MULTIDISCIPLINARY RESEARCH INSTITUTE INDIA

Open Access International Journal of Science and Engineering

ISSN: 2456-3293

Volume 9 Issue 1s (2026) | Pages: 118-127

Voice Assistance using ESP32 & Conversational AI

Ashwini Suryawanshi[1], Sharad Jagtap[2], Anil Naikwade[3], Priyanka Patange[4]

[1,2,3,4]Department of Electronics and Telecommunication Engineering, Anantrao Pawar College of Engineering and Research, Pune, India
[1]pshital29@gmail.com, [2]sharadjagtap0710@gmail.com, [3]naikwade88@gmail.com, [4]priyanka.pawar371@gmail.com

Peer Review Information

***Type**: Article*
***Received**: 29 March 2026*
***Revised**: 26 April 2026*
***Accepted**: 30 May 2026*
***Published**: 19 June 2026*

Abstract

Recent days Conversational AI is becoming more promising technology and it will speedily interact with Human machine. An advanced conversational AI model is developed by OpenAI using the ESP32 microcontroller and ChatGPT, to create a powerful voice assistant is main aim of building this project. The suggested technology seeks to revolutionize user engagement in intelligent settings by offering voice-activated, user-friendly access to data and services. The ESP32 microcontroller is the voice assistant's physical core. It has strong connectivity and can be used in many different ways. The ESP32 improves the assistant's usefulness beyond local processing by making it easy to connect to online sites [9]. The voice assistant work to become better capacity to understand and react to user inquiries in conversational English blending with ChatGPT, a state-of-the-art natural language processing and generation model. Configuring the ESP32 for audio input and output, putting advanced speech recognition algorithms into practice to record user commands, integrating ChatGPT for natural language processing capabilities, and creating an easy-to-use interface are all crucial steps in the development process. Moreover, during the development process, strict considerations are made about resource optimization, privacy, and security. This new voice assistant has a lot of features. It can give you personalised news, information, and weather updates as they happens by controlling a variety of smart home appliances, such as home theater systems, lights, and thermostats. Helping users keep track of their schedules by giving them personalised calendar events, alerts, and reminders, as well as strong web search features that make it easy to find useful information.

Keywords: Conversational AI; Privacy; Security; Resource Optimization; Natural Language Processing.

How to Cite This Article

Suryawanshi, A., Jagtap, S., Naikwade, A., & Patange, P. (2026). Voice Assistance using ESP32 & Conversational AI. *Open Access International Journal of Science and Engineering*, 9(1s), 118–127.

Introduction

The ESP32 microcontroller has a lot of features that make it great for voice assistant apps. For example, it has a dual-core CPU that can run at speeds of up to 240 MHz, which is more than enough power to do things like speech recognition and natural language processing that are needed for voice assistant features. The ESP32 has built-in Wi-Fi and Bluetooth, so the voice assistant can talk to other smart devices, use internet services, and get updates on the most

Current information in real time. Low Power Consumption: The ESP32's low power consumption design makes it perfect for voice assistant apps that need to save as much energy as possible and are always on [10].

Onboard Memory: The ESP32 can save user preferences, voice assistant settings, and other important data if it has enough onboard memory and can work with external storage devices like SD cards.

Support for Real-Time Operating Systems (RTOS): The ESP32 can run user-friendly programme code as well as an RTS. This lets you do a lot of things at once and process things in real time, exactly what you require in order to handle voice assistant tasks at the same time. A lot of help with development: The ESP32 ecosystem has a lot of development tools, like libraries, software development kits (SDKs), and community support, thereby making it easier to add and improve voice assistant features. There are many benefits to using conversational AI as a voice assistant, such as ChatGPT:

Natural Language Understanding: ChatGPT and other conversational AI models are very good at understanding natural language. This lets voice assistants correctly understand a wide range of user commands and questions. This feature makes users happier and makes interactions feel more natural.[1]

Conversational Flow: Conversational AI models let users have more interesting conversations with their voice assistants by making responses that sound a lot like how people talk. This flow of discussion makes the user and the helper feel closer to each other.

Conversational AI models, like ChatGPT, are flexible and adaptable to many different tasks and situations. This means that they can be used for more than just simple command-based interactions [12]. Conversational AI can do a lot of different things well, such as exchanging information, answering questions, or starting a casual conversation. Scalability: Conversational AI models can be implemented in settings with significant user traffic since they are scalable enough to manage numerous user interactions at once. Voice assistant performance and responsiveness are maintained even with high usage thanks to its scalability. Integration with Current Systems: Conversational AI models, such as ChatGPT, can be easily incorporated into current voice assistant ecosystems and platforms, enabling developers to take advantage of their capabilities without having to start from scratch [11]. Voice assistant solution implementation is accelerated by this integration, which also streamlines the development process.

To summarize, the incorporation of Conversational AI as a voice assistant results in increased user engagement, better interaction quality, and easier, more intuitive communication between users and speech-activated devices.

Related Work

Speech recognition using ESP 32 S3 N8R8

1. ESP Skainet:

Espressif's intelligent voice assistant, ESP-Skainet, is currently compatible with Speech Command Recognition and the Wake Word Engine.It is advised to use the ESP32-S3 spoken command recognition system, which has high-speed octal SPI PSRAM and AI instruction capability [12]. First, the most recent models will be used with ESP32-S3.ESP-Skainet is Espressif's smart voice assistant, which currently supports a wake-word engine (WakeNet), an offline speech-recognition engine (MultiNet) and acoustic front-end algorithms. Skainet is a part of Espressif's complete AIoT solution that combines ESP32 with an artificial intelligence (AI) development framework.

In the most practical manner, ESP-Skainet facilitates the construction of wake word detection and spoken commands recognition applications built around the ESP32 series chip from Espressif Systems[11]. Applications for speech command recognition and wake word detection can be quickly developed with ESP-Skainet.

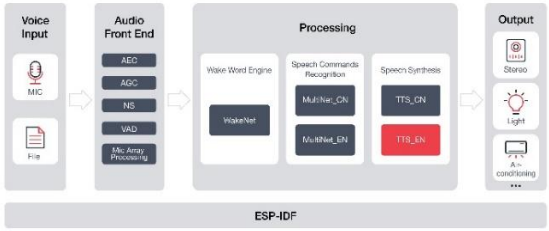

Fig. 1. *ESP Skainet Block Diagram*

Generally speaking, the following ESP-Skainet features will be supported:

2. WakeNet wake word Engine

WakeNet is a neural network-based wake word engine designed for low-power embedded microcontrollers. WakeNet can currently handle up to five wake words. The Flow diagram of the WakeNet

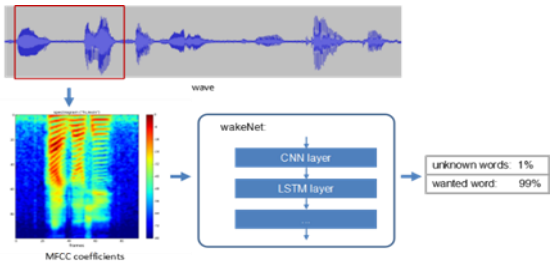

Fig. 2. *WakeNet Flow Diagram*

- Speech Feature

We utilize the MFCC technique in order to extract the speech spectrum information.. The input audio file is signed 16-bit encoded and has a sample rate of 16 KHz in mono. The window width and step size of each frame are 30 ms.

- Neural Network

The ninth version of the neural network structure has now been revised, and among the changes are:

1. There had been no purpose for WakeNet1, WakeNet2, WakeNet3, WakeNet4, WakeNet6, and WakeNet 7.
2. WakeNet5 is limited to ESP32 chip support.
3. WakeNet8 and WakeNet9, which are based on the Dilated Convolution structure, are limited to supporting the ESP32-S3 processor.
4. Keyword Triggering Method: We calculate the average recognition results (M) for several frames in a continuous audio stream and generate a smoothing prediction result in order to improve the accuracy of keyword triggering. A triggering command is only sent when the M value exceeds the predetermined threshold.

- WakeNet Procedure

1. Select WakeNet model
2. Run the WakeNet
3. The AFE currently includes WakeNet, which is turned on by default, and uses the AFE fetch interface to return the d
4. etection results.
5. To enable/disable WakeNet :
   afe_handle -> disable_wakenet(afe_data)
   afe_handle -> enable_wakenet(afe_data)

- MultiNet Command Word (Speech) Recognition Model

Based on ESP32, MultiNet is a model that can identify several speech command words offline. At the moment, 200 spoken commands—including personalized commands—can be recognized.

1. Recognize and assist English spoken instructions
2. Allow for user-defined instructions
3. Support for adding, removing, and changing commands while they're running
4. It supports up to 200 commands.
5. Both continuous and single recognition are supported.
6. Minimal weight and resource use a short latency of 500 ms
7. The concept has distinct partitions to facilitate users in applying over-the-air
8. The audio input used in MultiNet is the 16 KHz, 16 bit, mono audio that has been processed by the audio-front-end algorithm (AFE) [8]. Speech commands are recognized by means of audio signal recognition.

Flow diagram for commands recognition below

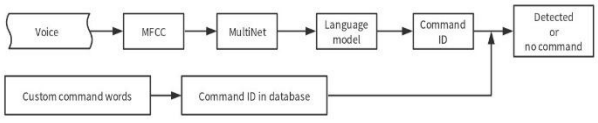

Fig. 3. *MultiNet Block Diagram*

MultiNet7 speaks in English using phonemes for speech instructions.

 # command_id, command_grapheme, command_phoneme

1,tell me a joke,TfL Mm c qbK

2, sing a song,Sgl c Sel

- Column 1: It is for command ID, which is the ID for each specific command in the code it should start from 1 and cannot be set to 0.
- Column 2: The command sentence, or command_grapheme. Unless the acronym is intended to be spoken differently, it is advised to use lowercase letters.
- Column 3: contains the optional command phoneme sequence, command_phoneme. we can use tool/multinet_g2p.py to convert graphemes to phonemes in order to populate this column; the preferred method is to insert the results at the third column.

MultiNet Procedure

- Load and initialize MultiNet model.
- Customize the commands.
- Running MultiNet on ESP IDF
- After turning on AFE and WakeNet, users can launch MultiNet, but they need to be aware of the following restrictions:
  - The MultiNet frame length needs to match the AFE fetch frame length.
  - 16 KHz, 16 bit, mono audio format support is provided. This format also applies to the data acquired through AFE fetch.
- Determine the frame length that MultiNet needs to pass. As an illustration [2]

  int EN_framesize = multinet > get_samp_framesize (model_data);
- Each frame that is given to MultiNet is described by its "EN_framesize." This size is precisely equal to the AFE-obtained amount of data points per frame [10].
- Start the speech recognition

We send the data from AFE fetch to the following API:

esp_mn_state_t mn_state = multinet->detect (model_data, buff);

the total size of buffer is

"EN_framesize * sizeof(int16_t)".

Esp 32 voice assiatant using Home Assistant OS

T An open-source operating system designed specifically to operate the well-known home automation platform Home Assistant is called Home Assistant OS.

- Centralized Management & Control: Home Assistant OS serves as a single center for home automation, making it easier to manage connected devices and automation schedules. By adding Home Assistant OS to voice assistant projects, developers enable customers to carry out intricate automation sequences and routines with just voice commands, which simplifies home management chores and boosts productivity all around [9].
- Adaptable Scripts and Automations: Because Home Assistant OS has so many customization options, developers can construct customized scripts and automations that are suited to specific needs and tastes. By utilizing these features, voice assistants that are linked with Home Assistant OS may carry out complex automation sequences in response to user inputs, giving smart home environments more flexibility and capability [2].
- Privacy and Security: Any smart home ecosystem must prioritize security and privacy, and Home Assistant OS does just that by implementing strong security features including user authentication, encryption, and secure remote access [7]. Developers may guarantee the security and privacy of user data by using Home Assistant OS in voice assistant projects, which builds user confidence and trust in the system.
- Support from the Community and Integrations: A thriving community of enthusiasts and developers supports Home Assistant OS by developing plugins, integrations, and unique components [4]. Developers can extend the functionality of their systems and improve interoperability with a vast array of smart home products and services by integrating voice assistants with Home Assistant OS, which gives them access to a plethora of community-developed resources.

1. Home assistant Assist pipeline

The Assist pipelines in Home Assistant are composed of different parts that come together to generate a voice assistant.

Various alternatives to select from for each component. Both text-to-speech and speech-to-text are fully local options.

- Whisper

Whisper is the speech-to-text Model by OpenAI model with multilingual support and it is Open-source. We make use of faster-whisper, a forked version. It takes about 8 seconds for a Raspberry Pi 4 to process incoming voice commands. It takes less than a second to complete on an Intel NUC.

Several speech processing tasks, such as multilingual speech recognition, speech translation, spoken language identification, and voice activity detection, are used to train a Transformer sequence-to-sequence model[1]. Several steps of a conventional speech-processing pipeline can be replaced by a single model since these tasks are collectively represented as a series of tokens that the decoder must anticipate [5]. A collection of unique tokens that function as task specifiers or classification goals are used in the multitask training format.

Table 1. *Whisper Processing Models*

Parameters	Multilingual model	Required VRAM	Relative Speed
39 M	tiny	~1 Gb	~32x
74 M	base	~1 GB	~16x
244 M	small	~2 GB	~6x
769 M	medium	~5GB	~2x
1550 M	large	~10 GB	~1x

Each of the five model sizes comes in four English-only versions that compromise accuracy for speed. Below is a list of the accessible models along with an estimate of how much memory they require and how quickly they infer from the large model [3]. Numerous factors, such as the hardware that is available, could affect the actual speed.

- Piper

Piper is our text-to-speech system. Optimized for the Raspberry Pi 4, Piper is text-to-speech model with numerous trained voices [6]. Multiple languages are supported. When utilizing models of moderate quality on a Raspberry Pi, it can create 1.5s of voice per second [1].

The following system is used for naming choices: "name"-"quality>-<language>_<REGION> The <name> section is derived from either the speaker's name, if given, or the dataset used to train the voice.

Four levels of quality exist for a voice:

i. x_low - 16Khz, smallest/fastest
ii. low - 16Khz,
iii. fast medium - 22.05Khz, slower but better sounding
iv. high - 22.05Khz, slowest but best sounding

Only the medium models will operate at a tolerable pace on a Raspberry Pi 4. The low or x-low voices are preferable if audio quality is not important because they will sound much faster than medium.

- Wyoming Protocol

An interposes event protocol over stdin/stdout for Rhasspy v3.
The command line interfaces of many voice programs are comparable [1]. For instance, the majority of text-to-speech applications receive text via standard input and output a WAV file or a file to standard output. example:

echo “Input text” | text-to-speech > output.wav

Standard input/output protocols would work with any language, operating system, etc. But certain voice applications require the production or consumption of audio or event streams. For instance, if a speech-to-text system can process audio while it's being captured, it might provide a result considerably more quickly.

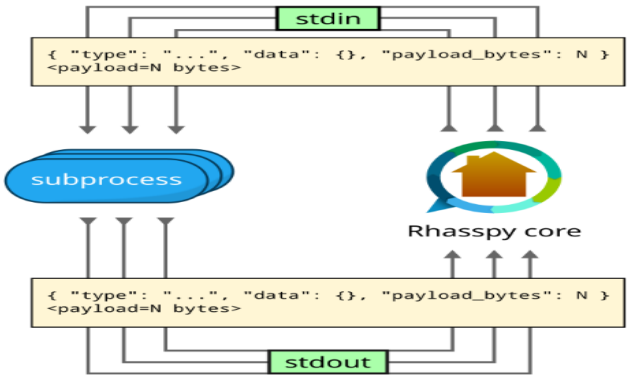

Fig. 4. *Wyoming Protocol Process*

- Event streams

Though they are designed to be byte streams, standard input/output can also be simply modified to function as event streams that transport binary data. This allows us to send events indicating that the stream has ended as well as audio segments to speech-to-text programs, for example. Everything without a plug or a broker.

Each event in the Wyoming protocol is:

A single line of JSON with an object:

1. MUST have a type field with an event type name
2. MAY have a data field with an object that contains event-specific data
3. MAY have a payload_length field with a number > 0
4. If payload_length is given, exactly that may bytes follows

Example:

{"type": "audio-chunk", "data": {"rate": 16000, "width", "channels": 1},
"payload_length": 2048 }
<2048 bytes>

- Using events over standard input/output unfortunately means we cannot talk to most programs directly [4]. Fortunately, programs with similar command-line interfaces can share and write simple adapters [5]. While speaking events to Rhasspy, the adapter calls the underlying software using a standard protocol such as "text in, WAV out."

- Event Types: Voice programs vary significantly in their options, but programs within the same domain have the same minimal requirements to function:

mic

1. Audio input
2. Outputs fixed-sized chunks of PCM audio from a microphone, socket, etc.
3. Audio chunks may contain timestamps

wake

1. Wake word detection
2. Inputs fixed-sized chunks of PCM audio
3. Outputs name of detected model, timestamp of audio chunk

asr

1. Speech to text
2. Inputs fixed-sized chunks of PCM audio
3. Inputs an event indicating the end of the audio stream (or voice command)
4. Outputs a transcription

vad

1. Voice activity detection
2. Inputs fixed-sized chunks of PCM audio
3. Outputs events indicating the beginning and end of a voice command.

intent

1. Intent recognition
2. Inputs text
3. Outputs an intent with a name and entities (slots)

handle

1. Intent/text handling
2. Does something with an intent or directly with a transcription
3. Outputs a text response

tts

1. Text to speech
2. Inputs text
3. Outputs one or more fixed-sized chunks of PCM audio

snd

1. Audio output
2. Inputs fixed-sized chunks of PCM audio
3. Plays audio through a sound system

The following event types are currently defined:

Table 2. *Event types*

Domain	Type	Data
audio	audio-start	timestamp,rate, width, channels
audio	audio-chunk	timestamp,rate,width, channels
audio	audio-stop	timestamp
wake	detection	name, timestamp
wake	not-detected	
vad	voice-started	timestamp
vad	voice-stopped	timestamp
asr	transcript	text
intent	recognize	text
intent	intent	name, entities
intent	not-recognized	text
handle	handled	text
handle	not-handled	text
tts	synthezize	text
snd	played	

Methodology

Installation of a local Assist pipeline

To convert text to speech and vice versa, install the add-ons.

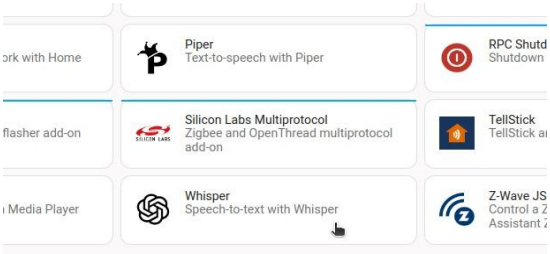

Fig. 5. *Whisper And Piper Add-ons*

1. Install the Open Wake Word add-on in order to utilize a wake word.
2. Start the add-ons by configuring them properly.
3. Navigate to the integrations under Settings > Devices & Services once the add-ons have started.
4. Wyoming integration discovering Piper and Whisper
5. You can choose Configure for each integration. After everything is set up, Piper and Whisper are visible in a single integration, along with Open Wake Word if desired.

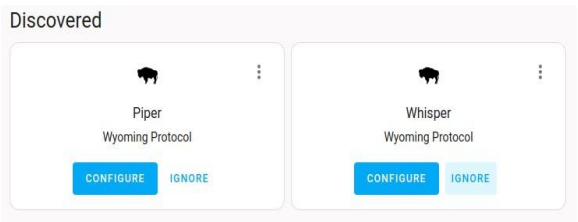

Fig. 6. *Configuring Wyoming protocol for Integration*

Setting up Assistant Pipeline

1. Go to S Select Add assistant after choosing Settings > Voice assistants.
2. Troubleshooting: If this screen displays no assistants, then the default configuration is not being used [12]. In this scenario, your configuration needs to include the following.yaml document:
   # Example configuration.yaml entry

assist_pipeline:

1. Enter the appropriate name.
2. Choose the language that you would like the model to learn.
3. Choose Home Assistant from the list of conversation agents.
4. Select faster-whisper as speech to text Model.
5. Select piper as text to speech Model.
6. It is possible to select form multiple language variants.
7. There are predefined wake words can choose one of them.

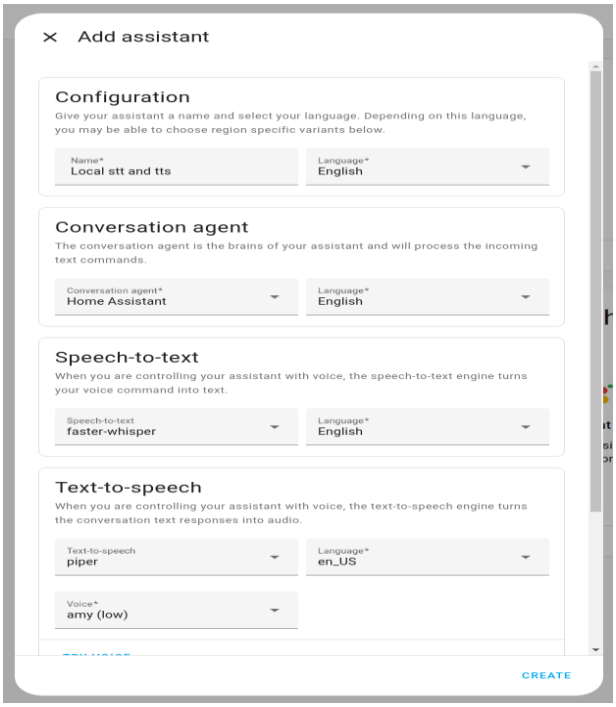

Fig. 7. *Setting up appropriate integrations in voice assiatant pipeline*

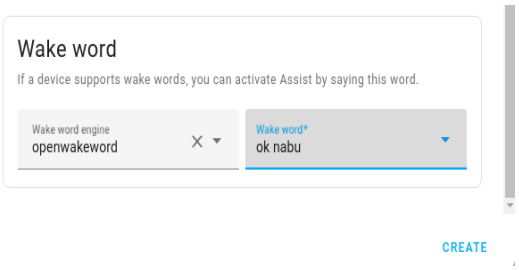

Fig. 8. *Selecting Wake word*

Voice commands can be processed locally on your device.Exposing Device to Assist is the must do procedure or it will be uncontrollable with the voice.

Working

The assistant pipeline in a home assistant system with a voice assistant function usually goes through a series of stages in order to receive user instructions and deliver pertinent answers [7]. Following are the working steps

- Speech Input: The user talks to the voice assistant and gives it commands or asks it questions. The voice input is recorded by a microphone that is connected to the system.

- Speech Recognition: A module for voice recognition receives the recorded speech input and turns it into text [3]. This module might be able to accurately transcribe the audio input by using things like automated speech recognition (ASR).
- Natural Language Understanding (NLU): After the text is transcribed, it goes to the NLU module, which looks at what the user wants and pulls out relevant information from the text. NLU methods like entity extraction and intent recognition are used to fully understand what the user wants.
- Classification of Intent: The system figures out what the user wants to do and what action or reaction is needed based on the information it has gathered [2]. For example, you could call the user who wants to turn on the lights a "control device."
- Action Selection: After figuring out which action the user wants, the system picks the best way to meet that need. This could mean sending commands to smart home devices to do things like turn on lights, change the temperature, or play music. It could also mean getting information from the smart home database.[5]
- Response Generation: The system notifies the user of the result by generating a response once the required tasks have been completed. The voice assistant may respond by saying this out loud or, if one is available, by displaying it on a screen.
- The response is designed to be informative and user-friendly, providing feedback on the executed action or acknowledging the user's request.
- User Interaction Loop: To further include the user, the system could also include a feedback loop. This could entail making recommendations, posing follow-up queries, or, depending on the circumstances of the discussion, providing more details.

Applications

- Automation and Control: we can automate operations, change settings, and take hands-free control of a variety of smart home systems and devices that are connected to the ESP32 by using this system.
- Information Retrieval: Users can ask voice assistants for information like news headlines, weather updates, calendar events, and more. The ESP32 makes it possible to get and send this information in real time.
- Accessibility: Conversational AI-powered voice assistants increase inclusivity and usability by making technology more approachable for people with impairments or those who would rather interact hands-free.
- Entertainment and Education: Voice assistants can entertain and educate users by interacting with them through storytelling, educational quizzes, trivia games, and other activities.
- Remote Control and Monitoring: Users can remotely control and monitor their smart home devices using voice commands, even when they are away from home, leveraging the ESP32's connectivity capabilities for remote access.
- Language Translation: Integrating conversational AI with ESP32 allows voice assistants to serve as language translation tools, enabling users to communicate with people who speak different languages by translating speech in real-time.

Results

In our project, we have successfully achieved the following aspects:

- Cost-Effectiveness: Our choice of using ESP32 microcontrollers for voice assistants aligns with the goal of affordability. By opting for these microcontrollers, we have ensured that our project remains cost-effective, making it accessible to hobbyists, do-it-yourself, and small-scale projects with budget constraints[1].
- Custom Voice Models: Voice assistants can be taught with custom voice models that are suited to particular accents, dialects, or speech patterns thanks to conversational AI, which will increase understanding and accuracy for users from a variety of linguistic backgrounds.
- Open-Source Community: Leveraging the vibrant open-source communities surrounding ESP32 microcontrollers and conversational AI frameworks, we have utilized a wealth of resources, tutorials, and community-contributed libraries. This has significantly expedited our development process and encouraged innovation within our project.
- Data Ownership and Control: In our implementation, we prioritize data ownership and control by deploying voice assistants with ESP32 microcontrollers. We ensure that voice data processing and storage occur locally on the device, reducing reliance on cloud services and addressing privacy concerns.
- Natural Language Understanding: Through the integration of conversational AI, our voice assistant demonstrates advanced natural language understanding capabilities. It comprehends and responds to user commands and queries in a manner that mimics human-like interaction, enhancing usability.
- Seamless Integration: Thanks to the Wi-Fi and Bluetooth connectivity options provided by ESP32 microcontrollers, our voice assistant seamlessly integrates with other smart home appliances and services[1]. This enables users to control various devices within their smart home ecosystem using voice commands.

By incorporating these features into our project, we have created a versatile and user-friendly voice assistant solution that meets the needs of diverse users while ensuring affordability, privacy, and seamless integration with smart home environments.

Conclusion

The voice assistant function of the home assistant system can efficiently comprehend user requests, take appropriate action, and respond with information by using this assistant pipeline. This improves user experience and makes it possible to connect with smart home services and devices in a seamless manner.

References

1. Vikas Hassija1 , Arjab Chakrabarti 2 , Anushka Singh 2 , Vinay Chamola 3,4, (Senior Member, Ieee), And Biplab Sikdar 5 , (Senior Member, IEEE) "Unleashing the Potential of Conversational AI: Amplifying Chat-GPT's Capabilities and Tackling Technical Hurdle", Digital Object Identifier 10.1109/ACCESS.2023.3339553, 5 December 2023
2. V. Gaur and N. Saunshi, ''Symbolic math reasoning with language models,'' in Proc. IEEE MIT Undergraduate Res. Technol. Conf. (URTC), Sep. 2022, pp. 1–5.
3. K. Chen, T. Zhao, M. Yang, L. Liu, A. Tamura, R. Wang, M. Utiyama, and E. Sumita, ''A neural approach to source dependence based context model for statistical machine translation,'' IEEE/ACM Trans. Audio, Speech, Language Process., vol. 26, no. 2, pp. 266–280, Feb. 2018.
4. M. Yang, C. Li, Y. Shen, Q. Wu, Z. Zhao, and X. Chen, ''Hierarchical human-like deep neural networks for abstractive text summarization,'' IEEE Trans. Neural Netw. Learn. Syst., vol. 32, no. 6, pp. 2744–2757, Jun. 2021
5. S. Kusal, S. Patil, J. Choudrie, K. Kotecha, S. Mishra, and A. Abraham, ''AI-based conversational agents: A scoping review from technologies to future directions,'' IEEE Access, vol. 10, pp. 92337–92356, 2022.
6. F.-Y. Wang, J. Li, R. Qin, J. Zhu, H. Mo, and B. Hu, ''ChatGPT for computational social systems: From conversational applications to human-oriented operating systems,'' IEEE Trans. Computat. Social Syst., vol. 10, no. 2, pp. 414–425, Apr. 2023.sensitive generation of conversational responses,'' 2015, arXiv:1506.06714.
7. S. Wang, Y. Yang, Z. Wu, Y. Qian, and K. Yu, ''Data augmentation using deep generative models for embedding based speaker recognition,'' IEEE/ACM Trans. Audio, Speech, Language Process., vol. 28,pp. 2598–2609, 2020.
8. S. AlZu'bi, A. Mughaid, F. Quiam, and S. Hendawi, ''Exploring the capabilities and limitations of ChatGPT and alternative big language models,'' Artif. Intell. Appl., pp. 1–5, Apr. 2023.
9. G. A. Santos, G. G. de Andrade, G. R. S. Silva, F. C. M. Duarte,J. P. J. da Costa, and R. T. de Sousa, ''A conversation-driven approach for chatbot management,'' IEEE Access, vol. 10, pp. 8474–8486, 2022.
10. K.-M. Kim, W.-J. Ryu, J.-H. Park, and S. Lee, ''MeChat: In-device personal assistant for conversational photo sharing,'' IEEE Internet Comput., vol. 23, no. 2, pp. 23–30, Mar. 2019.
11. Ashwini S. Shinde, Sharad S. Jagtap, Vishal Puranik, "Identification and sorting of power quality disturbances using signal processing with GUI," 10.1109/ICAECCT.2016.7942556,IEEE June 2017.
12. Jagtap Sharad Sarjerao;G. Sudhagar, "A Deep Learning Approach to Irrigation Management in Smart Agriculture" IEEE Pune Section International Conference (PuneCon), IEEE December 2023