SP-15—Chat GPT
GPT-Lingo
Final Report

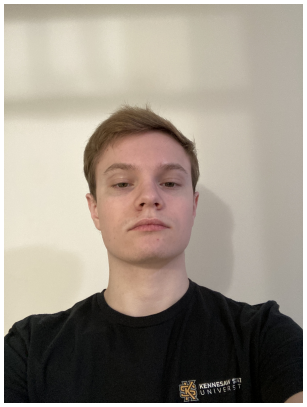
CS 4850 - Section 01 – Spring 2023

Multilingual Chatbot for Language Learning - Desktop Application

Website Link | Demo Video Link

Abstract

The objective of this senior project is to develop a desktop application that leverages natural language processing technology using Chat GPT, aimed at helping individuals learn new languages. The proposed tool allows users to engage in real-time conversations with the chatbot, enabling them to receive prompt feedback and corrections in a range of languages. The project's overarching aim is to create an interactive and efficient language learning solution suitable for individuals of all ages and proficiency levels.

			
Joshua Michael	**Yemi Agesin**	**Aidan Mitchell**	**Jacob Schirm**

Team Members:

Joshua Michael	(404) 914-8670	jpmichael450@gmail.com
Yemi Agesin	(857) 500-0745	yemifun01@gmail.com
Aidan Mitchell	(770) 712-7205	mitchellaidan2@gmail.com
Jacob Schirm	(678) 334-7499	jacobschirm5@gmail.com
Sharon Perry (Advisor)	(770) 329-3895	sperry46@kennesaw.edu

Introduction and Scope

This multilingual chatbot for language learning is a software system designed to assist language learners in practicing their skills and improving their proficiency in whatever foreign language they wish to learn. Our chatbot will provide users with a conversational platform to interact with, offering personalized feedback and language-specific content based on the user's language proficiency level.

The chatbot will support multiple languages, including English, Spanish, French, and German. Users will be able to interact with the chatbot in their preferred language, and the chatbot will use Chat GPT as its natural language processing (NLP) engine. Chat GPT is a state-of-the-art NLP model that uses deep learning techniques to generate natural language responses, which will allow the chatbot to provide more human-like responses and improve the user experience.

The chatbot will be accessible via an app interface, which will allow users to interact with the chatbot from any device with an internet connection. The web interface will provide users with a simple and intuitive interface to interact with the chatbot, including features such as conversation history, progress tracking, and access to language-specific content.

This provided language-specific content such as easy definitions and translations of the AI’s responses, and more unique dialog based on the user's language proficiency level. The chatbot will also use machine learning techniques to adapt to the user's learning style and preferences, providing personalized feedback and content to improve the user's language skills.

The development of the chatbot will follow an agile development methodology, with regular iterations and feedback from stakeholders to ensure the chatbot meets their needs and requirements. The development team will be responsible for ensuring the chatbot meets high standards of security, privacy, and performance, and will conduct regular testing and quality assurance to ensure the chatbot operates as intended.

Requirements Analysis

Functional Requirements:

User Interface:

1.1 The chatbot should have an intuitive graphical user interface (GUI) that allows users to interact with the system.

1.2 The GUI should have a text input field for users to enter their messages and a text output field that displays the chatbot's response.

1.3 The GUI should also provide options for users to choose their preferred language and set the level of proficiency in the language.

Language Processing:

2.1 The chatbot should be able to process natural language input from the user in different languages, including English, Spanish, French, German, etc.

2.2 The chatbot should use Chat GPT to generate appropriate responses in the language selected by the user.

2.3 The chatbot should be able to recognize and respond to common languages learning queries, such as requests for word definitions, verb conjugations, and grammar explanations.

User Management:

3.1 The chatbot should save the user's profiles, including their names, preferred languages, and levels of proficiency.

3.2 The chatbot should track the user's general progress in language learning and provide appropriate feedback to help the user improve their skills.

3.3 The chatbot should provide personalized language learning content based on the user's level of proficiency, interests, and learning goals.

System Performance:

4.1 The chatbot should be responsive and fast, with minimal latency between the user's input and the chatbot's response.

4.2 The chatbot should be able to handle multiple user requests simultaneously without performance degradation.

4.3 The chatbot should have appropriate error-handling mechanisms to handle unexpected user input and system failures.

Non-Functional Requirements:

Performance: The application must respond to user input within a reasonable time frame. The response time for generating a response from the Chat GPT model should be no more than a few seconds.

Security: The application must ensure that user data is kept secure and private. All communication between the user and the chatbot should be private.

Scalability: The application should be designed to handle a large number of users simultaneously. It should be scalable to handle increasing numbers of users without a significant decrease in performance.

Reliability: The application should be highly reliable and available at given times, and fall back to the previous Chat GPT versions if high demand is dealt with. It should be able to handle unexpected errors and exceptions without crashing or losing data.

Compatibility: The application should be compatible with different operating systems, including Windows, MacOS, and Linux.

User Interface: The application should have a user-friendly interface that is easy to navigate and understand. The interface should be designed to meet the needs of non-technical users, and it should be aesthetically pleasing.

Maintainability: The application should be easy to maintain and update. The code should be well-documented and well-organized so that other developers can easily understand and modify it. The application should be designed to be extensible and adaptable to changing user requirements.

System Architecture: The multilingual chatbot application will be developed as a desktop application using Python programming language, which can be run on a Windows operating system. The application will consist of two main components:

Frontend: The frontend of the application will be designed using a graphical user interface (GUI) library such as PyQT. The GUI will provide the user with an interface to interact with the chatbot. The user will be able to enter text input in their preferred language and receive responses from the chatbot in the language they have selected.

Backend: The backend of the application will be responsible for processing the user's input and generating a response. It will use the ChatGPT model to generate responses to the user's input. The user's grammar can be checked for mistakes and offered corrections using API resources such as Smodin or Python libraries. There will also be options offered to do user speech-to-text, as well as text-to-speech from the output of ChatGPT to make the flow of conversation more smooth. Speech-to-text is handled using Whisper from OpenAI and text-to-speech can be handled by one of many different APIs or libraries.

System Design: The multilingual chatbot system will be developed as a Python Windows application with a modular design that allows for easy maintenance, scalability, and integration with external services. The system will be designed using a client-server architecture with the client being a graphical user interface and the server being the natural language processing (NLP) and machine learning engine that provides the conversational capabilities of the chatbot.

The system will consist of the following modules:

Graphical User Interface (GUI) Module: This module will be responsible for handling the user interface and user interactions with the chatbot. The GUI will be designed using PyQt, a powerful and popular GUI toolkit for Python.

NLP and Machine Learning Module: This module will be responsible for processing natural language input, understanding the user's intent, generating appropriate responses, and learning from the user's interactions. This module will be built using the generalized large-language ChatGPT model, which is well known for its conversational abilities as an AI.

Data Storage and Retrieval Module: This module will be responsible for storing user data, such as user preferences, progress, and chat histories. This module will utilize a SQL database to store and retrieve user data.

Application Programming Interface (API) Integration Module: This module will be responsible for integrating external services such as translation services, speech-to-text and text-to-speech services, and other third-party NLP services. The API Integration module will be designed to be modular and easily extendable, allowing for future integrations with new external services.

The system will be designed to be scalable, allowing for the easy addition of new features and integration with external services. The system will also be designed to be maintainable, with a clear separation of concerns between different modules and clear documentation and coding standards.

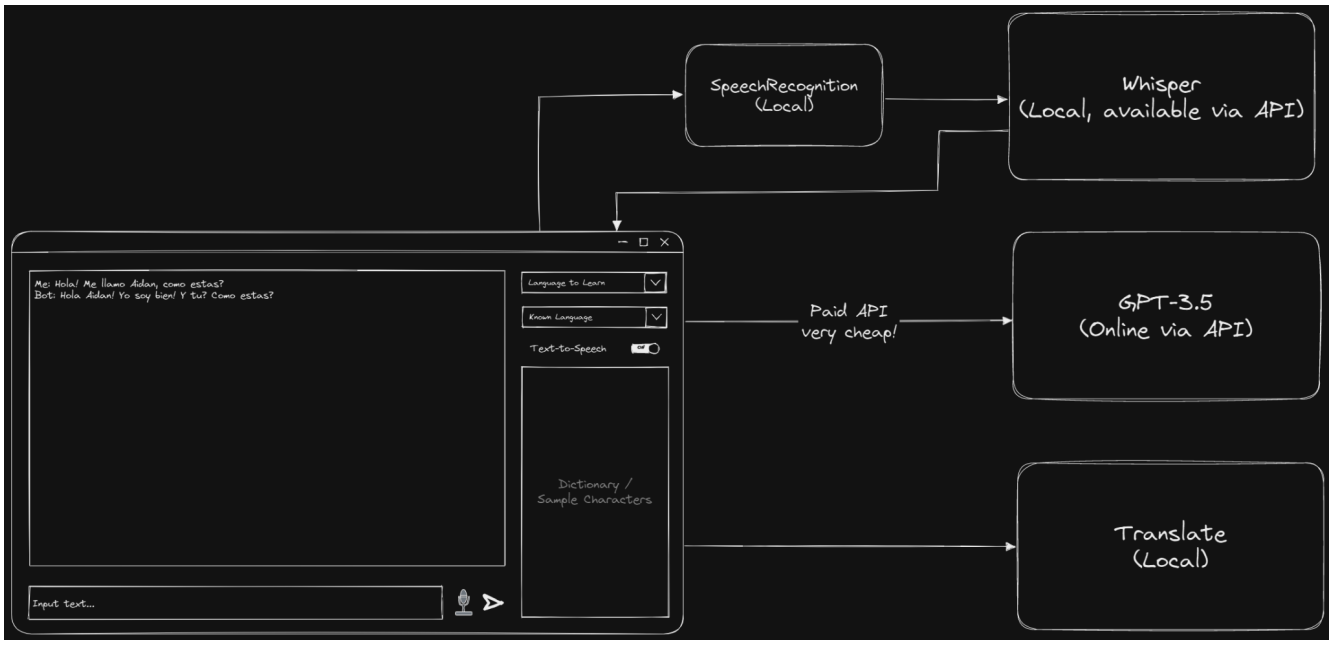


Figure 1.1

System Test Plan: The system test plan aims to ensure that the Multilingual Chatbot for Language Learning meets the requirements set forth in the functional and non-functional requirements sections of this SRS document. The testing will be performed on a Python Windows application.

Test Approach
The testing approach will include the following activities:

- Unit Testing: Each module of the application will be tested in isolation to ensure that the individual components are functioning as expected.
- Integration Testing: The individual modules will be combined and tested as a group to ensure that they work together correctly.
- System Testing: The entire application will be tested as a whole to ensure that all features work together in an intended manner.
- Acceptance Testing: The system will be tested by several users to ensure that the program works as intended, and is intuitive to use.

Test Cases
The following test cases will be executed to verify that the system meets the requirements:

Unit Test: Language Detection
- Verify that the language detection module correctly identifies the language of input text, and can handle input text in multiple languages.

Unit Test: Grammatical Correction
- Verify that the multi-lingual grammatical correction works and makes useful/helpful grammatical corrections and recommendations.

Integration Test: Speech Recognition and Text-to-Speech
- Verify that the speech recognition and text-to-speech modules work together correctly and can handle multiple different languages.

System Test: Conversational Flows
- Verify that the chatbot can engage in a conversation with a user in a single language and switch to a different respective language when necessary

Test Environment:
The testing environment will include the following:

Operating System: Windows 10
Python version: 3.10 or higher

Planning

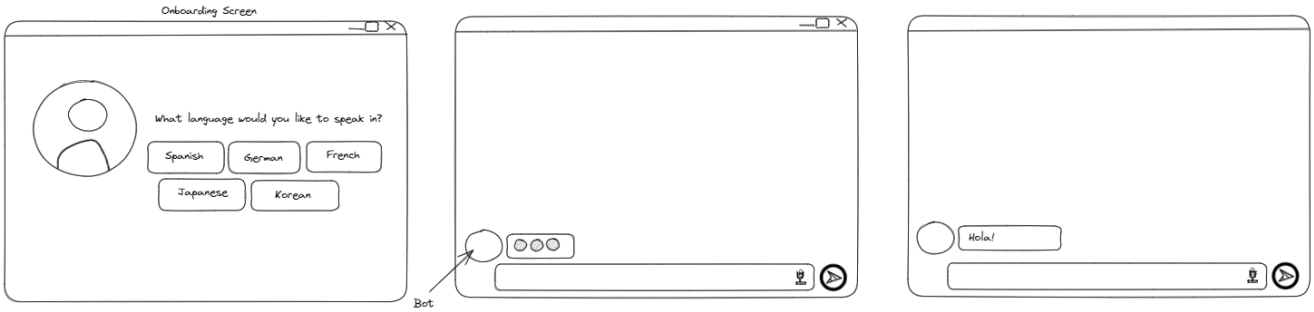
Featured below are our documents for how we planned to complete our tasks and our design and mockups for how the application would look.

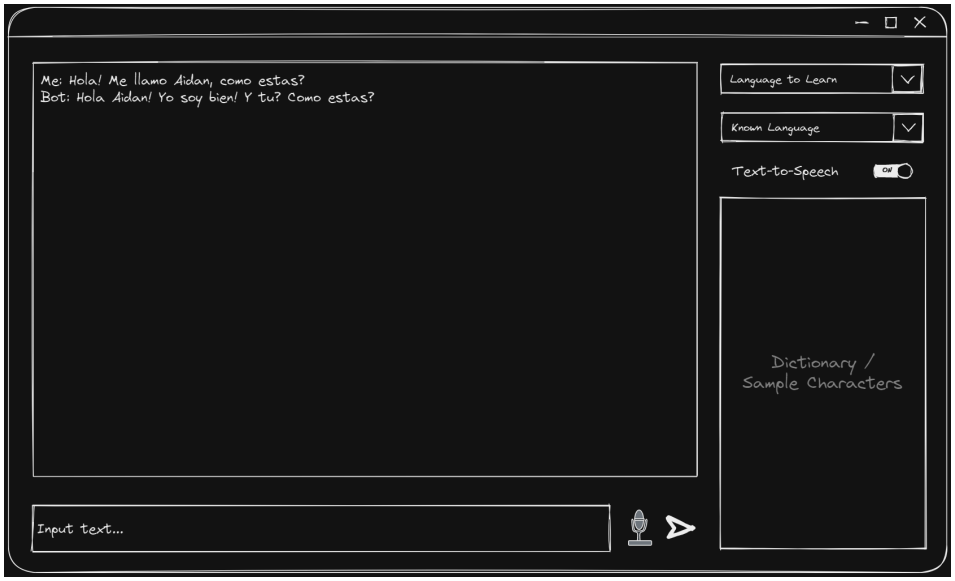
Project Name: SP-15 Chat GPT
Report Date: 2/3/2023

*Spring Break

Deliverable	Tasks	Complete%	Current Status Memo	Assigned To	Milestone #1 01/22	01/29	02/05	02/12	Milestone #2 02/19	02/26	03/05	03/12	Milestone #3 03/19	03/26	04/02	04/09	C-Day 04/16	04/23
REQUIREMENTS	Kick-Off Meeting	0%		Group	3													
	Define Requirements	0%		Josh		5												
	Review Requirements with Advisor	0%					2											
PROJECT DESIGN	Define Tech Required	0%		Josh				5										
	Database Design	0%							5	5								
	Front-End Design	0%								5								
	Back-End Design	0%																
	Develop GUI Prototype	0%								10		5						
	Test prototype	0%		Josh								5						
DEVELOPMENT	Review prototype design	0%		Josh	3	3												
Front End	Import and Configure PyQt	0%		Yemi			5	3										
	Create Buttons Library	0%		Yemi					5	3								
	Create StyleSheet	0%		Yemi								5						
	Create Launch Screen	0%		Yemi									3					
Back End	Research Solutions	100%	research complete	Aidan	3													
	Test Speech-to-Text	75%	needs testing with other models	Aidan		1												
	Connect Speech-to-Text to Mic	5%	needs testing	Aidan			5	3										
	Implement Speech-to-Text	5%	needs testing	Aidan					5									
	Test grammatical correction	5%	needs testing	Aidan						3		3						
	Implement grammatical correction	5%	needs testing	Aidan								3						
	Test Text-to-Speech	5%	needs testing	Aidan									3					
	Implement Text-to-Speech	5%	needs testing	Aidan									5	3				
	Selection Text Translation	0%		Jacob										5	3			
General	Project Website Setup	0%													5	10		
	Test Product	0%														5		
FINAL SUBMISSION	Final Report Document	0%															10	
	Final Report Poster	0%															5	
	Video Demonstration	0%															5	
	Presentation Prep	0%																10
	Final Project Submission	0%																3
			Total work hours	178	9	9	12	11	15	26	0	21	11	8	8	15	20	13

Legend
Planned
Delayed
Number Work In Man Hours





Development

Version Control

It all started with syncing up everyone's development environment. Since there was only a couple of us who needed to work on the codebase directly, we thought it wouldn't be too hard. Using GitHub as the repository host and then using git locally on our computers to handle the version control was set up during our first meeting.

Package Manager

Our first challenge came when the test code and application would work on one computer but when another person would pull the code the same code was not working. This is when we learned what a virtual environment was and how you can make one in Python to make sure that you are using the same version of Python. Python 3.9 and 3.10 are completely different and when we first started writing the backend API Python 3.10 was being used. The next thing we needed to implement was a package manager that would make sure all the dependencies that needed to be installed. This was done by using pip and freezing all the requirements to a text file. So when the next person needed to use the packages and have the same setup, all they had to do was pip install the requirements text file, and voila all the dependencies were the same.

Packages

The packages we used were openai, PyQt, Whisper, SpeechRecognition, Numpy, Torch, GTTS, PyDub, and Translate. We used openai for accessing and using a Large Language Model that was close to ChatGPT. At the time of development, ChatGPT's API was not released, and switching to that API upon release would break a lot of our code. PyQt was

used for the GUI and since Python and Qt are cross-platform, it allows our application to be used on Windows, Linux, and MacOS. Whisper and SpeechRecognition allowed us to capture audio from the user and transform their audio into a matrix. To fully implement the Text-To-Speech functionality we needed access to libraries that could handle large matrix calculations and Numpy and Torch give us those tools. Google Text-To-Speech or GTTS was a package that could handle a lot of the actual processing of the text-to-speech for us. From PyDub we used the AudioSegment and playback functions to allow us to cut only a single portion of the user's voice and "play" it to the GTTS. Translate, well, that helped us automatically translate both the bot and the user's language to their native language.

Challenges

Before we built the front end, we made sure that the back end worked. While we didn't have extensive unit tests, as it is hard to test on the logic that is outside the codebase, such as in our case as we would send user input to OpenAI. To test our code, we just set up manual tests in which we would verify that each function was doing what it was supposed to do based on pre-expectations.

The biggest challenge with the back end was multithreading and improving the performance of the app. After we had the initial front end working and had the basic functionality, it became apparent that the app was too slow. After using the mic button, and then sending it to the bot, the application would either freeze for 20-30 seconds, or would crash, and sometimes both. This was due to a multitude of issues but the main pattern of the issue was that different parts of the code were not waiting to continue before another part was finished. So before sending the message, we put the voice recognition and the sending of the bot on two separate threads, and then when the user hit send, after using the voice

recognition, the two tasks would start and finish separately. Then the message would be sent to openai and before displaying something, we would wait for the message to come and then display it.

After implementing this another problem arises, and this is when we learned that in GUI applications, the GUI must run on the main thread, and no other thread is allowed to touch the GUI. This was the reason for a lot of the crashes. To try and mitigate this we tried to import an asynchronous library that would allow the main thread to both send and only display the UI once it got a response from openai. This didn't work out too well as we needed to rewrite the classes and functions to be compatible with the async.io library. What we ended up doing was adding a co-routine, we got an average response time from openai and told the thread that before updating the UI.

Implementing the front end wasn't too bad, PyQt is a great library and comes with a lot of stuff out of the box. All we had to do was make our GUI class inherit from QMainWindow, to treat our class as the main window, and then add children widgets to it. A lot of widgets we used were dropdown menus, Labels, Text Views, and Line edits.

Results

The preliminary results of the language learning desktop application project are promising. We have successfully created a Qt GUI for the app, which is a significant step toward developing a user-friendly and accessible interface. Additionally, the speech-to-text feature has been successfully implemented, allowing users to interact with the app using their voice.

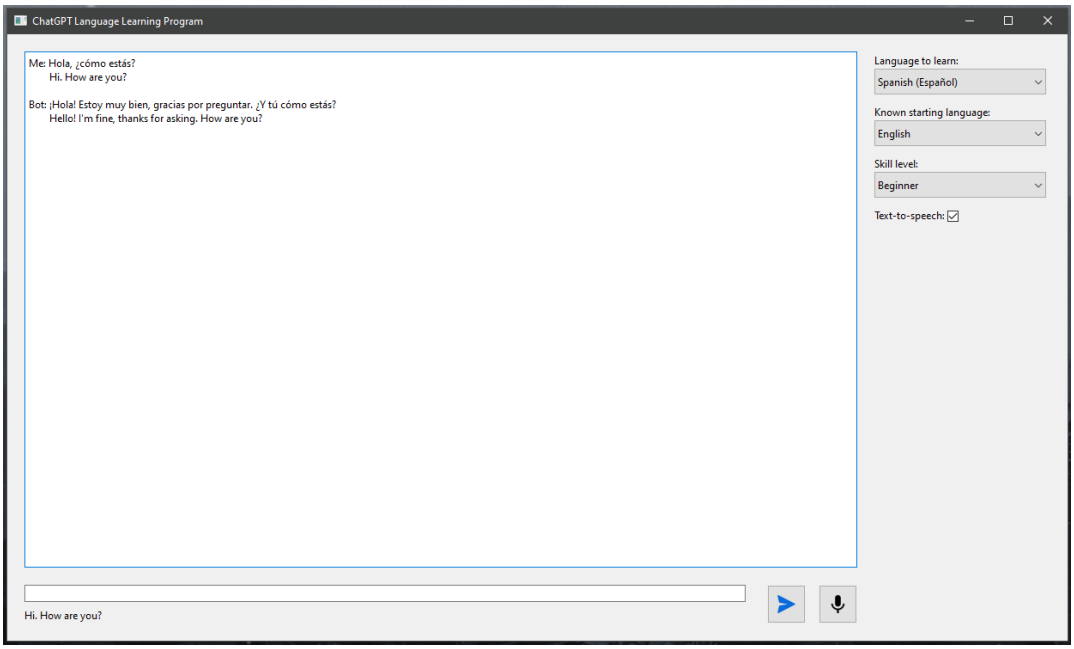


Figure 1.2

The ability to select between three different languages as target and input languages is another positive development, enabling users to customize their learning experience based on their needs and preferences. Furthermore, the translations provided by the app suggest that the natural language processing technology using Chat GPT is functioning as intended, which is crucial for the app's success as a language learning tool. Overall, these preliminary results indicate that the project is on track to achieve its goal of creating an interactive and effective language-learning solution.

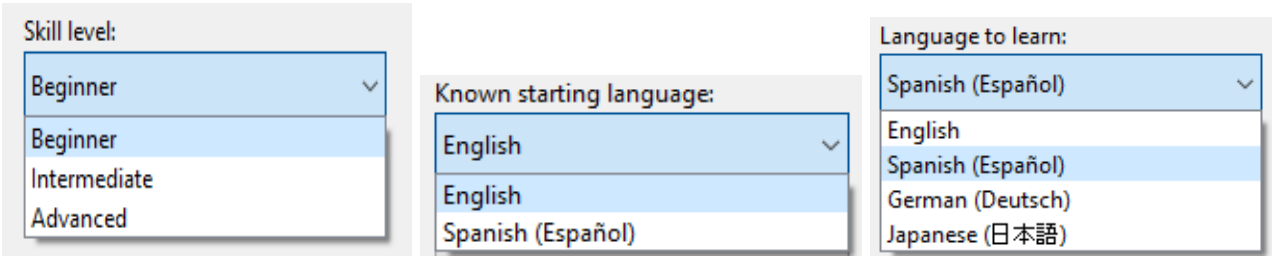


Figure 1.3

Conclusion

By utilizing natural language processing with Chat GPT, the proposed desktop application provides a unique opportunity for users to engage in real-time conversations in various languages, which is a critical aspect of language learning. Additionally, the instant feedback and corrections offered by the chatbot can help users identify and correct errors in their speech, leading to a more effective learning experience. Overall, this project has the potential to create an interactive and immersive language learning tool that can cater to learners of all ages and proficiency levels.