Implementation and Evaluation of LLM-Based Conversational Systems on a Low-Cost Device

Koga Sakai Yuji Uehara Shigeru Kashihara
Faculty of Information Science and Technology
Osaka Institute of Technology
Osaka, Japan
{e1q21042, e1n21017, shigeru.kashihara}@oit.ac.jp

***Abstract*—The rapid evolution of artificial intelligence (AI) technologies has highlighted the potential of generative AI, particularly large language models (LLMs), to revolutionize various sectors by automating content creation, enhancing personalized education, supporting medical diagnostics, and creating immersive entertainment experiences. However, deploying these advanced models often requires substantial computational resources, posing a challenge in resource-limited environments. This paper explores implementing LLM-based conversational systems on a low-cost device, specifically the Raspberry Pi. We designed and implemented a conversational system utilizing four LLMs, ChatGPT, Bard, Llama, and Rinna, and evaluated their performance in terms of execution time and computational resource usage. Our findings reveal that API-based models (ChatGPT and Bard) are more efficient in processing time and resource consumption, making them suitable for real-time applications. In contrast, locally-run models (Llama and Rinna) provide the advantage of offline operation despite higher computational demands.**

***Index Terms*—LLM, Conversational system, Raspberry Pi, Chat GPT, Bard, Llama, Rinna**

I. Introduction

The evolution of artificial intelligence (AI) technologies has spotlighted the emergence of generative AI. Traditional AI models have focused primarily on prediction, classification, and analysis. However, large language models (LLMs), such as the Generative Pre-trained Transformer (GPT), can now effortlessly generate various data types, including text and images. This text and image generation enhancement significantly benefits multiple sectors, potentially revolutionizing content production, education, healthcare, and entertainment. For instance, in content production, it can automate the creation of articles and videos. In education, personalized learning materials can be created [1]. In healthcare, it can support auxiliary diagnosis [2]. In entertainment, it can create realistic virtual characters and scenes.

While the advances in generative AI hold immense promise, they also pose a significant risk of exacerbating the digital divide, particularly in environments with limited ICT resources. However, low-cost, versatile devices such as the Raspberry Pi (RasPi) are gaining attention due to their potential to reduce barriers to access to generative AI. This potential could make a significant contribution to bridging the growing divide. It is crucial to explore the performance of LLMs on devices with limited processing power or in situations that do not have network connectivity, as this can further enhance the use of generative AI.

This paper focuses on the practical aspects of implementing LLMs on a low-cost device, the RasPi. Traditionally, conversational systems have been implemented using natural language processing (NLP) and machine learning technologies [3]. Previous research in this area has attempted to facilitate human-like conversations. The recent emergence of LLM technology provides an opportunity to solve this challenge [4]. We then design and implement a conversational system using four LLMs: ChatGPT, Bard, Llama, and Rinna. Processing power can be limited depending on the LLM used. In a conversational system, such limitations can lead to longer processing times, increased user latency, and increased conversational stress. We will also evaluate the impact of the different LLMs used in the system on execution time and computational resources.

The organization of this paper is as follows: Section II reviews related work in conversational AI and LLMs, highlighting previous approaches and identifying gaps that our work aims to fill.

TABLE I: Comparison of Systems and Their Features

System	Hardware	Multilingual Support	Extension Ease	Using LLM
DIVA [5]	RasPi	No	Yes	No
AISHA [6]	Amazon Alexa	Yes	No	Yes
LingoAI [7]	PC	Yes	No	Yes
C^2-PILS [8]	PC	No	Yes	Yes
uTalk [9]	PC	Yes	Yes	Yes
Stack-chan [10]	M5Stack	No	Yes	Yes
Our System	RasPi	Yes	Yes	Yes

Section III explains the system designs, and Section IV explains how our system was implemented. Section V outlines the methodology used to evaluate the system's response time and resource utilization performance. It presents the results of our experiments, providing insights into the efficiency and limitations of our approach. Finally, Section VI concludes with our remarks.

II. Related Work

In this section, we discuss related work on conversational systems. Table I shows a comparison of the features of this study and related studies. Before the advent of LLM, Bajaj et al. proposed an NLP-based conversational system, "DIVA [5]." It could provide home automation and BI tools on RasPi. However, natural conversation is difficult due to the search-based knowledge. As a system using LLM, Lee B. et al. proposed "AISHA [6]" in the medical field to answer medical questions. In the educational field, Nakazato et al. proposed a foreign language teaching system, "LingoAI [7]," while "C^2-PILS [8]" and "uTalk [9]" have shown that LLM-based conversational systems are helpful for customer service. Unlike existing products, LLM chatbots do not require predefined responses, allowing for more natural and fluid conversations. One product that takes advantage of this is "Stack-chan [10]," which uses M5Stack.

Four LLMs are employed in this study: ChatGPT, Bard, Llama, and Rinna. These LLMs were chosen because they run on RasPi. ChatGPT, provided by OpenAI, is widely used due to their readily available API [11]. Bard, provided by Google, can generate responses integrated with other Google services currently provided by Gemini. Llama, a model of Meta that is redistributable and has fostered various derivative models, and Rinna, developed by Rinna Corporation, is a GPT language model known for its high versatility. This paper examines the possibilities offered by using these LLMs in a low-cost device, i.e., RasPi.

III. System Designs

In this paper, we implement a conversational system using LLMs. The system initiates a user-friendly conversation with a simple call of its assigned name. User inputs are converted into text, processed by the LLM to generate responses, and voiced through speech synthesis. This user-centric system also includes additional functions such as photo-taking and multilingual capabilities, enhancing its ease of use and convenience.

Figure 1 illustrates the system design. The RasPi is equipped with a microphone, speaker, and camera. During a conversation, the system activates when the user calls its assigned name through the microphone in function A. The user's speech is recorded and converted into text in function B, and processed by the LLM to generate a response in function C. This response is then vocalized through the speaker in function D.

The system also includes a camera and multilingual support functions as optional features. The camera can take photos using the user's voice command and send them to contacts via the LINE application in function E. The multilingual feature allows users to switch the language of interaction via voice commands in function F.

IV. Implementation

In this section, we describe the implementation of the conversational system designed in Section III. As shown in Figure 1, the system consists of six functions. Sections IV-A to IV-D explain the conversational system's core functions, while Section IV-E covers implementing optional features. These functions are implemented on a Raspberry Pi 4 Model B with 8GB of RAM. Table II lists the libraries and APIs used for implementing these functions.

A. Wake Word Detection

The system initiates a conversation when the user calls out the system's assigned name. Figure 2A depicts the processing flow of the wake word detection function. In a standby state, the system waits for

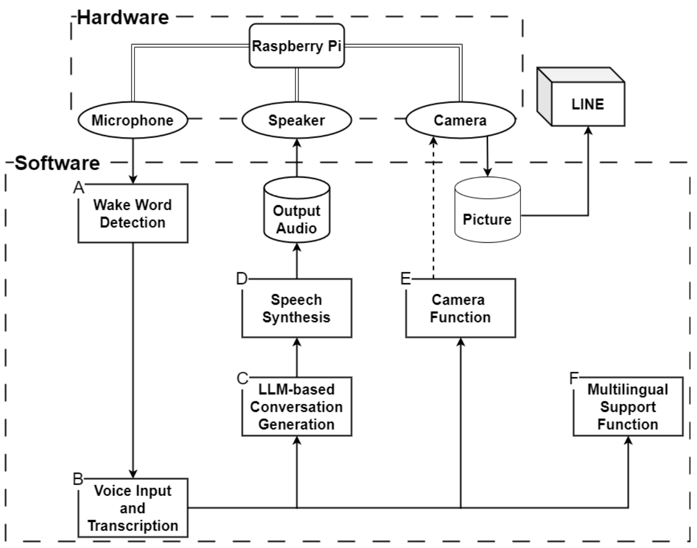


Fig. 1: System Design

TABLE II: List of Libraries and APIs Used for Functions

Function	Library or API	Purpose
A. Wake word detection	pykakasi [12]	Katakana conversion
B. Voice input	speech_recognition [13]	Recording microphone input
B. Transcription	openai-whisper [14]	Operating Whisper locally
	openai [15]	Operating Whisper API
C. LLM-Based conversation generation	openai [15]	Operating ChatGPT
	bard-api [16]	Operating Bard
	transformers [17]	Running models on HuggingFace
	llama-cpp-python [18]	Load Llama model
D. Speech synthesis	gTTS [19]	Operating Google Text-to-Speech
	pyaudio [20]	Playing sound
E. Camera	opencv-python [21]	Capturing camera input
	requests [22]	Send request to LINE app

TABLE III: Large Language Models

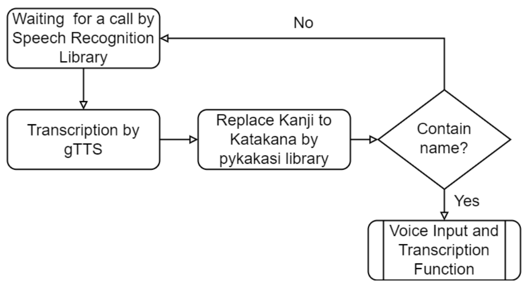
LLM Name	Model
ChatGPT	gpt-4-1106-preview
Bard	bard-api (unofficial)
Llama	Llama-2-7B-Chat-GGML
Rinna	japanese-gpt-neox-3.6b-instruction-sft

the user to say the assigned name. Google Speech Recognition, supported by the “SpeechRecognition” library, converts the audio input to text. The function then checks if the converted text contains the registered name. The conversation begins if the name is detected; otherwise, the system continues to wait.

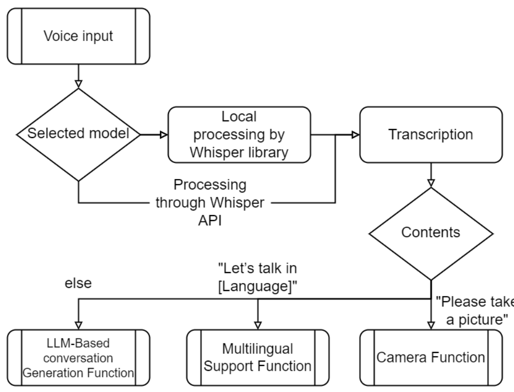
When this system is used in Japanese, it cannot recognize the same reading if converted using different kanji. In Japanese, multiple kanji characters apply to the same reading. To address this linguistic challenge, the system utilizes “Pykakasi”, a Python Natural Language Processing library, to perform accurate recognition by converting to katakana because the katakana can uniquely represent the pronunciation of Japanese readings. Also, this system uses the Whisper API to convert speech into text during conversation. However, the cost of the Whisper API is reduced by using Google Speech Recognition, which is available free of charge. Note that the input speech is saved in WAV format.

B. Voice Input and Transcription

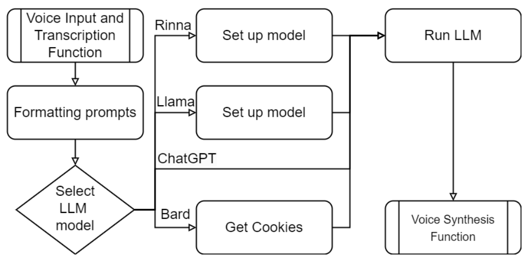
After the wake word detection, the system records the user’s speech and transcribes it into text. Figure 2B illustrates the processing flow of this functionality. The system employs SpeechRecognition to capture audio input and save it in WAV format. The audio file is then transcribed using OpenAI’s Whisper [11]. Different actions are triggered based on the transcribed text: if a voice command to activate the camera function is detected, the camera function is activated; if a command to switch


Waiting for a call by Speech Recognition Library
No
Transcription by gTTS
Replace Kanji to Katakana by pykakasi library
Contain name?
Yes
Voice Input and Transcription Function

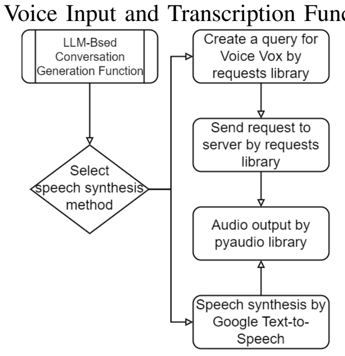
(A) Wake Word Detection Function


Voice input
Selected model
Local processing by Whisper library
Transcription
Processing through Whisper API
Contents
else
"Let's talk in [Language]"
"Please take a picture"
LLM-Based conversation Generation Function
Multilingual Support Function
Camera Function

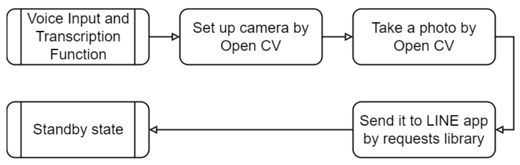
(B) Voice Input and Transcription Function



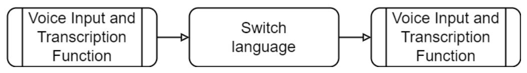
(C) LLM-Based Conversation Generation Function


LLM-Bsed Conversation Generation Function
Select speech synthesis method
Create a query for Voice Vox by requests library
Send request to server by requests library
Audio output by pyaudio library
Speech synthesis by Google Text-to-Speech

(D) Speech Synthesis Function


Voice Input and Transcription Function
Set up camera by Open CV
Take a photo by Open CV
Send it to LINE app by requests library
Standby state

(E) Camera Function


Voice Input and Transcription Function
Switch language
Voice Input and Transcription Function

(F) Multilingual Support Function

Fig. 2: Processing Flows for Functions

languages is detected, the multilingual feature is activated. Otherwise, the conversation processing continues. While processing the text with an LLM can provide more flexible context capture, this function does not use LLM processing to reduce response time. Note that the audio files created by this functionality are automatically deleted at the end of the conversation.

Both local and API models of Whisper are available for transcription. While the local model is free, it requires significant computational resources, making it unsuitable for use on a RasPi, considering processing time. Though incurring costs, the API model has a lower computational load and supports near-real-time operation.

C. LLM-Based Conversation Generation

The system can generate conversations using one of four LLMs: ChatGPT, Bard, Llama, or Rinna. Table III lists the API models for each LLM, and Figure 2C illustrates the processing flow of this functionality. For ChatGPT, the system uses the gpt-4-1106-preview provided by OpenAI. Since no official API for Bard was available at the time of system development, the “bard-api” available on GitHub is used [16], allowing conversations using browser cookie information. These APIs require network processing.

On the other hand, Llama and Rinna use models provided on Hugging Face [23] [24]. The original model for Llama distributed by Meta could not be run on RasPi due to memory limitations. Therefore, using the “Llama.cpp” library, the model of Llama was converted to GPT-Generated Unified Format (GGUF) to enable it to run on the device.

D. Speech Synthesis

The text generated by the LLM is converted to speech and output through the speaker. Figure 2D illustrates the processing flow of the speech synthesis.

The system uses the Python library "gTTS" (Google Text-to-Speech) to generate MP3 audio files, which are played back using "PyAudio". While gTTS only generates robotic voices, more natural voices can be produced using VoiceVOX, Japan's widely used speech synthesis software. However, due to the difficulty of running VoiceVOX on the Raspberry Pi's OS, the engine must be activated on another computer when using VoiceVOX.

E. Optional Features

This section describes the optional features implemented in the system: the camera function and the multilingual switch function. Figure 2E illustrates the processing flow of the camera function. The function uses a web camera connected to the Raspberry Pi to capture images using OpenCV's video capture function, which is then sent to the LINE application. The LINE is one of the popular messaging applications in Japan, and images can be sent using LINE's notify service, which provides a token to send images to assigned recipients or groups. Potential applications of the camera function include monitoring and personalized use cases such as facial recognition for user identification, allowing personalized conversations based on preferences and history.

Additionally, the conversational system supports switching to languages other than Japanese. Figure 2F illustrates the processing flow of the multilingual switch function. When a target language voice command is detected during the voice input and transcription functions, the system switches to that language. However, it limits the supported languages to those recognized by Whisper.

V. System Evaluation for Different LLMs

This section describes the evaluation of our conversational system implemented using various LLMs. We assess four types of LLMs: ChatGPT, Bard, Llama, and Rinna, focusing on (1) the execution time of each process and (2) the CPU and memory usage.

A. Evaluation Metrics and Methods

Since LLM generates different amounts of responses depending on the question content, we employ the following three questions to evaluate the system performance:

(i) Tell me about the French Revolution.
(ii) Create a story with fantasy elements.
(iii) Tell me about today's weather.

Each of these questions was prepared with the intention of answering the questions regarding (i) the accuracy of logical responses, (ii) creativity and literary expression, and (iii) accurate, real-time information. All assessments were conducted in Japanese.

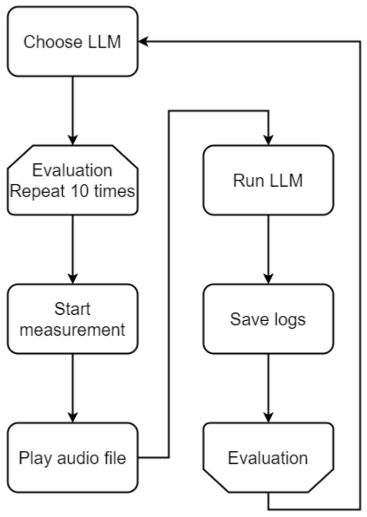


Fig. 3: Experiment Flow

We tested each of the four LLMs ten times with the three questions, following two evaluation criteria:

(1) Execution time of each process.
(2) CPU and memory usage.

For evaluation item (1), we measured the execution time for the transcription, text generation by LLM, and speech synthesis functions that constitute the conversation process. The Python "time" module was used for these measurements. Preliminary experiments indicated that external noise and variations in speech speed affect the transcription time. Therefore, we standardized the input conditions using pre-recorded audio files and measured the time from voice input to speech synthesis. For evaluation item (2), we used the Python "psutil" library to measure CPU and memory usage. In this experiment, the items were evaluated on 10 times in order to obtain an overall trend of the items evaluated in each LLM. The experiment was conducted in the flow shown in Figure 3.

B. Execution Time of Each Process

Figure 4 shows the overall processing time for the three questions, and Table IV presents the average values. The graphs indicate Llama and Rinna have longer execution times for all questions, likely because they run locally. Although they can operate on a RasPi, they are unsuitable for conversational systems. In contrast, ChatGPT and Bard process

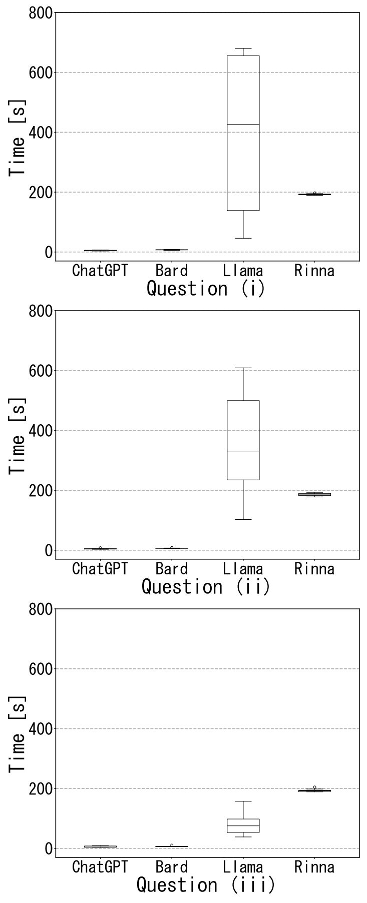


Fig. 4: LLM Execution Times

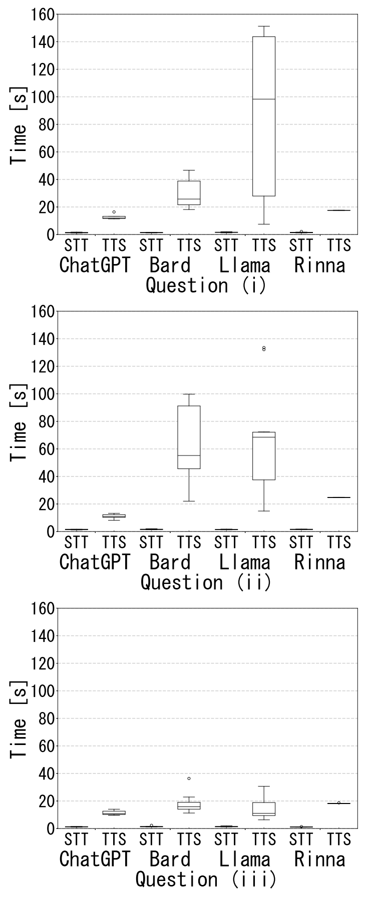


Fig. 5: Speech-to-Text and Text-to-Speech Execution Times

TABLE IV: LLM Average Processing Times

Model	Question (i)	Question (ii)	Question (iii)
ChatGPT	4.81 s	4.77 s	5.68 s
Bard	7.22 s	6.77 s	6.76 s
Llama	387.83 s	360.56 s	81.74 s
Rinna	192.72 s	185.36 s	193.81 s

TABLE V: CPU and RAM Usage

	CPU Usage	RAM Usage
ChatGPT	2.66%	20.76%
Bard	2.63%	21.20%
Llama	51.91%	23.18%
Rinna	87.60%	39.17%

requests via API over the Internet, resulting in shorter execution times.

Given the significant impact of different LLMs, we evaluate the execution time for transcription and speech synthesis processes. Figure 5 shows the processing times for the transcription (Speech-to-Text: STT) and speech synthesis (Text-to-Speech: TTS) functions for each LLM. As noted, pre-recorded audio files were used for transcription, with the lengths of the audio files for questions (i), (ii), and (iii) being 4 seconds, 5 seconds, and 3 seconds, respectively. The average time from audio completion to transcription was 1.45 seconds. However, there was variability in the speech synthesis execution time, influenced by the length of the generated text. Future improvements to ensure smooth conversation require adjustments to the amount of generated text.

C. CPU and Memory Usage

Table V presents the average CPU and memory usage for each LLM during execution. CPU usage was relatively low for ChatGPT (2.66%) and Bard (2.63%), while Llama (51.91%) and Rinna (87.60%) showed higher usage. Memory usage was in the 20% range for ChatGPT, Bard, and Llama, but Rinna approached 40%. These results indicate that ChatGPT and Bard, using APIs, have relatively low CPU and memory usage, while Llama and

Rinna, operating locally, exhibit particularly high CPU usage.

The evaluation shows that ChatGPT demonstrates the most stable performance among the LLMs. However, Bard displayed a unique ability to incorporate location information into its responses. Conversely, Llama and Rinna, functioning offline, offer the advantage of operating without a network connection. Therefore, selecting an appropriate LLM based on the ICT environment and application is crucial due to the differing characteristics and performance of each LLM.

D. Discussions

The evaluation results demonstrate the viability of utilizing LLM in an offline environment and the challenges associated with it. The findings suggest that to achieve rich conversations in an offline environment, the LLM itself must be accelerated or the processing capabilities of a modest computer must be enhanced. Furthermore, the use of lighter-weight LLMs may alleviate the burden and facilitate the development of a functional system.

Models such as Llama are also capable of customizing their responses through fine-tuning. As a result, it is feasible to develop devices that provide highly accurate medical information and to utilize them for educational purposes even in regions with limited Internet access, such as developing countries. In this evaluation, we did not assess the accuracy of the responses or the level of user satisfaction. These should be evaluated for the practical application of devices that can access LLM in an offline environment.

VI. Conclusions

This paper demonstrated the feasibility of implementing large language models (LLMs) on a low-cost device like the Raspberry Pi for a conversational system. Among the four LLMs evaluated, ChatGPT, Bard, Llama, and Rinna, ChatGPT, and Bard, which use external APIs, showed the shortest execution times and efficient resource usage, making them ideal for real-time applications. Conversely, Llama and Rinna, running locally, exhibited longer processing times and higher resource consumption but could operate without network connectivity. The findings suggest that while API-based models are optimal for environments with reliable internet access, locally-run models are advantageous in offline settings or where data privacy is crucial.

References

[1] D. Domenichini, F. Chiarello, V. Giordano, and G. Fantoni, "Llms for knowledge modeling: Nlp approach to constructing user knowledge models for personalized education," in *Adjunct Proceedings of the 32nd ACM Conference on User Modeling, Adaptation and Personalization*, ser. UMAP Adjunct '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 576–583. [Online]. Available: https://doi.org/10.1145/3631700.3665231

[2] S. J. Zhang, S., "A chatbot based question and answer system for the auxiliary diagnosis of chronic diseases based on large language model," *Sci Rep*, vol. 14, p. 17118, 2024.

[3] E. Adamopoulou and L. Moussiades, "Chatbots: History, technology, and applications," *Machine Learning with Applications*, vol. 2, p. 100006, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666827020300062

[4] C. Y. Kim, C. P. Lee, and B. Mutlu, "Understanding large-language model (llm)-powered human-robot interaction," in *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, ser. HRI '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 371–380. [Online]. Available: https://doi.org/10.1145/3610977.3634966

[5] D. Pramod, "Conversational system, intelligent virtual assistant (iva) named-diva using raspberry pi," *International Journal of Engineering Research & Technology*, vol. 8, 2020.

[6] B. Lee, O. Hossain, Z. R. Urbaez, A. T. Kshetri, and K. Mehta, "Leveraging chatgpt and amazon alexa to empower healthcare workers in sierra leone," in *2023 IEEE Global Humanitarian Technology Conference (GHTC)*, 2023, pp. 334–339.

[7] N. Hiroyuki, N. Hiroaki, and T. Hiroaki, "Application of a voice dialogue system using generative ai and 3d character for language learning (in japanese)," *RESEARCH REPORT OF JSET CONFERENCES*, vol. 2023, no. 3, pp. 36–40, 10 2023. [Online]. Available: https://cir.nii.ac.jp/crid/1390860764355040384

[8] S. Bhutada, S. S. D. Uppala, S. Nagavajyula, and T. Pottigari, "C2-pils: A chatbot using chat-gpt for pharma industry and life sciences," in *2023 International Conference on Advanced Computing Technologies and Applications (ICACTA)*, 2023, pp. 1–6.

[9] H. Azzuni, S. Jamal, and A. Elsaddik, "utalk: Bridging the gap between humans and ai," in *2024 IEEE International Conference on Consumer Electronics (ICCE)*, 2024, pp. 1–4.

[10] S. Ishikawa, "stack-chan," https://github.com/meganetaaan/stack-chan/, Github, accesed: 2024/05/31.

[11] "Openai developer platform," https://platform.openai.com/overview, OpenAI, accesed: 2024/05/31.

[12] H. Miura, "pykakasi," https://pypi.org/project/pykakasi/, PyPI, accesed: 2024/05/31.

[13] A. Z. (Uberi), "Speechrecognition," https://pypi.org/project/SpeechRecognition/, PyPI, accesed: 2024/05/31.

[14] OpenAI, "openai-whisper," https://pypi.org/project/openai-whisper/, PyPI, accesed: 2024/05/31.

[15] ——, "openai," https://pypi.org/project/openai/, PyPI, accesed: 2024/05/31.

[16] dsdanielpark, "Bard-api," https://github.com/dsdanielpark/Bard-API, Github, accesed: 2024/05/31.

[17] T. H. F. team, "transformers," https://pypi.org/project/transformers/, PyPI, accesed: 2024/05/31.

[18] A. Betlen, "llama-cpp-python," https://pypi.org/project/llama-cpp-python/, PyPI, accesed: 2024/05/31.

[19] P. N. Durette, "gtts," https://pypi.org/project/gTTS/, PyPI, accesed: 2024/05/31.

[20] H. Pham, “Pyaudio,” https://pypi.org/project/PyAudio/, PyPI, accesed: 2024/05/31.

[21] O. Team, “opencv-python,” https://pypi.org/project/opencv-python/, PyPI, accesed: 2024/05/31.

[22] K. Reitz, “requests,” https://pypi.org/project/requests/, PyPI, accesed: 2024/05/31.

[23] Rinna, “japanese-gpt-neox-3.6b-instruction-sft,” https://huggingface.co/rinna/japanese-gpt-neox-3.6b-instruction-sft, Hugging Face, accesed: 2024/05/31.

[24] TheBloke, “Llama-2-7b-chat-ggml,” https://huggingface.co/TheBloke/Llama-2-7B-Chat-GGML, Hugging Face, accesed: 2024/05/31.