 1. However, it is an improvement of the project in example 1 as it can support multiple files, which are PDF, DOCX, TXT and CSV. In addition, the fact that both projects use Streamlit for creating the user interface indicates that it is a good choice to use Streamlit. Nevertheless, the same issues from Example 1 still persist in Example 2. The chatbot's conversational functionality

is limited. This means that the user is only able to interact with the chatbot when a file has been uploaded. Therefore, if the user tried to interact with the chatbot without uploading any file, it would result in an error because at that point in time, the knowledge base/vector store is not initialised so the chatbot will not be able to answer the user's questions. Moreover, even if a file is uploaded, if the user tries to ask the chatbot questions that are not related to the file, the chatbot may either produce an error, give responses reflecting uncertainty like "I don't know" or give unusual or inappropriate responses. Last but not least, the chatbot is only restricted to text input and output. It did not incorporate the ability to take voice input and produce voice output.

Example 3: Building Conversational AI Chatbot with NLP

The project concerns building a conversational Chatbot with Python that integrates an AI algorithm with speech recognition processes. In order to achieve this goal, it will use pre-trained Transformer models and speech recognition APIs (Mondal, 2023).

The **implementation** of this chatbot is illustrated below:



1. Install Packages for Building Chatbot

- a. Install SpeechRecognition library for converting speech to text (Mondal, 2023)
- b. Install gTTS for converting text to speech and to speak it out (Mondal, 2023)
- c. Install transformers and tensorflow for Language Model (Mondal, 2023)
- d. Create the ChatBot class

2. Speech Recognition Function

- a. Goal is to transform from human spoken languages into a form that can be understood by AI or the computer (Mondal, 2023)
- b. Google API is used to achieve this (Mondal, 2023)

3. Choosing Language Model for Chatbot

- a. Use the DialogGPT model developed by Microsoft through the analysis of countless discussions and continuous dialogues occurring on the Reddit platform within a specific timeframe (Mondal, 2023).

Sample run of the program:

```
All model checkpoint layers were used when initializing TFGPT2LMHeadModel.  
All the layers of TFGPT2LMHeadModel were initialized from the model checkpoint at microsoft/DialoGPT-medium.  
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFGPT2LMHeadModel  
----- Starting up Dev -----  
Listening...  
Me --> hello dear  
Dev --> Hello, dear!  
Listening...  
Me --> hello Dev  
Dev --> Hello I am Dave the AI, what can I do for you?  
Listening...  
Me --> what is the weather  
Dev --> It's a bit chilly.  
Listening...  
Me --> what is the time  
Dev --> 20:10  
Listening...  
Me --> ERROR  
Dev --> Sorry, come again?  
Listening...  
Me --> thanks  
Dev --> cool!  
Listening...  
Me --> thanks  
Dev --> I'm here if you need me!  
Listening...  
Me --> ok exit  
Dev --> Have a good day  
----- Closing down Dev -----
```

Conclusion of Chatbot System:

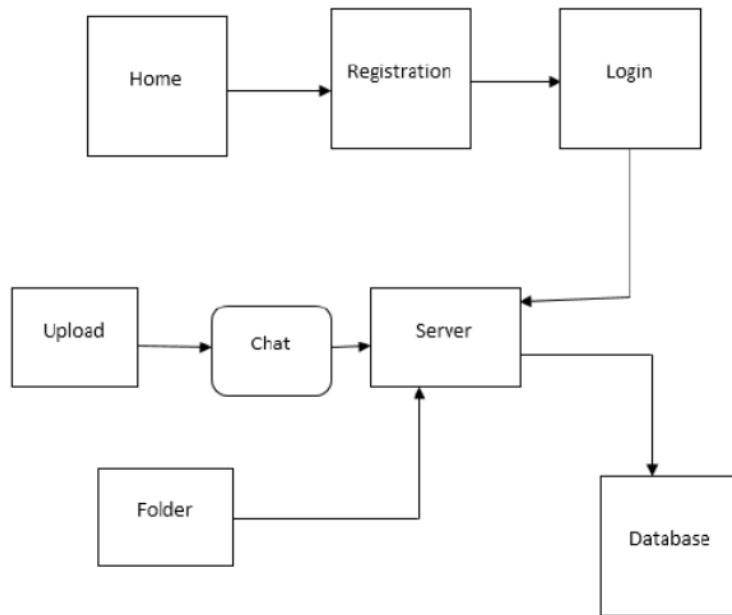
Mondal (2023) states that the conversational chatbot can be used as a basis for creating one that emulates human communication. The code is versatile and can function as the fundamental components for similar chatbot projects (Mondal, 2023).

Evaluation/Discussion of Chatbot System (Example 3):

The system has introduced an interesting idea of accepting voice input from the user. When the voice input is received, it will be converted into text and the output from the chatbot will be in voice form as well. It has also introduced the interesting concept of assigning a non-living entity like this chatbot a name to increase the user-friendliness of the system. Although the appealing aspect of this project is its conversational abilities with the user, it did not cover the aspect of accepting file from the user and answering questions based on it. Besides that, it does not provide the user with the opportunity to enter text as an input.

Example 4: Smart Chatbot for Navigating Repository Content through Guidance

The project is about developing an automated live chat support service whereby a web chatbot application is used to answer user's questions based on the files that he/she uploads into the system's database (Shinde et al., 2019). Below is the system's architecture diagram:



The system follows a sequential process for intelligent interaction with users based on uploaded files.

- First, users input or ask questions related to the uploaded file.
- Subsequently, these questions, intertwined with the uploaded file's context and user input, are displayed in a queue for streamlined processing.
- The system then selects a pertinent object from the user's question and compares it with streaming questions. As part of the process, the system reads the content of the uploaded file, breaking down the user's question into lines for comparison with streaming questions. Each line is scrutinised to check the alignment with the input question/object, utilising data from the uploaded file and streaming questions.
- The generated answer is then seamlessly appended to the original question, creating a comprehensive response that is presented to the user.

This stepwise approach ensures an effective and contextually rich interaction, culminating in the user receiving a display of their original question along with the system-generated answer (Shinde et al., 2019).

Conclusion of Chatbot System:

Shinde et al. (2019) stated that the system is beneficial in several ways. Firstly, the accessibility of chatbots around the clock ensures continuous execution of commands without fatigue, operating diligently throughout the entire year. Secondly, their inherent flexibility makes them adaptable for use across diverse industries, showcasing their versatility as virtual assistants. Furthermore, Shinde et al. (2019) emphasised that the ease of communication with users is a significant advantage, fostering seamless interaction. Additionally, the system's capacity to reduce human

effort in reading files was highlighted, positioning chatbots as efficient aids in information processing. Finally, Shinde et al. (2019) underscored the system's ease of maintenance, emphasising its practicality and user-friendly nature. However, simultaneous responses are not possible for multiple users, and only files in a specific format are accepted.

Evaluation/Discussion of Chatbot System (Example 4):

The system possesses notable features. Firstly, it incorporates a database that permanently stores user-uploaded files, eliminating the need for repeated uploads during subsequent interactions. Therefore, in terms of storing data, this system is more superior than the other 3 projects that precede it. Additionally, a security measure requires users to register successfully before accessing the system. However, it does not incorporate the use of speech-to-text and text-to-speech technologies, and it also lacks support for simultaneous responses. This limitation implies that users might experience delays as they wait for their questions to be answered.

g) Relevant Concepts & Techniques

- **Technique/Concept 1: Text-Summarisation**

In the field of NLP, models for Text Summarization autonomously condense various forms of content, such as documents, papers, podcasts, and videos, into concise and significant excerpts. These models leverage sophisticated advancements in Deep Learning and Machine Learning research to accomplish this task (Foster, 2023).

Numerous text summarization techniques have been developed over the past few decades, making it challenging to provide a single explanation of how text summarization functions. However, these methods can be categorised based on their overarching approaches to addressing the text summarization challenge (Foster, 2023).

One particularly clear and useful distinction lies in the classification of **Extractive** and **Abstractive** text summarization methods:

Extractive methods aim to derive the most relevant information directly from the given text. Extractive text summarization represents the more conventional approach, primarily due to its relative simplicity when compared to abstractive methods (Foster, 2023).

In contrast, **Abstractive** methods strive to generate an original body of text that effectively summarises the content of the source text. The inherent complexity of this task is evident, as it involves a considerable degree of freedom, not limited to merely presenting a subset of the original text. Despite their increased intricacy, Abstractive methods yield more flexible and arguably more faithful summaries, especially in the era of Large Language Models (Foster, 2023).

An **example** of a model that can be used for Text Summarisation is BART. BART, a model pre-trained on the English Language, is a transformer model characterised by a bidirectional (similar to BERT) encoder and an autoregressive (resembling GPT) decoder in a sequence-to-sequence (seq2seq) architecture. In its pre-training phase, BART undergoes two key steps: **(1)** introducing noise to the text using a random noising function, and **(2)** training the model to reconstruct the original text. When specifically adapted for tasks like text generation (e.g., summarization, translation), BART demonstrates notable effectiveness. However, it also performs well in comprehension tasks such as text classification and question answering. This specific

version has undergone fine-tuning on the CNN Daily Mail dataset, which comprises an extensive collection of text-summary pairs (Hugging Face, n.d.).

- **Technique/Concept 2: CountVectorizer**

The CountVectorizer, a valuable feature offered by the scikit-learn library in Python, enables the transformation of a given text into a vector by considering the frequency (count) of each word throughout the entire text. This proves beneficial when dealing with multiple texts, allowing the conversion of each word in each text into vectors for subsequent text analysis (Verma, 2022). For example, we have the following code:

```
document = [ "One Geek helps Two Geeks", "Two Geeks help Four Geeks", "Each Geek helps many other Geeks at GeeksforGeeks." ]
```

CountVectorizer constructs a matrix where every distinct word is depicted by a column, and each instance of text from the document corresponds to a row in the matrix. The content of each cell signifies the frequency of the respective word within that specific text sample (Verma, 2022).

	at	each	four	geek	geeks	geeksforgeeks	help	helps	many	one	other	two
document[0]	0	0	0	1	1	0	0	1	0	1	0	1
document[1]	0	0	1	0	2	0	1	0	0	0	0	1
document[2]	1	1	0	1	1	1	0	1	1	0	1	0

Based on the table above, we can see that:

1. The table features 12 distinct words portrayed as columns, with 3 text samples representing rows
2. Each cell within the table contains a numerical representation denoting the count of the respective word in that specific text.
3. All words have been converted to lowercase, and the columns are organised alphabetically.

Within the CountVectorizer, the words are not saved as strings; instead, they are assigned specific index values. For instance, 'at' would be assigned index 0, 'each' would be assigned index 1, 'four' would be assigned index 2, and so forth (Verma, 2022).

0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	1	1	0	0	1	0	0	0	1
0	0	1	0	2	0	1	0	0	0	0	1
1	1	0	1	1	1	0	1	1	0	1	0

- **Technique/Concept 3: TF-IDF Vectorizer**

TF-IDF stands for Term Frequency Inverse Document Frequency, a widely used algorithm for converting text into a numerical representation that is meaningful for training machine learning algorithms for prediction (Chaudhary, 2021). In other words, it involves determining the relevance of a word in a text collection or corpus. The significance of a word is directly proportional to its frequency within the text but is adjusted based on the word's frequency in the overall corpus or dataset (Riturajsaha, 2023).

The term “TF-IDF” consists of several terminologies, which are mentioned below:

1. **Term Frequency (TF)**

- a. Within document d, frequency signifies the count of occurrences of a specific word, denoted as t. Consequently, the relevance of a word increases with its frequency in the text (Riturajsaha, 2023).
- b. Given the non-significant order of terms, a vector can be employed to characterise the text using bag-of-term models. Each term in the document corresponds to an entry in the vector, with its value representing the term frequency (Riturajsaha, 2023).

2. **Document Frequency (DF)**

- a. This assesses the meaning of the text, akin to TF, across the entire corpus collection. The distinction lies in document d, where TF is the frequency count for a term t, while DF represents the occurrences of term t in the document set N (Riturajsaha, 2023).
- b. DF indicates the number of papers within which the word is present (Riturajsaha, 2023).

3. **Inverse Document Frequency (IDF)**

- a. It evaluates the relevance of a word, with the primary objective of finding pertinent records that fit the demand. As TF treats all terms as equally important, it becomes feasible to use term frequencies not only to gauge the term's weight in the document (Riturajsaha, 2023).

- **Technique/Concept 4: Cosine Similarity**

Cosine similarity is a measure used in information retrieval and text mining to assess how similar two entities are often represented as vectors in a multi-dimensional space. It calculates the cosine of the angle between these two vectors, which can represent various data types such as text documents, users' preferences, or product descriptions. The resulting similarity ranges from -1, meaning exactly opposite, 0, indicating orthogonality (decorrelation), and 1, indicating identicalness. However, in the context of text similarity and many other applications, the vectors are usually non-negative (as in frequencies of terms), so the similarity lies between 0 and 1.(Cosine Similarity, n.d.)

Use:

Cosine similarity is prevalent in recommendation systems (like suggesting movies or products). For instance, in a recommendation system, the movies watched or liked by a user are transformed into a user vector, and cosine similarity helps to find other users with similar tastes. Cosine similarity can also be used in text analysis by comparing how similar two documents are in

content. In chatbot, the cosine similarity can be used to understand the user's intent from their message. The chatbot can recognise what the user intends to convey and respond appropriately accordingly, maintaining a coherent and contextually relevant conversation. Another application is for FAQ matching, where each FAQ can be vectorised, and when a user question comes in, the bot can find the most similar FAQ entry to provide an appropriate response. If a low cosine similarity is found in the new query, it might need to update chatbot knowledge and its understanding to tackle user questions. (Karabiber, n.d.)

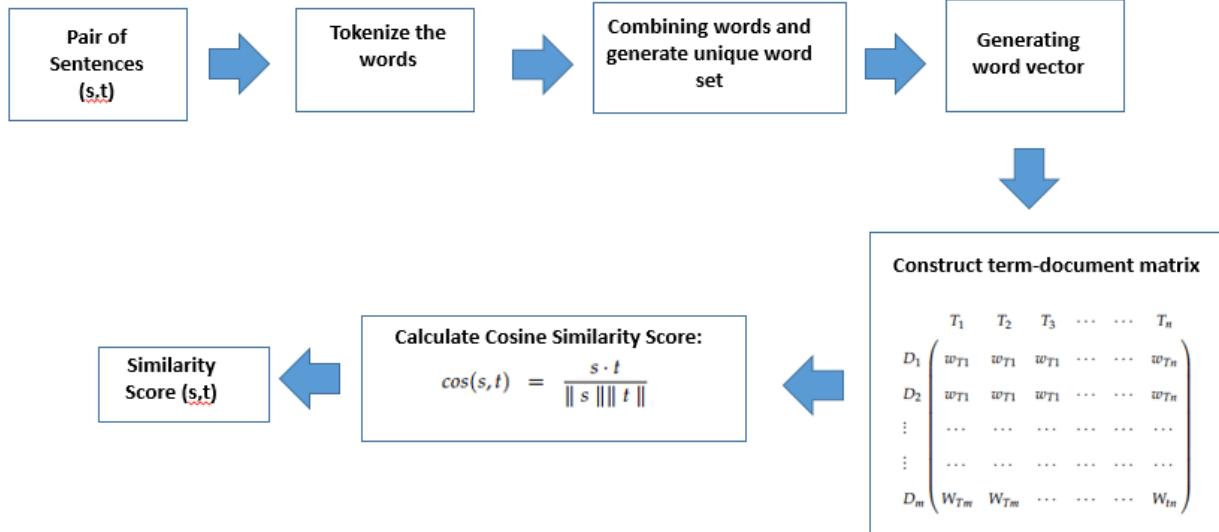


Figure g.4.1: How to calculate cosine similarity between sentences? (Alhadi et al., 2019)

- **Technique/Concept 5: Jaccard Similarity**

Also known as Jaccard Index, it is used to compute the agreement of 2 data sets, which is calculated as the size of the intersection of 2 sets divided by their union size (Amrikahr, 2023).

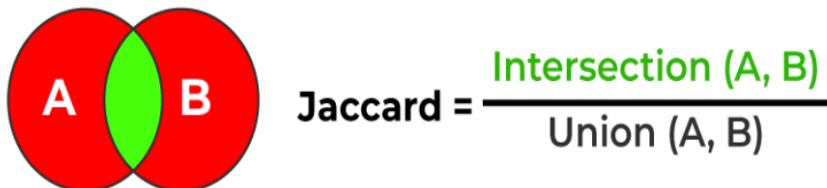


Figure g.5.1: Formula for Jaccard Similarity (Amrikahr, 2023)

Usage

Jaccard similarity proves particularly effective when the arrangement of items is inconsequential, focusing solely on the presence or absence of elements. This measure is widely employed in:

- **Text Analysis:** In the realm of natural language processing, Jaccard similarity serves as a tool for comparing texts, text samples, or even individual words (Amrikahr, 2023).
- **Social Network Analysis:** In the domain of social networks, Jaccard similarity is applied to discern similarities between user profiles or groups (Amrikahr, 2023).

- **Recommendation Systems:** Jaccard similarity proves beneficial in recommendation systems by identifying similar items or products based on user behaviour (Amrikahr, 2023).
- **Data Deduplication:** Employing Jaccard similarity facilitates the detection of duplicate or closely resembling records within a dataset (Amrikahr, 2023)

- **Technique/Concept 6: Speech-To-Text**

Speech-to-text (STT) technology, also known as automatic speech recognition (ASR), is a computational technique that converts spoken language into written text.

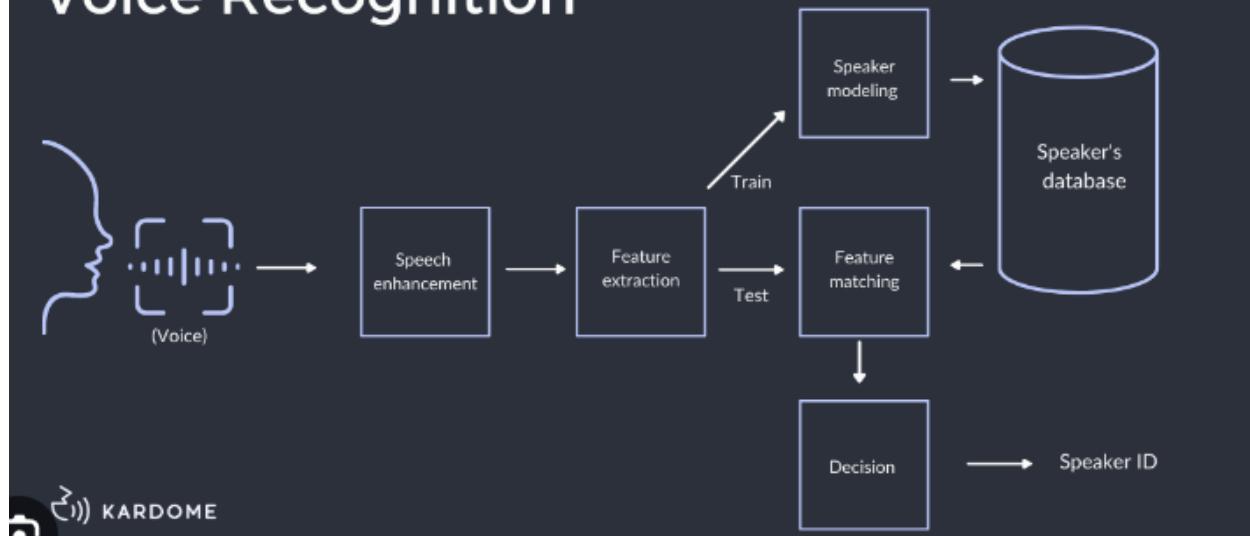
The system will capture the spoken words through the microphone and convert them into a digital acoustic signal. After that, it will undergo pre-processing, normalising and reducing the noise to deal with different speeds, accents, and speech peculiarities. Later, it will break down speech into phonemes (the minor sound units) to identify distinctive features representing spoken sounds. The phonemes will later be used to match the sound to a set (phonetic set) using model Hidden Markov Models (HMMs) or Deep Neural Networks (DNNs) to recognise the patterns in the speech.

To improve the system's accuracy, it will use the database of vocabulary and language rules to interpret the words and predict which words are likely to appear in that context. Finally, the system combines the outputs of the acoustic and language models to produce a text transcription of the spoken words (What Is Speech-To-Text Software? - Definition from Techopedia, 2019).

Usage

The technology is used in a wide range of applications, such as accessibility tools for the disabled, where they can use voice-to-text technology instead of typing it out. Besides that, it can be used on transcription services where a person's words in a meeting, lecture, or conversation can be recorded in text. Another usage is language translation, where a person can convert the spoken language into another text. In the chatbot context, applying this technology allows users to have a broader range of choices, whether it is typing or using the voice of text technology, so it can be more convenient, especially for someone with difficulty typing (Tate, 2022).

Voice Recognition



- **Technique/Concept 7: Text-To-Speech**

Definition & Purpose:

Text to speech (TTS) is a natural language model that translates text into audio presentation format (Rouse, 2012). TTS is considered as an assistive technology as it is sometimes called as “read aloud” technology (ReadingRockets, 2023). The model will try to capture the document text to accurately record them as text. Text to Speech technology is now a common technology that renders audio from text to assist users who are unable to read. TTS takes words from computers or different digital devices and converts it to audio format for pronunciation. In an application, it helps kids to improve reading, listening and writing. (ReadingRockets, 2023). In order for TTS to work, it involves several steps because TTS uses advanced algorithms and also machine learning techniques. Below is the overview of how TTS works.

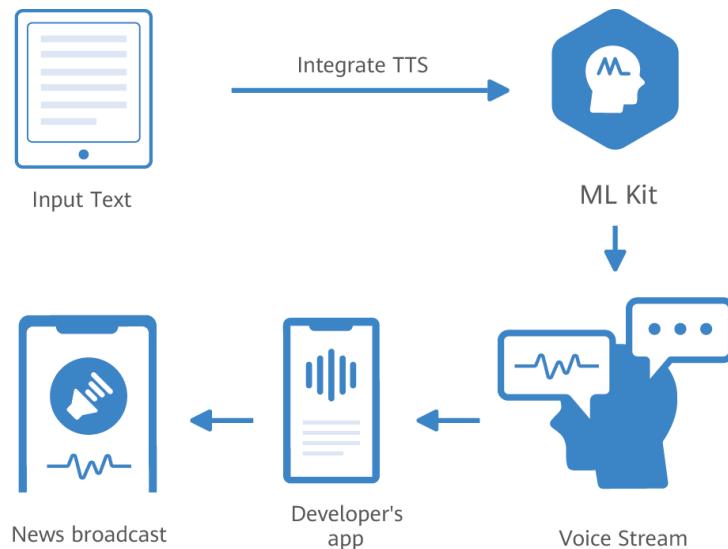
How does it work:

Imagine a tiny translator living inside your chatbot. When the bot crafts a text response, this translator leaps into action, it acts as a model in the back system to translate the text into voice (Dwivedi et al., 2023). First, it meticulously analyses the words one by one, their pronunciations, and the whole sentence's structure, paying close attention to commas and exclamation points as they whisper hints about tone and pauses which is a very crucial part for pronunciation (The most common punctuation marks in English. IDP IELTS Malaysia, n.d.). With this breakdown in hand, the translator then picks a voice, either a pre-recorded human one or an AI-generated whisper. Using algorithms, it breathes life into the text, adding pitch variations, emphasising key words, and inserting natural pauses just like when we talk (Lumen Learning, n.d.)

Finally, the translated sound waves flow out of your device, transforming the chatbot's text into a voice that speaks to you directly. This, in essence, is the magic of text-to-speech in chatbots, adding a layer of human-like interaction and making your conversations feel more natural and engaging. This creates a life like conversation with the chatbot and the human being.

Types of TTS tools:

Google Translate, Amazon Polly (AWS, n.d.), LOVO, Web Player, Synthesys



- **Technique/Concept 8: Translation**

Definition & Purpose:

Chatbot translation is a technology that translates messages sent to or from a chatbot (Language I/O, 2023). Chatbot translators act like magic bridges, connecting people separated by tongues. They weave words into tapestries of understanding, seamlessly switching between languages mid-flow. Business thrives as customer service transcends borders, conversations blossom despite language barriers, and voices are heard, no matter the native tongue. This powerful technology smashes cultural walls, creating a world where understanding reigns and communication truly knows no bounds and everyone can interact and understand each other without having a clue of the language in this era of technology (Kimber, 2021).

How does it work:

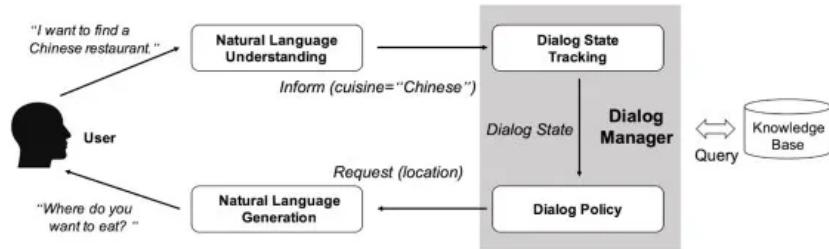
In the swirling ocean of languages, chatbot translation acts as a magical bridge, connecting users and chatbots even when they speak different tongues of languages, no matter its dialects (Barros, 2023). Imagine you ask a question in Chinese to a customer service chatbot. The chatbot, acts as a savvy language detective of a language master, uses its skills to identify your Chinese and then throws your message to a powerful translation engine (Hore, 2023).

This engine, like a skilled linguist, uses its vast knowledge to turn your Chinese into fluent English (Hore, 2023). Armed with your query, the chatbot analyses its meaning and intent, just like it would in English. Then, it crafts a helpful response in English and uses the same translation engine to send it back to you in clear, understandable Chinese. This intricate dance of translation happens bi-directionally, ensuring flawless communication no matter the language.

This translation magic opens doors to a world of possibilities, from global customer service to seamless multilingual communication (Hore, 2023). As technology advances, this bridge between languages will only become stronger, making the world truly a place where anyone can chat with anyone, anywhere and anytime in any country with the right correct language programmed into the algorithm.

Types of chatbot translation tools:

ChatGPT, Google Bard, Microsoft AI Co-pilot, Watsonx Assistant (IBM, n.d.)



Above is an illustration of how a chatbot translates english to chinese through a number of processes of using complex algorithms and knowledge base (trained dataset) to identify and translate the requested language to the user.

Methodology

The complete architecture of our proposed File Chatbot System is represented by the following three figures. The details concerning **Figures e.2** and **e.3** will be covered in depth in **Ryan Kho Yuen Thian's** and **Ong Weng Kai's** task allocations.

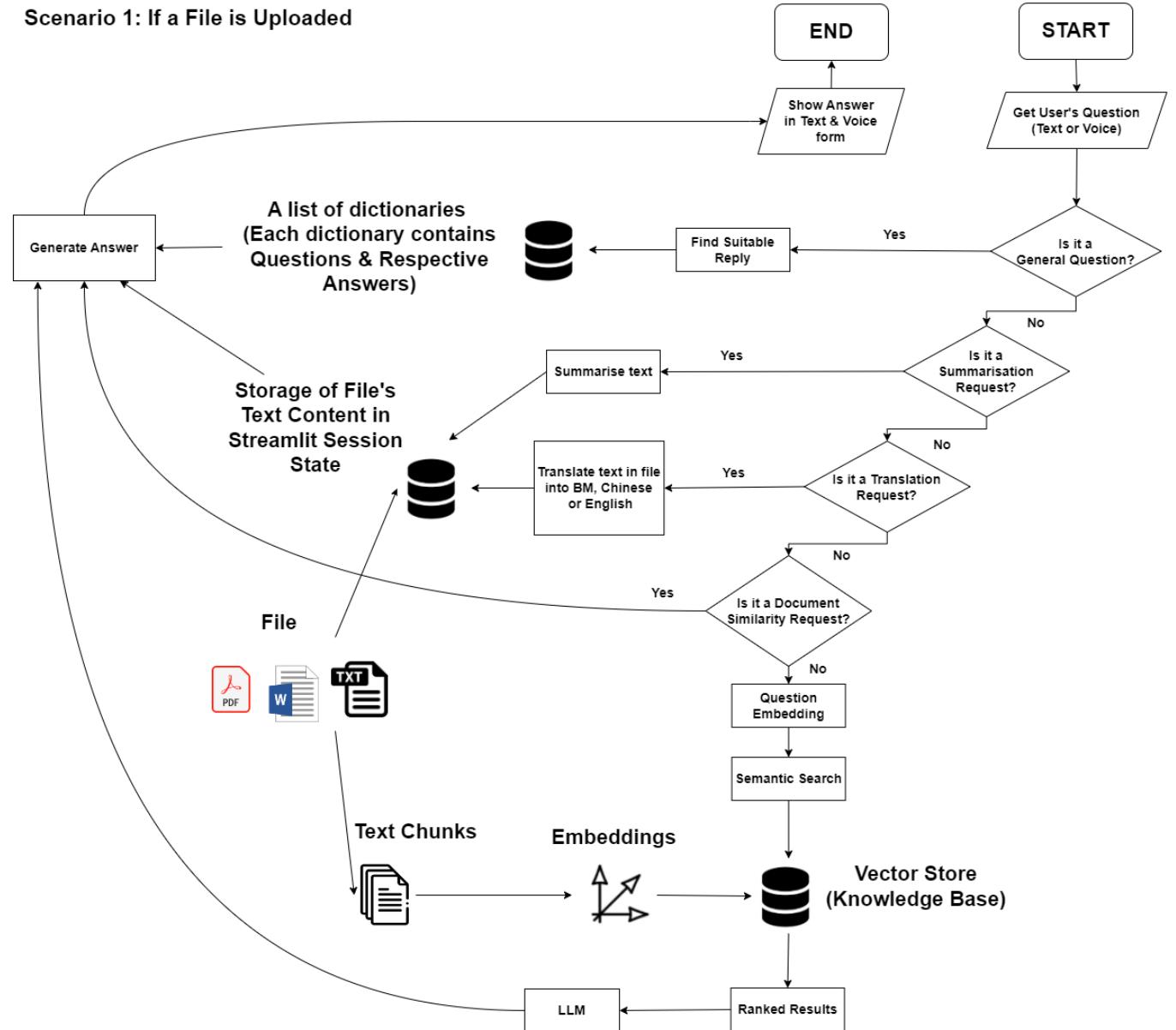


Figure e.1: If a File is Uploaded

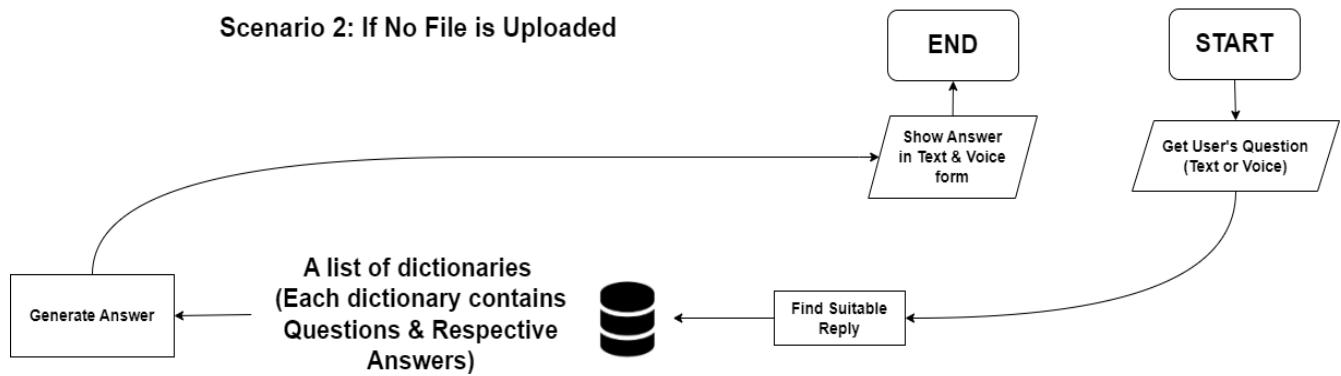


Figure e.2: If No File is Uploaded

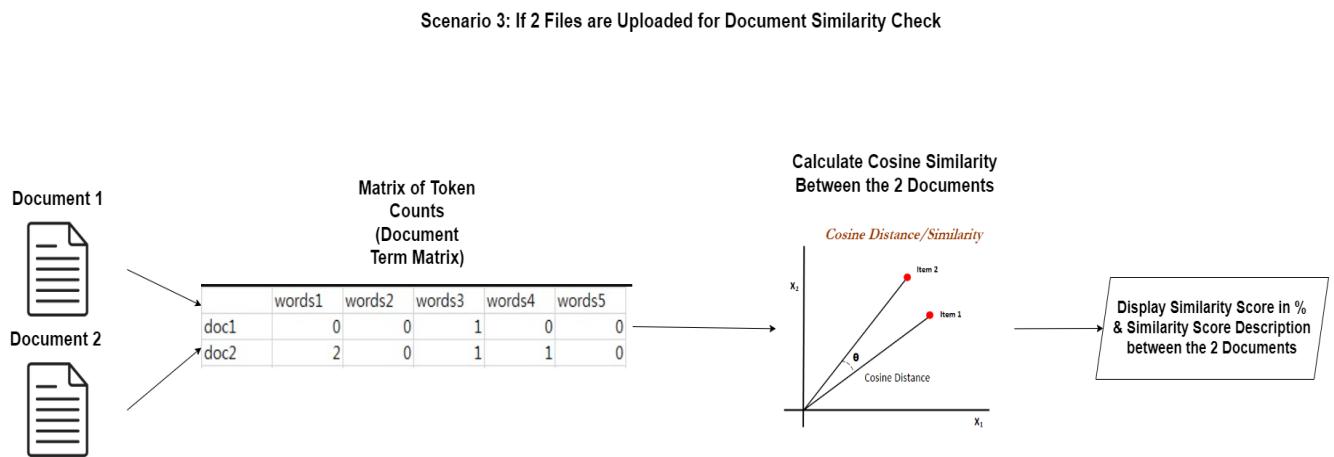


Figure e.3: If 2 Files are Uploaded for Document Similarity Check

Everyone

1. Answering File Related Questions

Referring to **Figure e.1**, for the File Chatbot to analyse and retrieve essential information from the uploaded files, six tasks must be completed to ensure its functionality.

- **Task 1: Retrieving Text from Various File Formats**

Before delving into the processing of the uploaded file's contents, we need to integrate a mechanism for extracting its text content. However, considering that the system accepts PDF, DOCX, or TXT files, the method for extracting text will vary depending on the file type.

a) To extract text from PDF

```
def getPdfText(pdfText):
    text = ''
    pdf_reader = PdfReader(pdfText)
    for page in pdf_reader.pages:
        text += page.extract_text()
    return text
```

b) To extract text from DOCX

```
def getWordText(docTxt):
    doc = Document(docTxt)
    text = "\n".join([paragraph.text for paragraph in doc.paragraphs])
    return text
```

c) To extract text from TXT

```
def getTextFile(txt):
    text = txt.getvalue().decode('utf-8')
    return text
```

The FileChatbot will only accept one file at a time from the user although we could have allowed the user to upload multiple files simultaneously. This deliberate choice serves to simplify the processing logic and enables the chatbot to concentrate on extracting pertinent information from a singular source. This streamlined approach not only enhances efficiency but also reduces the potential for complexity in user interactions. Users can provide specific questions or instructions related to the single file, promoting a more straightforward and focused dialogue. Managing one file at a time ensures resource efficiency and contributes to an improved user experience by ensuring a targeted and accurate analysis of each document.

- **Task 2: Splitting Text into Chunks**

In Natural Language Processing (NLP), the procedure known as '**Text chunking**' entails transforming unstructured text data into coherent units. This chunking process is valuable for identifying particular entities or phrases within the text, thereby improving the efficiency of information retrieval (Todeschini, 2023).

To achieve this, we need a **TextSplitter**, a tool designed to partition text into smaller fragments, typically aligning with semantic units like sentences. However, their intelligence extends beyond mere division; it involves a strategic combination of these fragments (Espíndola, 2023). The process unfolds in the following manner:

- The text is **initially divided** into compact fragments, each possessing semantic meaning, such as sentences.
- These smaller fragments are then **systematically combined** into larger units until a specific size is achieved, typically determined by some predefined function.
- Once the designated size is reached, the amalgamated fragment becomes an **independent unit of text**. Subsequently, a new text fragment begins, incorporating some degree of overlap to preserve contextual connections between the fragments.

We will be using `CharacterTextSplitter`, which allows the ability to specify a personalised delimiter for a more precise segmentation.

```
def getTextChunks(rawText):  
    textSplitter = CharacterTextSplitter(  
        separator = "\n",  
        chunk_size = 950,  
        chunk_overlap = 200,  
        length_function = len  
    )  
    txtchunks = textSplitter.split_text(rawText)  
    return txtchunks
```

- **Task 3: Building a Vector Store/Knowledge Base**

A **vector store** is a frequently employed method for storing and searching through unstructured data. It involves embedding the data and storing the resulting vectors. During query time, the unstructured query is also embedded, and the embedding vectors that are 'most similar' to the query are retrieved. The vector store manages the storage of embedded data and handles vector searches on your behalf (LangChain, n.d.).

For the **vector store**, we chose **FAISS**. This is because it is an open-source library designed for effective clustering of dense vectors and similarity search. It also possesses the capability to conduct searches on extensive vector sets that surpass RAM capacity. The library incorporates multiple techniques for similarity search, employing vector comparisons through L2 distances, dot products, and cosine similarity (Awan, 2023).

We chose the "**sentence-transformers/all-MiniLM-L6-v2**" model for its effectiveness in generating meaningful sentence **embeddings**, owing to its compact and efficient design. The proficiency of this model lies in converting sentences into embeddings rich in semantics, suitable for a range of NLP tasks, such as semantic search and clustering (mlflow, n.d.). Overall, with the utilisation of this model, we can guarantee users that the Chatbot application will efficiently generate embeddings, thereby minimising the wait time for the complete processing of the uploaded file.

```
...
def getVectorStore(textChunks):
    ...
    #model_id = 'sentence-transformers/all-MiniLM-L6-v2'
    #modelID = 'sentence-transformers/all-MiniLM-L6-v2'
    #modelID = 'distilbert-base-uncased-distilled-squad'
    #embeddings = HuggingFaceInstructEmbeddings(model_name="hkunlp/instructor-xl")
    #modelKwargs = {'device': 'cpu'}
    #embeddings = HuggingFaceEmbeddings(
    #    model_name='sentence-transformers/all-MiniLM-L6-v2', model_kwargs={'device': 'cpu'}
    #)
    vectorStore = FAISS.from_texts(texts=textChunks, embedding=embeddings)
    ...
    return vectorStore
...
```

- **Task 4: Building a Conversation Chain**

To ensure the FileChatbot delivers meaningful responses, the integration of a conversational AI model is crucial. In this endeavour, we opt for a model available on Hugging Face's model hub. Specifically, for the **LLM aspect**, the chosen model is **google/flan-t5-xxl**.

This is because **Flan-T5**, developed by Google researchers, is an open-source commercially available LLM. It represents a variation of the T5 (Text-To-Text Transfer Transformer) model, which stands out as a cutting-edge language model trained within a "text-to-text" framework (Dutta, 2023). OpenAI's LLMs, such as GPT-3.5, GPT-4, and GPT-4 Turbo, are not available for free use, despite their high level of sophistication.

For this LLM model, we have set the **temperature** to 0.5 for various reasons. A higher temperature like 1.0 produces more diverse and creative but potentially less focused and coherent text. On the other hand, a lower temperature like 0.5 makes the **output more deterministic and focused**, generating more conservative and precise text, which is what we want. Moreover, this parameter can only accept **positive integers, which means that the value cannot be 0**. As for the **max_length**, we set it to 512 because the model's maximum token size is **512**.

Last but not least, a **conversation chain** is needed as it is essential for storing and overseeing the conversation history and context. These chains play a crucial role in enabling the model to comprehend the ongoing conversation, facilitating the delivery of responses that are both coherent and contextually relevant.

```
def getConversationChain(vectorStore):
    ...
    llm = HuggingFaceHub(repo_id="google/flan-t5-xxl", model_kwargs={"temperature":0.5, "max_length":512})
    memory = ConversationBufferMemory(memory_key='chat_history', return_messages=True)
    conversationChain = ConversationalRetrievalChain.from_llm(
        ...
        llm=llm,
        ...
        retriever=vectorStore.as_retriever(),
        ...
        memory=memory
    )
    ...
    return conversationChain
...
```

- **Task 5: Handling User Input**

After the user has uploaded a file, the user can ask any question related to the file. This is because the FileChatbot will be using the vector store (knowledge base) and NLP models to generate the relevant answers.

However, there is a possibility that the user may ask questions that are out of topic or not understandable by the chatbot. Therefore, the **handleUserInput** function as seen below has been programmed to handle this kind of issue (will be covered in the individual task allocations).

```
def handleUserInput(userQues):  
    ques_ans = [  
        {'question': ["Hi", "Hi.", "Hi!", "Hello", "Hello.", "Hello!", "Hi there", "How are you", "I  
        'answer' : ["Hello!", "Good to see you again!", "Hi there, how can I help?"] },  
        {'question': ["Bye", "Bye.", "Bye!", "See you later!", "Goodbye!", "Nice chatting to
```

- **Task 6: Launching the Chatbot with Streamlit**

The initiation of the chatbot using Streamlit is a straightforward process. To simplify the usage of the chatbot, you can consult the **readme** file we've provided. The primary application required for launching the chatbot is Anaconda, a freely available and open-source application. You can utilise the Anaconda prompt by entering the specified command line "streamlit run file-path" to launch the chatbot successfully through your default web browser.

In the event of **errors**, they might arise due to the **absence of all the necessary packages** required for the chatbot to execute its programmed features. To address this, you can install the essential packages and libraries through the Anaconda prompt, ensuring that you have an active **internet connection** for the successful download of these packages. It's crucial to note that without an internet connection, the installation of packages or libraries to the Anaconda prompt will not be feasible. Ensuring the installation of all required packages or libraries is essential for the proper functioning of the chatbot. Even when the application is being run, it is important to ensure that your device has a stable internet connection since this system uses api.

RYAN KHO YUEN THIAN

1. Handling Spelling Mistakes or Typos (RYAN KHO YUEN THIAN)

If the user chooses to interact with the File Chatbot via text rather than voice, there is a possibility that the user may misspell some of the words, which could lead to inaccurate results being returned by the Chatbot. Therefore, I will be using the **autocorrect library**, which is one of the packages for creating a spell-checker in Python (Treadway, 2019). As the user will be interacting with the chatbot in English, I'll be creating a **Speller** object that uses the **English Language** from autocorrect's Speller class (Treadway, 2019). This object will be used for correcting misspellings if any (Treadway, 2019).

I will be creating a function called **correctSentence** to correct any misspellings in the user's query. Below is the general outline of what the function does:

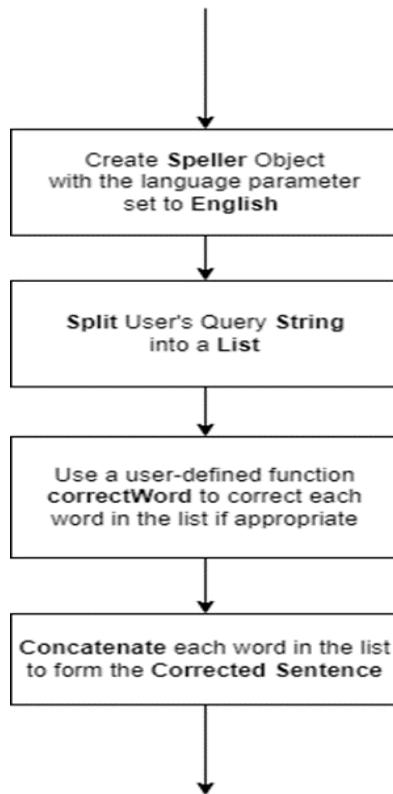


Figure R.1 Operations done by **correctSentence** function

Within the **correctSentence** function, it uses another function called **correctWord** to determine whether each word in the sentence needs to be corrected. This function is illustrated in Figure R.2.

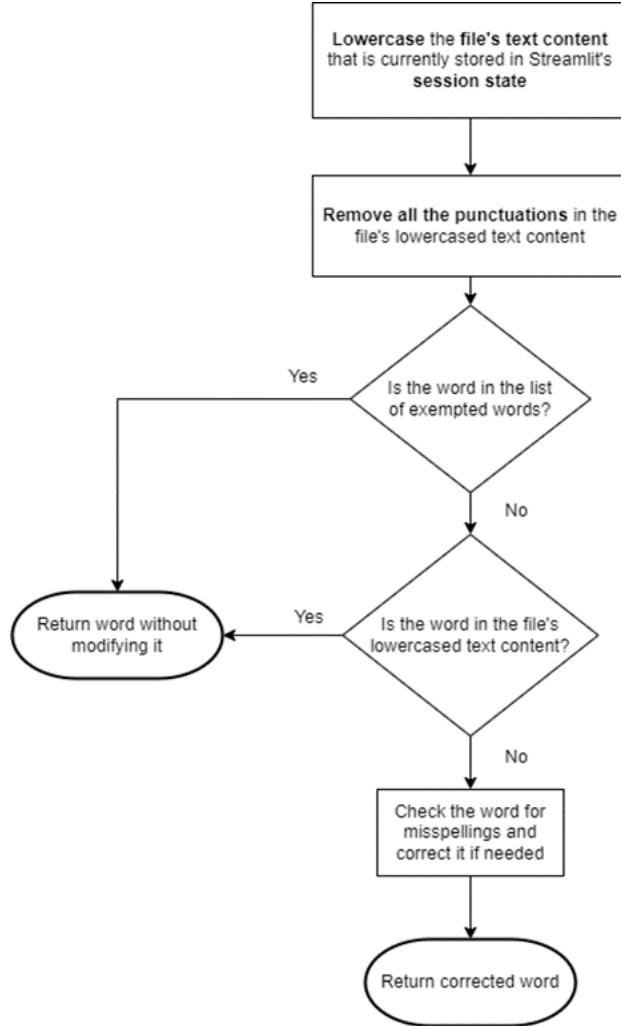


Figure R.2: Operations done by `correctWord` function

There are a few reasons why this function has several if-else statements. **Firstly**, if I straightaway use the function from the Speller class to autocorrect each word in the user's sentence, it may unintentionally modify one or some of the words in the sentence even if they are valid. For example, if the user has the word "chatbot" in the sentence, the function from the Speller class will change it to "chariot". However, if the user enters "chat bot", it will remain unmodified. Therefore, I had to create a list of words to be exempted from being corrected to prevent the user's valid words from being modified. **Secondly**, Speller may unintentionally autocorrect one of the user's words that exists in the uploaded file but may not be a Standard English word. For example, if a user uploads a fictional story, which is about a character called 'Groot', and asks the chatbot "Who is Groot?", Speller will autocorrect the sentence to "Who is Root?". Hence, I had to write an if-else statement to check whether the word in the user's query exists in the uploaded file. If the word is not in the exempted list of words and not in the uploaded file, I will allow Speller to check the word and autocorrect it if necessary.

The codes for the `correctSentence` and `correctWord` functions can be found in the figure on the next page. Please take note that `correctWord` is a function defined within the `correctSentence` function.

```
def correctSentence(userQues, exemptWords=['chatbot','chatbot.','chatbot?','chatbot!','chatbots','chatbot','cha
...     check = Speller(lang='en')
...     def correctWord(word):
...         punc = '''!()-[]{};:'"\,;<>./?@#$%^&*_~'''
...         modified_text = st.session_state.fileText.lower()
...
...         for element in modified_text:
...             if element in punc:
...                 modified_text = modified_text.replace(element, '')
...             if word.lower() in exemptWords:
...                 return word
...             elif re.sub(r'^\W+|\W+$', '', word).lower() in modified_text.split():
...                 return word
...             else:
...                 return check(word)
...     correctedWords = [correctWord(word) for word in userQues.split()]
...     correctedSentence = ''.join(correctedWords)
...     return correctedSentence
```

2. Document Similarity (RYAN KHO YUEN THIAN)

Assessing the similarity between documents is a frequent task in natural language processing (NLP). This process is essential in various microservices, including tasks such as information retrieval and translation, where it serves a crucial function (Nair, 2022). In order to compare the similarity between documents, I will be using the similarity metric Cosine Similarity and CountVectorizer from sklearn (explained in Research Background).

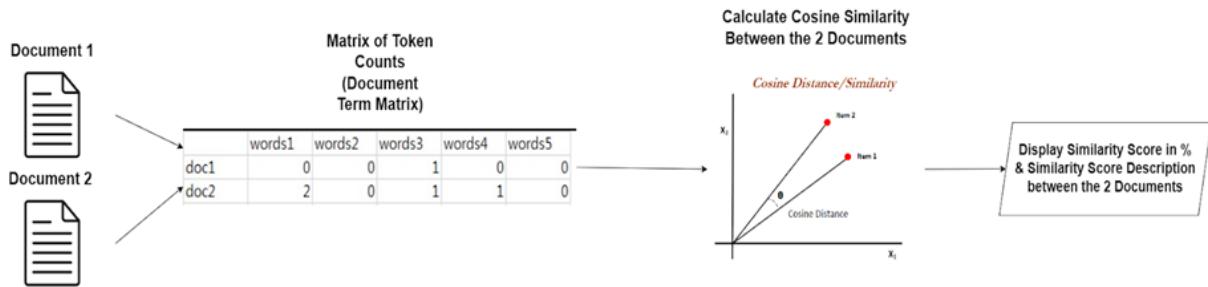


Figure R.3: The Process of Comparing Similarity between Documents

Referring to Figure R.3, after the user uploads two files, which could be either PDF, DOCX or TXT, their text contents will be extracted and passed into a function called **chkDocSimilarity** that takes 2 arguments, which are the 2 files. Within the function, their contents are placed in a list. Next, **CountVectorizer** is used to tokenise and count the words in the 2 files, creating a sparse matrix representation (Supe, 2023). After that, sklearn's cosine_similarity function is used to calculate the **cosine similarity** between the 2 files, which are represented by the sparse matrix, and the value of cosine similarity is returned in % form so that it is easily interpreted by the user.

The two figures show how this document similarity functionality is implemented. The first figure shows the definition of the function that is used to compute the document similarity while the second figure shows the function being used.

```
def chkDocSimilarity(file1, file2):
    docs = [file1, file2]
    countVectorizer = CountVectorizer()
    sparseMatrix = countVectorizer.fit_transform(docs)
    similarityScore = cosine_similarity(sparseMatrix, sparseMatrix)[0, 1]
    return similarityScore
```

```

    .with st.spinner("Processing"):
        #For 1st File
        if fileChoice1 == 'PDF':
            rawText1 = getPdfText(uploadedFile1)
        elif fileChoice1 == 'TXT':
            rawText1 = getTextFile(uploadedFile1)
        else:
            rawText1 = getWordText(uploadedFile1)

        #For 2nd File
        if fileChoice2 == 'PDF':
            rawText2 = getPdfText(uploadedFile2)
        elif fileChoice2 == 'TXT':
            rawText2 = getTextFile(uploadedFile2)
        else:
            rawText2 = getWordText(uploadedFile2)
        st.write(f"***Similarity score between the two files:*** {chkDocSimilarity(rawText1, rawText2):.2%}")
        pctVal = chkDocSimilarity(rawText1, rawText2)
        if pctVal > 0.8:
            st.write("**Verdict:** Very High Similarity 😊")
        elif pctVal > 0.6:
            st.write("**Verdict:** High Similarity 😃")
        elif pctVal > 0.4:
            st.write("**Verdict:** Moderate Similarity 😒")
        elif pctVal > 0.2:
            st.write("**Verdict:** Low Similarity 😐")
        else:
            st.write("**Verdict:** Very Low Similarity 😞")

```

3. Answering General Questions with/without Uploading a File (RYAN KHO YUEN THIAN)

Although it is important for the File Chatbot to be able to provide accurate answers to the user based on the uploaded files, it is also crucial to increase the user-friendliness of the system. Therefore, I will be integrating basic conversational capabilities into the chatbot, which gives it the ability to greet the user, say farewell, respond to words of gratitude and inform the users about the tasks that it can perform. This is achieved by considering 2 scenarios: whether a file is uploaded or not.

- **Scenario 1: No file is uploaded**

If no file is uploaded, the system will execute a user-defined function called **showGeneralResponse**. Within this function, there is an initialised list of dictionaries where each dictionary contains a list of questions and a list of the respective answers. To determine whether the user's question is similar to one of the questions, I will be using **cosine similarity** and **jaccard similarity**. While the system iterates over the predefined questions, if it finds a pair consisting of the user's question and the system's question that has a **cosine similarity** and **jaccard similarity** that are larger than or equal to the **largest cosine similarity and jaccard similarity** at that point in time, it will check whether both questions are **at least 60% similar** in terms of jaccard similarity. If yes, the system will stop searching and return an appropriate reply to the user. Otherwise, the system will continue iterating until it finds the most similar question or until all the questions have been checked.

However, do **take note** that right **before the cosine similarity** between two texts is calculated, I will perform some **text preprocessing**. The initial step involves **tokenising** and **lemmatising** the texts, **eliminating stopwords** and subsequently generating **TF-IDF vectors** for each text.

Ultimately, it computes the **cosine similarity** between the vectors using the `cosine_similarity` function from `sklearn.metrics.pairwise` (Otten, 2023).

To illustrate **why it is important to consider both similarity metrics**, imagine that there are 2 strings “What do you like?” and “What can you do?”. If we measure the cosine similarity between these 2 strings, it may indicate that they are quite similar even though we know that they are actually referring to different things. Therefore, to further verify whether they are actually similar in meaning, jaccard similarity is used to compute the agreement between them by dividing the size of the intersection of the 2 sets by their union size. If they are at least 60% similar, the system will consider them as similar in meaning. Otherwise, the system will not consider them as similar in meaning.

For the case where there is no similarity match between the user’s question and all of the system’s questions, the chatbot will apologise to the user, indicating that it lacks the answer to the user’s question, and provide a url, enabling the user to search for the answer on Google.

- **Scenario 2: A file is uploaded**

If the user did upload a file, the system will execute a user-defined function called `handleUserInput`, which has the following decision-making structure.

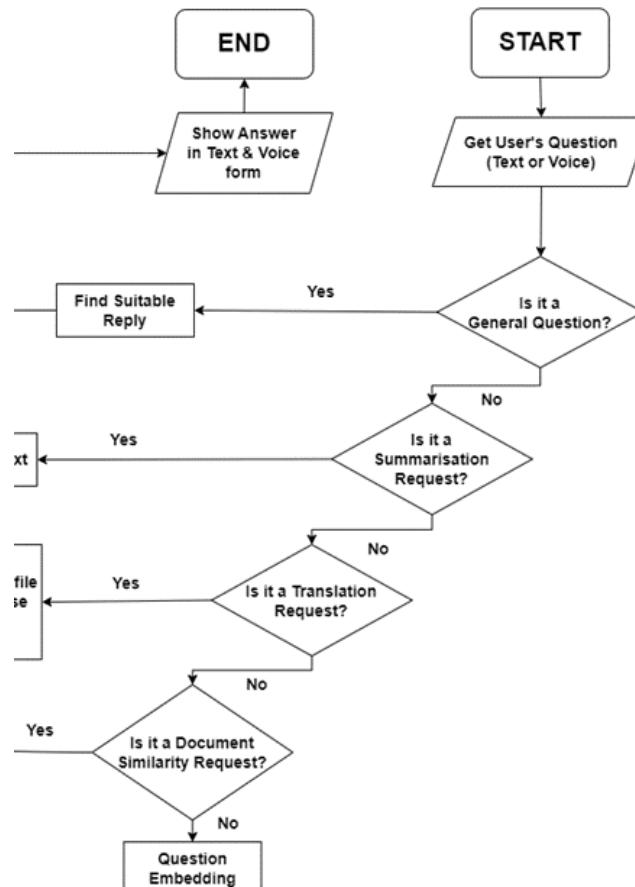


Figure R.4: The Decision-Making Structure of Handling User’s Question

Within this function, there is an initialised list of dictionaries where each dictionary contains a list of questions and a list of the respective answers. To determine whether the user's question is similar to one of the questions, I will be using **cosine similarity** and **jaccard similarity**. While the system iterates over the predefined questions, if it finds a pair consisting of the user's question and the system's question that has a **cosine similarity** and **jaccard similarity** that are larger than or equal to the **largest cosine similarity and jaccard similarity** at that point in time, it will check whether both questions are **at least 60% similar** in terms of jaccard similarity. If yes, the system will stop searching and return an appropriate reply to the user. Otherwise, the system will continue iterating until it finds the most similar question or until all the questions have been checked.

However, if all the system's questions have been checked, and they are not at all similar to the user's question, the system will subsequently examine whether the user's question falls into categories, such as a summarization request, a translation request, a document similarity request, a file-related question or an out-of-topic question (refer to Figure R.4).

For more information on how cosine similarity and jaccard similarity are calculated, please refer to **Figures R.5 and R.6**. For more information on how these 2 functions interact with each other, please refer to **Figure R.7**.

```
def cosineSimilarity(userQues, query):
    ....tokens1 = word_tokenize(userQues)
    ....tokens2 = word_tokenize(query)
    ....lemmatizer = WordNetLemmatizer()
    ....tokens1 = [lemmatizer.lemmatize(token) for token in tokens1]
    ....tokens2 = [lemmatizer.lemmatize(token) for token in tokens2]

    ....# Remove stopwords
    ....stop_words = stopwords.words('english')
    ....tokens1 = [token for token in tokens1 if token not in stop_words]
    ....tokens2 = [token for token in tokens2 if token not in stop_words]
    ....
    ....tfid_vectorizer = TfidfVectorizer()
    ....vector1 = tfid_vectorizer.fit_transform(tokens1)
    ....vector2 = tfid_vectorizer.transform(tokens2)
    ....similarityBetweenTexts = cosine_similarity(vector1, vector2)
    ....return similarityBetweenTexts[0,0]
```

Figure R.5: Code for Calculating Cosine Similarity Between 2 Texts

```

def jaccardSimilarity(set_1, set_2):
    ... intersection = len(set_1.intersection(set_2))
    ... union = len(set_1.union(set_2))
    ... return intersection / union if union != 0 else 0

def getTokens(text):
    ... # Simple tokenization by splitting on spaces
    ... return set(text.lower().split())

def determineRelationship(text_1, text_2, threshold=0.6):
    ... tokens_1 = getTokens(text_1)
    ... tokens_2 = getTokens(text_2)

    ... similarity = jaccardSimilarity(tokens_1, tokens_2)

    ... if similarity >= threshold:
        ... return True
    ... else:
        ... return False

```

Figure R.6: How Jaccard Similarity is calculated and how it determines whether the 2 texts are similar to each other.

```

found = False
largest_index = 0
largest_cosine_similarity = 0
largest_jaccard_similarity = 0
list_ind = 0

for qalist in ques_ans:
    for key, value in qalist.items():
        if key == 'question':
            for item in value:
                current = item
                if cosineSimilarity(userQues, current) >= largest_cosine_similarity and jaccardSimilarity(getTokens(userQues), getTokens(current)) >= largest_jaccard_similarity and found == False:
                    found = determineRelationship(userQues, current)
                    largest_index = list_ind
                    largest_cosine_similarity = cosineSimilarity(userQues, current)
                    largest_jaccard_similarity = jaccardSimilarity(getTokens(userQues), getTokens(current))
                    if found:
                        break
    list_ind += 1

```

Figure R.7: Code showing how the functions interact with each other

THONG CHENG HOW

1. Text to speech (THONG CHENG HOW)

```
def text_to_speech(text, language='en'):

    # Specify the language for gTTS
    if language == "en":
        lang = 'en'
    elif language == "ms":
        lang = 'ms' # Malay
    elif language == "zh-TW":
        lang = 'zh-TW' # Traditional Chinese
    else:
        lang = 'en' # Default to English if the language is not recognized

    # Create a gTTS object
    tts = gTTS(text=text, lang=lang, slow=False)
```

Everytime the chatbot gives a response to the user, the **audio file will automatically** be generated for the user to click on to listen. Based on the language of the text generated by the chatbot, the chatbot will follow based on the **if-else rules** to correspond to the audio that the chatbot should play to the user. As for the languages used, the chatbot has only **three languages** which are: English, Bahasa Melayu and Taiwan, Chinese. I will be using the library from **Google-Text-To-Speech library**. Although currently in the market there are many different text-to-speech libraries, gTTS suits our programmed chatbot the best.

This is because gTTS is a python library and CLI tool to interface with Google's Translate's text-to-speech (gTTS) API (python, 2023). gTTS is also a well known software that converts written text to spoken words generated by chatbots (Cahyo, 2022). **Google's text-to-speech algorithm** allows multiple different languages to be used for playing the audio file at once. Some features that stood out to me is the ability to customise speech-specific sentences to be tokenized for unlimited length of text to speech that can be played (python, 2023). This means no matter how long some sentences generated by the chatbot, the audio file can be played by users. With this, the user using our code must install the library package of **pip install gTTS**, in order to initiate the features of gTTS (python, 2023).

Another main feature that stood out is the ability to **switch** between **different languages**, for example English to Chinese and vice versa (python, 2023). Not only that, gTTS is also capable of pronunciation corrections (python, 2023). Previously we used pyttsx3 as our main library to play the audio files to the user. Although the main advantage of pyttsx3 library is the ability to use the audio files offline, we faced Many challenges and obstacles to create the audio file to perform the result, such as the difficulty to implement different languages to the audio files, as the code will be complicated and result in large amounts of errors to solve. Another challenge is the complexity of understanding the code by implementing different types. The way to implement pyttsx3 is also very different from gTTS.

Pyttsx3 implementation

```
import pyttsx3

# Initialize the TTS engine
engine = pyttsx3.init()

# Set the rate at which the text should be spoken
rate = engine.getProperty('rate')
engine.setProperty('rate', rate-50)

# Set the volume of the spoken text
volume = engine.getProperty('volume')
engine.setProperty('volume', volume+0.25)

# Set the voice to use
voices = engine.getProperty('voices')
engine.setProperty('voice', voices[1].id)

# Set the text to be spoken
text = "Hello, this is a test of the text-to-speech system using pyttsx3."

# Save the spoken text to an audio file
engine.save_to_file(text, 'output.mp3')

# Disconnect the TTS engine
engine.stop()
```

gTTS implementation

```
from gtts import gTTS
from playsound import playsound

# Set the text to be spoken
text = "Hello, this is a test of the text-to-speech system using gTTS."

# Create the TTS object
tts = gTTS(text)

# Save the audio file
tts.save("hello.mp3")

# Play the audio file
playsound("hello.mp3")
```

By comparing the two codes, the obvious choice was to choose from gTTS instead of pyttsx3 because of its easier way of implementation and the structure of code and it is more efficient to use. Google's Text-to-Speech just needs to create a **text-to-speech object** and save the audio file to the user's device and the voice is already to be played. As for pyttsx3, the user needs to **set the voice, initialise the engine, save the file to the user's device and stop the audio** once the audio file has successfully been played.

Diagram to illustrate how does text to speech operate in the chatbot

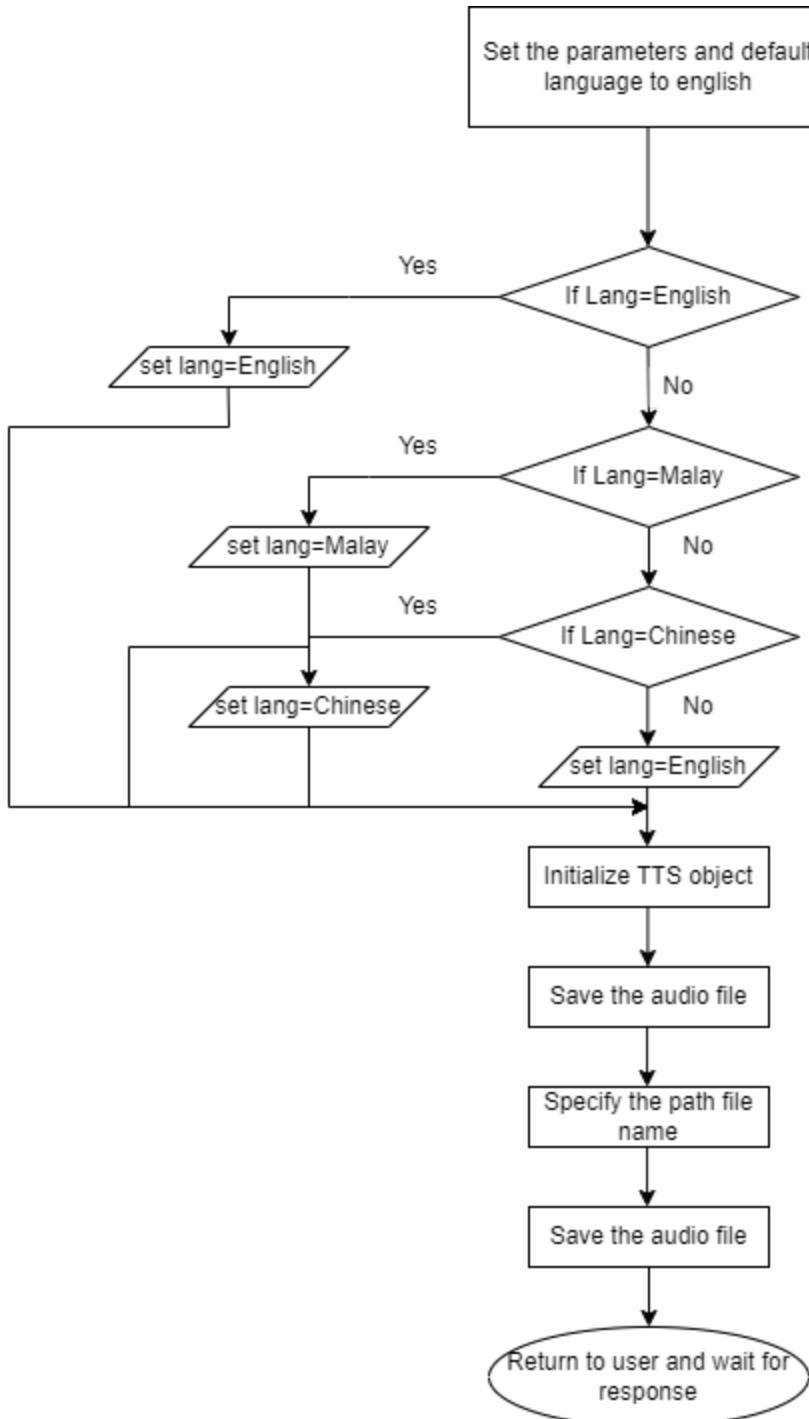


Figure 1: Process of text-to-speech

From the start, the context of the chatbot will **generate** the response of the specific language. gTTS API will determine the type of language from the if else function created in the code. Once the language has been identified, the TTS object will be generated. The audio file will be processed and the file path specifies where the file will be **saved** on the **individual's computer**. The audio file will be processed by the chatbot, and the user can either select to play the audio file or not to play the audio file.

2. Translates text into different languages (THONG CHENG HOW)

```
def translateText(userQues):
    malay_keywords = ["malay", "translate to malay", "into malay", "to bahasa melayu", "to bahasa malaysia", "melayu"]
    mandarin_keywords = ["mandarin", "translate to mandarin", "into mandarin", "to chinese", "to mandarin", "translate to chinese"]
    english_keywords = ["english", "translate to english", "into english", "to english", "english translation", "english version", "english text"]

    userQuesLower = userQues.lower()

    if any(keyword in userQuesLower for keyword in malay_keywords):
        return translateToMMText(st.session_state.fileText)
    elif any(keyword in userQuesLower for keyword in mandarin_keywords):
        return translateToMandarin(st.session_state.fileText)
    elif any(keyword in userQuesLower for keyword in english_keywords):
        return translateToEnglish(st.session_state.fileText)
    else:
```

Translation can be done if the user wants to translate the contents of the file to another different language. Some phrases that the chatbot understands as translation requests are “**Translate file to chinese**” and “**Translate file to malay**”. The chatbot will automatically translate the contents to the requested language that the user wants to. However, to do so, the user needs to install the **recommended package** in order for the chatbot to translate the contents to the correct format. For the translation to work smoothly, packages need to be installed, which is the google translator, which translates the content to the required language.

Translator	Supported Language Count	Advantage
Google	108	support the most languages in the world
Yandex	99	support more languages in the world, support word to emoji, unstable
Bing	77	support more languages in the world
Sogou	61	support more languages in the world
Baidu	28	support more languages, support professional field
Tencent	17	support more languages
Youdao	14	support more languages
Alibaba	12	support more languages, support professional field
DeepL	24	high quality to translate but response slowly, unstable
Caiyun	6	high quality to translate but response slowly, support professional field

Figure 2.1: Language supported by each API

For our chatbot, we chose the **google translator API**. This is because google translator has the highest supported language count compared to other language translators. Although Google translator **cannot support emoji**, it does not affect our performance and doesn't really impact the usability of the chatbot.

How does google translator work ?

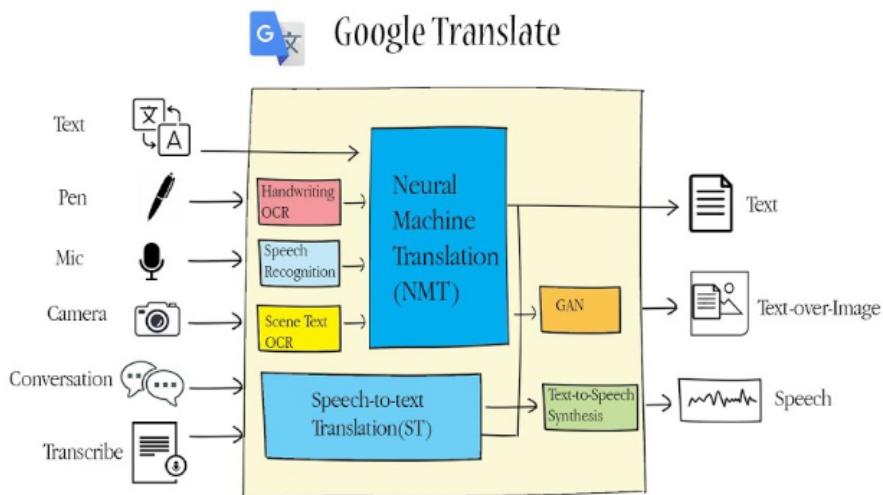


Figure 2.2: How Google Translator works ?

From the above diagram, this is the general way of how google translates the content to the requested format. The main engine translating the contents is the **Neural Machine Translation (NMT)**. NMT is a deep learning algorithm that uses the state-of-the-art machine learning tools to predict the likelihood of a sentence (Wiggins, 2023). NMT is trained partially before analysing the data, therefore the performance test is better. The traditional translation algorithm is using **Statistical Machine Translation (SMT)**. However, NMT is far more capable and more accurate in terms of word by word (Wiggins, 2023)

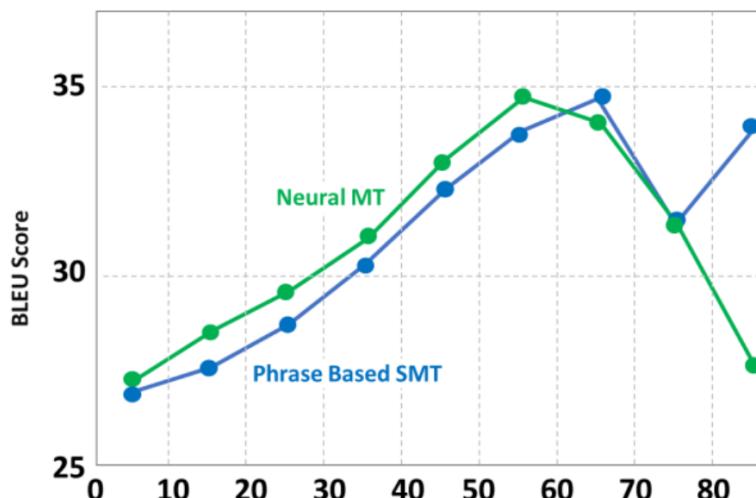


Figure 2.3: Performance of NMT vs SMT

As for NMT, 6/8 of points have higher quality of translation in terms of 10 to 60 subwords. Only 70 to 80 subwords are in the favour of SMT.

How our chatbot does the translation

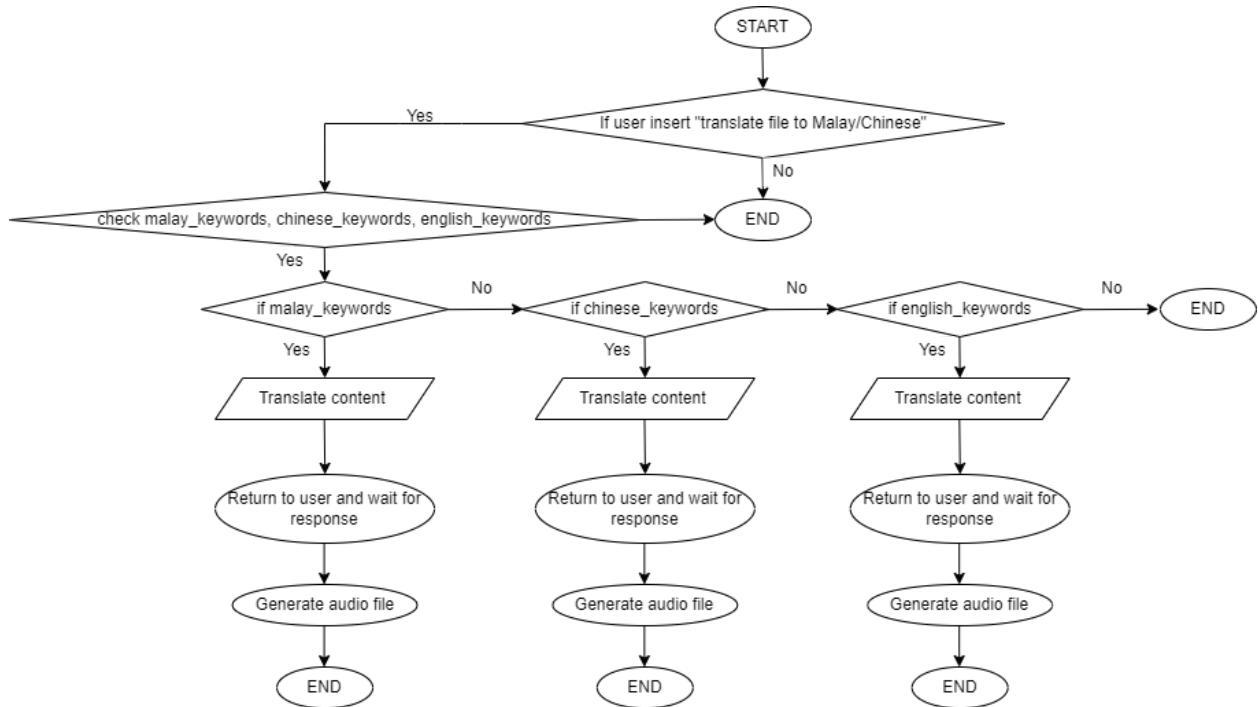


Figure 2.4: Illustration of google translation on chatbot

Based on the illustration above, if the user has entered the key words of translate, file and specify the specific language, the translator algorithm by google will translate the content in the file to the specific language. Audio file will then be generated for the chatbot response.

3. Summarisation (THONG CHENG HOW)

```
def summarize_text(text, max_chunk_length=700, max_summary_length=130):
    # Initialize the summarization pipeline
    summarizer = pipeline("summarization", model="facebook/bart-large-cnn")
```

If there is any file uploaded, the user can ask the chatbot to summarise the contents in the file.

As the text content from a file can be very lengthy, the `summarize_text` function will first split the text into chunks with a certain size. After that, for each chunk of text, I will use a summarizer model called `facebook/bart-large-cnn` to summarise it. After every `chunk` is summarised, the chunks are then concatenated together to form the complete summary of the file content (Chanda, 2023).

Illustrations of the process of summarisation

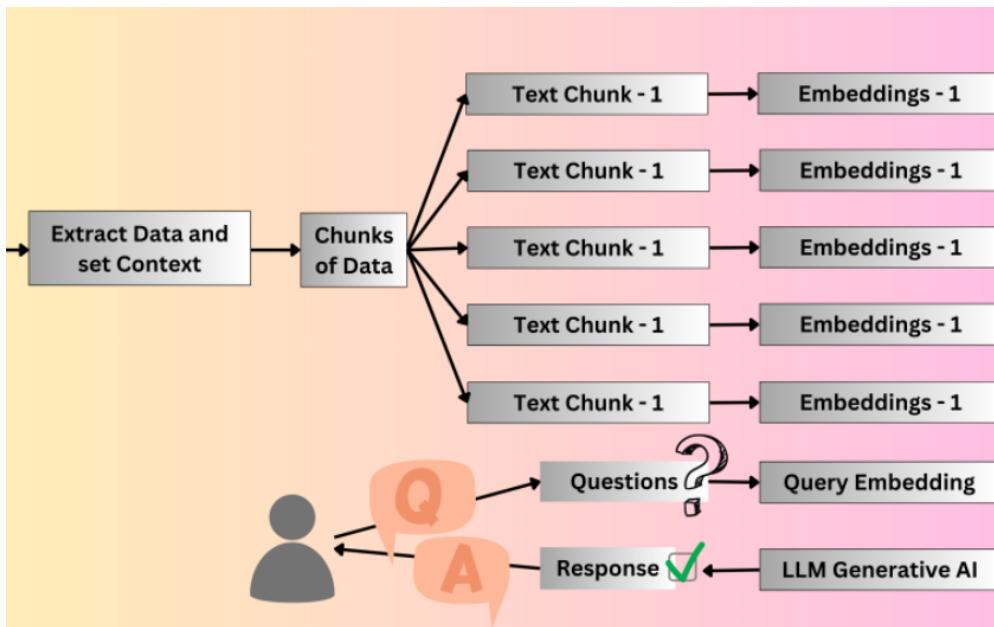


Figure 3.1: Illustration of the Process of Summarization

```
def isSummarizationRequest(userQues):
    summarization_keywords = ["summarize", "summarise", "summary", "brief", "overview"]
    context_keywords = ["file", "document", "text"]
    question_forms = ["can you", "please", "could you", "would you"]
    user_input_lower = userQues.lower()
    contains_summarizationKey = any(keyword in user_input_lower for keyword in summarization_keywords)
    contains_contextKey = any(keyword in user_input_lower for keyword in context_keywords)
    contains_questionForm = any(form in user_input_lower for form in question_forms)
    contains_addRule = "file summary" in user_input_lower or "provide an overview" in user_input_lower
    return (
        (contains_summarizationKey and contains_contextKey) or
        (contains_questionForm and contains_addRule) or contains_summarizationKey
    )
```

Figure 3.2: Code used to summarise the file's text content.

Figures 3.1 and 3.2 show the process of how the File Chatbot summarises the contents of the uploaded file.

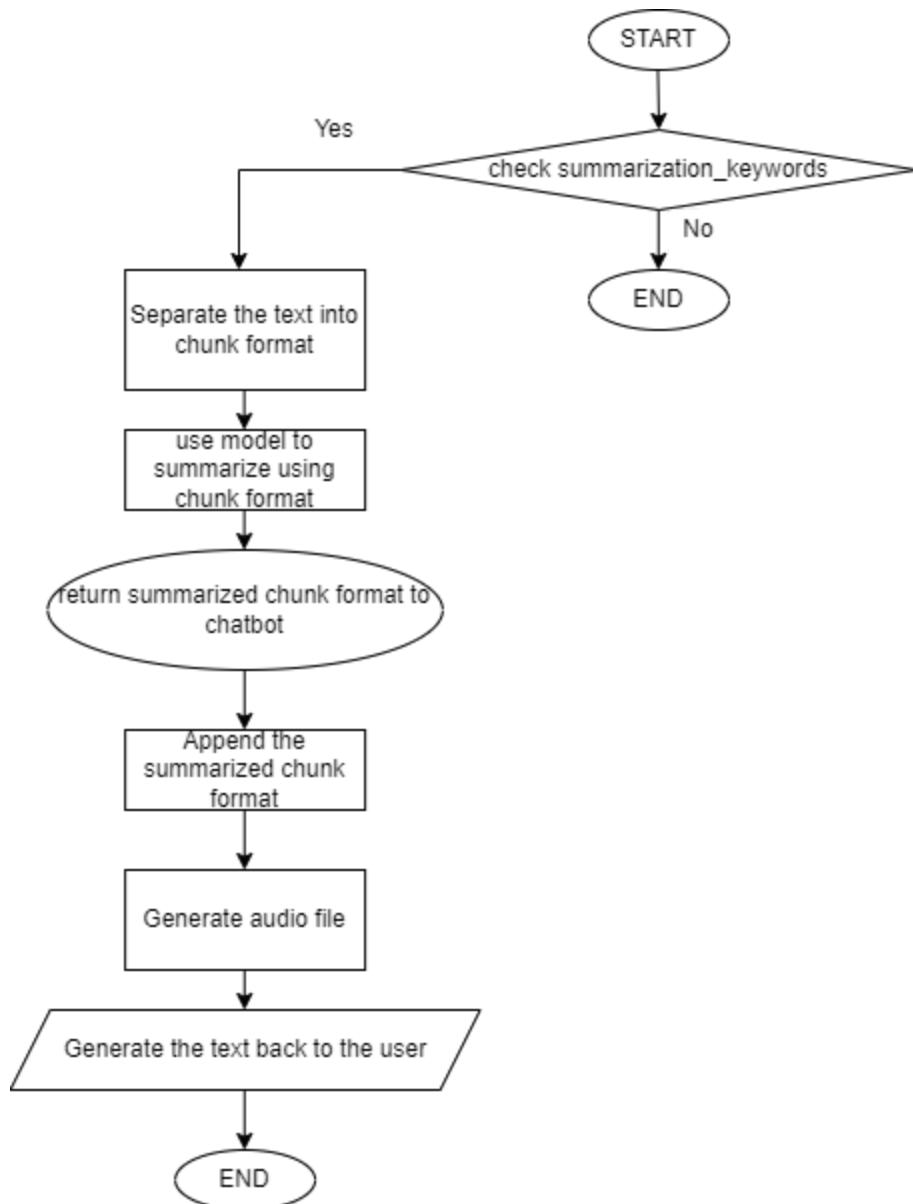


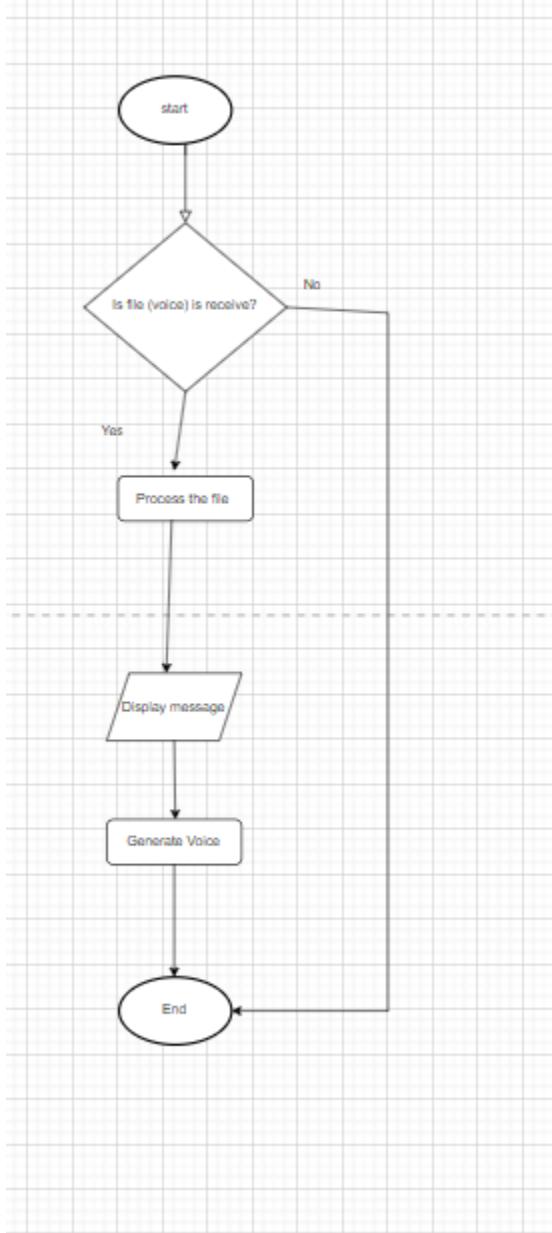
Figure 3.3: Flowchart illustration

Referring to Figure 3.3, the user must enter keywords that indicate that he/she wants the Chatbot to summarise the file's content. So, if it is a summarisation request, this will initiate the summarise function. The common steps for the chatbot to summarise the file's content are separating the content into chunks, using a summarisation model to summarise each chunk content and appending the contents of each summarised content to form the fully summarised content that will be returned to the user.

ONG WENG KAI

1. Speech to Text (ONG WENG KAI)

The user can choose to use the voice text feature, where the user can click the “Start Recording” button located on the left sidebar. If the button is clicked the file(voice) will be processed and transcribed into text and displayed. So, the chatbot can generate responses based on that.



```
speech_to_text(start_prompt="Start recording ⏹", stop_prompt="Stop recording ⏹", language='en', use_container_width=False, just_once=True, key='STT')
```

The user uses the recording button at the sidebar to record their voice. The "speech_to_text" function from the `streamlit_mic_recorder` library allows the Streamlit app to capture and transcribe voice input from the user. The parameter "`language='en'`", specifies the language setting for the speech recognition. '`en`' stands for English, indicating that the speech-to-text function should expect and process English

speech. Thus, it can improve the accuracy in speech recognition as it informs the system about the phonetics and syntax to anticipate. "**just_once=True**" is set to ensure it will only run once per trigger. It will not continuously listen and convert speech to text but will do so for a single instance of speech until stopped. "**use_container_width=False**" is used to control the design layout, and "**key='STT'**" is an identifier for "Speech To Text"

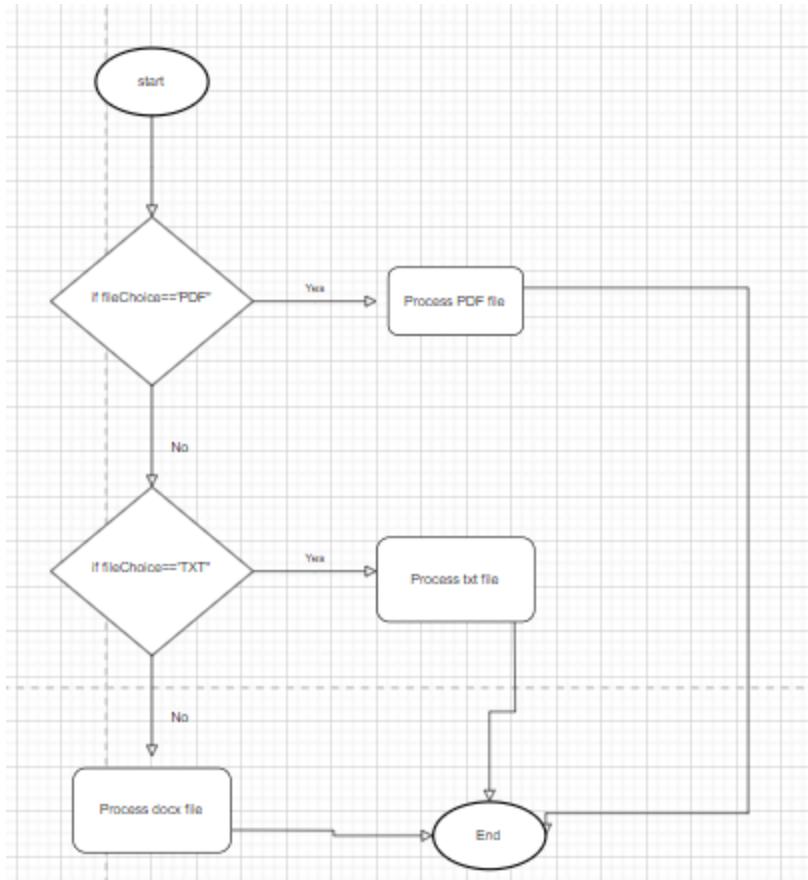
```
if userVoice:
    userVoiceText = userVoice[0].upper() + userVoice[1:]
    st.session_state.text_received.append(userVoiceText)
    st.session_state.messages.append({"role": "user", "content": userVoiceText})
    with st.chat_message("user", avatar="https://cdn-icons-png.flaticon.com/512/3177/3177440.png"):
        st.markdown(userVoiceText)

with st.chat_message("assistant", avatar="https://cdn-icons-png.flaticon.com/512/8943/8943377.png"):
    if uploadedFile:
        language, ans = handleUserInput(userVoiceText)
    else:
        ans, ind = showGeneralResponse(userVoiceText)
        addon = random.choice(
            [
                "👉 Please upload a file to start our conversation. Thank you!",
                "Please let me have a look at your file! Can't wait to read it 😊",
                "👉 Please upload your file on the left so that I can answer your questions! ",
            ]
        )
        if ind != 1:
            ans += ' ' + addon
    st.session_state.messages.append({"role": "assistant", "content": ans})
    displayChatBotMsg(ans, language)
```

First, it will check if there is a user voice if the voice is received, then capitalise the first letter. The processed text is then appended to a session state variable that keeps track of received text (**st.session_state.text_received**). The text stored in **st.session_state** allows information to be persistent between app reruns. The "**handleUserInput**" function will be used to analyse the content based on the file uploaded by the user (Voice) so the chatbot can later give an appropriate response. If the request received from the user is a general question, then it will give a random response to the user to upload a file and encourage or prompt interaction. The message between the user and chatbot can be displayed through "**st.chat_message**". Lastly, the session state keeps the message exchanged in the conversation.

2. Accept PDF/DOCX/TXT (ONG WENG KAI)

The user has to select the file type from the dropdown menu; if the file chosen by the User is PDF, then the file that can be processed is only PDF. If the user chooses TXT, then it will process the TXT file. If the user chooses Word/DOCX, only that particular file type will be processed by the Chatbot. The mechanic behind it is using simple if-else statements. It first checks whether the file type is "PDF". If Yes, then the file will be processed as PDF. If not, it checks whether it is a "TXT" file type; if yes, the file will be processed as a "TXT" file, else the file will be treated as a "DOCX" file. The details of the explanation will be covered in the following sections.



```

st.sidebar.subheader("1. Choose A File for Queries:")
complete = False
menu = ["PDF", "TXT", "DOCX"]
fileChoice = st.sidebar.selectbox("i) Choose file type", menu)

if fileChoice == "PDF":
    uploadedFile = st.sidebar.file_uploader("ii) Upload your file here", type="pdf", accept_multiple_files=False, key=1)

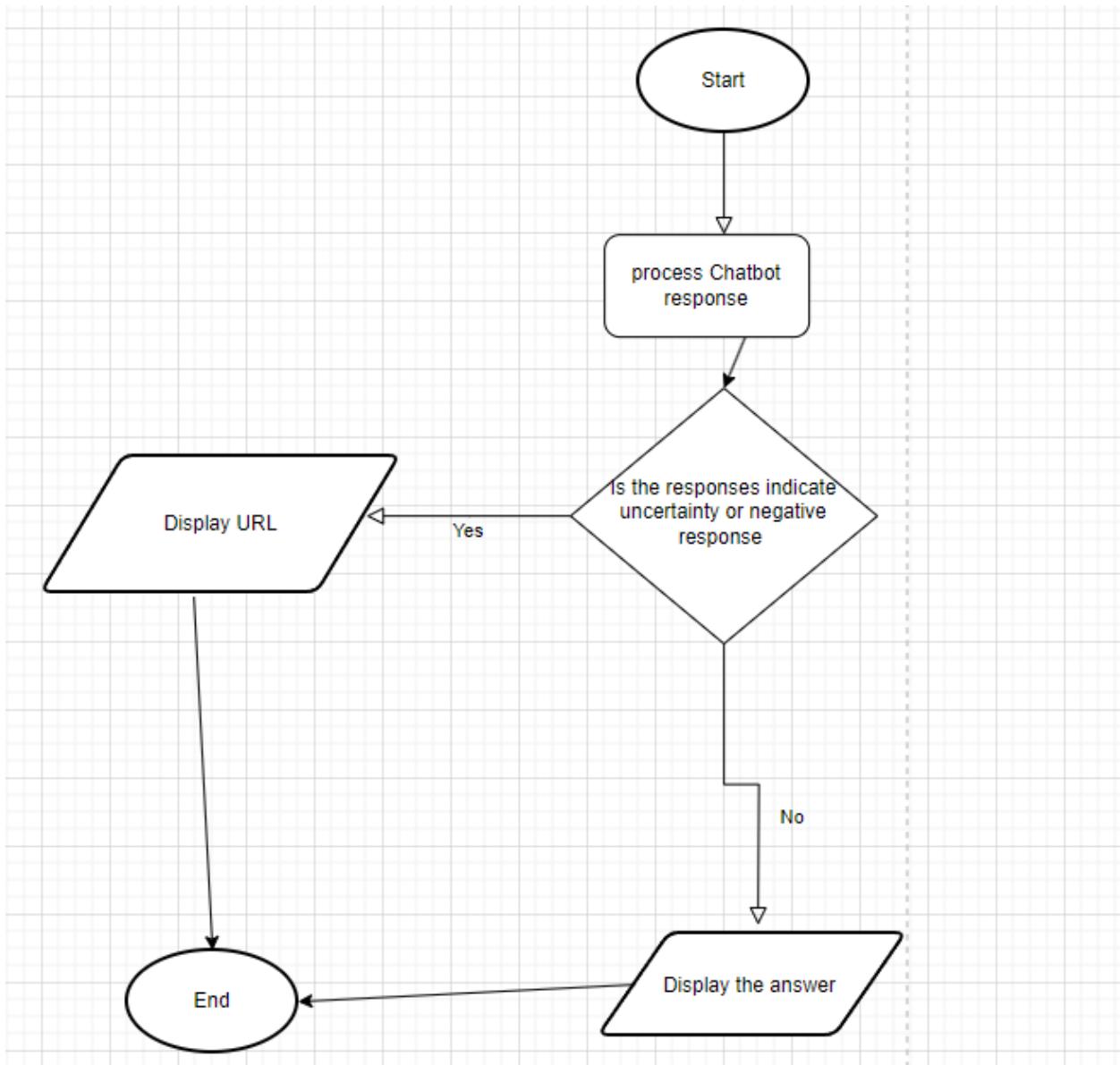
elif fileChoice == "TXT":
    uploadedFile = st.sidebar.file_uploader("ii) Upload your file here", type="txt", accept_multiple_files=False, key=1)
else:
    uploadedFile = st.sidebar.file_uploader("ii) Upload your file here", type="docx", accept_multiple_files=False, key=1)

```

First, the users can select the file type they wish to process, and then the file they upload will be selected and processed. The user can upload the file in a sidebar of the Streamlit app with a subheader of "1. Choose A File for Queries:". The menu list stores all the file types that can be processed. The "st.sidebar.selectbox" creates a dropdown menu in the sidebar. Users can select one option in the menu list to choose the file type they wish to process. After the user selects the file and the file is uploaded, the system will process the file through "st.sidebar.file_uploader" and the "accept_multiple_files=False" ensures only one file can be uploaded at a time.

3. Handling unrelated question (ONG WENG KAI)

The graph below illustrates the basic idea of how “handling unrelated question” function works; first, after receiving the text from the user, it will process and then get the chatbot response. After getting the chatbot's answer, the function will evaluate whether the question answered shows high uncertainty or a negative response. If yes, then it will give the user a Google link together with the user query. Otherwise, it will display the chatbot's answer as it does not show high uncertainty. The details of the explanation will be covered in the following sections.



```

try:
    response = st.session_state.conversation({'question': userQues})
    st.session_state.chat_history = response['chat_history']
    #userVoiceText = userVoice[0].upper() + userVoice[1:]
    #new code
    ans = response['answer'][0].upper() + response['answer'][1:]
    really_dk = False
    if ans.lower() == "i don't know," or ans.lower() == "i don't know." or ans.lower() == "i don't know" or ans.lower() == "i do not kno
        really_dk = True

```

It will first process the user's question. The script processes a user's question (`userQues`) and obtains a response from a chat model (`response = st.session_state.conversation({'question': userQues})`). After that, it will check if the chatbot's response indicates uncertainty or a negative response (e.g., "I don't know," "I do not know," "No," etc.). If the response is one of these, it sets a flag (`really_dk = True`). So the following function will be executed, creating a link to a Google search of the user's question. The "`ans`" variable is used to capitalise the first letter of the response.

```

encoded_query = urllib.parse.quote_plus(userQues)
google_search_url = f"https://www.google.com/search?q={encoded_query}"
ans += " I apologise 😊."
ans += f" [Search on Google]({google_search_url})"

```

The function `urllib.parse.quote_plus` in the Python standard library allows a string to be safely encoded as part of a URL, which is necessary because URLs can only contain certain characters. The `userQues` represents the user's question. So this function allows the user to place the user's question as part of the URL link, and the function is used to convert characters into a format that can be transmitted over the Internet without causing errors.

After that, it will construct a URL for a Google search. The base URL for a Google search is `https://www.google.com/search?q=`, and this is concatenated with the encoded query. Creating a direct URL to a Google search with the user's query will provide a seamless way for users to continue their research on Google directly.

Results

a) User Interface

The following two images show how the application looks like to the user when it first loads. On the left sidebar, there are 3 headings, providing instructions and options on how the user can interact with the File Chatbot.

The image displays two screenshots of the FileChatty application interface. Both screenshots show a sidebar on the left and a main chat area on the right.

Screenshot 1 (Top): Welcome to FileChatty

- Sidebar (Left):**
 - Welcome to FileChatty 🤖**
 - 1. Choose A File for Queries:**
 - i) Choose file type: PDF
 - ii) Upload your file here:
 - Drag and drop file here (Limit 200MB per file • PDF)
 - Browse files
 - 2. Interact with FileChatty via Text or Voice!**
 - i) If text, please type your message in the chatbox on the right.
 - ii) If voice, please click the button below.
Start recording
 - 3. Document Similarity**

Main Area (Right): Chat about Your Uploaded File

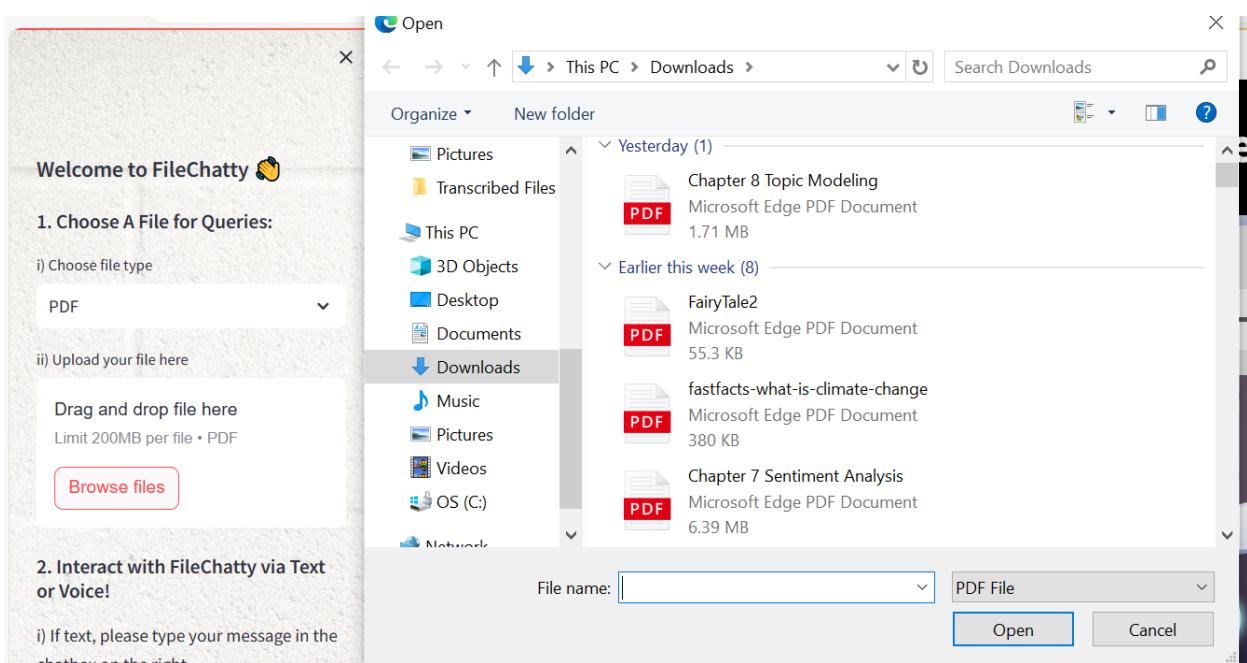
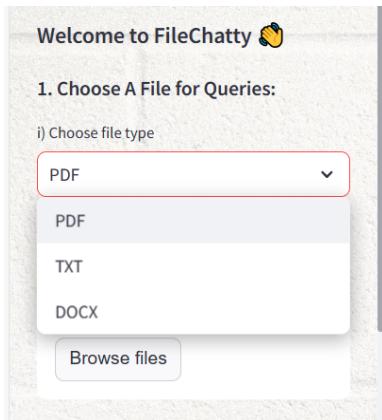
 - A friendly white robot with blue eyes and a speech bubble is the central figure.
 - A message from the bot: "Hi, I'm FileChatty 😊. Let's chat via text or voice!"
 - A progress bar shows 0:00 / 0:06.
 - A text input field: "Chat with FileChatty" with a send button.

Screenshot 2 (Bottom): Document Similarity

 - Sidebar (Left):**
 - 3. Document Similarity**
 - i) Upload 2 documents**
 - Choose file type for File 1: PDF
 - Upload your File 1:
 - Drag and drop file here (Limit 200MB per file • PDF)
 - Browse files
 - Choose file type for File 2: PDF
 - Upload your File 2:
 - Drag and drop file here (Limit 200MB per file • PDF)
 - Browse files
 - ii) Click on 'Process Files'**
 - Main Area (Right): Chat about Your Uploaded File**

b) Accepting Multiple File Types (PDF, DOCX or TXT)

The two figures below illustrate the File Chatbot's ability to accept either PDF, TXT or DOCX. If the user selects 1 particular file type in the drop-down list, the user's choice of file to be uploaded will only be restricted to files of the selected file type.



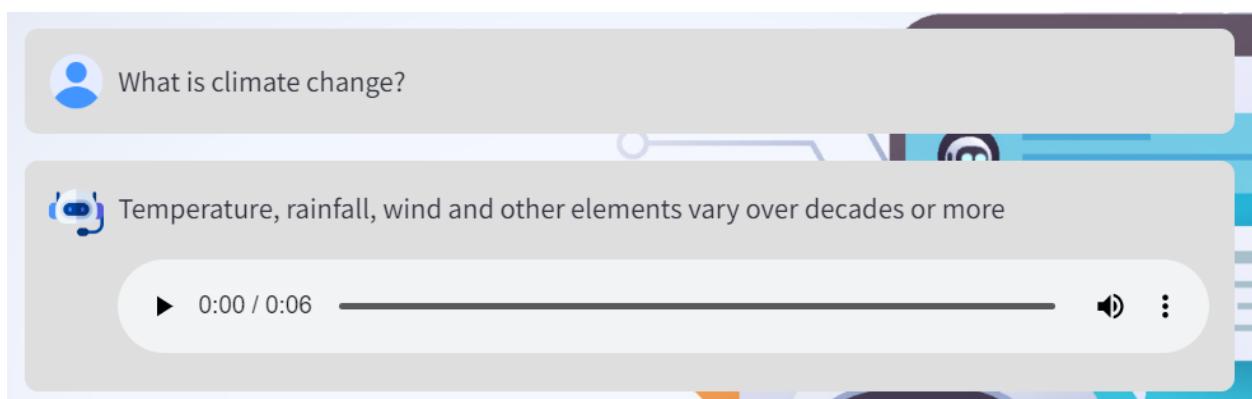
c) Validation of Responses to File-Related Questions (includes summarisation & translation)

Given that the user uploads a file called ‘fastfacts-what-is-climate-change.pdf’, below are the set of questions, expected answers and actual answers from the File Chatbot. We will evaluate the responses using 2 metrics, which are **Human Evaluation** and **Cosine Similarity (using <https://tilores.io/cosine-similarity-online-tool>)**. However, for summarisation and translation tasks, we will only assess using Human evaluation.

1. What is climate change?

Expected Answer: a natural process where temperature, rainfall, wind and other elements vary over decades or more

Actual Answer:



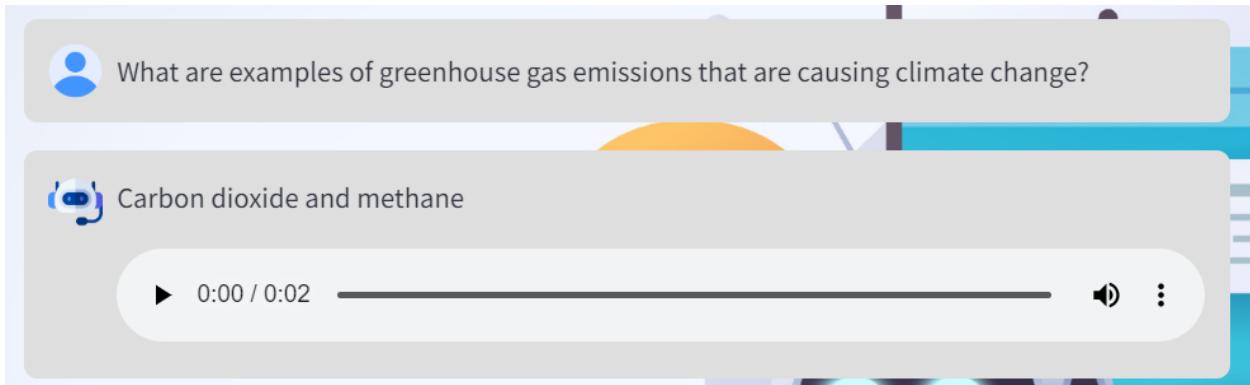
Remark: The expected answer and actual answer are almost the same. Therefore, the chatbot's response to this question is very accurate. This is supported by the cosine similarity score below.

A screenshot of the cosine similarity online tool. It shows two text inputs labeled "Text 1" and "Text 2". "Text 1" contains the bot's response: "Temperature, rainfall, wind and other elements vary over decades or more". "Text 2" contains the expected answer: "a natural process where temperature, rainfall, wind and other elements vary over decades or more". Below the inputs is a pink "Compare" button. To the left of the "Compare" button is a "Share Results:" section with a link: "https://tilores.io/cosine-similarity-online-tool?tl=Temperature,%20rainfall,%20wind%20and%20other%20elements%20vary%20over%20decades%20or%20more". To the right of the "Compare" button is a "Copy" button. At the bottom, there is a table with one row. The table has three columns: "Similarity Algorithm", "Similarity", and "Match*". The first column contains "Cosine", the second column contains "88.30%", and the third column contains "true".

2. What are examples of greenhouse gas emissions that are causing climate change?

Expected Answer: Carbon dioxide and methane

Actual Answer:



Remark: The expected answer and actual answer are exactly the same. Therefore, the chatbot's response to this question is very accurate. This is supported by the cosine similarity score below.

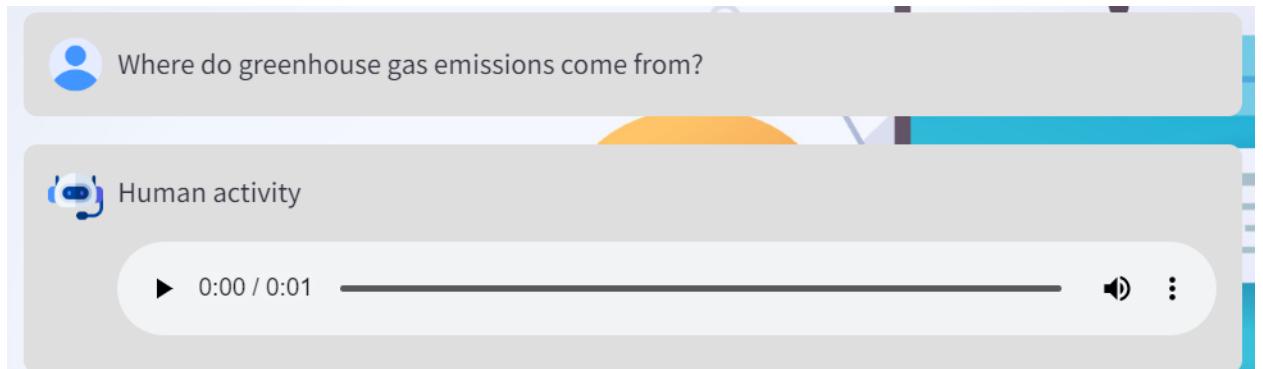
A screenshot of the cosine similarity online tool. It shows two text inputs: "Text 1" containing "Carbon dioxide and methane" and "Text 2" containing "Carbon dioxide and methane". A pink "Compare" button is centered between the inputs. Below the inputs, there is a "Share Results:" section with a URL "https://tilores.io/cosine-similarity-online-tool?tl=Carbon%20dioxide%20and%20methane&t2=" and a "Copy" button. At the bottom is a table with one row:

Similarity Algorithm	Similarity	Match*
Cosine	100.00%	true

3. Where do greenhouse gas emissions come from?

Expected Answer: These come from burning fossil fuels such as gasoline for driving a car or coal for heating a building. Clearing land and forests can also release carbon dioxide. Landfills for garbage are another source. Energy, industry, agriculture and waste disposal are among the major emitters.

Actual Answer:



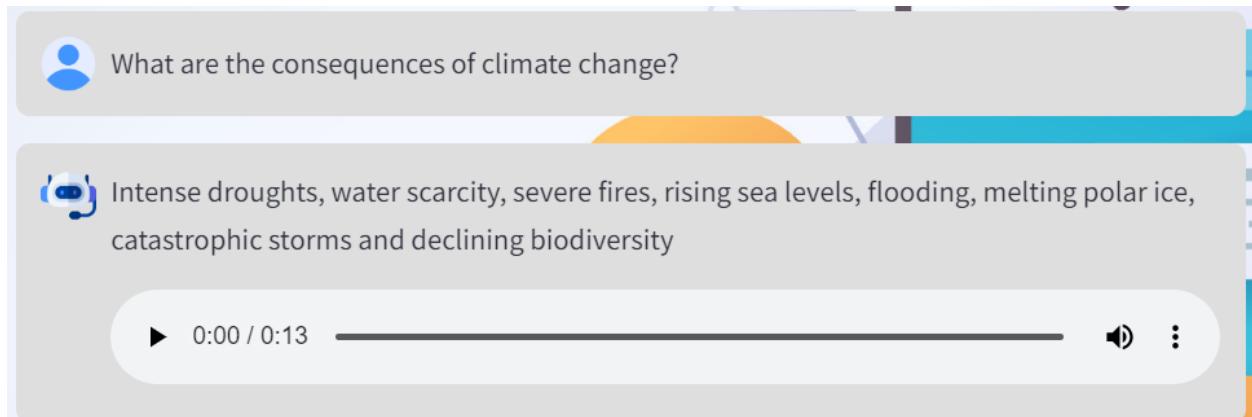
Remark: Although both the expected and actual answers have a similar gist, the actual answer is lacking some details. Therefore, for this question, the chatbot was not very accurate and detailed. This is supported by the cosine similarity score below.

A screenshot of a web-based cosine similarity calculator. It shows two text inputs: "Text 1" containing "Human activity" and "Text 2" containing "These come from burning fossil fuels such as gasoline for driving a car or coal for heating a building. Clearing land and fo". A "Compare" button is centered between the inputs. Below the inputs, there's a "Share Results:" section with a link "https://tilores.io/cosine-similarity-online-tool?t1=Human%20activity&t2=These%20come%20f" and a "Copy" button. At the bottom is a table with three columns: "Similarity Algorithm", "Similarity", and "Match*". The table has one row with "Cosine" in the algorithm column, "18.60%" in the similarity column, and "false" in the match column.

4. What are the consequences of climate change?

Expected Answer: Intense droughts, water scarcity, severe fires, rising sea levels, flooding, melting polar ice, catastrophic storms and declining biodiversity

Actual Answer:



Remark: The expected answer and actual answer are exactly the same. Therefore, the chatbot's response to this question is very accurate. This is supported by the cosine similarity score below.

The image shows a web-based cosine similarity tool. It has two text input fields: "Text 1" containing "Intense droughts, water scarcity, severe fires, rising sea levels, flooding, melting polar ice, catastrophic storms and declinir" and "Text 2" containing "Intense droughts, water scarcity, severe fires, rising sea levels, flooding, melting polar ice, catastrophic storms and declinir". Below the inputs is a "Compare" button. To the left of the results table is a "Share Results:" link with a URL "https://tilores.io/cosine-similarity-online-tool?t1=Intense%20droughts,%20water%20scarcity,%20severe%20fires,%20rising%20sea%20levels,%20flooding,%20melting%20polar%20ice,%20catastrophic%20storms%20and%20declinir&t2=Intense%20droughts,%20water%20scarcity,%20severe%20fires,%20rising%20sea%20levels,%20flooding,%20melting%20polar%20ice,%20catastrophic%20storms%20and%20declinir". To the right is a "Copy" button. A table below shows the comparison results:

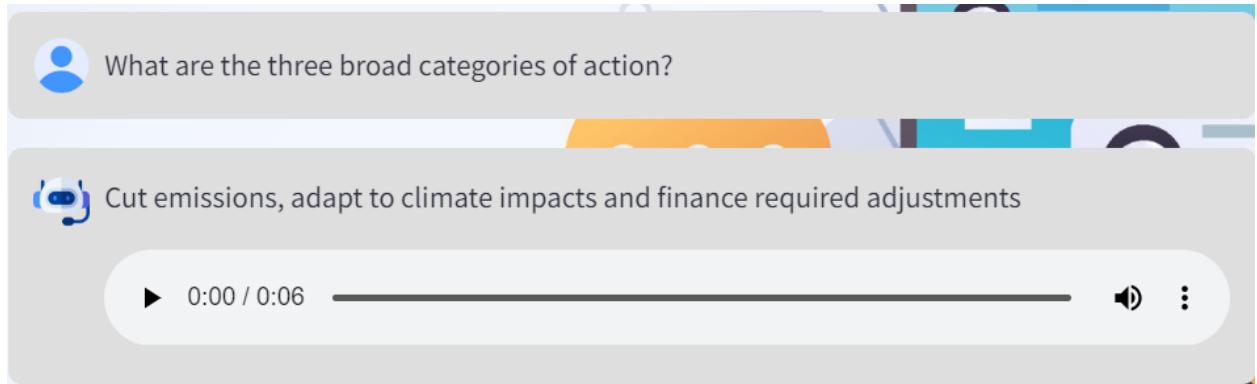
Similarity Algorithm	Similarity	Match*
Cosine	100.00%	true

* Match is considered true when similarity is 80% or above, distance is 3 or below, and phonetic encoding is equal.

5. What are the three broad categories of action?

Expected Answer: Cut emissions, adapt to climate impacts and finance required adjustments

Actual Answer:



Remark: The expected answer and actual answer are exactly the same. Therefore, the chatbot's response to this question is very accurate. This is supported by the cosine similarity score below.

The screenshot shows a comparison tool for text similarity. It has two input fields labeled "Text 1" and "Text 2", both containing the text "Cut emissions, adapt to climate impacts and finance required adjustments". Below the inputs is a large orange "Compare" button. To the left of the button is a "Share Results:" section with a link <https://tilores.io/cosine-similarity-online-tool?t1=Cut%20emissions,%20adapt%20to%20clima> and a "Copy" button. At the bottom is a table showing the results:

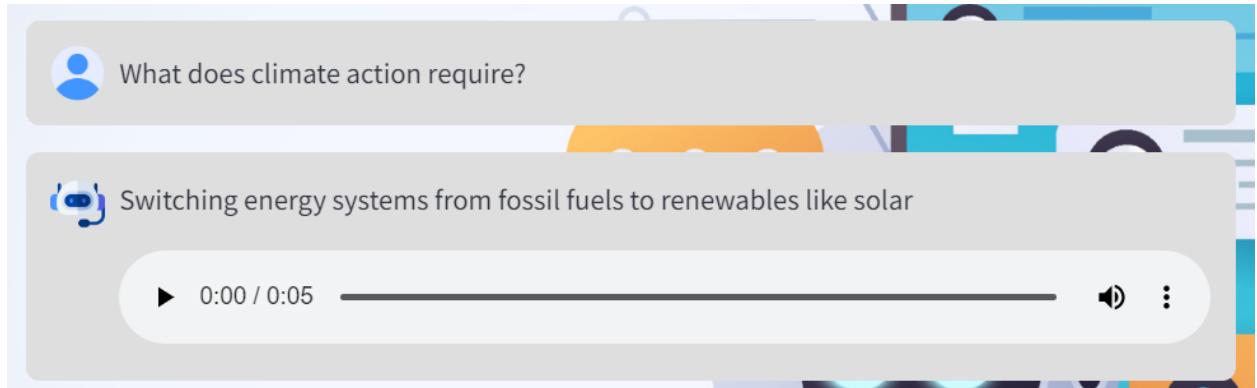
Similarity Algorithm	Similarity	Match*
Cosine	100.00%	true

* Match is considered true when similarity is 80% or above, distance is 3 or below, and phonetic encoding is equal.

6. What does climate action require?

Expected Answer: Switching energy systems from fossil fuels to renewables like solar

Actual Answer:



Remark: The expected answer and actual answer are the same. Therefore, the chatbot's response to this question is very accurate. This is supported by the cosine similarity score below.

A screenshot of the cosine similarity online tool. It shows two text inputs: "Text 1" containing "Switching energy systems from fossil fuels to renewables like solar" and "Text 2" containing the same text. A pink "Compare" button is centered below the inputs. To the left of the button is a "Share Results:" field with a URL and a "Copy" button to its right. Below the URL is a table comparing similarity algorithms. The table has three columns: "Similarity Algorithm", "Similarity", and "Match*". The first row shows "Cosine" as the algorithm, "100.00%" as the similarity, and "true" as the match status. A note at the bottom states: "* Match is considered true when similarity is 80% or above, distance is 3 or below, and phonetic encoding is equal.".

7. What is one critical step for industrialised countries to fulfil their commitment?

Expected Answer: Provide \$100 billion a year to developing countries so they can adapt and move towards greener economies.

Actual Answer:



Remark: The expected answer and actual answer are almost the same although the chatbot has left out a few details. Although the cosine similarity score below indicates that they are not highly similar, we will still classify the chatbot's response as accurate as the phrase “so they can adapt and move towards greener economies” is not a very important detail.

A screenshot of a web-based cosine similarity tool. It shows two text inputs: "Text 1" containing "Provide \$100 billion a year to developing countries" and "Text 2" containing "Provide \$100 billion a year to developing countries so they can adapt and move towards greener economies.". A "Compare" button is centered between the two inputs. Below the inputs, there is a "Share Results:" field with a URL: [https://tilores.io/cosine-similarity-online-tool?t1=Provide%20\\$100%20billion%20a%20year%20t2=Provide%20\\$100%20billion%20a%20year%20so%20they%20can%20adapt%20and%20move%20towards%20greener%20economies](https://tilores.io/cosine-similarity-online-tool?t1=Provide%20$100%20billion%20a%20year%20t2=Provide%20$100%20billion%20a%20year%20so%20they%20can%20adapt%20and%20move%20towards%20greener%20economies), and a "Copy" button. At the bottom, a table provides the results of the comparison:

Similarity Algorithm	Similarity	Match*
Cosine	76.40%	false

* Match is considered true when similarity is 80% or above, distance is 3 or below, and phonetic encoding is equal.

8. Summarise

Expected Answer: Anything related to climate change and the actions needed to address this issue

Actual Answer:

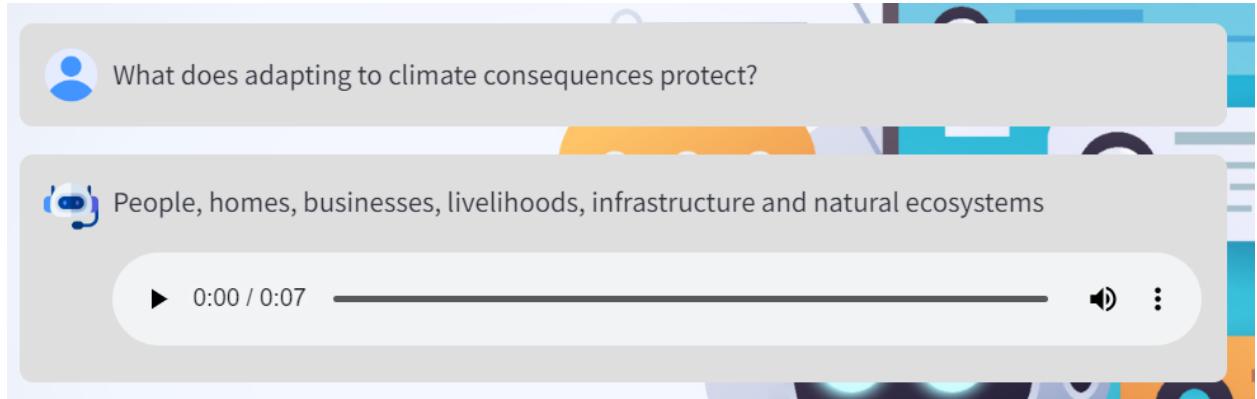
The screenshot shows a File Chatbot interface. At the top, there's a blue circular icon with a white person symbol, followed by the word "Summarise". Below this, a message from a blue robot icon reads: "Certainly! Here's a brief summary: Climate change can be a natural process where temperature, rainfall, wind and other elements vary over decades or more. In millions of years, our world has been warmer and colder than it is now. But today we are experiencing rapid warming from human activities, primarily due to burning fossil fuels. Greenhouse gas concentrations are at their highest levels in 2 million years. The earth is about 1.1°C warmer than it was in the 1800s. Energy, industry, agriculture and waste disposal are among the major emitters of carbon dioxide. People are experiencing climate change in diverse ways. It affects our health, ability to grow food, housing, safety and work. Some of us are already more vulnerable, such as people living in small island developing States. In the future, the number of 'climate refugees' is expected to rise. Current path of carbon dioxide emissions could increase global temperature by as much as 4.4°C by the end of the century. Everyone must take climate action, but people and countries must do more. The 100 least-emitting countries generate 3 per cent of total emissions. The 10 largest emitters contribute 68 per cent. Climate change is a huge challenge, but we already know many solutions. Switching energy systems from fossil fuels to renewables like solar will reduce emissions driving climate change. While a growing coalition of countries is committing to, we have to start right now. Fossil fuel production must decline by roughly 6 per cent per year between 2020 and 2030. About half of emissions cuts must be in place by 2030 to keep warming below 1.5°C. Adaptation will be required everywhere, but must be prioritized now for the most vulnerable. Climate action requires significant financial investments by governments and businesses. One critical step is for industrialized countries to provide \$100 billion a year to developing countries so they can adapt. We can pay the bill now, or pay dearly in the future." A red box highlights the "Chat with FileChatty" button at the bottom.

Remark: The summary generated by the File Chatbot accurately reflects the contents of the PDF file and has included all the main points.

9. What does adapting to climate consequences protect?

Expected Answer: People, homes, businesses, livelihoods, infrastructure and natural ecosystems

Actual Answer:



Remark: The expected answer and actual answer are exactly the same. Therefore, the chatbot's response to this question is very accurate. This is supported by the cosine similarity score below.

A screenshot of the cosine similarity online tool. It shows two text inputs: "Text 1" containing "People, homes, businesses, livelihoods, infrastructure and natural ecosystems" and "Text 2" containing the same text. A pink "Compare" button is centered between them. Below the inputs, there is a "Share Results:" section with a link "https://tilores.io/cosine-similarity-online-tool?t1=People%20homes,%20businesses,%20livelihi" and a "Copy" button. At the bottom, a table compares the similarity algorithm, similarity score, and match status. The table has three columns: "Similarity Algorithm" (Cosine), "Similarity" (100.00%), and "Match*" (true). A note at the bottom states: "* Match is considered true when similarity is 80% or above, distance is 3 or below, and phonetic encoding is equal."

10. Who are already more vulnerable to climate impacts?

Expected Answer: people living in small island developing States

Actual Answer:



Remark: The expected answer and actual answer are almost the same although the chatbot has left out a small detail. Nevertheless, we will classify the chatbot's response as accurate. This is supported by the cosine similarity score below.

The screenshot shows a web-based cosine similarity tool. It has two input fields labeled "Text 1" and "Text 2". "Text 1" contains "Small island developing States" and "Text 2" contains "people living in small island developing States". A large orange "Compare" button is centered between the text fields. Below the buttons, there is a "Share Results:" section with a URL "https://tilores.io/cosine-similarity-online-tool?t1=Small%20island%20developing%20States&t2=people%20living%20in%20small%20island%20developing%20States" and a "Copy" button. At the bottom, a table provides the results:

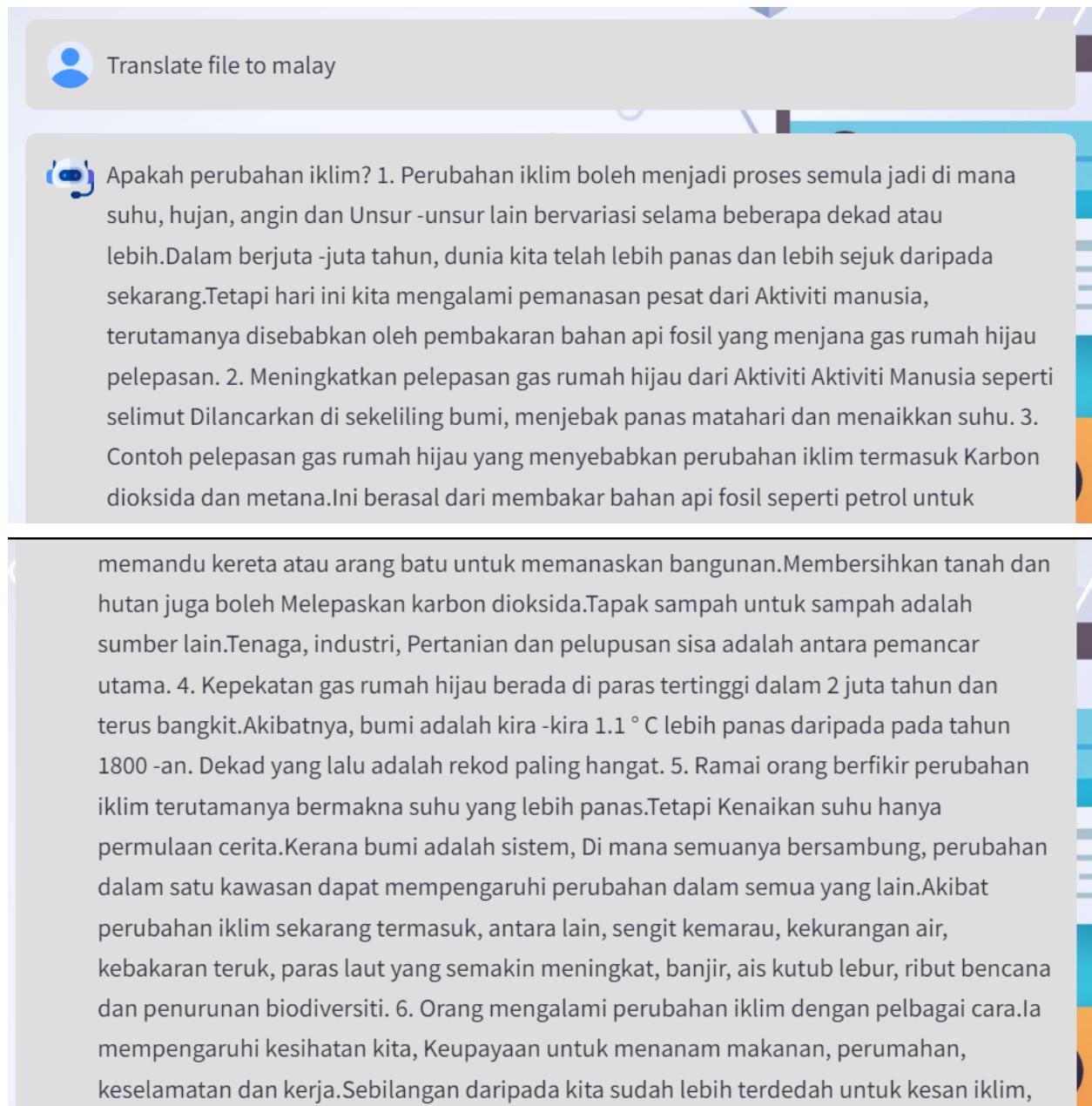
Similarity Algorithm	Similarity	Match*
Cosine	82.20%	true

* Match is considered true when similarity is 80% or above, distance is 3 or below, and phonetic encoding is equal.

11. Translate file to Malay

Expected Answer: A response in Bahasa Malaysia that has the same meaning and gist as the original English text

Actual Answer:



Translate file to malay

Apakah perubahan iklim? 1. Perubahan iklim boleh menjadi proses semula jadi di mana suhu, hujan, angin dan Unsur -unsur lain bervariasi selama beberapa dekad atau lebih. Dalam berjuta-juta tahun, dunia kita telah lebih panas dan lebih sejuk daripada sekarang. Tetapi hari ini kita mengalami pemanasan pesat dari Aktiviti manusia, terutamanya disebabkan oleh pembakaran bahan api fosil yang menjana gas rumah hijau pelepasan. 2. Meningkatkan pelepasan gas rumah hijau dari Aktiviti Aktiviti Manusia seperti selimut Dilancarkan di sekeliling bumi, menjebak panas matahari dan menaikkan suhu. 3. Contoh pelepasan gas rumah hijau yang menyebabkan perubahan iklim termasuk Karbon dioksida dan metana. Ini berasal dari membakar bahan api fosil seperti petrol untuk memandu kereta atau arang batu untuk memanaskan bangunan. Membersihkan tanah dan hutan juga boleh Melepaskan karbon dioksida. Tapak sampah untuk sampah adalah sumber lain. Tenaga, industri, Pertanian dan pelupusan sisa adalah antara pemancar utama. 4. Kepekatan gas rumah hijau berada di paras tertinggi dalam 2 juta tahun dan terus bangkit. Akibatnya, bumi adalah kira-kira 1.1°C lebih panas daripada pada tahun 1800-an. Dekad yang lalu adalah rekod paling hangat. 5. Ramai orang berfikir perubahan iklim terutamanya bermakna suhu yang lebih panas. Tetapi Kenaikan suhu hanya permulaan cerita. Kerana bumi adalah sistem, Di mana semuanya bersambung, perubahan dalam satu kawasan dapat mempengaruhi perubahan dalam semua yang lain. Akibat perubahan iklim sekarang termasuk, antara lain, sengit kemarau, kekurangan air, kebakaran teruk, paras laut yang semakin meningkat, banjir, ais kutub lebur, ribut bencana dan penurunan biodiversiti. 6. Orang mengalami perubahan iklim dengan pelbagai cara. Ia mempengaruhi kesihatan kita, Keupayaan untuk menanam makanan, perumahan, keselamatan dan kerja. Sebilangan daripada kita sudah lebih terdedah untuk kesan iklim,

seperti orang yang tinggal di negara -negara membangun pulau kecil. Keadaan seperti kenaikan paras laut dan pencerobohan air masin telah maju ke titik di mana keseluruhannya Komuniti terpaksa berpindah. Pada masa akan datang, bilangan "pelarian iklim" adalah dijangka meningkat. 7. Setiap peningkatan dalam pemanasan global. Dalam laporan 2018, beribu -ribu saintis dan pengulus kerajaan bersetuju bahawa mengehadkan kenaikan suhu global tidak lagi daripada 1.5°C akan membantu kita mengelakkan kesan iklim yang paling teruk dan mengekalkan yang boleh didiami iklim. Namun jalan pelepasan karbon dioksida semasa dapat meningkatkan global suhu sebanyak 4.4°C menjelang akhir abad. Fakta Fakta Cepat 8. Pelepasan yang menyebabkan perubahan iklim datang dari setiap bahagian dunia dan mempengaruhi semua orang, tetapi sesetengah negara menghasilkan lebih banyak daripada yang lain. 100 sekurang-kurangnya- Pemancar negara menjana 3 peratus daripada jumlah pelepasan. 10 pemancar terbesar menyumbang 68 peratus. Semua orang mesti mengambil tindakan iklim, tetapi orang dan negara Mewujudkan lebih banyak masalah mempunyai tanggungjawab yang lebih besar untuk

bertindak terlebih dahulu. 9. Perubahan iklim adalah satu cabaran besar, tetapi kita sudah tahu banyak penyelesaian. Ini boleh menyampaikan manfaat ekonomi sambil meningkatkan kehidupan kita dan melindungi alam sekitar. Kita juga mempunyai perjanjian global untuk membimbing kemajuan, seperti Konvensyen Rangka Kerja PBB mengenai perubahan iklim dan perjanjian Paris. Tiga kategori tindakan yang luas adalah: potong Pelepasan, menyesuaikan diri dengan kesan iklim dan kewangan yang diperlukan pelarasan. 10. Menukar sistem tenaga dari bahan api fosil ke tenaga boleh diperbaharui seperti solar akan mengurangkan Pelepasan yang mendorong perubahan iklim. Tetapi kita mesti bermula sekarang. Sementara berkembang Gabungan Negara berkomitmen untuk mengeluarkan pelepasan sifar pada tahun 2050, kira -kira separuh daripada Pemotongan pelepasan mestilah disediakan pada tahun 2030 untuk terus memanaskan badan di bawah 1.5°C . Bahan api fosil Pengeluaran mesti menurun kira -kira 6 peratus setahun antara 2020 dan 2030. 11. Mengadaptasi akibat iklim melindungi orang, rumah, perniagaan, mata pencaharian, Infrastruktur dan ekosistem semulajadi. Ia meliputi kesan semasa dan yang

pencaharian, Infrastruktur dan ekosistem semulajadi. Ia meliputi kesan semasa dan yang mungkin di masa depan. Adaptasi diperlukan di mana-mana, tetapi mesti diprioritaskan sekarang. Orang yang terdedah dengan sumber yang paling sedikit untuk mengatasi bahaya iklim. Kadar pulangan boleh tinggi. Sistem amaran awal untuk bencana, misalnya, menyelamatkan nyawa dan harta, dan boleh memberikan manfaat sehingga 10 kali kos awal. 12. Kita boleh membayar bil sekarang, atau membayar mahal pada masa akan datang. Tindakan iklim memerlukan pelaburan kewangan yang ketara oleh kerajaan dan perniagaan. Tetapi tidak bertindak iklim jauh lebih mahal. Satu langkah kritikal adalah untuk negara-negara perindustrian untuk memenuhi mereka komitmen untuk menyediakan \$ 100 bilion setahun ke negara-negara membangun supaya mereka dapat menyesuaikan diri dan bergerak ke arah ekonomi yang lebih hijau. Sumber: IPCC (1), WMO (4, 7, 10), WMO (4), IPCC (4, 7), Tindakan Iklim PBB (8, 10), Bank Dunia (11).



Remark: The File Chatbot has accurately translated the English text into Bahasa Malaysia. Hence, we will classify the response as accurate.

12. Translate document to chinese

Expected Answer: A response in Chinese that has the same meaning and gist as the original English text

Actual Answer:

The screenshot shows a mobile application interface with a light gray header bar at the top. On the left is a blue circular profile icon with a white person symbol. To its right, the text "Translate document to chinese" is displayed in a black sans-serif font. Below this, the main content area is divided into two sections by a thin horizontal line. The upper section contains a blue circular icon with a white person symbol, followed by the question "什么是气候变化?". The lower section contains the translated text in Chinese, which is a multi-paragraph document about climate change causes, impacts, and global efforts.

什么是气候变化? 1. 气候变化可以是自然过程，温度，降雨，风和其他元素数十年或更长时间有所不同。在数百万年中，我们的世界已经比现在温暖更冷。但是今天我们正在经历着快速的变暖 人类活动，主要是由于燃烧的化石燃料会产生温室气体 排放。2. 增加人类活动的温室气体排放的作用像毯子包裹在地球上，捕获太阳的热量并提高温度。3. 引起气候变化的温室气体排放的示例包括二氧化碳和甲烷。这些来自燃烧的化石燃料，例如汽油 用于驾驶汽车或煤炭来加热建筑物。清理土地和森林也可以释放二氧化碳。垃圾填埋场是另一个来源。能源，工业，农业和废物处理是主要发射器之一。4. 温室气体浓度在200万年的最高水平上，并且继续上升。结果，地球比1800年代温度约为1.1°C。最后十年是有记录的最温暖的。5.许多人认为气候变化主要意味着温度更高。但温度升高只是故事的开始。因为地球是一个系统，在所有连接的地方，一个区域的变化都会影响所有人的变化其他的。气候变化的后果现在包括强烈 干旱，缺水，严重的火灾，海平面上升，洪水，融化极地冰，灾难性风暴和生物多样性下降。6.人们以多种方式经历气候变化。它影响我们的健康，能够种植食物，住房，安全和工作。我们中有些人已经更脆弱了为了气候影响，例如居住在小岛发展中国家的人们。状况 像海平面上升和盐水的入侵已经发展到整个地点 社区必须搬迁。将来，“气候难民”的数量是预计会上升。7.全球变暖问题的每一项增加。在2018年的报告中，成千上万的科学家 政府审稿人同意，将全球温度升高到不再 超过1.5°C可以帮助我们避免最严重的气候影响并保持宜居 气候。然而，当前的二氧化碳排放路径可能会增加全球 到本世纪末，温度高达4.4°C。快速事实事实 8.引起气候变化的排放来自世界各地 影响每个人，但有些国家的生产要比其他国家多得多。100最小 发射国家产生了总排放量的3%。10个最大的发射器 贡献68%。每个人都必须采取气候行动，但是人和国家 创建更多的问题具有首先采取行动的最大责任。9.气候变化是一个巨大的挑战，但我们已经知道许多解决方案。这些可以 提供经济利益，同时改善我们的生活并保护环境。我们也有全球协议来指导进度，例如《联合国框架公约》 关于气候变化和巴黎协议。三

们也有全球协议来指导进度，例如《联合国框架公约》关于气候变化和巴黎协议。三个广泛的行动是：剪切排放，适应气候影响和金融需要调整。10.将能源系统从化石燃料转换为诸如太阳能之类的可再生能源排放推动了气候变化。但是我们必须立即开始。而成长国家联盟将于2050年承诺零排放净排放量，约占一半排放量必须在2030年进行，才能保持低于 1.5°C 的温暖。化石燃料在2020年至2030年之间，每年的产量必须下降约6%。11.适应气候后果可保护人，房屋，企业，生计，基础设施和自然生态系统。它涵盖了当前的影响以及可能在未来。到处都需要适应，但必须现在优先考虑脆弱的人拥有最少的资源来应对气候危害。以...的速率返回可能很高。例如，用于灾难的预警系统，例如挽救生命和财产，最多可提供最初成本的10倍。12.我们现在可以支付账单，或将来付出高额付款。气候行动需要政府和企业的大量金融投资。但是气候无所作为要贵得多。一个关键的步骤是工业化国家履行其承诺每年向发展中国家提供1000亿美元，以便他们可以适应并转向更绿色的经济体。资料来源：IPCC (1) , WMO (4、7、10) , WMO (4) , IPCC (4、7) , 联合国气候行动 (8、10) , 世界银行 (11) 。

Remark: The File Chatbot has accurately translated the English text into Chinese. Hence, we will classify the response as accurate.

13. Overall Remark about the Chatbot's Responses

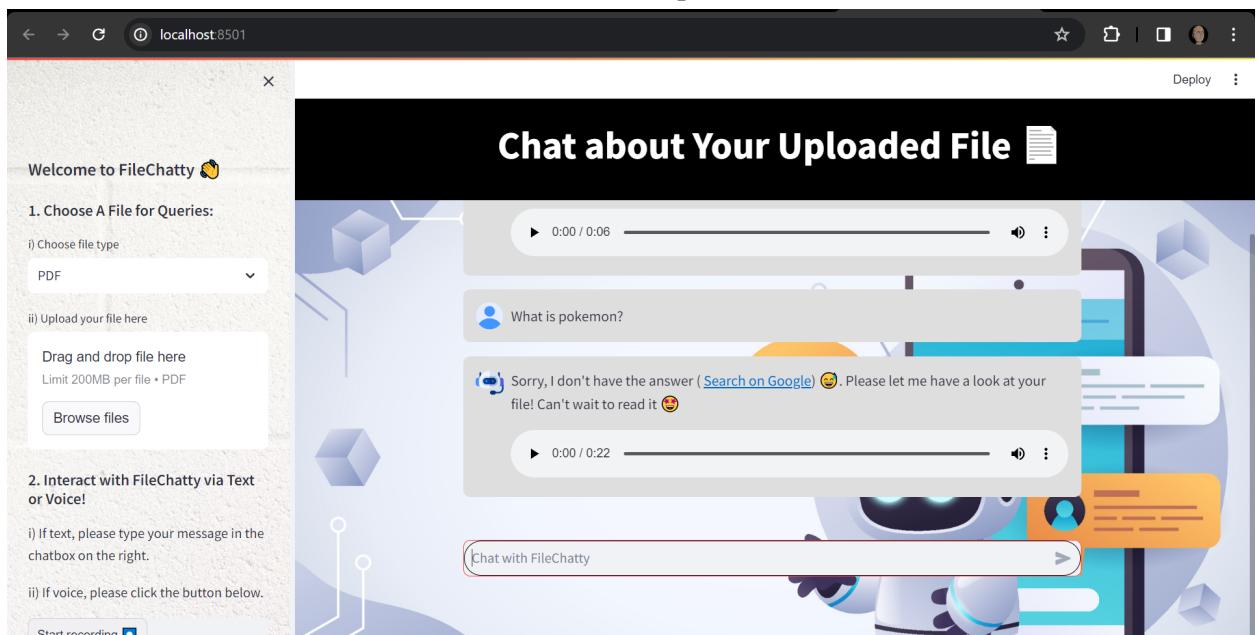
Out of the 12 questions asked, the Chatbot was able to accurately answer 11 of them, which are based on the file uploaded. This demonstrates that the Chatbot is able to understand the user's queries most of the time.

d) Answering General or Unrelated Questions with/without Uploading a File

Besides asking file-related questions, the user can also ask general questions regardless of whether a file is uploaded. If the chatbot does not know how to answer the question, it will apologise and provide a url link for the user to search for the answer on Google, which is helpful for this kind of scenario. Below are some scenarios of asking questions not related to the file.

1. Asking general/unrelated questions without uploading a file

Scenario 1: Chatbot does not know the answer to the question



Scenario 2: Chatbot knows the answer to the question

Welcome to FileChatty 🤖

1. Choose A File for Queries:

- i) Choose file type PDF
- ii) Upload your file here
Drag and drop file here Limit 200MB per file + PDF
Browse files

2. Interact with FileChatty via Text or Voice!

- i) If text, please type your message in the chatbox on the right.
- ii) If voice, please click the button below.
Start recording

Chat about Your Uploaded File 📄

What can you do?

Glad you've asked! I can answer questions related to the uploaded file, translate text to Mandarin and Bahasa Melayu, accept both audio and text inputs, provide responses in both audio and text, accept PDF, DOCX, or TXT files, compare document similarity, summarize text and support general conversation capabilities. 😊 Please upload your file on the left so that I can answer your questions!

0:00 / 0:36

Chat with FileChatty

2. Asking general/unrelated questions with a file uploaded

Scenario 1: Chatbot does not know the answer to the question

Welcome to FileChatty 🤖

1. Choose A File for Queries:

- i) Choose file type PDF
- ii) Upload your file here
Drag and drop file here Limit 200MB per file + PDF
Browse files

FairyTale2.pdf 55.3KB

File preview

Once upon a time, in a bright and bustling kingdom, lived a kind a

Chat about Your Uploaded File 📄

A video game

What is the recipe for chicken rice?

I don't know. I apologise 😞 [Search on Google](#)

0:00 / 0:01

0:00 / 0:19

Chat with FileChatty

Scenario 2: Chatbot knows the answer to the question

The image displays two screenshots of a web application interface, likely a chatbot or AI service, running on localhost:8501.

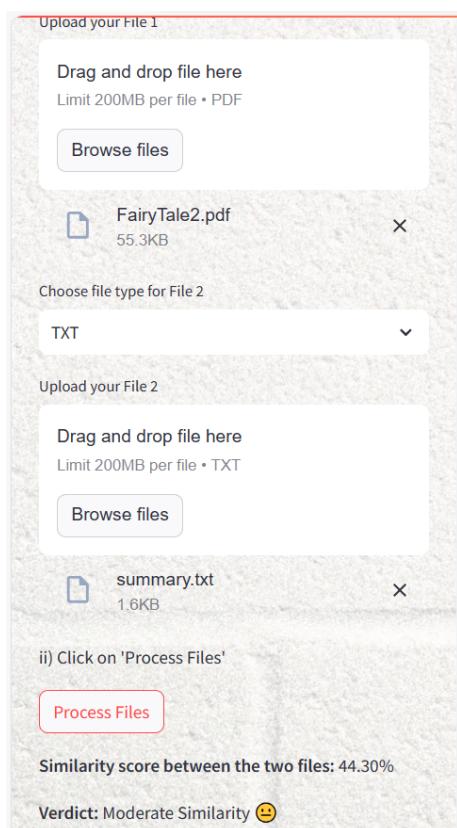
Screenshot 1: The left panel shows the "FileChatty" interface with a "Welcome to FileChatty" message and a file upload section. The user has chosen "FairyTale2.pdf" (55.3KB). The right panel shows a "Chat about Your Uploaded File" interface. A message from the bot says, "I don't know. I apologise. Search on Google". A user asks, "Who is Pikachu?", and the bot responds, "A Pokemon." There is a "Search on Google" link above the bot's response.

Screenshot 2: The left panel shows the same "FileChatty" interface. The right panel shows the "Chat about Your Uploaded File" interface. A user asks, "What can you do?", and the bot responds with a detailed description of its capabilities: "Glad you've asked! I can answer questions related to the uploaded file, translate text to Mandarin and Bahasa Melayu, accept both audio and text inputs, provide responses in both audio and text, accept PDF, DOCX, or TXT files, compare document similarity, summarize text and support general conversation capabilities." A "Search on Google" link is also present above the bot's response.

e) Document Similarity Check

In order for this feature to function, the user has to upload 2 files, which could be either PDF, TXT or DOCX. After specifying the file type, to upload a file in each file uploader, the user has to click on “Browse files” to insert the file for the chatbot to process. After two files are uploaded, the user has to click on the “Process Files” button in order for the File Chatbot to calculate the similarity between the 2 files.

The figure below shows a user comparing the content of a PDF file with the content of a TXT file. The similarity analysis will show a similarity score in % and a description that depicts the level of similarity between the files.



f) Speech-to-Text and Text-to-Speech

Apart from using text to interact with the chatbot, the user can also use voice to chat with it. On the left sidebar, the user can click on the “Start recording” button and when the user has finished saying his/her sentence, he/she must click on “Stop recording” to allow the chatbot process the query the user is asking.

After processing the user’s query, the chatbot will generate a textual response and an audio response. If the response is in English, the voice generated will have a Native English speaker’s accent. If the response is in Bahasa Malaysia, the voice generated will have a Native Malay speaker’s accent. If the response is in Chinese, the voice generated will have a Native Taiwanese Chinese speaker’s accent.

Below is a sample question we have asked the chatbot “Who is the princess” and the correctly generated response.

<p>2. Interact with FileChatty via Text or Voice!</p> <p>i) If text, please type your message in the chatbox on the right.</p> <p>ii) If voice, please click the button below.</p> <p>Start recording <input checked="" type="button"/></p>	<p>2. Interact with FileChatty via Text or Voice!</p> <p>i) If text, please type your message in the chatbox on the right.</p> <p>ii) If voice, please click the button below.</p> <p>Stop recording <input checked="" type="button"/></p>
---	--

The screenshot shows the FileChatty interface. On the left, there is a sidebar with instructions for interacting via text or voice, and a section for document similarity with file upload options. On the right, there is a main chat area with a message history. The message history shows a user asking "Who is the princess" and the bot responding with "Eliana". Below the chat area is a button labeled "Chat with FileChatty".

g) Handling Spelling Mistakes or Typos

Behind the scenes, whenever a user sends a textual message to the chatbot, it will first check for spelling mistakes in the user's response in order for the chatbot to understand the user's query and return the most relevant answer. If there are spelling errors or typos, the chatbot will automatically correct them.

Below, we show a user entering an incorrect spelling of the word "Chinese", which is "Chinase". When the user pressed the 'Enter' key, our chatbot managed to autocorrect the word's spelling to "Chinese". Because of this, it allowed the chatbot to understand that the user actually wanted it to translate the text in the uploaded file to Chinese, enabling it to generate the right response.



Discussion and Conclusion

1. Achievement of Objectives

The following were our objectives when we set out on this project:

1. To automate the process of extracting information from documents, which are PDF, DOCX and TXT files.
2. To provide assistance to users in navigating and understanding documents, making it easier for them to find the information they need in a short amount of time.
3. To enhance user engagement by providing a user-friendly interface for extracting information from documents with a click of a button.
4. To enrich the capabilities of natural language understanding to interpret and respond to user queries related or not related to the content of the uploaded documents.
5. To allow users to make queries via text or voice and to receive responses via text or voice.
6. To allow users to translate the contents of the documents uploaded to another language, which is either Bahasa Malaysia or Mandarin.
7. To provide a fast and efficient way to summarise the document, possibly within seconds, depending on the size of the document.
8. To allow the user to check for document similarity by comparing two files (the file types of the 2 files do not have to be the same)..
9. To correct spelling mistakes made by the user while names and special terms remain unchanged.
10. To use pre-processing (lemmatisation, stop words removal) before performing Cosine Similarity.

We have met **objectives 1 & 2** as we have managed to develop a File Chatbot Application that accepts PDFs, DOCXs and TXTs and offers various functionalities, ranging from answering file-related queries to summarisation.

We have also met **objectives 3 & 5** as we have provided ways for the user to interact with the chatbot via text or voice and to receive the response via text or voice. We have provided convenient and intuitive means for the user to upload their files. We have also made it easy for the user to compare the similarity between documents by clicking just a few buttons. The right blend of background images, colours, fonts and the layout of elements has been used to entice the user to use our application.

Our File Chatbot has also met **objective 4** as the File Chatbot has been programmed to handle scenarios where the user has uploaded a file or has not uploaded a file and to handle questions that may be either file-related, general or irrelevant. This will increase the application's overall user-friendliness and conversational abilities.

Our project has also achieved **objectives 6 and 7** as the user can easily request for these tasks to be done via text or voice and the responses are generated relatively quickly depending on the amount of text in the document.

We have also achieved **objective 8** as we allow the user to upload any two files to perform similarity comparison and to increase the interpretability of the similarity score, we even provided the similarity description.

Last but not least, the project has met **objectives 9 and 10** as the File Chatbot has increased resistance to issues relating to spelling mistakes and determining whether the user's question is related to one of the questions preprogrammed in the system.

2. The Significance of Result

The significance of our results in developing a Multi-File Type Chatbot using a Large Language Model (LLM) lies mainly in its **accuracy and effectiveness**. By successfully comprehending user queries across multiple file types, our chatbot establishes itself as a reliable source of information, fostering user satisfaction and trust. Besides answering the user's queries, our chatbot also provides other **value-added services** such as translation of the file contents from the English language to either Bahasa Malaysia or Chinese Mandarin. Our chatbot's ability to read **multiple file types**, which are common-occurring (PDF, Word, TXT), provides convenience to users, thereby increasing user satisfaction. Additionally, for the convenience of users, our chatbot allows communication with it to be in **audio** instead of written text. When performing research, a very useful time-saving feature is being able to obtain a **summary** of a document's contents. Our chatbot has such a feature. Moreover, there may be occasions where the user may want to compare two documents for **similarity**. Our chatbot has this capability too, displaying the degree of similarity as a percentage.

A robust chatbot should gracefully manage situations where the **user input is not perfect**. Regardless, the chatbot should handle the situation gracefully and ensure a smooth conversational experience despite the non-ideal situation. An example of this is when the user enters ambiguous queries or **questions not related to the file's contents**. If a file was indeed uploaded by the user, the chatbot will first determine whether the user's question is a general question. If it is not, our chatbot will then check whether the question falls under a summarisation request, a translation request or a document similarity request. If it is not one of them, the user's question will be passed to the LLM. In this situation, our chatbot will check if the LLM manages to answer the question. If not, our chatbot will apologise saying that it failed to answer the question. Instead of just stopping there, it also tries to be helpful by passing the query to Google and directing the user to the URL. Occasionally, the LLM may issue a "**tokens limit**" error message when processing a request. In this situation, the user will be shielded from this cryptic error message else he/she will be bewildered. By being **resilient** to user errors and LLM errors, our chatbot still manages to maintain a coherent conversation with the user despite encountering a negative situation.

Another example is when the user enters **mis-spelled words**, they are auto-corrected. However, auto-correction should not occur for names and special words such as technical terms. Our chatbot is able to distinguish between the two different situations and take the appropriate actions or not. By responding appropriately to these non-ideal situations, our chatbot improves the **overall user experience and demonstrates a resilient performance**. If this aspect is not addressed properly, it will bring down the overall user experience.

The **user interface** of our chatbot is ChatGPT-like, providing familiar UI controls and a user-friendly appearance. Again to demonstrate the same resilient behaviour, validation checks have been implemented to prevent errors. When uploading a file/document to query on, the user first needs to indicate the File Type which can only be of 3 types: PDF (default), TXT or DOCX. When the file type is indicated, only files of that type will be eligible for selection. For example if PDF is the indicated file type, then only PDF files will be displayed to the user for selection. This control ensures that it is impossible to select a DOCX or TXT file when the selected file type is PDF. Another validation check is when comparing 2 documents for similarity. Only after the 2 files have been selected and uploaded will the "Process Files" button be enabled.

3. Limitations

Although all the objectives are achieved within the scope of this assignment, our chatbot still contains limitations. Below are some limitations:

- Limited token limit for google/flan-t5-xxl LLM
 - Token limit is 512 for max length
 - Models with low token limits may struggle to process long documents or lengthy pieces of text. This limitation can be problematic in scenarios where comprehensive analysis or understanding of lengthy content is required.
- Not being able to upload more than 1 file for file-related questions
 - This was actually a design decision. We restricted it to 1 file only because we discovered that whenever multiple files were uploaded, it took a long time to get a response. Another reason the chatbot was programmed to accept only 1 file was to simplify the processing logic and enable the chatbot to focus on extracting relevant information from a singular source.
- The translation feature is limited to translating into two languages, which are Chinese and Bahasa Malaysia
 - This means that it prevents the user from translating the content of their uploaded file to other languages like Tamil, Bahasa Indonesia, etc
- For file uploads, the chatbot only accepts three file types, which are PDF, TXT and DOCX type
 - Restricting uploads to specific types may hinder users who have documents in alternative formats relevant to their queries.
 - Limiting file uploads to a few types may restrict the chatbot's applicability and use cases.
- Needs a stable internet connection to run the application smoothly
 - Since the FileChatbot uses HuggingFace API, an internet connection will be needed to access and use HuggingFace's models.
- If the user uploads a large file, the FileChatbot may take some time to answer the user's file-related questions and perform text summarisation.
 - Processing and analysing a large file takes a longer time, especially if the chatbot has to extract information or perform complex operations on the content.
- Responses by the chatbot may occasionally be inaccurate or irrelevant
 - It depends on the type of data that the LLM was trained on, its contextual understanding ability and how it was fine-tuned.
- The Speech-to-Text function performs better when users interact at a slower pace.
 - Some users speak quickly while some speak with slang and at a slower pace. The chatbot may not be able to figure out the exact word and sometimes miss out important key words mentioned by the user.

- The creation of an audio file as the chatbot's response to the user may take longer time if the textual response that is generated by the chatbot is too long
 - Longer texts will generally take more time for Google Text-to-Speech (gTTS) to process and convert into speech.

With these limitations listed above, at section 4, we explain in detail what should be improved in the future.

4. Future improvements

While the application offers promising features, it's important to note the limitations mentioned above. Therefore future improvement is crucial to make our chatbot more efficient for our users. Currently, users can only upload one file at a time, and file types are restricted to PDF, TXT, and Word documents. To improve this feature of file upload, we could include more file options for the user like Microsoft Excel, CSV, PNG and more. Furthermore, an area for potential future improvements is allowing users to upload more than one file to the chatbot. In short, by enabling users to upload several files at a time that can be of various file types, it will allow the File Chatbot to support a wide range of use cases. Whether it's for research, data analysis, or obtaining information from different sources, this flexibility makes the chatbot a versatile tool for users with diverse needs.

In addition to enhancing file upload functionality, there is potential for improving translation capabilities in future updates. Expanding the range of supported languages would enable the chatbot to cater to a more diverse international user base and attract new users. In the future, we could implement a user feedback system, allowing users to provide input. This feedback would be valuable for enhancing the functionalities of the File Chatbot to better meet user expectations.

To improve the File Chatbot's responses, we could consider using OpenAI's embeddings and LLM model. OpenAI Embeddings deliver swift and effective functionality, with models hosted on robust cloud infrastructure accessible via API. This means that users do not need to have extensive local setup. As for OpenAI's LLM, it is trained on extensive datasets, providing advanced capabilities in language generation and comprehension. This LLM is capable of performing tasks, such as question-answering, summarisation and text generation.

Furthermore, we acknowledge that our speech-to-text functionality necessitates users to speak at a slower pace compared to other chatbots. Nevertheless, we are committed to enhancing this feature for our users. Exploring an alternative model, which may yield superior performance compared to our current model, is under consideration as part of our efforts to improve the overall user experience.

Currently, the audio files generated by the chatbot are stored on the user's device, which may not be beneficial for users with limited device storage and may not be convenient for users to regularly delete those audio files. To address this, we could try saving the audio files on the cloud instead of on the user's device.

We will continuously enhance our chatbot's intelligence, exploring the nuances of natural language to shape conversations that will be not only accurate but truly engaging. This journey into the evolving realm of AI will be fueled by dedication, and with each step, we will unlock new possibilities for meaningful connections with our users and the chatbot.

References

- Alhadi, A. C., Deraman, A., Jalil, M. M. A., Yussof, W. N. J. W., & Mohemad, R. (2019). *The process for Cosine similarity between sentences*. ResearchGate. Retrieved from <https://www.researchgate.net/publication/337664616/figure/fig1/AS:831360558174209@1575223211972/The-process-for-Cosine-similarity-between-sentences.ppm>.
- Amrikahr. (2023). *Jaccard Similarity*. GeeksForGeeks. Retrieved from <https://media.geeksforgeeks.org/wp-content/uploads/20230811131746/How-to-Calculate-Jaccard-Similarity-in-Python.png>.
- Amrikahr. (2023, August 11). *How to calculate Jaccard similarity in Python*. GeeksforGeeks. <https://www.geeksforgeeks.org/how-to-calculate-jaccard-similarity-in-python/>
- Author: Fatih Karabiber Ph.D. in Computer Engineering, Fatih Karabiber Ph.D. in Computer Engineering, Psychometrician, E. R., & LearnDataSci, E. B. F. of. (n.d.). *Cosine similarity*. Learn Data Science - Tutorials, Books, Courses, and More. <https://www.learndatasci.com/glossary/cosine-similarity/>
- Awan, A. A. (2023, November 17). *The 5 best vector databases you must try in 2024*. KDnuggets. <https://www.kdnuggets.com/the-5-best-vector-databases-you-must-try-in-2024>
- Barros, M. V. (2023, September 3). *Chatgpt vs. human language*. Medium. <https://medium.com/listening-in-tongues/chatgpt-vs-human-language-ed16b68b729b>
- Cahyo, F. (2022, December 30). *Creating a text-to-speech system in python*. Medium. <https://medium.com/@fabiansyahcahyo/creating-a-text-to-speech-system-in-python-3c83297b336e>
- Chaindesk. (2023). *Chat with word*. <https://www.chaindesk.ai/products/chat-word>
- Chanda, D. (2023, December 8). *Build your own custom chatbot (without LLM hallucinations and on your distinct data)*. LinkedIn. <https://www.linkedin.com/pulse/build-your-own-custom-chatbot-guide-dipendu-chanda>
- Chao, K. (2023, July 2). *Q&A bot using Langchain, Huggingface embedding, OpenAI LLM*. Tales of a Programmer. <https://randomdotnext.com/q-a-bot-using-langchain-huggingface-embedding-openai/>
- ChatPDF. (n.d.). *Chat with any PDF*. <https://www.chatpdf.com/>
- Chaudhary, M. (2021, January 28). *TF-IDF vectorizer scikit-learn*. Medium. <https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a>

Datahat. (2023). *Building a Chatgpt like Chatbot using Langchain and Hugging Face || Step by step Langchain tutorial*. Retrieved December 31, 2023, from <https://www.youtube.com/watch?v=cKjh5ZOWqus>.

Dutta, M. (2023, September 20). *Unlocking Langchain & Flan-T5 XXL: A guide to efficient document querying*. Analytics Vidhya.
<https://www.analyticsvidhya.com/blog/2023/09/unlocking-langchain-flan-t5-xxl-a-guide-to-efficient-document-querying/#:~:text=Flan%2DT5%20is%20a%20commercially,%2Dto%2Dtext%E2%80%9D%20framework>

Dwivedi, Y. K., Kshetri, N., Hughes, L., Slade, E. L., Jeyaraj, A., Kar, A. K., Baabdullah, A. M., Koohang, A., Raghavan, V., Ahuja, M., Albanna, H., Albashrawi, M. A., Al-Busaidi, A. S., Balakrishnan, J., Barlette, Y., Basu, S., Bose, I., Brooks, L., Buhalis, D., ... Wright, R. (2023). Opinion paper: “so what if chatgpt wrote it?” multidisciplinary perspectives on opportunities, challenges and implications of Generative Conversational AI for Research, practice and policy. *International Journal of Information Management*, 71, 102642.
<https://doi.org/10.1016/j.ijinfomgt.2023.102642>

Espíndola, G. (2023, September 5).  *text splitters: Smart text division with Langchain*. Medium.
<https://gustavo-espindola.medium.com/%EF%B8%8F-%EF%B8%8F-text-splitters-smart-text-division-with-langchain-1fa8ac09eb3c>

Foster, K. (2023, November 9). *Text summarization for NLP: 5 best apis, AI models, and ai summarizers*. News, Tutorials, AI Research.
<https://www.assemblyai.com/blog/text-summarization-nlp-5-best-apis/>

Garrido, L. (2018). *General methodology for a text-based conversational agent*. ResearchGate. Retrieved from
https://www.researchgate.net/figure/General-methodology-for-a-text-based-conversational-agent_fig1_327391803.

Ghebache, S. (2023, September 27). *Build powerful chat assistant for pdfs and articles without openai key*. Analytics Vidhya.
<https://www.analyticsvidhya.com/blog/2023/09/chat-assistant-for-pdfs-and-articles-without-openai-key/#h-word-embedding>

GTTS. PyPI. (2023, December 20). <https://pypi.org/project/gTTS/>

Hugging Face. (n.d.). *Facebook/Bart-Large-CNN · huggingface*. facebook/bart-large-cnn · Hugging Face.
<https://huggingface.co/facebook/bart-large-cnn>

Hore, S. (2023, May 12). *Chatbot evolution: Chatgpt vs. rule-based*. Analytics Vidhya.
<https://www.analyticsvidhya.com/blog/2023/05/chatbot-evolution-chatgpt-vs-rule-based/>

Huawei OS Developer. Document. (2023, December 8).

<https://developer.huawei.com/consumer/en/doc/HMS-Plugin-Guides/text-to-speech-0000001052489003>

IBM. (n.d.).

https://www.ibm.com/products/watsonx-assistant?utm_content=SRCWW&p1=Search&p4=4370077029995900&p5=e&gclid=CjwKCAiAnL-sBhBnEiwAJRGigsrXJKLby2pIetZFAVjOkFoRMZTfL2Jwo-EnYGuKz8QCMEvxIMMXOhoCzUQQAvD_BwE&gclsrc=aw.ds

K, D. (2023, October 12). How to develop a multi-file chatbot?. Analytics Vidhya.

<https://www.analyticsvidhya.com/blog/2023/09/how-to-develop-a-multi-file-chatbot/>

Kanade, V. (2023, September 26). *Large language model types, working, and examples: Spiceworks.*

Spiceworks. <https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-llm/>

Kapronczay, M. (2022, December 13). *A beginner's guide to language models.* Built In.

<https://builtin.com/data-science/beginners-guide-language-models>

Kimber, R. (2021, March 8). *How technology can cross language barriers.* LinkedIn.

<https://www.linkedin.com/pulse/how-technology-can-cross-language-barriers-richard-kimber>

LangChain. (2023). *Introduction.*  Langchain.

https://python.langchain.com/docs/get_started/introduction

LangChain. (n.d.). *Vector Stores.*  Langchain.

https://python.langchain.com/docs/modules/data_connection/vectorstores/

Language I/O. (2023, June 27). *Email and ticket translation.*

<https://languageio.com/chatbot-translation/#:~:text=Chatbot%20translation%20software%20is%20technology,in%20more%20than%20150%20languages.>

Lumen Learning. (n.d.). *Public speaking: Emphasis, Volume, and Pauses.* Lumen.

<https://courses.lumenlearning.com/wm-publicspeaking/chapter/emphasis-volume-and-pauses/>

mlflow. (n.d.). *Introduction to sentence transformers and mlflow.* Introduction to Sentence Transformers and MLflow - MLflow 2.9.2 documentation.

<https://mlflow.org/docs/2.9.2/lms/sentence-transformers/tutorials/quickstart/sentence-transformer-s-quickstart.html>

Mondal, A. (2023, October 25). *How to build your AI chatbot with NLP in python?.* Analytics Vidhya.

<https://www.analyticsvidhya.com/blog/2021/10/complete-guide-to-build-your-ai-chatbot-with-nlp-in-python/>

MonicaChatPDF. (n.d.). Your CHATGPT AI assistant chrome extension. Monica.

<https://monica.im/webapp/doc-chat>

- Nair, A. (2022, January 17). Comparing documents with similarity metrics. Medium.
<https://towardsdatascience.com/comparing-documents-with-similarity-metrics-e486bc678a7d>
- Otten, N. V. (2023, December 6). Top 7 ways to implement document & text similarity in python: NLTK, scikit-learn, Bert, Roberta, FastText and pytorch. Spot Intelligence.
<https://spotintelligence.com/2022/12/19/text-similarity-python/#:~:text=There%20are%20several%20ways%20to,for%20natural%20language%20processing%20tasks.&text=This%20code%20first%20tokenizes%20and,IDF%20vectors%20for%20the%20texts>
- PDFGear. (2023). *PDFgear chatbot*. PDFgear. <https://www.pdfgear.com/chat-pdf/>
- ReadingRockets. (2023). *Text-to-speech technology: What it is and how it works*. Reading Rockets.
<https://www.readingrockets.org/topics/assistive-technology/articles/text-speech-technology-what-it-and-how-it-works>
- Research, D. E. (2022, April 12). *Language translation using Python*. Medium.
<https://towardsdatascience.com/language-translation-using-python-bd8020772ccc>
- Riturajsaha. (2023, January 19). *Understanding TF-IDF (term frequency-inverse document frequency)*. GeeksforGeeks.
<https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/>
- Rouse, M. (2012, November 19). *What is speech-to-text software? - definition from Techopedia*. Speech-to-Text Software. <https://www.techopedia.com/definition/23767/speech-to-text-software>
- Rouse, M. (2012, June 7). *What is text to Speech (TTS)? - definition from Techopedia*. Techopedia.
<https://www.techopedia.com/definition/23843/text-to-speech-tts>
- Sánchez-Díaz, X., Ayala-Bastidas, G., Fonseca-Ortiz, P., & Garrido, L. (2018). A knowledge-based methodology for building a conversational chatbot as an intelligent tutor. *Advances in Computational Intelligence*, 3–3. https://doi.org/10.1007/978-3-030-04497-8_14
- Shinde, S., Nahar, S., Shelke, S. N., Mishra, A., & Sapre, S. (2019). Intelligent chatbot for guided navigation of repository contend. *IJARCCE*, 8(5), 25–28.
<https://doi.org/10.17148/ijarcce.2019.8506>
- Supe, K. (2023, June 7). Understanding cosine similarity in python with scikit-learn. Memgraph.
<https://memgraph.com/blog/cosine-similarity-python-scikit-learn>
- Tate, L. (n.d.). *The difference between speech and voice recognition*. RSS.
<https://www.kardome.com/blog-posts/difference-speech-and-voice-recognition>

The most common punctuation marks in English. IDP IELTS Malaysia. (n.d.).

<https://ielts.idp.com/malaysia/prepare/article-punctuation>

Todeschini, S. (2023, July 20). *How to chunk text data-A comparative analysis*. Medium.

<https://towardsdatascience.com/how-to-chunk-text-data-a-comparative-analysis-3858c4a0997a>

Treadway, A. (2019, December 10). *3 packages to build a spell checker in Python*. Open Source Automation. <https://theautomatic.net/2019/12/10/3-packages-to-build-a-spell-checker-in-python/>

Usher, S. (2021). *Free Machine Learning Services on AWS*. Amazon.

https://aws.amazon.com/free/machine-learning/?gclid=CjwKCAiA-bmsBhAGEiwAoaQNmnPefxVkB-HAKRItHipO4OFgF4CT9YVb5-BNWAh0QJNbMRUAAIo-ERoC4d8QAvD_BwE&trk=23910ecf-e1fa-45f7-aa60-5b1dc7e0a225&sc_channel=ps&ef_id=CjwKCAiA-bmsBhAGEiwAoaQNmnPefxVkB-HAKRItHipO4OFgF4CT9YVb5-BNWAh0QJNbMRUAAIo-ERoC4d8QAvD_BwE%3AG%3As&s_kwcid=AL%214422%213%21637354260642%21p%21%21g%21%21text+to+speech+apps%2119037249484%21146784777591

Verma, K. (2022, July 7). *Using countvectorizer to extracting features from text*. GeeksforGeeks.

<https://www.geeksforgeeks.org/using-countvectorizer-to-extracting-features-from-text/>

Wahid, A. A. (2023, June 27). *Langchain chatbot for multiple pdfs: Harnessing GPT and Free Huggingface LLM Alternatives*. Medium.

<https://medium.com/@abdullahw72/langchain-chatbot-for-multiple-pdf-harnessing-gpt-and-free-huggingface-llm-alternatives-9a106c239975>

Wiggins, D. (2023, March 6). *What is Neural Machine Translation (NMT)?*. Omnisclien Technologies.

<https://omniscien.com/faq/what-is-neural-machine-translation/>

YouTube. (2023). *How Google Translate works- Architecture explained*. YouTube. Retrieved December 31, 2023, from <https://www.youtube.com/watch?v=z3XKMnu2pgg>.