



**NEAR EAST UNIVERSITY
INSTITUTE OF GRADUATE STUDIES
DEPARTMENT OF ARTIFICIAL INTELLIGENCE**

**REAL-TIME VOICE-ACTIVATED CHAT-BOX DIGITAL ASSISTANT USING
ESP32 AND OPENAI API**

M.Sc. THESIS

EriOluwa ODUNLAMI

**Nicosia
June, 2024**

ODUNLAMI ERIOLUWA

**REAL-TIME VOICE-ACTIVATED
CHAT-BOX DIGITAL ASSISTANT
USING ESP32 AND OPENAI API**

MASTER THESIS

2024

NEAR EAST UNIVERSITY
INSTITUTE OF GRADUATE STUDIES
DEPARTMENT OF ARTIFICIAL INTELLIGENCE ENGINEERING

**REAL-TIME VOICE-ACTIVATED CHAT-BOX DIGITAL ASSISTANT USING
ESP32 AND OPENAI API**

M.Sc. THESIS

EriOluwa ODUNLAMI

Supervisor





Professor Fadi AL-TURJMAN

Nicosia

June, 2024


Approval

We certify that we have read the thesis submitted by **EriOluwa ODUNLAMI** titled “ **REAL-TIME VOICE-ACTIVATED CHAT-BOX DIGITAL ASSISTANT USING ESP32 AND OPENAI API**” and that in our combined opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Masters of Artificial Intelligence Engineering.


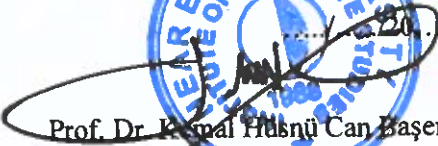
Examining Committee	Name-Surname	Signature
Head of the Committee:	Prof Dr. Fadi Al-Turjman	
Committee Member:	Assoc. Prof. Dr Abdullahi Umar Ibrahim	
Committee Member:	Asst. Prof. Dr Mubarak Auwal Saleh	
Supervisor:	Prof Dr. Fadi Al-Turjman	

Approved by the Head of the Department

...../...../2024


Prof. Dr. Fadi Al-Turjman
Head of Department

Approved by the Institute of Graduate Studies



Prof. Dr. Kemal Hüsni Can Başer
Head of the Institute

Declaration

I hereby declare that all information, documents, analysis and results in this thesis have been collected and presented according to the academic rules and ethical guidelines of Institute of Graduate Studies, Near East University. I also declare that as required by these rules and conduct, I have fully cited and referenced information and data that are not original to this study.

 Odunlami EriOluwa Ayomide

...../...../.....

Day/Month/Year

Acknowledgments

I would like to express my heartfelt gratitude to several individuals who have supported me throughout this project.

First and foremost, I thank God for His guidance and wisdom. I am deeply indebted to my supervisor, Prof. Fadi Al-Turjmann, for his invaluable guidance, expertise, and unwavering support. I also extend my sincere appreciation to my colleague, Sarumi Usman, who has been an immense help throughout this journey.

To my loving parents, Dr. and Mrs. Odunlami, I thank you for your constant encouragement, love, and support.

Lastly, I would like to thank my friends and colleagues who have contributed to my growth and success in various ways.

Odunlami EriOluwa

Abstract**REAL-TIME VOICE-ACTIVATED CHAT-BOX DIGITAL ASSISTANT
USING ESP32 AND OPENAI API****Odunlami, EriOluwa****MS.c, Department of Artificial Intelligence Engineering****June, 2024, 53 pages**

The essence of this thesis is to give comprehensive details of the design and implementation of a voice-activated chatbot system or 'Chat-box'. The chat-box's purpose is simply to record questions from the user and then answer the questions, with the aid of OpenAI API 'speaking' via a speaker. To make this possible, two ESP-32 microcontrollers were used in a master-slave configuration. The Master ESP-32 was connected to a microphone which recorded the questions asked and then the master ESP-32 converts the questions from audio to text. The text is then sent to the slave ESP-32 which then forwards the received question to ChatGPT using OpenAI API, the answers gotten from ChatGPT is then heard from a speaker, hence giving the experience of talking with a smart box. The chat-box was later implemented alongside a robotic head with moving mouth called 'Dux' which gives the feeling of a talking robot.

This system shows how ESP-32 can be applied in conversational Cloud and AI applications. The results show how effective this design is in real-time interactions in different scenarios.

Key Words: Chat-box, Chatbot, ESP-32, OpenAI, ChatGPT, API

Table of Contents

Approval	III
Declaration	IV
Acknowledgments	V
Abstract	VI
Table of Contents	VII
List Of Tables	VIII
List Of Figures	IX
List Of Abbreviations	X
CHAPTER I	1
Introduction	1
1.1 STATEMENT OF THE PROBLEM	1
CHAPTER II	7
Literature Review	7
2.1 Background	7
2.2 Speech-To-Text Technology	7
2.3 ESP-NOW Protocol	8
2.4 ChatGPT	9
2.5 Conclusion	11
CHAPTER III	14
Methodology	14
CHAPTER IV	34
Results Analysis	34
References	41
Appendix A	46
Code	46
Appendix X	60
Similarity Report	60

List Of Tables

Table 1: Table showing available conversational Ais with their accuracy, response time and reference.	9
Table 2: Table comparing accuracy of Chat-box across three scenarios	38
Table 3: Table comparing Average time of response, Audio clarity rating and Quality of response across all three scenarios	39
Table 4: Average Overall Response Time across all three scenarios	40

List Of Figures

Figure 1: Image of ESP32 (Photo credit: https://www.aranacorp.com/en/_trashed/)	10
Figure 2: Flowcharts describing operations of the chat-box.....	18
Figure 3: Microphone box containing master esp32 and microphone.....	19
Figure 4: Chat-box (slave ESP32 circuit with speaker) incorporated with dux-head project.....	28
Figure 5: Wireless microphone using ESP-NOW on breadboard.....	29
Figure 6: MAX98357 circuit diagram with a speaker and microcontroller (Photo credit: https://www.dfrobot.com/product-2614.html).....	29
Figure 7: MAX98357 diagram and label (Photo credit: https://cdn.shopify.com/s/files/1/1509/1638/files/2BreakoutBoardBezeichnung_1024x1024.png?v=1648737062)	31
Figure 8: Chat-box testing phase on breadboard (Slave ESP32-MAX98352 Amplifier-Speaker Connection)	35

List Of Abbreviations

AI:	Artificial Intelligence
App:	Application
DL:	Deep Learning
CNN:	Convolutional Neural Network
VGG:	Visual Geometry Group
ResNet:	Residual Network
ML:	Machine Learning
DOM:	Document Object Model
API:	Application Programming Interface
RNN:	Recurrent Neural Network
CPU:	Central Processing Unit
GPU:	Graphics Processing Unit
WSGI:	Web Server Gateway Interface
GD:	Gradient Descent
SGD:	Stochastic Gradient Descent
Adam:	Adaptive Moment Estimation
GAN:	Generative Adversarial Network
FTP:	File Transfer Protocol
VAR:	Variant
RAM:	Random Access Memory
HTML:	HyperText Markup Language
PHP:	Hyper PreProcessor
SQL:	Structured Query Language
UI:	User Interface
CLI:	Command Line Interface
URL:	Uniform Resource Locator

CHAPTER I

Introduction

In recent years, Artificial Intelligence (AI) has become increasingly prevalent in various industries, revolutionizing the way tasks are automated and executed. One area that stands to benefit from AI technology is education. With the advent of AI assistants, students have the opportunity to access personalized learning experiences and receive immediate feedback on their progress. The development of a student AI assistant chat-box device aims to enhance the educational experience by providing an interactive platform where students can ask questions, seek clarification, and engage in dialogue to deepen their understanding of academic concepts. By leveraging AI technology and integrating it into a physical device, this project seeks to bridge the gap between traditional teaching methods and modern advancements in AI technology, ultimately creating a more dynamic and efficient learning environment for students.

1.1 STATEMENT OF THE PROBLEM

In recent years, the integration of artificial intelligence (AI) technology in educational settings has gained significant momentum. The emergence of chat-box devices as student assistants represents a cutting-edge development in this space. These AI-powered chat-boxes are designed to provide personalized support to students, offering assistance with a wide range of academic tasks and inquiries. By leveraging natural language processing techniques, these chat-boxes can engage students in interactive conversations, deliver instant feedback, and adapt their responses based on individual needs. The implementation of a student AI assistant chat-box device holds immense potential to enhance the learning experience for students, offering a convenient and efficient way to access educational resources and support. As such, the development of these innovative devices aligns with the broader goal of harnessing AI technology to optimize learning outcomes and facilitate a more personalized approach to education. Its incorporation into educational institutions can revolutionize the way students engage with course material and receive academic guidance, ultimately fostering a more dynamic and adaptive learning environment (Krishan Kant Singh Mer et al., 2021-04-23).

1.2 AIMS AND OBJECTIVES OF THE PROJECT

While trying to enhance student learning experiences, the incorporation of artificial intelligence (AI) technologies has become increasingly very common and one particular project that exemplifies this trend involves the development of a student assistant chat-box utilizing key components such as ChatGPT, esp32 microcontroller, a microphone, and a speaker. This innovative device purpose is to provide students with a personalized and interactive tool for academic support and guidance while also removing the distraction of the screen by leveraging the capabilities of ChatGPT for natural language processing and response generation, the chat-box can engage in meaningful conversations with students to address their queries and facilitate learning. Additionally, the integration of esp32, a versatile microcontroller, alongside a microphone and speaker allows for seamless

communication between the chat-box and the user. The best thing about this project is that it showcases how the convergence of AI technologies and IoT devices can revolutionize the way students access educational resources and assistance in a user-friendly and efficient manner.

1.3 SIGNIFICANCE OF STUDY

In the realm of educational settings, the integration of Artificial Intelligence (AI) technology holds immense importance. AI can personalize learning experiences by analyzing individual learning patterns and adapting teaching methods accordingly. This personalized approach can cater to diverse learning needs, ensuring that each student receives tailored support and feedback. Additionally, AI technology can automate administrative tasks, allowing educators to focus more on innovative teaching strategies and student engagement. AI-powered tools can also provide timely and actionable insights into student performance, enabling instructors to intervene promptly when necessary. Moreover, AI can facilitate interactive and engaging learning experiences through virtual tutors, personalized recommendations, and real-time feedback mechanisms. As stated by , the utilization of AI in education has the potential to revolutionize traditional teaching methods and enhance overall educational outcomes, making it a crucial component in modern educational environments.

1.3.1 Context for Implementing Student AI Assistant Chat-box Device

In the realm of educational technology, the integration of Artificial Intelligence (AI) into student learning environments has become increasingly prevalent. AI chat-boxes have emerged as a promising tool to enhance student engagement, provide personalized assistance, and offer immediate feedback. With the rise of remote and hybrid learning models, the need for innovative solutions to support student learning is more critical than ever. Implementing a Student AI Assistant Chat-box Device offers a unique opportunity to address these challenges by creating a virtual companion that can interact with students, answer questions, and guide them through their educational journey. By leveraging the capabilities of AI technologies, educators can enhance the learning experience, promote self-directed learning, and foster a more interactive and personalized learning environment. As Kingsley (Kingsley Okoye et al., 2024-06-07) have highlighted, the integration of AI chat-boxes in education has shown promising results in improving student outcomes and promoting a more dynamic and adaptive learning experience.

1.3.2 Current Educational Landscape

The current educational landscape is increasingly incorporating technology to enhance learning experiences and address the diverse needs of students. With the rise of personalized learning and adaptive technologies, educators are leveraging AI and chat-box devices to provide individualized support and feedback to students in real-time. These devices have the potential to cater to students' specific learning styles, preferences, and pace, ultimately improving their academic performance and engagement. Additionally, the integration of AI chat-boxes in education allows for more efficient communication between students and teachers, facilitating instant access to resources, feedback, and assistance. As educational institutions strive to embrace digital transformation and meet the demands of modern learners, the development of student AI assistant chat-box devices presents a compelling solution to optimize the learning process and promote student success . Moreover, by leveraging cutting-edge technologies such as esp32, microphone, and

speaker, this project aims to revolutionize the way students interact with educational content and support systems (Miao et al., 2021-04-08).

1.4 LIMITATIONS

In traditional learning environments, students often face various challenges that can hinder their academic progress. One common difficulty is the lack of personalized attention from educators, as classes typically have a large number of students, making it challenging for teachers to cater to individual learning needs. This can result in students feeling lost or overlooked, leading to a decline in motivation and engagement with the material. Additionally, the rigid structure of traditional classrooms may not accommodate different learning styles, making it difficult for some students to fully grasp the content. Furthermore, access to resources and support outside of the classroom can be limited, leaving students struggling to find assistance when needed. These challenges underscore the need for innovative solutions, such as AI-powered chat-box devices, to provide personalized support and guidance to students in a more accessible and efficient manner (Miao et al., 2021-04-08).

1.5 Introduction of AI technology as a solution to enhance student learning experiences

As technology continues to advance, it has become increasingly important to leverage innovative solutions in education to enhance student learning experiences. The introduction of Artificial Intelligence (AI) technology has shown great potential in revolutionizing the way students interact with educational content. AI-powered chat-boxes can provide personalized assistance to students, offering immediate feedback, explanations, and resources tailored to their individual needs. By incorporating AI technology into the learning process, educators can create a more engaging and dynamic environment that fosters collaboration, critical thinking, and problem-solving skills. Research has shown that students are more likely to retain information when they are actively engaged in the learning process, and AI-powered chat-boxes have the ability to provide that level of engagement. This innovative approach not only enhances student learning experiences but also prepares them for a future where technology plays an increasingly prominent role in all aspects of society (Sarah Nagle et al., 2022-03-15).

1.6 Motivation for Developing Student AI Assistant Chat-box Device

In today's educational landscape, the integration of technology is becoming increasingly vital in enhancing learning outcomes for students. The motivation behind developing a student AI assistant chat-box device stems from the need to provide personalized and accessible support to learners. By incorporating artificial intelligence capabilities into a chat-box, students can receive immediate responses to their queries, enabling them to engage more actively with their studies. Moreover, the use of such innovative technology can promote independent learning and problem-solving skills among students, preparing them for the challenges of the digital age. Additionally, the student AI assistant chat-box device holds the potential to alleviate the burden on educators by offering timely assistance to students outside of classroom hours, thus fostering a more self-regulated and efficient learning environment. This initiative aligns with the broader objectives of enhancing student learning experiences and promoting technological literacy in education (P. Otero et al., 2022-08-05).

1.7 Enhancing Student Engagement and Learning

In modern educational settings, the concept of enhancing student engagement and learning has become a focal point for educators and researchers alike. By integrating technology into the learning environment, educators aim to provide students with interactive and personalized experiences that cater to their individual needs and learning styles. One promising tool that has emerged in this realm is the utilization of AI chat-boxes as student assistants. These intelligent virtual agents can support students in navigating complex concepts, providing instant feedback, and offering personalized learning recommendations. By incorporating AI chat-boxes into the classroom, educators can create a more dynamic and interactive learning environment that fosters student engagement and motivation. Research has shown that students are more likely to be actively involved in their learning when they are provided with opportunities for interactive and personalized learning experiences. By harnessing the power of AI technology, educators can enhance student engagement and facilitate more effective learning outcomes (Arcangelo Castiglione et al., 2018-09-23).

1.7.1 Explanation of how AI chat-boxes can provide personalized assistance to students

In the realm of educational technology, AI chat-boxes present a promising avenue for enhancing student learning experiences through personalized assistance. By leveraging advanced natural language processing algorithms, AI chat-boxes can tailor their interactions with students based on individual preferences, learning styles, and needs. These chat-boxes can adapt their responses in real-time to provide customized support, guidance, and feedback to students as they navigate educational tasks and challenges. With the ability to analyze vast amounts of data and information, AI chat-boxes can offer personalized recommendations for study materials, resources, and learning strategies that align with each student's unique learning goals. Moreover, AI chat-boxes can assist students in setting personalized learning objectives, tracking their progress, and identifying areas for improvement. Through their adaptive capabilities, AI chat-boxes have the potential to revolutionize the way students engage with educational content and receive support, ultimately enhancing learning outcomes and fostering a more student-centered approach to education (Arcangelo Castiglione et al., 2018-09-23).

1.7.2 Benefits of interactive learning through AI technology

In the context of developing a student AI assistant chat-box device that integrates AI technology to enhance interactive learning experiences, the potential benefits identified through research highlight a paradigm shift in language pedagogy. Leveraging Artificial Intelligence chat-boxes offers a transformative approach to language education by providing immediate, personalized feedback to learners, thereby fostering a supportive learning environment tailored to individual needs and preferences (Konstantinos M. Pitychoutis, 2024). The integration of AI technology in education not only enables personalized learning experiences but also enhances student engagement through interactive platforms. Furthermore, the synergy between AI and Augmented Reality (AR) technologies presents opportunities for visualizations, simulations, and collaborative learning experiences, ultimately deepening students' understanding of complex concepts and promoting active knowledge sharing within the classroom (A. Zouhri et al., 2024).

This interconnectedness between AI technology and interactive learning methodologies underscores the immense potential for enhancing educational experiences and motivating the development of innovative student assistant chat-box devices that respond to the evolving needs of students and educators alike.

1.8 CONCLUSION

In summary, the development of a student AI assistant chat-box device utilizing Chatgpt, esp32, microphone, and speaker has the potential to revolutionize the way students interact with educational resources. By providing a voice-activated tool that can answer questions, provide information, and offer support in real-time, this device offers a personalized learning experience for students. The integration of artificial intelligence technology enables the chat-box to continuously learn and improve its responses, enhancing its effectiveness over time. Additionally, the use of esp32 ensures seamless connectivity and communication, making the device accessible and user-friendly. Overall, this innovative project not only showcases the capabilities of AI technology in education but also highlights the importance of adapting to the evolving needs of learners in the digital age. Further research and development in this area can lead to significant advancements in the field of educational technology, benefiting students and educators alike. (Tareq Ahram and Redha Taiar, 2023-04-13)

In today's fast-paced education system, students often struggle to access personalized assistance and resources tailored to their individual needs. This is where the Student AI Assistant Chat-box Device plays a crucial role. By utilizing artificial intelligence technology, this innovative device can provide students with instant support and guidance, enhancing their learning experience. The chat-box's ability to quickly respond to inquiries, provide relevant information, and offer personalized recommendations can significantly impact a student's academic performance and overall well-being. Furthermore, the convenience of having a virtual assistant available 24/7 can help alleviate the burden on educators and support staff, allowing them to focus on more complex tasks. Ultimately, the Student AI Assistant Chat-box Device not only improves academic outcomes but also fosters a more efficient and effective learning environment for students and educators alike. (Kingsley Okoye et al., 2024-06-07)

The integration of AI technology in educational settings has profound implications for enhancing the learning experience. By incorporating AI-driven tools, educators can personalize learning pathways to cater to individual student needs, thereby promoting a more effective and engaging learning environment. AI technology can also streamline administrative tasks, allowing teachers to allocate more time to student interactions and instructional delivery. Additionally, AI-enabled systems can provide real-time feedback on student performance, enabling timely interventions and personalized support. Through the utilization of AI, educational institutions can harness the power of data analytics to gain insights into student learning patterns and preferences, ultimately driving continuous improvement in instructional strategies and curriculum design. Moreover, the integration of AI technology aligns with the broader trend towards digitalization in education, equipping students with essential 21st-century skills and preparing them for the demands of an increasingly technology-driven society. (Miao et al., 2021-04-08)

As technology continues to advance at a rapid pace, the future implications of utilizing AI assistants for student support are promising. With the ability to access information

instantaneously and provide personalized assistance, AI assistants have the potential to revolutionize the way students learn and interact with educational resources. By leveraging machine learning algorithms and natural language processing capabilities, these AI assistants can adapt to students' individual needs and learning styles, offering tailored guidance and support in real-time. Furthermore, as AI technology evolves, the potential for advancements in student support through AI assistants is limitless. From providing interactive tutoring sessions to offering cognitive-behavioral therapy interventions, the future of AI-driven student support holds great promise in enhancing educational outcomes and improving overall student well-being. The implementation of AI assistants in student support services represents a significant leap forward in the realm of educational technology, paving the way for a more efficient and personalized learning experience for students across various academic disciplines.

CHAPTER II

Literature Review

2.1 Background

In recent times, there has been exponential advancement in the progress of Artificial Intelligence related technology like Speech-to-text for instance. Great progress has been made in terms of quality, how we interact with it, precision and efficiency. For instance, in the case of Speech-to-text technology, noticeable improvements has been noticed based on precision, authenticity and speed thereby increasing its versatility thanks to important algorithms and frameworks of interfaces like Google Cloud speech to text. Large Language Models (LLM) like ChatGPT are emerging and evolving at a scary rate, becoming more smart and human-like over time. If we combine both speech to text and LLM capability with the emergence of ESP-NOW communication protocol, this opens us up to various topnotch applications in various fields like Education (like this project for instance), Medicine, Agriculture etc. We will examine the development and current state of these technologies so that we can gain valuable insights into their application with respect to the Chat-box system.

2.2 Speech-To-Text Technology

Speech-to-text technology is a rapidly advancing technology that simply converts spoken words into text format. Speech-to-text technology has been around for a while and early systems have relied solely on rule-based frameworks and phonetic dictionaries. The results of these methods were decent but not reliable enough to work with. In modern times, Deep learning techniques like transformers and recurrent neural networks (RNN) are used to increase accuracy by handling diverse accents of the speaker, language and cutting out noise environments which affected results of early systems.

Speech-to-text and text-to-speech technologies have really come a long way. There has been a large improvement and advancement in terms of their accuracy, naturalness and latency reduction. For instance, the progress made in terms of naturalness has been significant. To put this in perspective, a research done by Sherman in 1993 shows how hard it takes computers to handle data from videos, let alone speech. Back then, even one second video created so much data that servers could not keep up at the time, they exhausted their available IP packets just for one second video. (Sherman, 1993) but nowadays, modern speech-to-text applications can handle speech quickly without needing as much computer power, making them improve their naturalness, thanks to advancements made in underlying algorithms and improved neural network architectures (Fang Chen et al., 2010-07-01).

Speech-to-text technology has improved greatly in humanizing computer interaction. These technologies can be seen in chatbots, personal assistants, and augmented reality simulations that use text-to-speech to create voices for characters, for example. Google Cloud Speech-to-text has been more influential in ushering this new era of 'more human-like' speech-to-text technologies. The voices sound more natural in the case of text-to-speech technology, the texts can bypass complexity of error correction and understand and interpret (not perfect yet) human speeches more accurately. Early speech-to-text systems

have relied previously on carefully researched guidelines to convert sound into text or vice versa but modern systems take advantage of human expertise to improve the quality of their interpretation with the aid of deep learning models, such as recurrent neural networks (RNNs) and transformer-based models like BERT and GPT-3. A recent study by Fang Chen et al (2010-07-01) further illustrates the use of neural network architectures eg Transformer can improve the performance of speech-to-text medical transcription systems. This techniques reduce noise to the bare minimum and then improves the quality of the output rendered.

Regarding to the Chat-box project, the Speech-to-text function's role is important. It is needed to listen to the question asked by the user and translate it accurately to text format so that it can be forwarded to ChatGPT. The accuracy of the Speech-to-text function is so crucial to the success of the project since ChatGPT needs to get the right question to send out the right answer. Google Cloud Speech-to-text API was used in this project and the accuracy was impressive as shown in the result chapter of this thesis.

2.3 ESP-NOW Protocol

ESP-NOW is a cutting-edge wireless communication protocol developed by Espressif Systems, specifically for IoT applications using ESP microcontrollers. It is known for its high dependability, low power consumption, and secure communication and because of this, ESP-NOW enables smooth and seamless data transmission between IoT devices without requiring traditional Wi-Fi network infrastructure. Because of the lightweight design of ESP microcontrollers, this makes it especially suitable for a lot of IoT devices, enabling fast and efficient device-to-device communication. Researchers show how important secure protocols like ESP-NOW can be in ensuring data integrity and confidentiality in modern IoT systems (Themba Ngobeni et al., 2024).

ESP-NOW works using peer-to-peer communication between two or more ESP microcontrollers, bypassing the need for external routers or Wi-Fi networks. Instead of connecting to a WiFi router, they exchange data directly by connecting using MAC addresses. It is more like an exclusive WiFi connection for ESP microcontrollers. One ESP device initiates the communication channel, and others connect securely in a broadcast or unicast mode. This protocol's low latency and lightweight design make it ideal for real-time applications, like the chat-box system integrating speech-to-text and text-to-speech functions. Its secure data handling and energy efficiency makes it more appealing for IoT deployments.

ESP-NOW has so many diverse applications in IoT use cases ranging from sensor networks, smart home automation, to industrial IoT. An example of its application can be seen its use in decentralized communication setups for smart lighting and home monitoring systems (Wang et al., 2018; Lee et al., 2019). Its ability to handle low-power, high-speed data transmission makes it indispensable and highly valuable in environments requiring robust and reliable connections. Case studies and literature also highlight its role in promoting automation and enhancing real-time interactions in IoT projects.

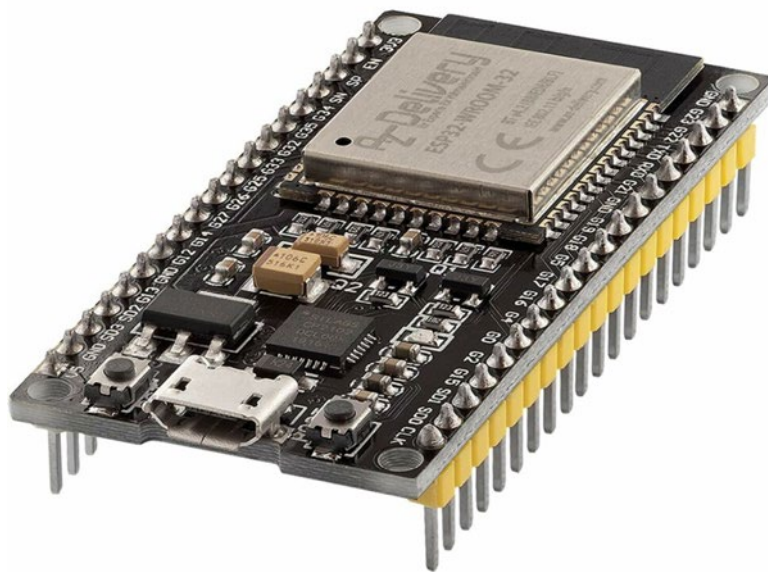


Fig 1: IMAGE OF ESP32 (Photo credit: https://www.aranacorp.com/en/_trashed/)

ESP-NOW has some vital advantages over other connection types like bluetooth and WiFi. First and foremost, it is more secured than bluetooth and WiFi connections. It also boasts of other advantages like low power consumption, reduced latency, and enhanced reliability compared to traditional protocols like Wi-Fi or Bluetooth (Tomonobu Senjyu). However, it has limitations, including a relatively short message size and dependency on ESP32 hardware. Despite these constraints, its simplicity and efficiency make it an optimal choice for IoT applications that prioritize energy-saving and direct device communication like the Chat-box project.

ESP-NOW's simplicity and versatility makes it the most ideal connection type employed in the Chat-box project. The ESP-NOW protocol's purpose is to relay the generated answer from the master ESP32 to the slave ESP32 which is connected to the speaker. It serves as a bridge between the two ESP32s and it makes the design much more flexible.

2.4 ChatGPT

Recently, OpenAI's ChatGPT has been in the forefront of the recent surge or waves of Large Language Models. The launch of ChatGPT opened humanity to the endless possibilities with LLMs. For instance, this recent surge of progress with Natural Language Processing (NLP) when combined with Large Language Models (LLMs) have greatly impacted systems of interaction with examples like ChatGPT. The AI language model developed by OpenAI has proved to be extremely useful in numerous fields such as Education. For instance, the OpenAI's golden goose improves interactions between students and their teachers and also even with their institutions through context awareness,

real-time interaction, and human-like response generation. The integration of ChatGPT into educational systems makes learning more personalized, effective and self-directed.

One thing that makes ChatGPT unique is that it provides a much more multi-dimensional and customized approach which is powered by AI. It enables students to learn information best suited to their personal needs and preference. Students can learn what they want to learn, how they want to learn, where and when they want to learn at their own convenient pace. With its ability to modify complexity based on user-set prompts, it is able to give out optimal information for people with varying levels of understanding. This flexibility in approach makes it helpful for both students who might be struggling and those ahead of their syllabus. ChatGPT is also a great way for people to bypass traditional means of education, making it simple to retrieve data without wasting precious time. Another major application of ChatGPT in the education arena is AI-powered tutoring. It helps learners in different subjects like maths, science, literature etc., and provides step-by-step solutions with explanations. Plus it's able to review essays, summarize academic papers, and assist with research project outlines. The availability of GPT-3 around the clock removes the boundaries of human tutors. This means that students can access a tutor at any convenient time. Language learning is one of the most prominent fields in which ChatGPT shines. Students can practice conversations in English due to its AI-driven technology. ChatGPT can also receive feedback on their pronunciation, and engage in grammar correction exercises that can help refine their language skills. It also enhances multilingual learners' comprehension of various languages due to its real-time translation features. ChatGPT serves as an AI language tutor, helping both novice and more advanced learners hone their communication skills.

ChatGPT can serve as an handy writing assistant for students and researchers by helping with brainstorming, writing, essay structure, summaries of books, and academic writing. ChatGPT can save a lot of time taken in studying research materials by giving accurate summaries and making it easy for the researcher to navigate the academic material. Additionally, it can offer citation recommendations and help with improving writing flow. ChatGPT also has the potential of writing ideas on its own but ethical issues surrounding the over-dependence on AI content shows that critical thinking and originality should dominate the use of ChatGPT in academia.

AI-driven technologies such as ChatGPT is a huge source of help for students with disabilities. Text-to-speech, speech-to-text, and personalized learning assistance are some features that aid accessibility for students with disabilities like visually impaired or dyslexic students. ChatGPT can also help neuro-diverse students by providing customized learning materials best suited for each different cognitive styles. These developments facilitate a more equitable learning experience.

As we all know that every rose has its thorns, it is worth noting that despite the numerous positive impacts from ChatGPT, the ChatGPT's 'rose' comes with its thorns; challenges and limitations which are also present in education. While this has its benefits, ChatGPT poses a problem in education. There are valid concerns about misinformation, biased responses and academic dishonesty (plagiarism to name one) from the AI LLM. Overdependence on AI-generated responses can also weaken students' critical and problem-solving skills which is counterproductive to the values of academia. Teachers need to lead students toward the responsible use of ChatGPT, and make sure it acts as a supplement rather than a replacement for traditional learning methods. AI offers several

advantages to education and can be defined as the future of it. ● AI Software Program Interface adapt learning to a personal level. Yet still, enough attention should be given to ensure that a balance is kept between AI-enabled and human-led education for a wholesome learning experience.

ChatGPT has taken education, whether it is about personalized learning, AI-based tutoring, language analysis, research assistance, or accessibility, to a never before seen level. Although the implementation of ChatGPT in education presents many advantages, it also contributes to challenges that needs meticulousness. Through responsible AI usage, ethical academic practices, and adequate human supervision, educators can harness the potential of ChatGPT while also maintaining educational integrity. While the potential of AI grows, so must our will to employ it to complement, not compete with, the existing educational scaffolds.

2.5 Conclusion

Recent studies in the field of interactive chat-box systems have shown a growing interest in incorporating advanced technologies such as natural language processing and artificial intelligence to enhance user experiences. These studies have highlighted the importance of improving the responsiveness and efficiency of chat-box systems by utilizing speech-to-text conversion and text-to-speech synthesis capabilities. However, there is a noticeable gap in the literature regarding the development of a seamless and integrated chat-box system using multiple microcontrollers like the ESP32. This project aims to address this gap by leveraging the ESP32's capabilities to create a more sophisticated and dynamic chat-box system. By integrating Speech-to-Text conversion, ESP-NOW communication, and ChatGPT response generation, this project seeks to enhance user interactions with the chat-box system. Through this research, we aim to contribute to the field by providing a more responsive and efficient chat-box system that is user-centered and technologically innovative.

Recent research has underscored the growing significance of interactive chat-box systems in various domains, highlighting their potential to enhance user engagement and streamline communication processes. Innovations in the fields of natural language processing, machine learning, and artificial intelligence have enabled the development of more sophisticated chat-box systems capable of understanding and generating human-like responses. However, existing literature suggests that there remains a gap in designing chat-box systems that seamlessly integrate speech-to-text conversion with intelligent response generation in real-time (Leah A Lievrouw et al., 2006-01-17). This is where the current project seeks to make a valuable contribution by leveraging ESP32 microcontrollers to create a responsive and efficient chat-box system. By combining speech recognition technology with advanced language models like ChatGPT, the aim is to enhance user interaction by providing more accurate and contextually relevant responses. Through this innovative approach, the project aims to address the limitations of existing chat-box systems and offer a more user-friendly and intuitive communication experience for users.

Previous research has widely explored the development of interactive chat-box systems, focusing on enhancing user experience and efficiency. Recent papers have highlighted the integration of machine learning algorithms to improve response accuracy and speed, as well as the utilization of natural language processing to enhance conversational capabilities. However, a gap in the existing research lies in the seamless integration of speech-to-text

and text-to-speech technology within chat-box systems, presenting a challenge for creating truly interactive and efficient platforms. This project aims to address this gap by utilizing two ESP32 microcontrollers to enable real-time communication and response generation, significantly advancing the field of interactive chat-box systems. By building upon current knowledge and enhancing the capabilities of existing technologies, this project seeks to create a more responsive and user-friendly chat-box system that offers a seamless conversational experience for users.

Table 1: Table showing available conversational AIs with their accuracy, response time and reference

Name of AI	Type of AI	Accuracy and Response time	Reference
Jibo	Conversational AI	Accuracy: 90% (claimed by manufacturer) Response time: 2-3 seconds	Jibo (n.d.), 2020
Kuri	Conversational AI	Accuracy: No publicly available data available Response time: No publicly available data available	Mayfield robotics Kuri
Alexa	Virtual Assistant Speaker	Accuracy: 95%, Response time: 1.5 seconds	Loup Ventures (2020) Alexa Blogs (2019)

Google Assistant	Voice Assistant	Accuracy: 95% Response time: 2.1 seconds	Stone Temple (2020)
Siri	Voice Assistant	Accuracy: 83.1% Response time: 2.5 seconds	Apple Support (2020)
Chat-box	Conversational AI	Accuracy: 93.8 % Response time: Varies based on network	Odunlami EriOluwa (2024)

CHAPTER III

Methodology

The aim of this project is to build a chat-box, a device that can help us communicate with ChatGPT and get a response. In order to achieve this, the device will receive audio from the user, transcribe the audio into text, forward the text as a query to ChatGPT via an OpenAI API key. The response from ChatGPT is then converted from text to an audio file which is played from the speaker, thereby becoming the chat-box. To get this done, two ESP-32s were employed to split the functions. The full list of components used is listed below:

1. Amplifier (MAX98357)
2. Microphone (INMP441-MEMS-I2S-MIC-MODULE)
3. Two ESP 32 (NODEMCU-32S)
4. Speaker
5. LEDs (3 pieces)
6. 1000 ohms resistor
7. Button

In order to achieve the objective of this project, three milestones must be achieved and they are; Speech-to-text function, ESP-NOW function and the Text-to-speech function. To bring this chat-box project to fruition, it is essential that focus is directed on the integration and seamless interaction between two ESP-32 microcontrollers. Each device plays a crucial and unique role in the overall functionality of the chat-box, ensuring efficient data processing and communication. The first ESP-32's job is basically to capture 'speech' (audio) with the aid of a microphone and converting the captured audio into text, taking advantage of the functions and capabilities of Google Cloud's Speech-to-Text API. This step is vital for enabling the chat-box to understand and process verbal inputs from users, thus forming the foundation for subsequent interactions.

The 'text' generated by the first ESP-32 is not the final result needed. The generated text is still the question that needs to be answered. That is where the second ESP-32 microcontroller comes in and in order to get the question to the second ESP-32, means of communication is needed to send text from the first ESP-32 to the other ESP-32 and to achieve that, ESP-NOW function was introduced. The robust communication link established between the two ESP-32 devices using ESP-NOW protocol allows for quick and reliable data transfer, ensuring that the text generated from the speech input is promptly transmitted to the second device. This efficient communication channel is crucial for maintaining a responsive and interactive user experience, as it offers range, minimizes delays and potential data loss, thereby enhancing the overall performance of the chatbot.

Finally, the second ESP-32 device takes the received question and interacts with the ChatGPT API to generate a meaningful response or reply. This response is then converted into speech using a text-to-speech library, completing the conversational loop. This final step not only demonstrates the integration of advanced AI capabilities with hardware components but also highlights the importance of delivering a natural and engaging audio response to the user. By focusing on these interconnected aspects, the project aims to create a sophisticated and user-friendly chatbot system.

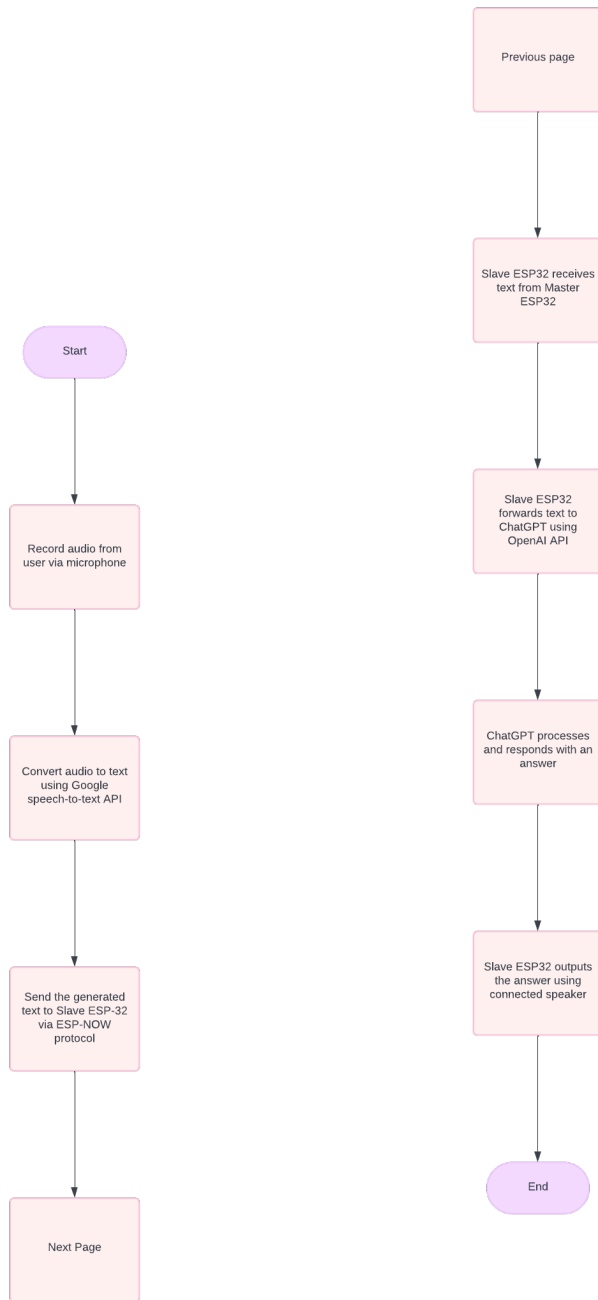


Fig 2: FLOWCHARTS DESCRIBING OPERATIONS OF THE CHAT-BOX

3.1 Speech To Text Function

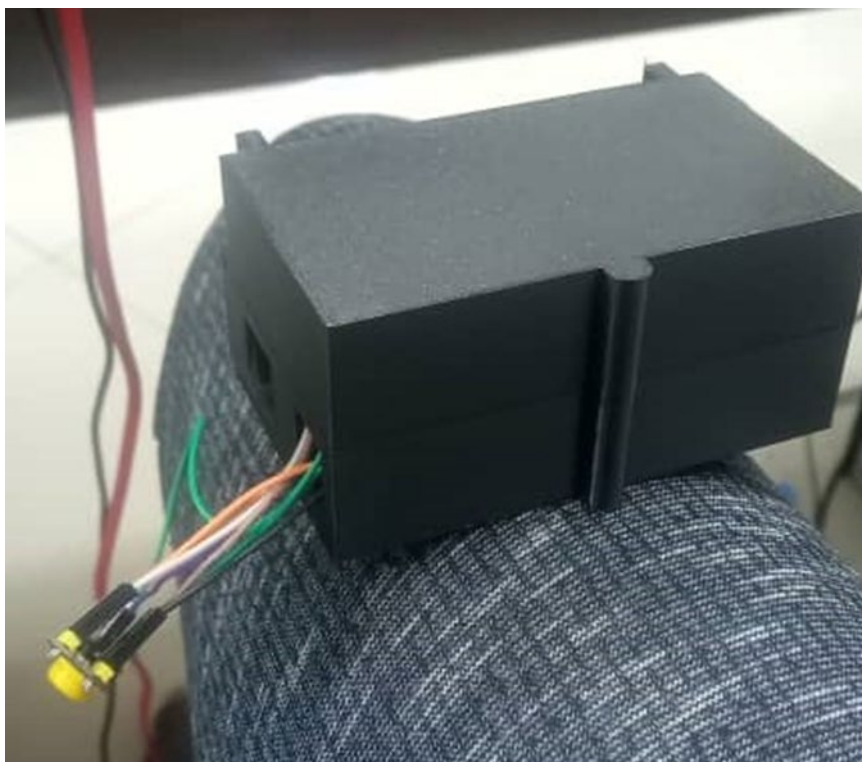


Fig 3: MICROPHONE BOX CONTAINING MASTER ESP32 AND MICROPHONE

For the speech-to-text section, the two major components used are ESP-32 (NODEMCU-32S) and a microphone (INMP441-MEMS-I2S-MIC-MODULE). As described above, the microphone receives audio which is converted to audio with the aid of Google Cloud Services.

3.1.1 Hardware Connection

The hardware connection is quite simple. The microphone (INMP441-MEMS-I2S-MIC-MODULE) has six pins; L/R, WS, SCK, SD, VDD and GND pins. The INMP441 is a high-performance, low-power, digital output, omnidirectional MEMS microphone. It communicates with the aid of the I2S (Inter-IC Sound) protocol. The VDD and L/R pins are connected together and then connected to the 3v3 pin of the ESP-32 microcontroller which supplies the microphone 3.3 volts. GND pin of the microphone goes to any ground pin of the ESP-32 to complete the power supply circuit of the speech-to-text section. Here are the functions of the remaining primary pins:

- WS (Word Select) Pin:
 - Also known as LRCLK (Left-Right Clock).
 - This pin determines which channel is currently active, either the left or the right.
 - It acts as a frame synchronization signal, indicating the start of a new audio frame.
 - When WS is low, data on the SD line corresponds to the left channel. When WS is high, data on the SD line corresponds to the right channel.
 - The WS pin is connected to the D22 pin of the ESP-32 microcontroller in this case.

- SCK (Serial Clock) Pin
 - Also known as BCLK (Bit Clock).
 - This pin provides the clock for the data transfer.
 - Each bit of data on the SD line is clocked out on each rising or falling edge of the SCK signal, depending on the I2S mode used.
 - It synchronizes the data transmission between the microphone and the receiving device.
 - The SCK pin is connected with the D26 pin of the ESP-32 microcontroller.
- SD (Serial Data) Pin
 - This pin is responsible for the actual audio data.
 - The data is relayed in a serial format, with the bits being clocked out on the SCK pin.
 - It alternates between the left and right audio data, controlled by the WS pin.
 - The SD pin is connected to the D34 pin of the ESP32 microcontroller.

In summary, the WS pin is used to select the channel (left or right), the SCK pin is the clock for data transmission (timer of the microphone), and the SD pin is responsible for the digital audio data. These three pins work together to transmit audio data from the INMP441 microphone to a microcontroller or another I2S-compatible device.

Two LEDs and a reset button were later added to serve as the User Interface of the circuit. The button turns the device on, the LED determines when to record the question and when the question has reached the slave ESP-32 microcontroller. With the hardware connections all taken care of, all that is left for the successful operation is the code implementation which is explained in the next section.

3.1.2 Code Implementation

The coding of the ESP-32 is done with the aid of the Arduino IDE using C++ language. The code used here is basically the combination of the ESP-NOW master's code from the ESP-32 library on Arduino IDE with the Speech-to-text code (by Techiesms on github). The code was split into eight groups, one main code and seven implementation codes. The main code's operation can be broken down into three major components; Initialization, Audio Processing and Data Transmission.

a) Initialization

++ (InitESPNow() Function):

In this stage, the networks required, both the WiFi and ESP-NOW networks are initialized. The ESP-NOW network is first initialized by disconnecting the ESP-32 from any other WiFi network to avoid interference as shown in the code below;

```
void InitESPNow() {
  WiFi.disconnect();
  if (esp_now_init() == ESP_OK) {
    Serial.println("ESPNow Init Success");
  } else {
```

```

    Serial.println("ESPNow Init Failed");
    ESP.restart();
}
}

```

++ Scanning for Slaves (ScanForSlave() Function):

After initializing the ESP-NOW network, the ESP-32 scans for WiFi networks to find the slave device by looking for SSIDs that start with "Slave". The Master ESP-32 searches for WiFi networks in AP mode. The slave ESP-32 SSID has already been set to be 'slave-1' to make it easy for the master ESP-32 to locate it. After locating the slave ESP-32, the MAC address of the slave ESP-32 microcontroller is extracted and stored as shown in the code below;

```

void ScanForSlave() {
    int8_t scanResults = WiFi.scanNetworks();
    bool slaveFound = 0;
    memset(&slave, 0, sizeof(slave));

    Serial.println("");
    if (scanResults == 0) {
        Serial.println("No WiFi devices in AP Mode found");
    } else {
        Serial.print("Found "); Serial.print(scanResults); Serial.println(" devices ");
        for (int i = 0; i < scanResults; ++i) {
            String SSID = WiFi.SSID(i);
            int32_t RSSI = WiFi.RSSI(i);
            String BSSIDstr = WiFi.BSSIDstr(i);

            if (PRINTSCANRESULTS) {
                Serial.print(i + 1);
                Serial.print(": ");
                Serial.print(SSID);
                Serial.print(" (");
                Serial.print(RSSI);
                Serial.println(")");
            }
            delay(10);
            if (SSID.indexOf("Slave") == 0) {
                Serial.println("Found a Slave.");
                Serial.print(i + 1); Serial.print(": "); Serial.print(SSID); Serial.print(" [");
                Serial.print(BSSIDstr); Serial.print("]"); Serial.print(" ("); Serial.print(RSSI);
                Serial.println(")");
                int mac[6];
                if (6 == sscanf(BSSIDstr.c_str(), "%x:%x:%x:%x:%x:%x", &mac[0], &mac[1],
                &mac[2], &mac[3], &mac[4], &mac[5])) {
                    for (int ii = 0; ii < 6; ++ii) {
                        slave.peer_addr[ii] = (uint8_t) mac[ii];
                    }
                }
                slave.channel = CHANNEL;
                slave.encrypt = 0;
            }
        }
    }
}

```

```

        slaveFound = 1;
        break;
    }
}
}

if (slaveFound) {
    Serial.println("Slave Found, processing..");
} else {
    Serial.println("Slave Not Found, trying again.");
}

WiFi.scanDelete();
}

```

++ Managing Slave Connection (manageSlave() Function):

The next lines of code checks if the slave device is already paired with the master ESP-32. If the slave device is not already connected, it tries to make sure the devices pair together. Different error messages are programmed also to aid in troubleshooting errors in the pairing process.

```

bool manageSlave() {
    if (slave.channel == CHANNEL) {
        if (DELETEBEFOREPAIR) {
            deletePeer();
        }

        Serial.print("Slave Status: ");
        bool exists = esp_now_is_peer_exist(slave.peer_addr);
        if (exists) {
            Serial.println("Already Paired");
            return true;
        } else {
            esp_err_t addStatus = esp_now_add_peer(&slave);
            if (addStatus == ESP_OK) {
                Serial.println("Pair success");
                return true;
            } else if (addStatus == ESP_ERR_ESPNOW_NOT_INIT) {
                Serial.println("ESPNow Not Init");
                return false;
            } else if (addStatus == ESP_ERR_ESPNOW_ARG) {
                Serial.println("Invalid Argument");
                return false;
            } else if (addStatus == ESP_ERR_ESPNOW_FULL) {
                Serial.println("Peer list full");
                return false;
            } else if (addStatus == ESP_ERR_ESPNOW_NO_MEM) {
                Serial.println("Out of memory");
                return false;
            } else if (addStatus == ESP_ERR_ESPNOW_EXIST) {

```

```

    Serial.println("Peer Exists");
    return true;
  } else {
    Serial.println("Not sure what happened");
    return false;
  }
}
} else {
  Serial.println("No Slave found to process");
  return false;
}
}
}

```

++ After confirming the ESP-NOW connection, the WiFi connection is set to stationary mode in the void setup section of the code, The WiFi connection part of the code is handled by the CloudSpeechClient.cpp implementation code as shown below. The ESP-32 scans for the WiFi network whose details (SSID and password) are given by the network-param.h implementation code.

```

CloudSpeechClient::CloudSpeechClient(Authentication authentication) {
  this->authentication = authentication;
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) delay(1000);
  client.setCACert(root_ca);
  if (!client.connect(server, 443)) Serial.println("Connection failed!");
}

```

b) Audio Processing and Transcription

++ Recording and Transcribing Audio:

The recording of the audio is achieved with the aid of four of the implementation codes (i2s.cpp, i2s.h, audio.cpp, audio.h). The I2S.cpp and I2S.h implementation codes initialize the interface based on the type of microphone used, the audio from the microphone is also read by the I2S.cpp and I2S.h implementation codes. The audio.cpp and audio.h implementation codes manages the audio gotten through the I2S interface, allocates memory on the ESP-32 for the audio data, generates a WAV file header and processes the audio data into a format suitable for a WAV file.

The recorded audio is then sent to Google Cloud's Speech-to-text API for transcription. This is handled by the CloudSpeechClient.cpp and CloudSpeechClient.h implementation codes. The audio and cloud speech client objects are deleted after processing to avoid overwhelming the ESP-32 microcontroller.

The code below simply shows line of codes from the main code that activates the operation of the implementation codes. The code below also shows serial monitor comments and LED pins set such a way to help the user know when to record his message.

```

Serial.println("\r\nRecord start!\r\n");
digitalWrite(15, HIGH);
digitalWrite(2, LOW);
Audio* audio = new Audio(ICS43434);

```

```

audio->Record();
Serial.println("Recording Completed. Now Processing...");
digitalWrite(15, LOW);
digitalWrite(2, HIGH);
CloudSpeechClient* cloudSpeechClient = new CloudSpeechClient(USE_APIKEY);
cloudSpeechClient->Transcribe(audio);
delete cloudSpeechClient;
delete audio;

```

c) Data Transmission

++ sendData() Function:

The transcribed text ('My_Answer') is converted to a C-style string. The text is then sent to the slave ESP-32 using ESP-NOW. Messages are placed to review the status of the send operation handling different error conditions as shown in the code below;

```

void sendData() {
    const char* data = My_Answer.c_str();
    const uint8_t *peer_addr = slave.peer_addr;
    Serial.print("Sending: "); Serial.println(data);
    esp_err_t result = esp_now_send(peer_addr, (uint8_t *)data, strlen(data) + 1);
    Serial.print("Send Status: ");
    if (result == ESP_OK) {
        Serial.println("Success");
    } else if (result == ESP_ERR_ESPNOW_NOT_INIT) {
        Serial.println("ESP NOW not Init.");
    } else if (result == ESP_ERR_ESPNOW_ARG) {
        Serial.println("Invalid Argument");
    } else if (result == ESP_ERR_ESPNOW_INTERNAL) {
        Serial.println("Internal Error");
    } else if (result == ESP_ERR_ESPNOW_NO_MEM) {
        Serial.println("ESP_ERR_ESPNOW_NO_MEM");
    } else if (result == ESP_ERR_ESPNOW_NOT_FOUND) {
        Serial.println("Peer not found.");
    } else {
        Serial.println("Not sure what happened");
    }
}
}

```

++ OnDataSent() Callback:

The callback is handled after sending data to verify and log the delivery status.

```

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
    char macStr[18];
    snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
             mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4],
             mac_addr[5]);
    Serial.print("Last Packet Sent to: "); Serial.println(macStr);
    Serial.print("Last Packet Send Status: "); Serial.println(status ==
    ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

```

}

In conclusion, the main code effectively sets up an ESP-32 as a master device that it:

- 1) Initializes ESP-NOW for communication.
- 2) Scans for and pairs with the slave ESP-32.
- 3) Captures the audio input, processes it and transcribes the audio using Google Cloud's Speech-to-Text service.
- 4) Sends the transcribed text to the slave ESP32 using ESP-NOW.
- 5) Handles communication and ensures successful data transmission.

This setup is essential for the first part of the project, allowing the first ESP32 to get, process and transcribe the audio, then send the resulting text to the slave ESP32 for further handling and response generation. The flowchart below further explains the workings of the Speech-to-text code.

flowchart TD

```

Start --> InitSerial
InitSerial[Initialize Serial Communication] --> SetupGPIO
SetupGPIO[Set Up GPIO Pins] --> InitWiFi
InitWiFi[Initialize WiFi in Station Mode] --> InitESPNow
InitESPNow[Initialize ESP-NOW] --> CheckESPNow
CheckESPNow{ESP-NOW Init Success?}
CheckESPNow -- Yes --> RegisterCallback
CheckESPNow -- No --> RestartESP32
RestartESP32[Restart ESP32] --> InitESPNow
RegisterCallback[Register ESP-NOW Send Callback] --> BeginSpeechRec
BeginSpeechRec[Begin Speech Recognition Process] --> RecordAudio
RecordAudio[Record Audio] --> TranscribeAudio
TranscribeAudio[Transcribe Audio Using Google Cloud] --> ScanSlave
ScanSlave[Scan for Slave Devices] --> CheckSlaveFound
CheckSlaveFound{Slave Found?}
CheckSlaveFound -- Yes --> ManageSlave
CheckSlaveFound -- No --> PrintNoSlave
PrintNoSlave[Print "No WiFi devices in AP Mode found"] --> End
ManageSlave[Manage Slave Connection] --> CheckPairSuccess
CheckPairSuccess{Pairing Successful?}
CheckPairSuccess -- Yes --> SendData
CheckPairSuccess -- No --> PrintPairFailed
PrintPairFailed[Print "Slave pair failed!"] --> End
SendData[Send Transcribed Text via ESP-NOW] --> PrintSendStatus
PrintSendStatus[Print Send Status] --> End
OnDataSent[On Data Sent Callback] --> PrintMACStatus
PrintMACStatus[Print MAC Address and Delivery Status] --> End
End[End]

```

3.2 ESP-NOW Function

ESP-NOW communication protocol is a crucial part of the project. It serves as an essential bridge between the two ESP-32 microcontrollers helping them complement each

other. The Speech-to-text section above showed how the ESP-NOW connection is integrated to the code but this section sheds more light on the essence of ESP-NOW communication.

ESP-NOW is a connectionless communication protocol designed by Espressif, made most especially for ESP32 and ESP8266 microcontrollers. It allows for low-latency, peer-to-peer wireless communication between the microcontrollers. In the case of the chat-box project, ESP-NOW ensures the communication between two ESP32 microcontrollers: one acting as the master ESP-32 microcontroller (which gets, processes and transcribes the speech) and the other as the slave ESP-32 microcontroller (responsible for relaying the transcribed text to ChatGPT and converting ChatGPT response to audio).

3.2.1 Key Steps in ESP-NOW Communication

1. Initialization of ESP-NOW:

Both the master and slave ESP-32 devices initialize the ESP-NOW protocol. This involves setting up the WiFi of the ESP-32 microcontrollers in station mode and initializing the ESP-NOW library to handle communication. Initialization is crucial for the devices to proceed with other functions of the code.

2. Device Discovery:

The master ESP32 scans for available WiFi networks to discover the slave device. The slave ESP32 broadcasts its presence by doing two things; operating in AP (Access Point) mode and also using a recognizable SSID (in this case, starting with "Slave"). This makes it easier for the master to identify and select the correct slave device for pairing.

3. Peer Information Setup:

Once the slave device is identified, the master ESP-32 extracts the MAC address of the slave and saves it, which is very important for direct communication between them. The master then prepares an `esp_now_peer_info_t` structure that includes the slave ESP-32's MAC address, communication channel, and encryption status.

4. Pairing Devices:

The master ESP-32 attempts to pair with the slave by adding it as a peer using the ESP-NOW protocol. Pairing ensures a trusted communication link between the two devices. This step ensures that the master can send data to the slave without any form of interruptions or errors.

5. Data Transmission:

The master ESP-32 microcontroller captures the speech (audio input) from a microphone and processes it using Google Cloud's Speech-to-Text service (by using `CloudSpeechClient.cpp` and `CloudSpeechClient.h` implementation code) to convert the spoken words of the user into text. This text data is then sent to the slave ESP32 via ESP-NOW communication protocol. The master formats/edits the text data into a suitable message structure and sends it to the slave's MAC address.

6. Receiving Data:

The slave ESP32, configured to listen for incoming ESP-NOW messages, receives the text data sent by the master. Upon receiving the data, the slave processes it and prepares a response using the OpenAI API. The response text is then converted back into speech using Google Cloud's Text-to-Speech service.

7. Handling Callbacks:

Both the master and slave ESP32 microcontrollers make use of callback functions to handle various events such as confirming successful message delivery, message reception, and detecting errors. These callbacks provide feedback on the communication status between the ESP-32 microcontrollers, allowing the devices to react accordingly (for instance, retrying transmission if it fails).

3.2.2 Workflow of the ESP-NOW Communication on the Two Devices

Master ESP32:

- Initializes ESP-NOW.
- Scans for the slave ESP-32.
- Extracts the slave ESP-32 MAC address.
- Pairs with the slave ESP-32.
- Captures audio input and converts it to text.
- Sends the text data to the slave using ESP-NOW.
- Handles callback events to confirm successful data transmission or retry if necessary.

Slave ESP32:

- Initializes ESP-NOW.
- Broadcasts its presence in AP mode for discovery.
- Receives pairing request from the master and pairs.
- Listens for incoming text data from the master.
- Processes the received text using the OpenAI API.
- Converts the response text to speech.
- Handles callback events to confirm message receipt and process incoming data.

3.2.3 Advantages of Using ESP-NOW

1. Low Latency:

ESP-NOW allows for near-instantaneous data transmission between the ESP-32s, which is important and valuable for real-time applications like a chat-box.

2. Low Power Consumption:

The ESP-NOW protocol consumes relatively low power since it is designed to be power-efficient, making it suitable for battery-operated devices.

3. Connectionless Communication:

The ESP-32 microcontrollers communicate directly independent (without the need for) of a router or access point, making the network setup much more simplified and reducing potential chances of failure.

4. Range:

The ESP-NOW connection offers range and flexibility to the design of the chat-box.

3.3 Text-To-Speech Function



Fig 4: CHAT-BOX (SLAVE ESP32 CIRCUIT WITH SPEAKER) INCORPORATED WITH DUX-HEAD PROJECT

The question has been converted from audio to text thanks to the Master ESP-32 microcontroller. The question needs to be processed, sent to ChatGPT API and the response gotten should be converted to audio which is then heard from a speaker. To achieve this, a slave ESP-32 microcontroller is connected to the amplifier (MAX98357) which is later connected to the speaker. An LED is connected to indicate when the text reaches the slave ESP-32 microcontroller and when the ChatGPT response has been converted to audio. More details about the hardware section of the design are discussed below;

3.3.1 Hardware Connection



Fig 5: WIRELESS MICROPHONE USING ESP-NOW ON BREADBOARD

The hardware connection involves simply connecting the MAX98357 amplifier to the ESP32 microcontroller, followed by connecting the speaker to the amplifier. This setup allows the digital audio signals processed by the ESP32 to be amplified and output through the speaker, facilitating clear and powerful sound for the chat-box project.

The MAX98357 Amplifier is a digital pulse-code modulation (PCM) input amplifier, that is often used in audio applications to drive (power) speakers. In the context of the chat-box project, the MAX98357 amplifier is connected to the slave ESP-32, which processes the received text data (from the master ESP32) and converts it to speech. The amplifier then drives a speaker to output the audio.

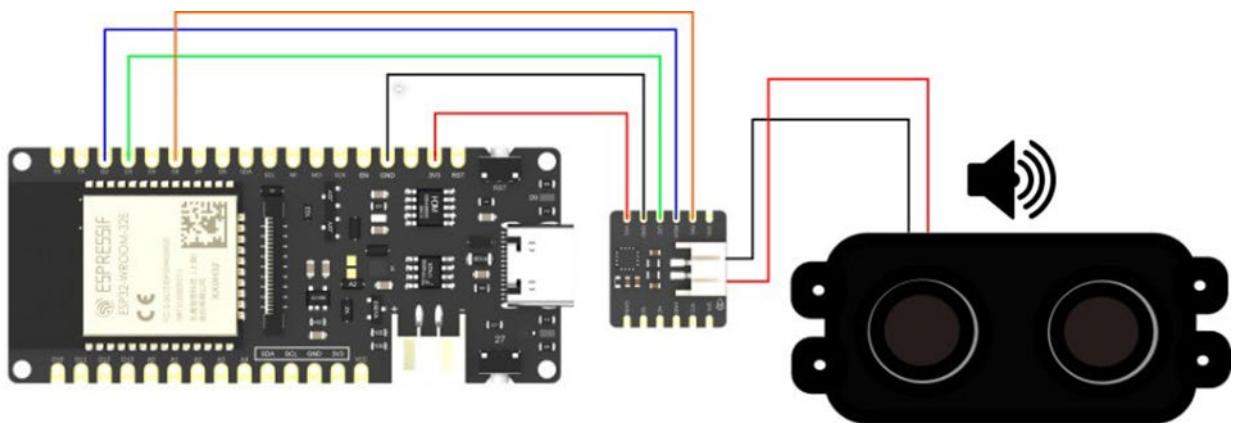


Fig 6: MAX98357 CIRCUIT DIAGRAM WITH A SPEAKER AND MICROCONTROLLER (Photo credit: <https://www.dfrobot.com/product-2614.html>)

Key Features of MAX98357:

- **Class D Amplification:** This is the reason why the MAX98357 amplifier is very efficient and why it consumes low amount of power.
- **I2S Audio Interface:** This feature directly interfaces with microcontrollers (for instance, ESP-32 microcontroller in this case) and digital audio sources.
- **Integrated Digital Signal Processing (DSP):** This feature improves and enhances the quality of the audio.
- **Low Quiescent Current:** The MAX98357 Amplifier consumes very low current and this makes it suitable for battery-powered applications.
- **Shutdown and Mute Control:** This feature allows for power management and aids in audio control.

Primary Pins and Their Connections:

1) **VIN (Power Supply Voltage):**

This pin receives power supply for the amplifier. It is connected to the 3v3 pin of the ESP-32 microcontroller which implies that it is being powered with 3.3 volts.

2) **GND (Ground):**

This is the other half of the power supply pin, this is the ground pin of the amplifier and it is connected to the ground pin of the ESP-32 microcontroller to complete the power supply circuit.

3) **SD (Shutdown) pin:**

The essence of this pin is to control the shutdown mode of the amplifier. Putting this pin low puts the amplifier into a low-power shutdown state. In this circuit, the SD pin is not needed.

4) **GAIN (Gain Control):**

This pin's function is to select the gain of the amplifier. For maximum audio sound, the gain pin was connected to the 3v3 pin of the ESP-32 microcontroller.

5) **DIN (Data Input):**

This pin receives the digital audio data in I2S format from the ESP-32 microcontroller. It is connected to output pin D25 of the ESP-32 (the pin was declared in the code of the microcontroller).

6) **BCLK (Bit Clock):**

For successful synchronization of the data, this pin receives the bit clock signal from the microcontroller. It is connected to the input pin D27 of the ESP-32 microcontroller.

Note: The input pin was declared in the code of the microcontroller.

7) **LRCLK (Left-Right Clock):**

This pin's job is to receive the left-right clock signal from the microcontroller, which shows the channel (left or right) of the audio data. It is connected to the input pin D26 of the ESP-32 microcontroller.

Note: The input pin was declared in the code of the microcontroller.

8) OUT+ (Positive Speaker Output):

The amplified audio signal is sent through this pin to the positive pin of the speaker since it is connected to the positive terminal of the speaker.

9) OUT- (Negative Speaker Output):

The amplified audio signal is sent through this pin to the positive pin of the speaker since it is connected to the positive terminal of the speaker.

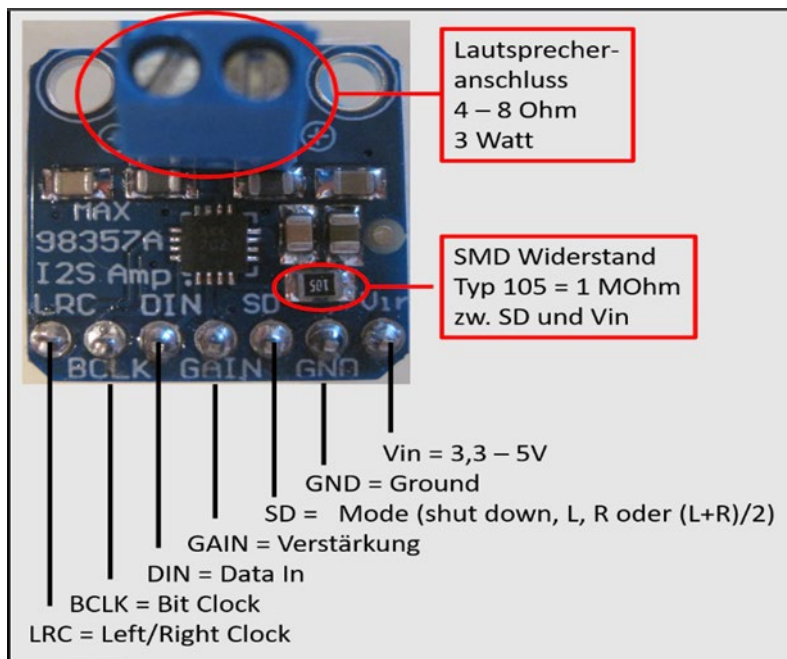


Fig 7: MAX98357 DIAGRAM AND LABEL (Photo credit: https://cdn.shopify.com/s/files/1/1509/1638/files/2BreakoutBoardBezeichnung_1024x1024.png?v=1648737062)

3.3.2 Code Implementation

This code shows in details the operation of the slave ESP-32 microcontroller, which receives questions from the master ESP-32 microcontroller, forwards them to the ChatGPT API for processing, and then converts the response to speech. Here's an analysis of how the code works, section by section:

Setup and Initialization

1) Libraries and Definitions:

The code includes the important libraries for WiFi, ESP-NOW, HTTP requests, and audio handling (Audio.h) as shown in the 'include' section of the code below;

```
#include <Arduino.h> // Provides basic Arduino functions.
#include <esp_now.h> // Manages ESP-NOW communication.
#include <WiFi.h> // Manages WiFi functionality.
#include <HTTPClient.h> // Allows for HTTP/HTTPS requests.
#include <ArduinoJson.h> // Handles JSON parsing.
#include "Audio.h" // Manages audio playback.
```

I2S pins are also defined here for audio output as shown in the section of the code below;

```
#define CHANNEL 1
#define I2S_DOUT 25
#define I2S_BCLK 27
#define I2S_LRC 26 // Define I2S pins and communication channel.
```

2) Global Variables:

Global variables like WiFi credentials, ChatGPT API token and parameters, question and Audio are declared in this phase as shown in the code below;

```
const char* ssid = "WIFI NAME";
const char* password = "WIFI PASSWORD"; // WiFi credentials.
const char* chatgpt_token = "ChatGPT-API"; // API token for accessing ChatGPT.
const char* temperature = "0";
const char* max_tokens = "45"; // Parameters for the ChatGPT API request.
String Question = ""; // Holds the received question from the master ESP32.
Audio audio; // Audio object for handling sound playback.
```

3) ESP-NOW Initialization (InitESPNow):

The first rule of initializing ESP-NOW is to disconnect the ESP-32 microcontroller from any existing WiFi connection to avoid interference similarly to how it was declared in the Master ESP-32 microcontroller's code. After this precaution, the ESP-NOW is initialized. If the initialization fails, the ESP32 tries again.

```
void InitESPNow() {
  WiFi.disconnect();
  if (esp_now_init() == ESP_OK) {
    Serial.println("ESPNow Init Success");
  } else {
    Serial.println("ESPNow Init Failed");
    ESP.restart();
  }
}
```

4) Configuring Device as Access Point (configDeviceAP):

At the last stage of the setup and initialization stage, the ESP-32 is setup as a WiFi Access Point with a predefined SSID and password on a specified channel. The success or failure of the AP configuration will be displayed on the serial monitor for troubleshooting purposes.

```
void configDeviceAP() {
  const char *SSID = "Slave_1";
  bool result = WiFi.softAP(SSID, "Slave_1_Password", CHANNEL, 0);
  if (!result) {
    Serial.println("AP Config failed.");
  } else {
    Serial.println("AP Config Success. Broadcasting with AP: " + String(SSID));
    Serial.print("AP CHANNEL "); Serial.println(WiFi.channel());
  }
}
```

Main Setup ('setup' Function)

1) Serial Communication:

Serial communication is started/initialized for debugging purposes as shown below;

```
void setup() {
  Serial.begin(115200);
  Serial.println("ESPNow/Basic/Slave Example");
}
```

2) WiFi and ESP-NOW Setup:

The ESP-32 is configured to operate in both AP and Station modes (WIFI_AP_STA). Command 'configDeviceAP' is called to set up the AP, after which the AP MAC address is printed. ESP-NOW is then initialized and it registers the receive callback function 'OnDataRecv'.

```
WiFi.mode(WIFI_AP_STA);
configDeviceAP();
Serial.print("AP MAC: "); Serial.println(WiFi.softAPmacAddress());
InitESPNow();
esp_now_register_recv_cb(OnDataRecv);
```

3) Connecting to WiFi:

The provided SSID and password is used to connect to a WiFi network to access the internet. The device then waits until the connection is established and then the serial monitor prints the local IP address.

```
WiFi.begin(ssid, password);
Serial.print("Connecting to ");
Serial.println(ssid);

while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.print(".");
}
Serial.println("connected");
Serial.print("IP address: ");
Serial.println(WiFi.localIP());
```

4) Audio Setup:

Lastly, the I2S pins are configured for audio output. The audio volume is set to 100.

```
audio.setPinout(I2S_BCLK, I2S_LRC, I2S_DOUT);
audio.setVolume(100);
}
```

ESP-NOW Data Reception (OnDataRecv Function)

Receive Callback:

When data is received via ESP-NOW, the setup is triggered. Then the MAC address of the sender is extracted and printed out. The message received is then saved into a string 'Question'.

```
void OnDataRecv(const uint8_t *mac_addr, const uint8_t *data, int data_len) {
  char macStr[18];
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4],
           mac_addr[5]);
  Serial.print("Last Packet Recv from: "); Serial.println(macStr);
}
```



```

char msg[data_len + 1];
memcpy(msg, data, data_len);
msg[data_len] = '\0';

Serial.print("Last Packet Recv Data: "); Serial.println(msg);
Serial.println("");

Question = String(msg);
}

```

Main Loop ('loop' Function)

1) Audio Loop:

Continuously calls audio.loop() to handle audio playback.

2) Processing the Question:

This process checks if a new question has been received (Question.length() > 0). The question is then formatted as a JSON string for the API request.

3) HTTP Request to ChatGPT:

HTTP client is initialized. The HTTP client is then configured to connect to the OpenAI API endpoint. The required headers are set, including the authorization header with the API token. Constructs the JSON payload with the model, prompt, temperature, and max tokens. The POST request is then sent to the API.

4) Handling the API Response:

The code checks if the HTTP request was successful. After confirming, it then Parses the JSON response to extract the generated text. Unnecessary characters are striped from the response and prepares it for audio playback. It then uses the 'audio.connecttospeech' method to convert the response text to speech. The Question variable is then reset to an empty string.

```

void loop() {
  audio.loop();

  if (Question.length() > 0) {
    String processedQuestion = "\"" + Question + "\"";
    Serial.println(processedQuestion);

    HTTPClient https;

    if (https.begin("https://api.openai.com/v1/completions")) {
      https.addHeader("Content-Type", "application/json");
      String token_key = String("Bearer ") + chatgpt_token;
      https.addHeader("Authorization", token_key);

      String payload = String("{\"model\": \"gpt-3.5-turbo-instruct\", \"prompt\": \"") +
        processedQuestion + String(", \"temperature\": ") + temperature + String(",
        \"max_tokens\": ") + max_tokens + String("}");

```

```

int httpCode = https.POST(payload);

if (httpCode == HTTP_CODE_OK || httpCode ==
HTTP_CODE_MOVED_PERMANENTLY) {
    String response = https.getString();
    DynamicJsonDocument doc(1024);
    deserializeJson(doc, response);
    String Answer = doc["choices"][0]["text"];
    Answer = Answer.substring(2);
    Serial.print("Answer : "); Serial.println(Answer);
    audio.connecttospeech(Answer.c_str(), "en");
} else {
    Serial.printf("[HTTPS] GET... failed, error: %s\n",
https.errorToString(httpCode).c_str());
}
https.end();
} else {
    Serial.printf("[HTTPS] Unable to connect\n");
}

    Question = "";
}
}
...

- audio.loop(): Maintains audio playback.
void audio_info(const char *info) {
    Serial.print("audio_info: "); Serial.println(info);
}

```

This code effectively sets up the slave ESP-32 microcontroller to receive questions from the master ESP32, and then forward these questions to the ChatGPT API, which then convert the received answers to speech. The use of ESP-NOW for device communication and I2S for high-quality audio output makes this setup suitable for real-time, interactive applications like the chat-box project.

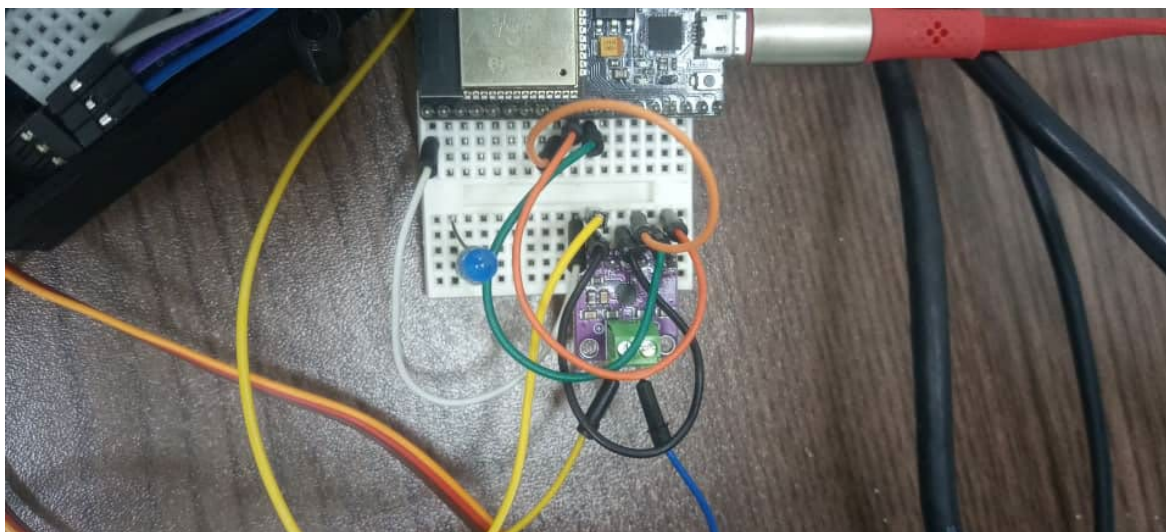


Fig 8: CHAT-BOX TESTING PHASE ON BREADBOARD (Slave ESP32-MAX98352 Amplifier-Speaker Connection)

CHAPTER IV

Results Analysis

4.1 Performance Metrics

In order to effectively evaluate the performance of the chat-box, the chat-bot was rated based on the following metrics; accuracy of speech-to-text conversion, latency in data transmission, quality of text-to-speech output, and overall system responsiveness. The accuracy of speech-to-text conversion was measured by comparing the transcribed text which the Google Cloud Speech-to-Text API produced with what was actually said. This is to make sure that the chatbot gets the accurate question as this is crucial to get the right response.

Data transmission from the Master ESP-32 to the slave ESP-32 was assessed by testing how long it takes the transcribed question from the master ESP-32 takes to reach the slave ESP-32 regardless of distance and obstacles. This was crucial for ensuring real-time interaction. The quality of text-to-speech output was also evaluated. It was evaluated based on clarity of the speaker, naturalness of response generated, and how valid the responses gotten were.

Finally, system responsiveness was measured by testing how long the chat-box takes to respond to the question asked. The total time taken from asking a question to receiving the spoken answer is what is measured at this stage, encompassing all stages of the process, including API interactions and text-to-speech conversion. These metrics were chosen to give a comprehensive assessment of how the chat-box will perform in real-world conditions. For instance, high accuracy in speech-to-text conversion makes sure that the questions are correctly interpreted by the system, which is important/fundamental for generating accurate responses. The Data transmission test is necessary to verify the reliability of the ESP-NOW connection in sending the question to the slave when in range for maintaining the flow of conversation and avoiding delays that could disrupt the user experience. The quality of text-to-speech output verifies the accuracy and clarity of the responses generated, while overall system responsiveness determines the chat-box's suitability for real-time applications. By systematically evaluating these metrics, the strengths and weaknesses of the chat-box system were identified, which will aid in future advancements.

4.1.1 Test Scenarios and Procedures

Three major scenerios were created to stimulate different conditions and see how the chatbot performs in these situations:

Scenario 1: Short and simple questions (e.g., "What is your name?")

Scenario 2: Complex and longer questions (e.g., "Can you explain the theory of relativity?")

Scenario 3: Noisy environment with background noise. This was created by playing loud video in the background while recording.

4.2 Results and Analysis

4.2.1 Accuracy of Speech-to-Text Conversion

The accuracy of the speech-to-text conversion was measured by comparing the transcribed text from the Google Cloud Speech-to-Text API with the actual spoken input. Ten basic questions were spoken and the accuracy was calculated based on how accurate it was at transcribing the spoken words. For instance, for scenario one, ten basic questions like ‘What is the capital of Turkey’ were said and the accuracy of the transcribed text was compared to the spoken words. The accuracy was calculated as the percentage of correctly transcribed words over the total number of words spoken.

$$\text{Accuracy (\%)} = (\text{CTW} / \text{TWS}) * 100$$

CTW = Correctly Transcribed Words
TWS = Total Words Spoken

Table 2: Table comparing accuracy of Chat-box across three scenarios

Scenario	Total Words Spoken	Correctly Transcribed Words	Accuracy (%)
1	60	60	100%
2	65	61	93.8%
3	60	51	85%

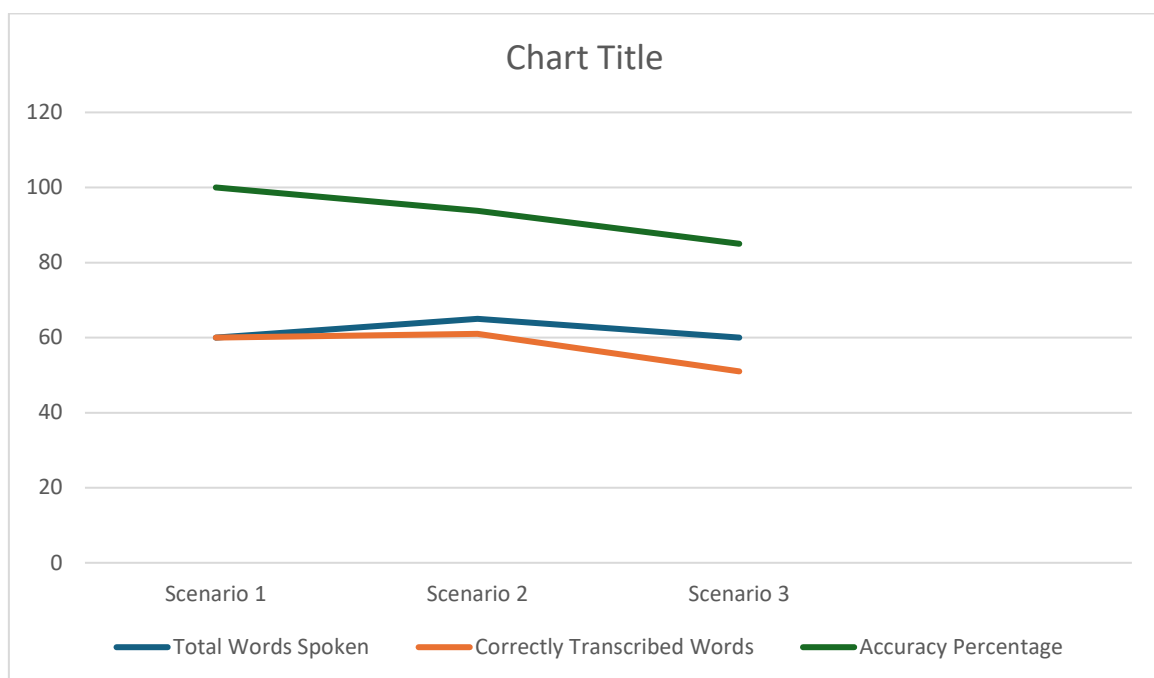


CHART EXPLAINING TABLE ABOVE

This result shows high accuracy in quiet environments with both simple questions and complex questions. The accuracy decreases in noisy environments but the decrease remains acceptable for real life applications.

4.2.2 ESP-NOW Data Transmission

The ESP-NOW data transmission was tested to see how fast and reliable the connection is for sending data between the two ESP-32 microcontrollers. This test was conducted by varying the distance between the ESP-32s and checking if the message is delivered successfully. The speed of the message transmission was also taken account of and at the end of the test, the ESP-NOW connection was reliable and fast in transmitting the question from the master ESP-32 to the slave ESP-32 microcontroller.

This test is important to make sure that the communication channel between the ESP-32 microcontrollers is reliable and not affect the outcome of the project negatively.

4.2.3 Quality of Text-to-Speech Output

After testing the speech-to-text transcription quality and the ESP-NOW connection, the quality of the Text-to-speech output was tested for the speed of response, clarity of audio and accuracy of response with respect to the question asked.

In order to rate this efficiently, ten questions from each scenario were asked and the responses were rated based on the time taken to get the response, the clarity of the audio and the accuracy of the response. The speed of response was calculated by checking the time period between receiving the question to the time which the response is heard on the speaker. The table below explains the results gotten;

NOTE: $ATR = TTR / 10$

ATR = Average Time of Response

TTR = Total Time of Response

$ACR = TACR / 10$

ACR = Average Audio Clarity Rating

TACR = Total Audio Clarity Rating

$QR = TQR / 10$

QR = Average Quality Response

TQR = Total Quality of Response

Table 3: Table comparing Average time of response, Audio clarity rating and Quality of response across all three scenarios

Scenarios	Average Time of response(s)	Audio Clarity rating (1-5)	Quality of response (1-5)
1	27.6	4.8	4.9
2	31.2	4.2	4.1
3	15.2	4.6	4.7

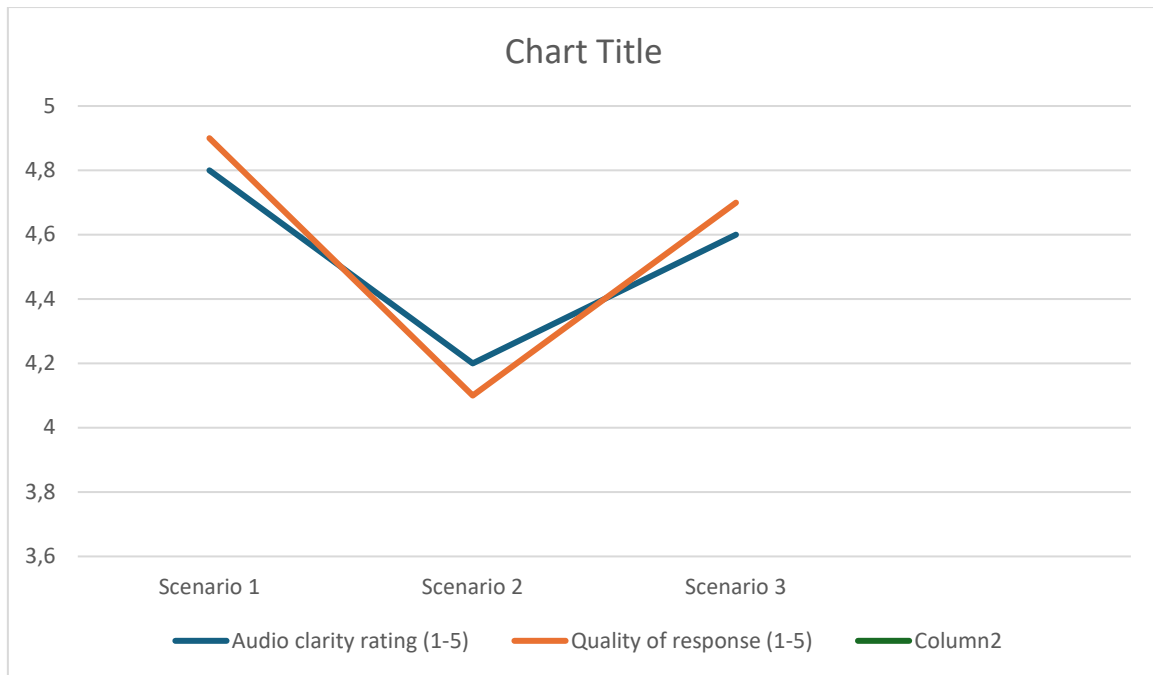


CHART SHOWING COMPARISON BETWEEN AUDIO CLARITY AND QUALITY OF RESPONSE

As shown by the table above, the average time of response is ‘all over the place’ but one thing the results across the three scenarios have in common is that the response time is slow and unpredictable. This is mostly because of slow internet connection. This makes it generate responses from OpenAI slow.

The Audio clarity had a good rating. The only issue with clarity is that the speaker placement affects the sound from the speaker. The quality of response depended on the response from ChatGPT and it was mostly satisfying and accurate except for some few tough questions.

4.2.4 System Responsiveness

The overall system responsiveness measures the total time it takes the chat-cox to respond to the question. This means that this is the measure of time it takes from the speech-to-text conversion, data transmission all the way to the response-to-speech conversion. This was done by asking ten questions each in the three scenarios and then calculating the average time it takes to respond to the questions. The table below shows the results gotten in details;

NOTE: Formula used to calculate average response time $ART = TRT/10$

ART = Average Response Time

TRT = Summation of response time recorded across ten trials per scenario

Table 4: Average Overall Response Time across all three scenarios

Scenario	Average Response Time (s)
1	49.5
2	67.0
3	51.8

The table above shows that the device takes almost a minute to respond to the questions asked with the lowest average time being 49.5 seconds. This shows that the chat-box takes a lot of time to respond to the question and this is a major drawback to the design.

4.3 Discussion of Results

The results from the chat-box had been mostly positive; the audio quality has been good enough, the chat-box transcribes the questions well, the response to the questions have been fairly accurate. The unique aspects and the limitations of this thesis are highlighted below;

4.3.1 Standout points of research

The standout parts of these research are;

- The chat-box uses a pretrained application on OpenAI to generate more unique answers especially when the questions are about Machine learning. This makes the answers from the chat-box more unique and different from what ChatGPT would normally give.
- Another unique area of the project is its wireless one-way communication (using ESP-NOW protocol) which makes communication more unique.

4.3.2 Limitations

- One major drawback is that it takes time to get a response. This is as a result of its reliability on the internet to transcribe the question asked and to get a response from OpenAI.
- Another drawback is that the text-to-speech only records audio for 3 seconds. This is to manage the credit given on the Google API.

Future improvements can focus on improving the response time and improving the time limit to record question.

References

Konstantinos M. Pitychoutis (2024). "Harnessing AI Chat-boxes for EFL Essay Writing: A Paradigm Shift in Language Pedagogy".
<https://www.semanticscholar.org/paper/64f739ceb4b63564b5a9552d91e9011f8b27ffe3>

A. Zouhri, M. El Mallahi (2024). "Improving Teaching Using Artificial Intelligence and Augmented Reality".
<https://www.semanticscholar.org/paper/4f985bf98c3195487149b4c83df70312d010fb13>

Krishan Kant Singh Mer, Vijay Bhaskar Semwal, Vishwanath Bijalwan, Rubén González Crespo (2021-04-23). "Proceedings of Integrated Intelligence Enable Networks and Computing". Springer Nature.
https://play.google.com/store/books/details?id=rworEAAAQBAJ&source=gbs_api

Kingsley Okoye, Samira Hosseini, Kamal Kant Hiran , Julius Nganji (2024-06-07). "Impact and implications of AI methods and tools for the future of education". Frontiers Media SA.
http://books.google.com/books?id=gjsNEQAAQBAJ&dq=What+is+the+significance+of+context+and+motivation+in+implementing+a+student+AI+assistant+chat-box+device%3F&hl=&source=gbs_api

Miao, Fengchun, Holmes, Wayne, Ronghuai Huang, Hui Zhang, UNESCO (2021-04-08). "AI and education". UNESCO Publishing.
http://books.google.com/books?id=yyE7EAAAQBAJ&dq=CONTEXT+AND+MOTIVATION+FOR+STUDENT+AI+ASSISTANT+CHAT-BOX+DEVICE+%2B+Current+Educational+Landscape&hl=&source=gbs_api

Miao, Fengchun, Holmes, Wayne, Ronghuai Huang, Hui Zhang, UNESCO (2021-04-08). "AI and education". UNESCO Publishing.
http://books.google.com/books?id=yyE7EAAAQBAJ&dq=Analyze+challenges+faced+by+students+in+traditional+learning+environments+to+develop+context+and+motivation+for+student+AI+assistant+chat-box+device.&hl=&source=gbs_api

Sarah Nagle, Elias Tzoc (2022-03-15). "Innovation and Experiential Learning in Academic Libraries". Rowman & Littlefield.
https://play.google.com/store/books/details?id=G9BbEAAAQBAJ&source=gbs_api

P. Otero, P. Scott, S.Z. Martin (2022-08-05). "MEDINFO 2021: One World, One Health — Global Partnership for Digital Innovation". IOS Press.
http://books.google.com/books?id=4pGREAAAQBAJ&dq=Analysis+of+motivation+behind+creating+AI+chat-box+device+for+student+assistance+in+educational+settings&hl=&source=gbs_api

Arcangelo Castiglione, Florin Pop, Massimo Ficco, Francesco Palmieri (2018-09-23). "Cyberspace Safety and Security". Springer.
http://books.google.com/books?id=_akfuWEACAAJ&dq=How+can+the+context+and+motivation+for+student+AI+assistant+chat-box+devices+be+leveraged+to+enhance+student+engagement+and+learning%3F&hl=&source=gbs_api

Arcangelo Castiglione, Florin Pop, Massimo Ficco, Francesco Palmieri (2018-09-23). "Cyberspace Safety and Security". Springer.
http://books.google.com/books?id=_akfuwEACAAJ&dq=How+can+AI+chat-boxes+personalize+assistance+for+students+in+the+development+of+a+contextual+and+motivational+student+AI+assistant+chat-box+device%3F&hl=&source=gbs_api

Tareq Ahram and Redha Taiar (2023-04-13). "Human Interaction & Emerging Technologies (IHET-AI 2023): Artificial Intelligence & Future Applications". AHFE Conference.
https://play.google.com/store/books/details?id=inq3EAAAQBAJ&source=gbs_api

Kingsley Okoye, Samira Hosseini, Kamal Kant Hiran , Julius Nganji (2024-06-07). "Impact and implications of AI methods and tools for the future of education". Frontiers Media SA.
http://books.google.com/books?id=gjsNEQAAQBAJ&dq=What+is+the+significance+of+deploying+AI+chat-boxes+as+student+assistants+and+analyzing+their+context+and+motivation+in+educational+settings%3F&hl=&source=gbs_api

Miao, Fengchun, Holmes, Wayne, Ronghuai Huang, Hui Zhang, UNESCO (2021-04-08). "AI and education". UNESCO Publishing.
http://books.google.com/books?id=yyE7EAAAQBAJ&dq=What+is+the+importance+of+integrating+AI+technology+into+educational+environments+for+developing+a+student+AI+assistant+chat-box+device%3F&hl=&source=gbs_api

R. Baecker, Peter Wolf, Kelly Rankin (2004). "The ePresence Interactive Webcasting and Archiving System: Technology Overview and Current Research Issues". 2004. pp. 2532-2537.
<https://www.semanticscholar.org/paper/d34e5165d316e8e6bbe7d97d160df729dc4fa532>

D. Cook, G. Guyatt (2001). "Colloid Use for Fluid Resuscitation: Evidence and Spin". 135. pp. 205-208.
<https://www.semanticscholar.org/paper/44c695fbdf664850834ef62e66a1ff40f3d35dd0>

Themba Ngobeni, Boniface Kabaso (2024). "Quantum-Secure Signalling Model for L1/L2 Next-Gen Interconnect and Roaming Networks Over IPX for NB-IoT Traffic: A Review".
<https://www.semanticscholar.org/paper/c57b14b23011051b1c883da1f90e31305ced57ea>

M. V. Stilpen, D. M. Avejonas (2023). "PRE-TEST: SPEECH THERAPY PROTOCOL FOR COGNITIVE ASSESSMENT.". 66. pp. 1031-1032.
<https://www.semanticscholar.org/paper/206137c5707e4c6c7879950d5f6db7ca0810096e>

J. Chukwuere (2024). "Today's Academic Research: The Role of ChatGPT Writing".
<https://www.semanticscholar.org/paper/85e4112943f5dd4001788ba3adc142b9091dc1>

K. P. Kuchinke (2023). "Grounding and Deepening Academic Writing in Human Resource Development: The Role of Selecting and Representing the Supporting Literature". 22. pp. 414-427.
<https://www.semanticscholar.org/paper/d68c79fb3001472ecbb379c68a7051b39aa33849>

C. Debrah, A. Darko, Albert P. C. Chan (2022). "A bibliometric-qualitative literature review of green finance gap and future research directions". 15. pp. 432-455. <https://www.semanticscholar.org/paper/0bc82cd97ee26c3b8f275f1a9c075a78d091f787>

Daniel Stefan, Valentina Vasile, Anca Oltean, Calin-Adrian Comes, Anamari-Beatrice Stefan, Liviu Ciucan-Rusu, E. Bunduchi, Maria-Alexandra Popa, Mihai Timus (2021). "Women Entrepreneurship and Sustainable Business Development: Key Findings from a SWOT–AHP Analysis". 13. pp. 5298. <https://www.semanticscholar.org/paper/f442a1b62cbdbf3ef9635a378750aec40d6c1daa>

Fausto Pedro García Márquez (2021-08-18). "Internet of Things". BoD – Books on Demand. http://books.google.com/books?id=fX4_EAAAQBAJ&dq=Literature+review+on+developing+chat-box+systems+using+ESP32+microcontrollers&hl=&source=gbs_api

Fang Chen, Kristiina Jokinen (2010-07-01). "Speech Technology". Springer Science & Business Media. https://play.google.com/store/books/details?id=jD3XKFBuN5QC&source=gbs_api

Tomonobu Senjyu . "Smart Trends in Computing and Communications". Springer Nature. http://books.google.com/books?id=Z9UHEQAAQBAJ&dq=Literature+review+on+ESP-NOW+vs.+other+IoT+wireless+protocols&hl=&source=gbs_api

Paul C. Cozby, Patricia E. Worden, Daniel W. Kee (1989). "Research Methods in Human Development". WCB/McGraw-Hill. http://books.google.com/books?id=NTkiAAAAMAAJ&dq=Literature+review+on+recent+developments+and+improvements+in+ESP-NOW+for+enhancing+performance+and+reliability.&hl=&source=gbs_api

Vedat Ozan Oner (2021-09-13). "Developing IoT Projects with ESP32". Packt Publishing Ltd. https://play.google.com/store/books/details?id=e4k4EAAAQBAJ&source=gbs_api

Et al. Beschi I S (2023). "Applications of Deep Learning and Machine Learning in Healthcare Domain – A Literature Review". <https://www.semanticscholar.org/paper/8ac9e14e98c71a13f364ba3a87be9b49e120f8d9>

Xinying Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John C. Grundy, Haoyu Wang (2023). "Large Language Models for Software Engineering: A Systematic Literature Review". abs/2308.10620. <https://www.semanticscholar.org/paper/000f964393dafa113a8e66734d63b2a145844159>

Basahel, A, Irani, Z (2010). "Examining the strategic benefits of information systems: A global case study". EMCIS2010. <https://core.ac.uk/download/336775.pdf>

British Academy of Management 32nd Annual Conference : Driving Productivity in Uncertain and Challenging Times, British Academy of Management, Camilleri, Adriana Caterina, Camilleri, Mark Anthony (2018). "The performance management and appraisal in higher education". British Academy of Management. <https://core.ac.uk/download/162326049.pdf>

Asim Zulfiqar (2024-01-19). "Hands-on ESP32 with Arduino IDE". Packt Publishing Ltd. https://play.google.com/store/books/details?id=5JnqEAAAQBAJ&source=gbs_api

Diana Ridley (2012-07-23). "The Literature Review". SAGE. https://play.google.com/store/books/details?id=WaNrAwAAQBAJ&source=gbs_api

Amina Al-Marzouqi . "Artificial Intelligence in Education: The Power and Dangers of ChatGPT in the Classroom". Springer Nature. http://books.google.com/books?id=ViL-EAAAQBAJ&dq=Literature+review+on+the+impact+of+ChatGPT+in+enhancing+interactive+user-friendly+chatbot+systems.&hl=&source=gbs_api

Darwish, Dina (2024-04-09). "Design and Development of Emerging Chatbot Technology". IGI Global. https://play.google.com/store/books/details?id=V9wBEQAAQBAJ&source=gbs_api

William Johnson (2024-04-09). "ChatGPT in the Classroom for Harnessing AI to Revolutionize Higher Education in Colleges and Universities". LEGENDARY EDITIONS. https://play.google.com/store/books/details?id=QbQAEQAAQBAJ&source=gbs_api

Zhao Ni, Mary L Peng, Vimala Balakrishnan, Vincent Tee, I. Azwa, Rumana Saifi, LaRon Nelson, David Vlahov, Frederick L Altice (2023). "Implementation of Chatbot Technology in Health Care: Protocol for a Bibliometric Analysis". 13. <https://www.semanticscholar.org/paper/1324c6b904eca5fa87eb116457da66dcf59aac8c>

A. Cau, Alison Müller, Samia El Joueidi, K. Chan, Jayson Park, M. Semakula, S. Nsanzimana, Peter Lodokiyaa, Linet Lumumba, Osman A. Abdullahi, C. Logie, A. Hayward, R. Lester (2021). "Digital mHealth and Virtual Care Monitoring and Support in Pandemics: Part 2 - A Rapid Review Assessing Strategies during COVID-19 (Preprint)". <https://www.semanticscholar.org/paper/02152661864f374bb5d0f7cb82c92d975d40f26a>

Yuvika Gupta, F. Khan (2024). "Role of artificial intelligence in customer engagement: a systematic review and future research directions". <https://www.semanticscholar.org/paper/25d4f4be2ff2475ec6cb509326bb6807aa230a65>

Lee Kowalsky (2001). "Internet conferencing tools for deaf and hard of hearing users". <https://www.semanticscholar.org/paper/dedac1cc074d1ab308469fe50716a5320679c70b>

Rafael Santana Queiroz, Lucas Marins Batista, Miguel Felipe Nery Vieira, Lucas Cruz da Silva, Bruno Caetano dos Santos Silva, Rodrigo Santiago Coelho (2023). "A Literature Review of Additive Manufacturing in the Fabrication of Soft Robots: Main Techniques, Applications, and Related Industrial-Sized Machines". <https://www.semanticscholar.org/paper/6ac5ee35cde4fc3f2150bdea0c17a9171d7d5de9>

Miguel A. Baque-Cantos, Cristhian Y. Moreira-Cañarte, Andrés Ultreras-Rodríguez, Daniel O. Nieves-Lizárraga, Felipe De J. González-Rodríguez, J. S. Moreira-Choez, Shirley T. Campos-Sánchez, Mariana De L. Cantos-Figueroa, Cristian Rincón-Guio (2023). "Technological Enablers and Prospects of Project Management in Industry 4.0: A Literature Review". <https://www.semanticscholar.org/paper/47a399eaa631fe48ba2781ff5919d5e09908eb58>

Shuvojit Nath, Sabrina Kirschke (2023). "Groundwater Monitoring through Citizen Science: A Review of Project Designs and Results". 61. <https://www.semanticscholar.org/paper/f14e91cf29d6de0ffcb40703746a8a4a543bf540>

André Coners, Benjamin Matthies (2022). "Perspectives on reusing codified project knowledge: a structured literature review". <https://www.semanticscholar.org/paper/c50ace415e2f4eca6520d30f904d82c60b5a5f72>

P. Muruganatham, S. Wibowo, S. Grandhi, N. Samrat, Nahina Islam (2022). "A Systematic Literature Review on Crop Yield Prediction with Deep Learning and Remote Sensing". 14. pp. 1990. <https://www.semanticscholar.org/paper/11413499875c438b3ce16fca7eefaf21850d4d9>

S. F. Khatib, Ahmed A. Elamer, D. F. Abdullah, S. Hazaea (2022). "The development of corporate governance literature in Malaysia: a systematic literature review and research agenda". <https://www.semanticscholar.org/paper/8360e3cf197324681a7237737a23ba6abbc3c5bc>

Leah A Lievrouw, Sonia M. Livingstone (2006-01-17). "Handbook of New Media". SAGE. http://books.google.com/books?id=P9HkFWEwfFUC&dq=Importance+of+interactive+chat-box+systems+in+literature+review.&hl=&source=gbs_api

Khosrow-Pour, Mehdi (2014-07-31). "Encyclopedia of Information Science and Technology, Third Edition". IGI Global. https://play.google.com/store/books/details?id=MJd_BAAAQBAJ&source=gbs_api

Diana Ridley (2012-07-23). "The Literature Review". SAGE. https://play.google.com/store/books/details?id=WaNrAwAAQBAJ&source=gbs_api

Appendix A

Code

TEXT-TO-SPEECH CODE

```

#include <Arduino.h>
#include <esp_now.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include "Audio.h"

#define CHANNEL 1
#define I2S_DOUT 25
#define I2S_BCLK 27
#define I2S_LRC 26

const char* ssid = "WIFI NAME";
const char* password = "WIFI PASSWORD";
const char* chatgpt_token = "ChatGPT token";
const char* temperature = "0";
const char* max_tokens = "45";
String Question = "";

Audio audio;

void InitESPNow() {
  WiFi.disconnect();
  if (esp_now_init() == ESP_OK) {
    Serial.println("ESPNow Init Success");
  } else {
    Serial.println("ESPNow Init Failed");
    ESP.restart();
  }
}

void configDeviceAP() {
  const char *SSID = "Slave_1";
  bool result = WiFi.softAP(SSID, "Slave_1_Password", CHANNEL, 0);
  if (!result) {
    Serial.println("AP Config failed.");
  } else {
    Serial.println("AP Config Success. Broadcasting with AP: " + String(SSID));
    Serial.print("AP CHANNEL "); Serial.println(WiFi.channel());
  }
}

void setup() {
  Serial.begin(115200);
  Serial.println("ESPNow/Basic/Slave Example");
}

```



```

WiFi.mode(WIFI_AP_STA);
configDeviceAP();
Serial.print("AP MAC: "); Serial.println(WiFi.softAPmacAddress());
InitESPNow();
esp_now_register_recv_cb(OnDataRecv);

WiFi.begin(ssid, password);
Serial.print("Connecting to ");
Serial.println(ssid);

while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.print(".");
}
Serial.println("connected");
Serial.print("IP address: ");
Serial.println(WiFi.localIP());

audio.setPinout(I2S_BCLK, I2S_LRC, I2S_DOUT);
audio.setVolume(100);
}

void OnDataRecv(const uint8_t *mac_addr, const uint8_t *data, int data_len) {
  char macStr[18];
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4],
           mac_addr[5]);
  Serial.print("Last Packet Recv from: "); Serial.println(macStr);

  char msg[data_len + 1];
  memcpy(msg, data, data_len);
  msg[data_len] = '\0';

  Serial.print("Last Packet Recv Data: "); Serial.println(msg);
  Serial.println("");

  Question = String(msg);
}

void loop() {

  audio.loop();

  if (Question.length() > 0) {
    String processedQuestion = "\"" + Question + "\"";
    Serial.println(processedQuestion);

    HTTPClient https;

```

```

if (https.begin("https://api.openai.com/v1/completions")) {
    https.addHeader("Content-Type", "application/json");
    String token_key = String("Bearer ") + chatgpt_token;
    https.addHeader("Authorization", token_key);

    String payload = String("{\"model\": \"gpt-3.5-turbo-instruct\", \"prompt\": ") +
processedQuestion + String(", \"temperature\": ") + temperature + String(", \"max_tokens\":
") + max_tokens + String("}");

    int httpCode = https.POST(payload);

    if (httpCode == HTTP_CODE_OK || httpCode ==
HTTP_CODE_MOVED_PERMANENTLY) {
        String response = https.getString();
        DynamicJsonDocument doc(1024);
        deserializeJson(doc, response);
        String Answer = doc["choices"][0]["text"];
        Answer = Answer.substring(2);
        Serial.print("Answer : "); Serial.println(Answer);
        audio.connecttospeech(Answer.c_str(), "en");
    } else {
        Serial.printf("[HTTPS] GET... failed, error: %s\n",
https.errorToString(httpCode).c_str());
    }
    https.end();
} else {
    Serial.printf("[HTTPS] Unable to connect\n");
}

    Question = "";
}
}

void audio_info(const char *info) {
    Serial.print("audio_info: "); Serial.println(info);
}

```

SPEECH-TO-TEXT CODE

```

#include <esp_now.h>
#include <WiFi.h>
#include "Audio.h"
#include "CloudSpeechClient.h"

esp_now_peer_info_t slave;
#define CHANNEL 0
#define PRINTSCANRESULTS 0
#define DELETEBEFOREPAIR 0

extern String My_Answer;

```

```

// Init ESP Now with fallback
void InitESPNow() {
  WiFi.disconnect();
  if (esp_now_init() == ESP_OK) {
    Serial.println("ESPNow Init Success");
  } else {
    Serial.println("ESPNow Init Failed");
    ESP.restart();
  }
}

// Scan for slaves in AP mode
void ScanForSlave() {
  int8_t scanResults = WiFi.scanNetworks();
  bool slaveFound = 0;
  memset(&slave, 0, sizeof(slave));

  Serial.println("");
  if (scanResults == 0) {
    Serial.println("No WiFi devices in AP Mode found");
  } else {
    Serial.print("Found "); Serial.print(scanResults); Serial.println(" devices ");
    for (int i = 0; i < scanResults; ++i) {
      String SSID = WiFi.SSID(i);
      int32_t RSSI = WiFi.RSSI(i);
      String BSSIDstr = WiFi.BSSIDstr(i);

      if (PRINTSCANRESULTS) {
        Serial.print(i + 1);
        Serial.print(": ");
        Serial.print(SSID);
        Serial.print(" (");
        Serial.print(RSSI);
        Serial.println("");
      }
      delay(10);
      if (SSID.indexOf("Slave") == 0) {
        Serial.println("Found a Slave.");
        Serial.print(i + 1); Serial.print(": "); Serial.print(SSID); Serial.print(" [");
        Serial.print(BSSIDstr); Serial.print("]"); Serial.print(" ("); Serial.print(RSSI);
        Serial.println("");
        int mac[6];
        if (6 == sscanf(BSSIDstr.c_str(), "%x:%x:%x:%x:%x:%x", &mac[0], &mac[1],
&mac[2], &mac[3], &mac[4], &mac[5])) {
          for (int ii = 0; ii < 6; ++ii) {
            slave.peer_addr[ii] = (uint8_t) mac[ii];
          }
        }
      }
      slave.channel = CHANNEL;
    }
  }
}

```

```

    slave.encrypt = 0;
    slaveFound = 1;
    break;
  }
}
}

if (slaveFound) {
  Serial.println("Slave Found, processing..");
} else {
  Serial.println("Slave Not Found, trying again.");
}

WiFi.scanDelete();
}

// Check if the slave is already paired with the master.
// If not, pair the slave with master
bool manageSlave() {
  if (slave.channel == CHANNEL) {
    if (DELETEBEFOREPAIR) {
      deletePeer();
    }

    Serial.print("Slave Status: ");
    bool exists = esp_now_is_peer_exist(slave.peer_addr);
    if (exists) {
      Serial.println("Already Paired");
      return true;
    } else {
      esp_err_t addStatus = esp_now_add_peer(&slave);
      if (addStatus == ESP_OK) {
        Serial.println("Pair success");
        return true;
      } else if (addStatus == ESP_ERR_ESPNOW_NOT_INIT) {
        Serial.println("ESPNow Not Init");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_ARG) {
        Serial.println("Invalid Argument");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_FULL) {
        Serial.println("Peer list full");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_NO_MEM) {
        Serial.println("Out of memory");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_EXIST) {
        Serial.println("Peer Exists");
        return true;
      } else {

```

```

        Serial.println("Not sure what happened");
        return false;
    }
}
} else {
    Serial.println("No Slave found to process");
    return false;
}
}

void deletePeer() {
    esp_err_t delStatus = esp_now_del_peer(slave.peer_addr);
    Serial.print("Slave Delete Status: ");
    if (delStatus == ESP_OK) {
        Serial.println("Success");
    } else if (delStatus == ESP_ERR_ESPNOW_NOT_INIT) {
        Serial.println("ESPNow Not Init");
    } else if (delStatus == ESP_ERR_ESPNOW_ARG) {
        Serial.println("Invalid Argument");
    } else if (delStatus == ESP_ERR_ESPNOW_NOT_FOUND) {
        Serial.println("Peer not found.");
    } else {
        Serial.println("Not sure what happened");
    }
}

// send data
void sendData() {
    const char* data = My_Answer.c_str();
    const uint8_t *peer_addr = slave.peer_addr;
    Serial.print("Sending: "); Serial.println(data);
    esp_err_t result = esp_now_send(peer_addr, (uint8_t *)data, strlen(data) + 1);
    Serial.print("Send Status: ");
    if (result == ESP_OK) {
        Serial.println("Success");
    } else if (result == ESP_ERR_ESPNOW_NOT_INIT) {
        Serial.println("ESPNow not Init.");
    } else if (result == ESP_ERR_ESPNOW_ARG) {
        Serial.println("Invalid Argument");
    } else if (result == ESP_ERR_ESPNOW_INTERNAL) {
        Serial.println("Internal Error");
    } else if (result == ESP_ERR_ESPNOW_NO_MEM) {
        Serial.println("ESP_ERR_ESPNOW_NO_MEM");
    } else if (result == ESP_ERR_ESPNOW_NOT_FOUND) {
        Serial.println("Peer not found.");
    } else {
        Serial.println("Not sure what happened");
    }
}
}

```

```

// callback when data is sent from Master to Slave
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  char macStr[18];
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
  Serial.print("Last Packet Sent to: "); Serial.println(macStr);
  Serial.print("Last Packet Send Status: "); Serial.println(status ==
ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void setup() {
  Serial.begin(115200);
  pinMode(15, OUTPUT);
  pinMode(2, OUTPUT);

  digitalWrite(15, LOW);
  digitalWrite(2, LOW);
  WiFi.mode(WIFI_STA);
  Serial.println("ESPNow/Basic/Master Example");
  Serial.print("STA MAC: "); Serial.println(WiFi.macAddress());
  InitESPNow();
  esp_now_register_send_cb(OnDataSent);

  // Begin speech recognition process
  Serial.println("\r\nRecord start!\r\n");
  digitalWrite(15, HIGH);
  digitalWrite(2, LOW);
  Audio* audio = new Audio(ICS43434);
  audio->Record();
  Serial.println("Recording Completed. Now Processing...");
  digitalWrite(15, LOW);
  digitalWrite(2, HIGH);
  CloudSpeechClient* cloudSpeechClient = new CloudSpeechClient(USE_APIKEY);
  cloudSpeechClient->Transcribe(audio);
  delete cloudSpeechClient;
  delete audio;

  // Scan for slaves and manage connections
  ScanForSlave();
  if (slave.channel == CHANNEL) {
    bool isPaired = manageSlave();
    if (isPaired) {
      sendData();
      digitalWrite(15, HIGH);
      digitalWrite(2, LOW);
      delay(1000);
      digitalWrite(15, LOW);
      digitalWrite(2, HIGH);
    } else {
      Serial.println("Slave pair failed!");
    }
  }
}

```

```

    }
  }
}

```

```

void loop() {
  // Empty loop as the main logic is handled in setup
}

```

AUDIO.CPP CODE

```
#include "Audio.h"
```

```

Audio::Audio(MicType micType) {
  wavData = new char*[wavDataSize/dividedWavDataSize];
  for (int i = 0; i < wavDataSize/dividedWavDataSize; ++i) wavData[i] = new
char[dividedWavDataSize];
  i2s = new I2S(micType);
}

```

```

Audio::~~Audio() {
  for (int i = 0; i < wavDataSize/dividedWavDataSize; ++i) delete[] wavData[i];
  delete[] wavData;
  delete i2s;
}

```

```

void Audio::CreateWavHeader(byte* header, int waveDataSize){
  header[0] = 'R';
  header[1] = 'I';
  header[2] = 'F';
  header[3] = 'F';
  unsigned int fileSizeMinus8 = waveDataSize + 44 - 8;
  header[4] = (byte)(fileSizeMinus8 & 0xFF);
  header[5] = (byte)((fileSizeMinus8 >> 8) & 0xFF);
  header[6] = (byte)((fileSizeMinus8 >> 16) & 0xFF);
  header[7] = (byte)((fileSizeMinus8 >> 24) & 0xFF);
  header[8] = 'W';
  header[9] = 'A';
  header[10] = 'V';
  header[11] = 'E';
  header[12] = 'f';
  header[13] = 'm';
  header[14] = 't';
  header[15] = ' ';
  header[16] = 0x10; // linear PCM
  header[17] = 0x00;
  header[18] = 0x00;
  header[19] = 0x00;
  header[20] = 0x01; // linear PCM
  header[21] = 0x00;
  header[22] = 0x01; // monoral
  header[23] = 0x00;
}

```

```

header[24] = 0x80; // sampling rate 16000
header[25] = 0x3E;
header[26] = 0x00;
header[27] = 0x00;
header[28] = 0x00; // Byte/sec = 16000x2x1 = 32000
header[29] = 0x7D;
header[30] = 0x00;
header[31] = 0x00;
header[32] = 0x02; // 16bit monoral
header[33] = 0x00;
header[34] = 0x10; // 16bit
header[35] = 0x00;
header[36] = 'd';
header[37] = 'a';
header[38] = 't';
header[39] = 'a';
header[40] = (byte)(waveDataSize & 0xFF);
header[41] = (byte)((waveDataSize >> 8) & 0xFF);
header[42] = (byte)((waveDataSize >> 16) & 0xFF);
header[43] = (byte)((waveDataSize >> 24) & 0xFF);
}

void Audio::Record() {
  CreateWavHeader(paddedHeader, wavDataSize);
  int bitBitPerSample = i2s->GetBitPerSample();
  if (bitBitPerSample == 16) {
    for (int j = 0; j < wavDataSize/dividedWavDataSize; ++j) {
      i2s->Read(i2sBuffer, i2sBufferSize/2);
      for (int i = 0; i < i2sBufferSize/8; ++i) {
        wavData[j][2*i] = i2sBuffer[4*i + 2];
        wavData[j][2*i + 1] = i2sBuffer[4*i + 3];
      }
    }
  }
  else if (bitBitPerSample == 32) {
    for (int j = 0; j < wavDataSize/dividedWavDataSize; ++j) {
      i2s->Read(i2sBuffer, i2sBufferSize);
      for (int i = 0; i < i2sBufferSize/8; ++i) {
        wavData[j][2*i] = i2sBuffer[8*i + 2];
        wavData[j][2*i + 1] = i2sBuffer[8*i + 3];
      }
    }
  }
}

```

AUDIO.H CODE

```

#ifndef _AUDIO_H
#define _AUDIO_H

```

```

#include <Arduino.h>

```



```

#include "I2S.h"

// 16bit, monoral, 16000Hz, linear PCM
class Audio {
  I2S* i2s;
  static const int headerSize = 44;
  static const int i2sBufferSize = 12000;
  char i2sBuffer[i2sBufferSize];
  void CreateWavHeader(byte* header, int waveDataSize);

public:
  static const int wavDataSize = 90000;          // It must be multiple of
  dividedWavDataSize. Recording time is about 1.9 second.
  static const int dividedWavDataSize = i2sBufferSize/4;
  char** wavData;                               // It's divided. Because large continuous
  memory area can't be allocated in esp32.
  byte paddedHeader[headerSize + 4] = {0};      // The size must be multiple of 3 for
  Base64 encoding. Additional byte size must be even because wave data is 16bit.

  Audio(MicType micType);
  ~Audio();
  void Record();
};

#endif // _AUDIO_H

```

CLOUDSPEECHCLIENT.CPP CODE

```

#include "CloudSpeechClient.h"
#include "network_param.h"
#include <base64.h>
#include <ArduinoJson.h>

WiFiClientSecure client;
String My_Answer = ""; // Global variable to hold the transcription result

CloudSpeechClient::CloudSpeechClient(Authentication authentication) {
  this->authentication = authentication;
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) delay(1000);
  client.setCACert(root_ca);
  if (!client.connect(server, 443)) Serial.println("Connection failed!");
}

CloudSpeechClient::~CloudSpeechClient() {
  client.stop();
  WiFi.disconnect();
}

void CloudSpeechClient::PrintHttpBody2(Audio* audio) {
  String enc = base64::encode(audio->paddedHeader, sizeof(audio->paddedHeader));

```

```

enc.replace("\n", "");
client.print(enc);
char** wavData = audio->wavData;
for (int j = 0; j < audio->wavDataSize / audio->dividedWavDataSize; ++j) {
    enc = base64::encode((byte*)wavData[j], audio->dividedWavDataSize);
    enc.replace("\n", "");
    client.print(enc);
}
}

void CloudSpeechClient::Transcribe(Audio* audio) {
    String HttpBody1 =
    "{\"config\": {\"encoding\": \"LINEAR16\", \"sampleRateHertz\": 16000, \"languageCode\": \"en-NG\"}, \"audio\": {\"content\": \"\"};
    String HttpBody3 = \"}\"}\r\n\r\n";
    int httpBody2Length = (audio->wavDataSize + sizeof(audio->paddedHeader)) * 4 / 3;
    String ContentLength = String(HttpBody1.length() + httpBody2Length +
    HttpBody3.length());
    String HttpHeaders = String("POST /v1/speech:recognize?key=") + ApiKey
        + String(" HTTP/1.1\r\nHost: speech.googleapis.com\r\nContent-Type:
    application/json\r\nContent-Length: ") + ContentLength + String("\r\n\r\n");
    client.print(HttpHeaders);
    client.print(HttpBody1);
    PrintHttpBody2(audio);
    client.print(HttpBody3);
    My_Answer = "";
    while (!client.available());

    while (client.available()) {
        char temp = client.read();
        My_Answer = My_Answer + temp;
    }

    Serial.print("My Answer - "); Serial.println(My_Answer);
    int position = My_Answer.indexOf('{');
    String jsonResponse = My_Answer.substring(position);
    Serial.print("Json data--"); Serial.println(jsonResponse);

    // Parse JSON to extract the transcribed text
    DynamicJsonDocument doc(2048);
    DeserializationError error = deserializeJson(doc, jsonResponse);
    if (error) {
        Serial.print(F("deserializeJson() failed: "));
        Serial.println(error.f_str());
        return;
    }

    const char* transcript = doc["results"][0]["alternatives"][0]["transcript"];
    Serial.print("Transcript: ");
    Serial.println(transcript);
}

```

```

    My_Answer = String(transcript);
}

```

CLOUDSPEECHCLIENT.H CODE

```

// CloudSpeechClient.h
#ifndef CLOUDSPEECHCLIENT_H
#define CLOUDSPEECHCLIENT_H

#include <WiFiClientSecure.h>
#include "Audio.h"

enum Authentication {
    USE_APIKEY,
    USE_SERVICEACCOUNT
};

class CloudSpeechClient {
public:
    CloudSpeechClient(Authentication authentication);
    ~CloudSpeechClient();
    void Transcribe(Audio* audio);
    void QueryChatGPT(const String& question, String& answer);

private:
    Authentication authentication;
    void PrintHttpBody2(Audio* audio);
};

#endif

```

I2S.CPP CODE

```

#include "I2S.h"
#define SAMPLE_RATE (16000)
#define PIN_I2S_BCLK 26
#define PIN_I2S_LRC 22
#define PIN_I2S_DIN 34
#define PIN_I2S_DOUT 25

// This I2S specification :
// - LRC high is channel 2 (right).
// - LRC signal transitions once each word.
// - DATA is valid on the CLOCK rising edge.
// - Data bits are MSB first.
// - DATA bits are left-aligned with respect to LRC edge.
// - DATA bits are right-shifted by one with respect to LRC edges.
I2S::I2S(MicType micType) {
    if (micType == M5GO || micType == M5STACKFIRE ) {
        BITS_PER_SAMPLE = I2S_BITS_PER_SAMPLE_16BIT;
        i2s_config_t i2s_config = {

```

```

        .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX | I2S_MODE_TX |
I2S_MODE_DAC_BUILT_IN | I2S_MODE_ADC_BUILT_IN),
        .sample_rate = SAMPLE_RATE,
        .bits_per_sample = BITS_PER_SAMPLE,
        .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
        .communication_format = (i2s_comm_format_t)(I2S_COMM_FORMAT_I2S_MSB),
        .intr_alloc_flags = 0,
        .dma_buf_count = 2,
        .dma_buf_len = 1024
    };
    i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);
    i2s_set_adc_mode(ADC_UNIT_1, ADC1_CHANNEL_6);
    i2s_set_clk(I2S_NUM_0, SAMPLE_RATE, BITS_PER_SAMPLE,
I2S_CHANNEL_STEREO);
    i2s_adc_enable(I2S_NUM_0);
}
else if (micType == ADMP441 || micType == ICS43434 ) {
    BITS_PER_SAMPLE = I2S_BITS_PER_SAMPLE_32BIT;
    i2s_config_t i2s_config = {
        .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX),
        .sample_rate = SAMPLE_RATE,
        .bits_per_sample = BITS_PER_SAMPLE,
        .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
        .communication_format = (i2s_comm_format_t)(I2S_COMM_FORMAT_I2S |
I2S_COMM_FORMAT_I2S_MSB),
        .intr_alloc_flags = 0,
        .dma_buf_count = 16,
        .dma_buf_len = 60
    };
    i2s_pin_config_t pin_config;
    pin_config.bck_io_num = PIN_I2S_BCLK;
    pin_config.ws_io_num = PIN_I2S_LRC;
    pin_config.data_out_num = I2S_PIN_NO_CHANGE;
    pin_config.data_in_num = PIN_I2S_DIN;
    i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);
    i2s_set_pin(I2S_NUM_0, &pin_config);
    i2s_set_clk(I2S_NUM_0, SAMPLE_RATE, BITS_PER_SAMPLE,
I2S_CHANNEL_STEREO);
}
}

int I2S::Read(char* data, int numData) {
    return i2s_read_bytes(I2S_NUM_0, (char *)data, numData, portMAX_DELAY);
}

int I2S::GetBitPerSample() {
    return (int)BITS_PER_SAMPLE;
}

```

I2S.H CODE

```
#ifndef _I2S_H
#define _I2S_H
#include <Arduino.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/i2s.h"
#include "esp_system.h"
```

```
enum MicType {
    ADMP441,
    ICS43434,
    M5GO,
    M5STACKFIRE
};
```

```
class I2S {
    i2s_bits_per_sample_t BITS_PER_SAMPLE;
public:
    I2S(MicType micType);
    int Read(char* data, int numData);
    int GetBitPerSample();
};
```

```
#endif // _I2S_H
```

NETWORK_PARAM.H CODE

```
#ifndef _NETWORK_PARAM_H
#define _NETWORK_PARAM_H
```

```
const char *ssid = "WIFI NAME";
const char *password = "WIFI PASSWORD";
const char* server = "speech.googleapis.com";
const char* root_ca="Input google cloud certificate";
const String ApiKey = "Input Google API key";
```

Appendix X

Similarity Report

The screenshot displays a similarity report interface. On the left, a vertical sidebar contains the text: "REAL-TIME VOICE-ACTIVATED CHAT-BOX DIGITAL ASSISTANT USING ESP32 AND OPENAI API" and "MASTER THESIS". The main area shows the document's header: "NEAR EAST UNIVERSITY", "INSTITUTE OF GRADUATE STUDIES", "DEPARTMENT OF ARTIFICIAL INTELLIGENCE", "REAL-TIME VOICE-ACTIVATED CHAT-BOX DIGITAL ASSISTANT USING ESP32 AND OPENAI API", "M.Sc. THESIS", and "EriOluwa ODUNLAMI". On the right, a "Match Overview" sidebar shows a total similarity of 15%. Below this, it lists five matches with their respective percentages and source types:

Match	Source	Percentage
1	forum.arduino.cc (Internet Source)	3%
2	docs.espressif.com (Internet Source)	2%
3	docs.neu.edu.tr (Internet Source)	1%
4	Submitted to Yakin Do... (Student Paper)	1%
5	qiita.com (Internet Source)	1%