HUMAN-ROBOT INTERACTION FOSTERED BY GENERATIVE ARTIFICIAL INTELLIGENCE: EXPLORING THE PRIVACY PERSPECTIVE

Sthéfano Bruno Santos Divino [*]
André de Lima Salgado [**]
Danilo Chagas Clemente [***]
Maria Eduarda Couto Grossi [****]
Rafael Bastos Andrade [*****]

Abstract: Assistive Social Robotics (SARs) refers to robots that assist human users. With technological advances and studies in Generative Artificial Intelligence, new research and projects in SARs are being developed. The OpenTera framework currently does not encompass the potential of these technologies to facilitate SARs' development. Additionally, it does not include specific conversation modules, which are essential for the robot's social functions. The objective of this study is to make adaptations to the OpenTera framework to incorporate the benefits of using the ChatGPT API in SARs' development. We also take into analysis the privacy threat involved in the design of such a prototype. The study and development were carried out by a mobile application in Flutter, using libraries for automatic speech recognition and speech production, which communicates with the ChatGPT API to respond to user queries. The study's results led to the development of a new module for adapting the OpenTera framework. Considering the results, the new module can be added to the framework, making it more adequate, with an integrated SARs module powered by generative artificial intelligence. This is an interdisciplinary and original study between Law and Computer Science, in which the results show that privacy and personal data are vulnerable to the use of new technologies.

[*] PhD and Master's degrees in Private Law from the Pontifical Catholic University of Minas Gerais, Brazil. Email: sthefanoadv@hotmail.com / ORCID iD: https://orcid.org/0000-0002-9037-0405

[**] Adjunct Professor in the Department of Computer Science, Federal University of Lavras, Minas Gerais, Brazil. Email: andre.salgado@ufla.br / ORCID iD: https://orcid.org/0000-0003-4396-1111

[***] Undergraduate student in the Department of Computer Science, Federal University of Lavras, Minas Gerais, Brazil. Email: danilo.clemente@estudante.ufla.br / ORCID iD: https://orcid.org/0009-0001-4432-7449

[****] Undergraduate student in the Department of Computer Science, Federal University of Lavras, Minas Gerais, Brazil. Email: maria.grossi1@estudante.ufla.br / ORCID iD: https://orcid.org/0009-0008-2155-6389

[*****] Undergraduate student in the Department of Computer Science, Federal University of Lavras, Minas Gerais, Brazil. Email: rafaelbastos2@gmail.com / ORCID iD: https://orcid.org/0009-0008-6235-269X

Keywords: Artificial Intelligence; Human-Robot Interaction; Privacy; Robot.

INTRODUCTION

The definition of Social Assistive Robots (SARs) proposed by Feil-Seifer and Mataric (2005) involves the intersection of Assistive Robotics (AR) and Social Interactive Robotics (SIR). In this context, AR refers to robots providing assistance or support to human users, while SIR pertains to robots interacting with users. Based on the definition presented by Feil-Seifer and Mataric (2005), we understand that the goal of SAR is to assist human users through social interaction, similar to assistive robotics. However, SAR distinguishes itself from SIR by having a more specific focus on social interaction. While in SIR, the robot aims to develop close and effective interactions with humans for the intrinsic value of the interaction itself, in SAR, the robot seeks to create close and effective interactions with a human user to offer assistance and promote measurable progress in areas such as recovery, rehabilitation, learning, etc

OpenTera is a microservices architecture framework developed to enable software and robotics engineers to prototype and deploy Human-Internet of Things (H-IoT) applications without starting from scratch each time. This comprehensive solution employs a series of independent modules for tasks such as data and session management, telemedicine, daily assistive tasks/actions, devices, and intelligent environments, all connected through the framework (Panchea et al., 2022).

Following advancements in Generative Artificial Intelligence (AI) studies and the release of ChatGPT, major companies have begun investing in studies and tools in this area. Google introduced Bard, a chatbot capable of responding to user queries (Collins, 2023). Microsoft, in collaboration with OpenAI, launched Bing Chat, which utilizes ChatGPT and can also respond to questions (Microsoft, 2023b). Additionally, there are generative image systems based on descriptions, such as Dall-E 2 and Midjourney (OpenAi, 2023b; Midjourney, 2023). It is crucial to note that, besides the mentioned examples, numerous other generative AIs have been released, and many studies and projects are ongoing in this field. With the rapid pace of innovation, new AIs will likely be launched soon, providing even broader capabilities and opportunities across various sectors.

Despite the benefits offered by generative AI, OpenTera currently does not encompass the potential of these technologies to facilitate the development of SARs. Furthermore, OpenTera does not include specific modules for conversation and emotion design, which are essential for social functions in human-robot interaction (Bartneck et al., 2020). In this context, this study aims to make adaptations to the OpenTera framework to incorporate the benefits of using the ChatGPT API in SAR development. Additionally, we take over a privacy evaluation employing threat modeling and legal discussions. This study aims to enhance the quality of SAR by developing new technologies for the rapid prototyping of robots at the same time that we take privacy by design. To achieve this, the following specific goals are proposed:

Develop a listening module for OpenTera in mobile development technology that can transcribe environmental speech into strings. Implement a module connecting to the ChatGPT API to send the transcribed string sequences.

a) Integrate a module querying the ChatGPT API to receive responses to prompts containing transcribed string sequences.
b) Construct a speech module that can transform ChatGPT responses into audio for the robot to articulate.
c) Design an animated GIF display module capable of representing emotions during the robot's listening and speech. Considering the market entry of the technologies employed in this work, our objectives have a predominantly technological focus, aiming to explore possibilities for mobile development for low-cost, rapid prototyping of SARs, to support future research in the field requiring such prototyping.

I. LITERATURE REVIEW

In this review, previous studies on topics addressed in this work will be presented, including automatic speech recognition, speech production, emotions in interaction, and generative artificial intelligence. It aims to provide an overview of the current state of knowledge on these subjects, identifying key theories to establish a solid foundation for the development of the present study, etc.

A. Automatic speech recognition

Speech recognition, also known as automatic speech recognition (ASR), computer speech recognition, or speech-to-text conversion, is a feature that enables a program to process human speech into written format (IBM, 2023).

According to Aldarmaki et al. (2022, p. 3), automatic speech recognition is defined as follows

> It is the process of automatically identifying patterns in a speech waveform. Patterns that can be detected from speech include the speaker's identity, language, emotion, and the textual transcription of the spoken utterance. The latter is generally what is sought in ASR.

In traditional ASR, as explained by Aldarmaki et al. (2022), models consist of three main components: an acoustic model, a pronunciation dictionary, and a language model. The acoustic model calculates the probability of acoustic units, such as phonemes, the smallest recognizable unit of speech, which can be modeled using Gaussian Mixture Models (GMMs) and Hidden Markov Models (HMMs). The language model calculates the probability of a sequence of words. The pronunciation dictionary is used to map the sequence of phonemes to words, generated from the combination of the two models.

Aldarmaki et al. (2022) further explain that in modern ASR, systems for recognition are end-to-end. Decoder-encoder architectures, the use of Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs) are employed to process input audio and generate a sequence of characters as output.

To work with automatic speech recognition in Flutter, various libraries are available. The azure_speech_recognition_null_safety library, implementing Azure Cognitive Speech Recognition for Flutter, allows users to utilize Azure's resources for speech recognition and transcription to text. Its documentation can be found in the public repository of Bregant (2020). The google_speech library implements the Google Speech-to-Text API plugin for the same purpose, available in the Junghans repository (2021). Flutter itself also provides a native speech-to-text feature, maintained by the community and documented in Software (2021). Various other libraries can be found on the official Dart package manager for Flutter, Google (2017).

The library chosen for the development of this work was speech-to-text for several reasons. In addition to supporting multiple languages, the developer considers it simple and easy to implement. It is perceived as requiring no additional configuration, enabling real-time speech transcription. Other libraries may require additional configurations, such as creating a Google and Microsoft account to obtain API keys, which could incur additional costs if transcription limits are exceeded (Google, 2023b; Microsoft, 2023c).

Flutter's native speech-to-text library is used for voice recognition through the device's microphone, converting the captured speech audio into text (Software, 2021).

In the developed application, three methods from the SpeechToText class in the library were used. The first method, initialize(), initializes the voice recognition service and returns true if the initialization is successful and false if any issues occur. The second method, listen(), initializes the listening mode and allows converting each recognized word into text. Finally, the stop() method interrupts listening if it is active (Software, 2021).

B. Speech production

As defined by Ning et al. (2019), speech production, also known as text-to-speech, is a technology that involves various areas such as acoustics, linguistics, digital signal processing, and statistics, with the primary goal of converting text input into speech.

> End-to-end neural Text-to-Speech (TTS) models typically first convert text into acoustic features (e.g., mel spectrograms) and then transform the mel spectrograms into audio samples (REN et al., 2019).

It is possible to work with speech production in Flutter using one of the libraries available on the official Google packages website (2017). There, a variety of options can be found that allow transforming any text into speech audio. For example, flutter_azure_tts implements the Microsoft Azure Cognitive Text-To-Speech API plugin, enabling the transformation of text into synthesized human speech in various languages (Fel, 2021). On the other hand, flutter_tts utilizes the device's resources to achieve the same result (Lutton, 2018).

For the development of this work, flutter_tts was chosen for the same reasons as the selection of speech_to_text. In addition to being robust and capable of producing speech in various languages, it is easy to implement according to the developer of this work. On the contrary, flutter_azure_tts may require additional configurations, such as the need to create a Microsoft account and obtain API keys for usage. Moreover, flutter_azure_tts may incur additional costs beyond a production limit (Microsoft, 2023c).

The flutter_tts library performs the reverse path of the speech-to-text library, i.e., it is used to transform any text into speech. This library also has several classes and methods to achieve the main goal (Lutton, 2018).

For development, three methods of the FlutterTts class were used. The first method is setLanguage(), used to set the language in which the words will be spoken. In this case, the language 'pt-BR,' representing Brazilian Portuguese, is passed as a parameter to the method. The second method is speak(), which is used to play the text as speech, expecting the text to be played as a parameter. Finally, the setCompletionHandler() method expects a callback function as a parameter. Its purpose is to call this function after the completion of the speak() method. In other words, the callback function is not called until the entire text is played (Lutton, 2018).

C. Emotions in interaction

Bartneck et al. (2020, p. 118) explain that in human-robot interactions, emotions are considered an important communication channel. In this sense, even if a robot has not been primarily designed to express emotions, people still tend to interpret them as if they were driven by emotional states. According to Bartneck et al. (2020, p. 120), "Typically, people design robots that convey emotions through facial expressions. The most common approach here is to mimic how people demonstrate emotions." In a human-robot interaction involving emotions, user satisfaction increases according to Dohsaka et al. (2014), as cited by Chaves and Gerosa (2021, p. 28), where:

> [...] proposed a chatbot that uses empathetic and self-directed emotional expressions to keep users engaged in a questionnaire-style dialogue. The research results revealed that empathetic expressions significantly improved user satisfaction. Furthermore, empathetic expressions also enhanced user ratings of peer agents in terms of intimacy, compassion, kindness, and encouragement.

Even though they are simplified emotions, animated gifs can effectively convey them. They translate into a visual language that can be easily understood and interpreted by those interacting with them. For example, an animated gif of a person smiling conveys joy and happiness, while a gif of someone frowning and appearing concerned may evoke a sense of anxiety or tension. Moreover, the repetitive nature of animated gifs allows emotions to be prolonged and emphasized. By repeating a facial expression several times, gifs can create a sense of emotional intensity, reinforcing the emotional message conveyed (Yang; Zhang; Luo, 2019).

Two animated gifs were chosen to simulate the functioning of the developed application throughout the work, both in the listening (speech-to-text) and speaking (text-to-speech) modes.

In the listening mode, the gif is represented by two eyes, symbolizing the application's attention to the person speaking (Robotics, 2021). Additionally, these eyes can convey emotions such as curiosity, interest, or empathy, reinforcing the application's ability to emotionally connect with the user. On the other hand, the gif in the speaking mode is represented by an emoji of a moving mouth, as if speaking, demonstrating various types of emotions such as joy, surprise, emphasis, or concern, depending on the context of the words spoken (Chuwy’s, 2019).

The gifs were obtained from the Giphy website, which provides free gifs for use and is available on Robotics (2021) and Chuwy’s (2019), used for the listening and speaking modes, respectively.

D. Generative AI

Zhang et al. (2023) explains that Generative Artificial Intelligence (Generative AI) is a set of tasks or applications that generate new content through AI methods from existing data. "Modern generative AI relies on various technical foundations, from model architecture and self-supervised pre-training to generative modeling methods (such as GAN and diffusion models)" (Zhang et al., 2023, p. 2).

After advances in Generative AI studies and the launch of ChatGPT, major companies have started to engage in studies and tools in this area. Google released Bard, a chatbot that responds to user queries (Collins, 2023). Microsoft, in partnership with OpenAI, launched Bing Chat, which utilizes ChatGPT and also responds to user questions (Microsoft, 2023b). There are also generative image models from introduced descriptions, such as Dall-E 2 and Midjourney (Openai, 2023b; Midjourney, 2023), respectively. It is important to note that several others have been launched in addition to those mentioned, and many studies and projects are underway in this field. With the rapid pace of innovation, new Generative AIs will likely be launched soon, offering even more resources and opportunities across various sectors.

ChatGPT is a generative artificial intelligence model developed by OpenAI. It is built on the Generative Pre-trained Transformer (GPT) architecture, which utilizes deep neural networks and machine learning techniques to understand and generate text (Heitler, 2023). The version used for the development of this work is GPT-3.5, one of the most advanced versions of OpenAI, trained with a wide range of data and knowledge until the end of the year 2021 (Natalie, 2023; OpenAI, 2023c). It also has a highly accessible API for easy integration and use across various platforms (OPENAI, 2023c).

For these reasons, it was chosen for the development of this work. For communication between the application and the OpenAI API, the native http library (Dart, 2018) was used. The API key and URL (Uniform Resource Locator) from the OpenAI API for communication between the parties were obtained from the official OpenAI website (OpenAI, 2023c). Using the library, a request was made using the POST method, passing the obtained API key in the request header and the user's speech, initially converted to text using the speech-to-text library, in the request body. After making the request, the API response is awaited, and if successful, the generated text is converted into speech, now using the text-to-speech library.

II. MATERIALS AND METHODS

The following sections outline the methods and techniques employed in conducting this study, as well as the necessary resources for its execution. The choice of methodological procedures was based on the objectives and the nature of the research.

A. Methods

This study aims to investigate the technological possibilities of adapting the OpenTera framework to incorporate the benefits of using generative Artificial Intelligence in the prototyping of Socially Assistive Robots (SARs). It is important to emphasize that the pursuit of improvements in human-robot interactions is not limited to projects exploring human needs or preferences but also encompasses exploration and technical advancement in this field (Bartneck et al., 2020, p. 57).

Therefore, this study is grounded in traditional methodological steps in the field (Bartneck et al., 2020, p. 57):

1. Definition of the research problem or question.
2. Construction of interaction.
3. Conducting tests and threat modeling.
4. Analysis of results.
5. Iteration of steps 2 to 4 until the problem or question is satisfactorily addressed.

The definition of the research problem is presented in the introductory section of this study: the OpenTera framework lacks modules for facilitating the prototyping of conversation and emotions in social robots. The subsequent sections describe the construction of SAR interaction using the OpenAI API, the development of a Flutter application, libraries, and APIs for speech recognition and translation of character strings into speech (audio) for the robot.

In these sections, tests conducted, and analyses performed to identify how the adaptation of OpenTera could be executed, enabling the incorporation of benefits from the API, are detailed. As part of the testing phase, a threat modeling process as described by Torr (2005) is incorporated. This process involves identifying potential threats and discussing them against the requirements of the Law, ensuring that the system is designed to withstand malicious attacks of all kinds.

All tests, threat modeling, and analyses were conducted based on the authors' experience as a software developer. We followed Sejnowski's (2023) steps to conduct a Turing test with the new application.

The Turing test is an evaluative process to verify a machine's ability to manifest intelligent behavior equivalent to or indistinguishable from a human being. The test involves a human evaluator posing questions to a human interlocutor and a machine interlocutor, disregarding which is which, and attempting to recognize which is the machine based on the responses. If the evaluator cannot differentiate the machine from the human, the machine passes the test (SejnowskI, 2023, p. 317).

It is worth noting that the author has one year and two months of experience as a developer in the industry, working for the company MGCode (<https://mgcode.com.br/>). During the first three months, the author worked as an intern in development. The remaining nine months were spent as a junior developer, working with Web development technologies (JavaScript, PHP, Angular, React, Java, and React Native).

B. Materials

This section presents all the materials and tools used for the development of the work. It details the installation of necessary tools and configurations that need to be made, as well as obtaining useful resources for using the ChatGPT API.

1. Installation and Configuration of Flutter on Windows

To install Flutter, the development environment must meet certain requirements, such as:

- Operating system Windows 10 or higher.
- 1.64GB of disk space, excluding IDEs.
- Windows PowerShell 5.0 or later.
- Git for Windows 2.x, with the option to use git commands in the prompt.

The latest Flutter SDK package can be downloaded from the official website and extracted to the desired installation location. For this project, C:\src was used as the installation path. The path should not contain special characters or spaces.

Then, the path to the Flutter bin folder should be added to the Path variable in the Windows environment variables. This will enable running Flutter commands directly in the command prompt, completing the installation.

Android Studio can be installed and configured by downloading and executing the executable file from the official website. The welcome screen will guide the user through the installation process. For this project, where the application was run on a physical device, the option to create an emulator was not selected. The user can choose the installation location and proceed with the installation. After the installation is finished, Android Studio can be launched.

Before installing the required components, the license policies must be accepted in the Android Studio interface. This step is mandatory to proceed with the installation.

The next step is to download the SDK Command-line Tools, which can be done in two simple steps. First, open Android Studio and access the SDK Manager option from the menu. Then, go to the SDK Tools tab and check the box for Android SDK Command-line Tools. Confirm the installation by clicking OK.

After installing the SDK Command-line Tools, two new environment variables need to be added: ANDROID_HOME and JAVA_HOME. These variables should contain the paths of the Android SDK (Software Development Kit) and JBR from Android Studio, respectively.

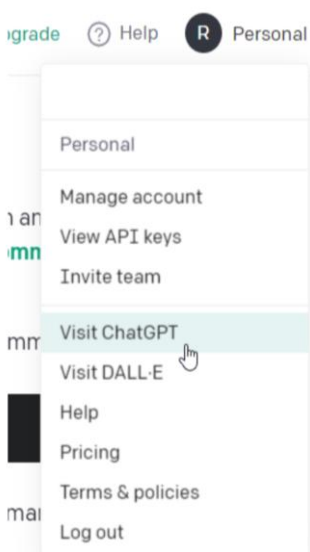
To install Microsoft Visual Studio with the C++ component, download the installer and run it. Select the option 'Desktop development with C++' to initiate the installation. This will configure Visual Studio with C++ for further use.

Visual Studio Code, also known as VSCode, is a convenient and comprehensive tool for developing Flutter applications, according to the author of this work. To install VSCode, download its executable (Microsoft, 2023a) and run it. Accept the terms at the beginning of the installation to proceed. Then, select all tool options for better use of VSCode and continue with the installation.

Once VSCode is installed and ready for use, the Flutter extension needs to be configured, which is discussed in the next subsection.

Obtaining the ChatGPT API key is straightforward. You need to access the OpenAI website navigate to the API documentation and log in (OpenAI, 2023a). After logging in, simply click on the profile in the upper right corner and go to View API keys, as shown in Figure 1.

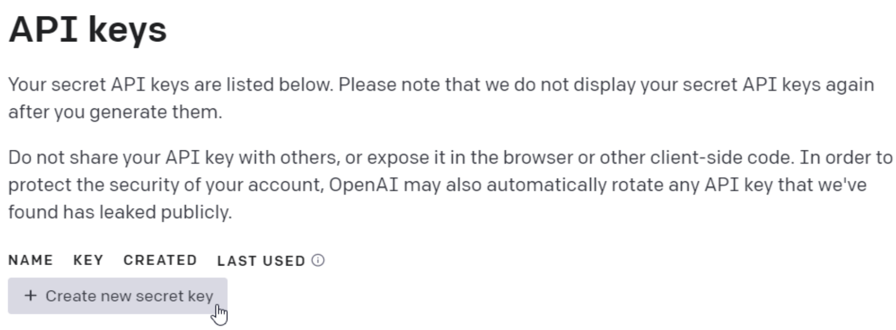
Figure 1 – Accessing ChatGPT API Keys.



Source: OpenAI (2023a).

On the API key page, it is necessary to create a new key by clicking on the Create new secret key button, as illustrated in Figure 3.20. Simply give it a name, save it, and keep it for later use.

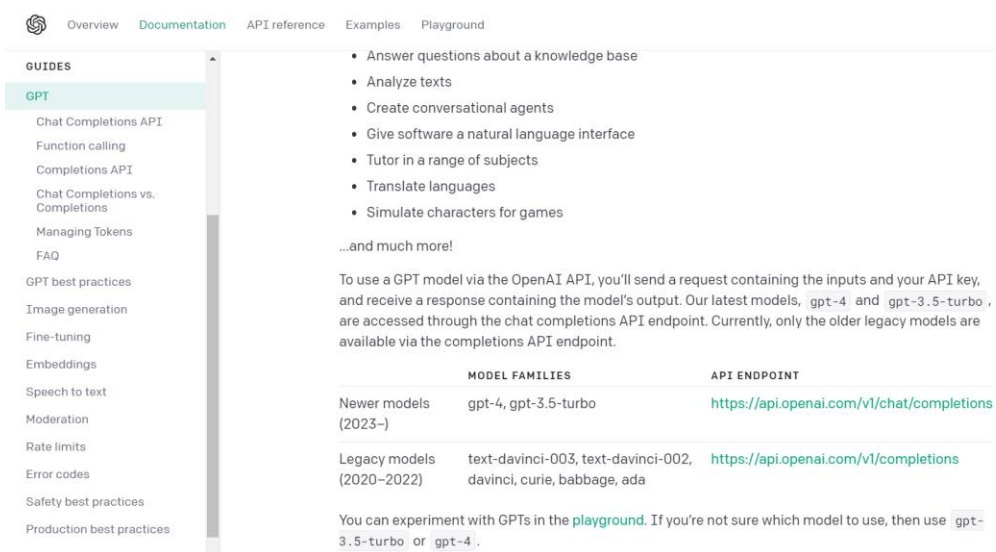
Figure 2 – Creating a New ChatGPT API Key.



Source: OpenAI (2023).

Navigate to the documentation and, in the left sidebar under Guides and select GPT to obtain the API URL. The API URL is present in a table with OpenAI models, as shown in Figure 3.21.

Figure 3 – Obtaining the ChatGPT API URL.



Source: OpenAI (2023b).

III. RESULTS AND DISCUSSIONS

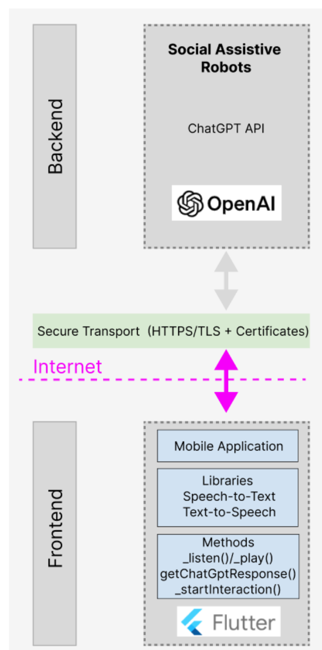
This section clearly presents how the project was built, and the results were obtained. The architectural vision of the new module appears as a proposal to integrate the *framework* OpenTera, called OpenTera++. It also details how the mobile application was built in Flutter, its operation, and its limitations.

A. OpenTera++ architectural vision

OpenTera++ was developed to be an integral part of OpenTera, working as a social assistive robot. It follows the pattern of the original *framework*, with a part in the backend and another in the front end, as well as an intermediate layer of HTTPS secure transport.

The backend is made up of the ChatGPT API that receives requests from the front end, processes them, and returns a response based on a question asked. front is composed of the Flutter mobile application, with the libraries used *speech-to-text* and *text-to-speech* and the methods created in the development of the application *_listen( ), _play(), getChatGptResponse(),* and *_startInteraction()*. In short, the application receives questions from the user through speech, transcribes them into text using the *speech-to-text* library, and sends it to the backend which processes the question and returns an answer that is transmitted to the user through the *flutter_tts* library.

Figure 4 – Social Assistive Robots Module



Source: OpenAI (2023c).

Therefore, the new module can handle interactions with users, process their queries, and get answers from ChatGPT. This allows OpenTera to offer an intelligent virtual assistant as an integral part of its functionality.

It can also be integrated with other OpenTera modules, such as data and session management. This allows the social bot to utilize relevant and contextual information to provide more accurate and personalized responses to users.

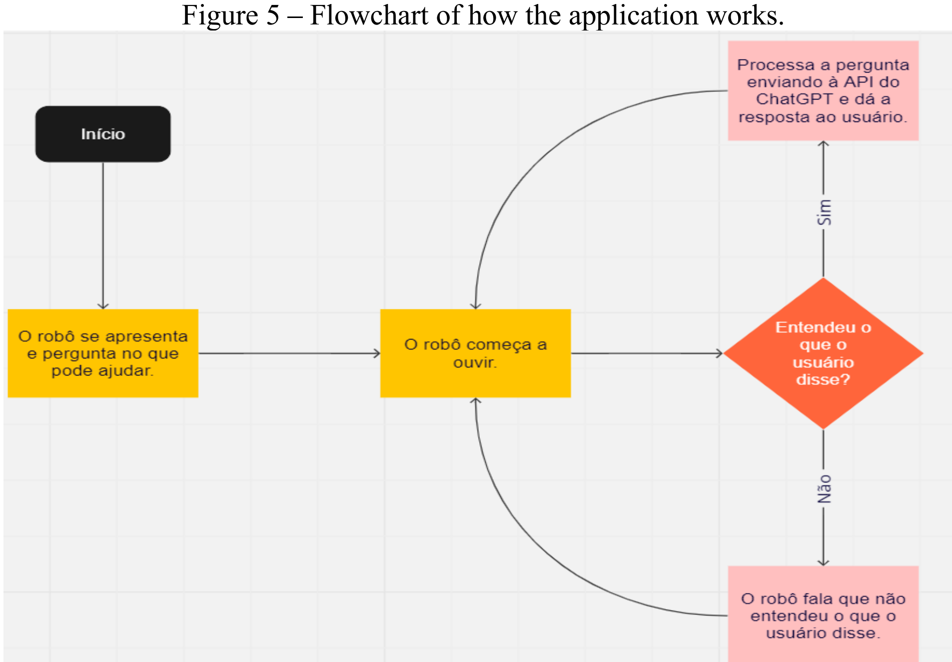
In summary, incorporating this new module, a social assistive robot based on the ChatGPT API, into the OpenTera *framework* can enrich the framework's functionality, offering users an intelligent and interactive virtual assistant. The proposed module was named OpenTera++ and its objective is to be an assistive social robot, it is shown in Figure 4.1.

B. OpenTera++ Conversation Modules

Developed in Flutter, the application built in this work was designed to be an intelligent social robot where the user can interact with it and ask questions (through voice) on the most varied topics that are processed and answered by the robot through ChatGPT. It's like having a conversation partner always available to answer your curiosities and help you with information.

When the application starts, the robot welcomes you and asks how it can help. Soon later, it starts listening through the device's microphone. If it understands what the user has said, it will send a request to the ChatGPT API and wait for the response. With the answer ready, it returns to the user and, through speech, answers the question asked previously. If it does not understand what the user said or if nothing is said, the robot will say that it did not understand and ask whether the question can be repeated. To illustrate, the operation is shown in Figure 5 flowchart.

Figure 5 – Flowchart of how the application works.



Source: Authors (2024).

For the OpenTera++ conversation module, four methods were developed, called *_startInteraction(), _listen(), _play()* and *getChatGptResponse()*. They will be explained and shown below.

Being one of the most important methods, *_startInteraction()* receives the current status of the interaction as a parameter. It is used to control the types of interaction between the robot and the user, which can be *start, error,* and *repeat*. The method is called with a status start when the application starts and adjusts the response text to welcome and conveys this in a speech to the user by calling the speak method from the library *flutter_tts*. The *error* status does the same thing as the *start* status, however the user response text changes. Finally, the *repeat* status is used when the API response is returned and has already been transmitted to the user. All three statuses, after fulfilling their function, call the *_listen()* method, which will soon be explained. The *_startInteraction()* method code is shown below.

```
Future < void > _startInteraction ( String status ) async {
if (status == 'start') {
    _text = 'Hello. How can I help you?';
  } else if (status == 'error') {
    _text = 'I didn't understand what you said, could you repeat?';
    _interaction = 0;
  }

  if ((status == 'start' || status == 'error') && _interaction == 0) {
    setState(() {
      _isListening = false;
      _isSpeaking = true;
    });

    await Future.delayed(const Duration(seconds: 1));
    await flutterTts.setLanguage('en-US');
    await flutterTts.speak(_text);

    flutterTts.setCompletionHandler(() {
      setState(() {
        _isSpeaking = false;
      });
      _listen();
    });
  } else if (status == 'repeat') {
    _isListening = false;
    await Future.delayed(const Duration(seconds: 1));
    _listen();
  }
}
```

The *_listen()* method has the main objective of listening through the device's microphone and transcribing everything that was said into text. It has two main outputs. If the user said nothing or said something that could not be understood, the *_startInteraction()* method is called passing the status error as a parameter, which was explained previously. Otherwise, the *listen()* method of the *speech_to_text* library, introduced earlier, is called to transcribe the words spoken by the user into text. When the user finishes speaking, the *_play()* method is called. The *_listen()* method created is shown below.

```
void _listen() async {
  if (!_isListening) {
    bool available = await _speech.initialize(
      onStatus: (val) => print('onStatus: $val'),
      onError: (val) {
        if (val.errorMsg == 'error_speech_timeout' ||
            val.errorMsg == 'error_no_match') {
          _startInteraction('error');
        }
      },
    );
    if (available) {
      setState(() => _isListening = true);
      _speech.listen(
        onResult: (val) async {
          setState(() {
            _text = val.recognizedWords;
          });
          if (val.finalResult) {
            _play();
          }
        },
      );
    }
  } else {
    setState(() => _isListening = false);
    _speech.stop();
  }
}
```

In the *_play()* method the main objective is to take the text that was transcribed from speech in the previous method and pass it as a parameter to the *_getChatGptResponse()* method, which is explained below and which processes the request to the ChatGPT API. The *_play()* method then waits for *_getChatGptResponse()* to return the API response and calls the speak method of the *flutter_tts* library to respond to the user through speech. When the entire speak method is finished, the *_startInteraction()* method will be called taking the status *repeat* as a parameter, starting the flow again as can be seen in Figure 4.2. The code is shown below.

```
void _play() async {
  _voice = await getChatGptResponse(_text);

  setState(() {
    _isListening = false;
    _isSpeaking = true;
  });

  await flutterTts.setLanguage('en-US');
  List<int> latin1Bytes = latin1.encode(_voice);
  String textUtf8 = utf8.decode(latin1Bytes);
  await flutterTts.speak(textUtf8);

  flutterTts.setCompletionHandler(() {
    setState(() async {
      _isSpeaking = false;
      _isListening = true;
      _interaction++;
      _startInteraction('repeat');
    });
  });
}
```

Finally, the *_getChatGptResponse()* method is responsible for making a request using the HTTP web communication protocol and the POST method through Flutter’s *http* library for the ChatGPT API. It basically consists of obtaining the text per parameter and making the request. If the API response is successful, the method returns the text generated in the response, otherwise an exception is thrown. Note that in the headers object the *Authorization* attribute contains the *API key*, obtained in the previous section, after the word *Bearer*. It is also worth mentioning that in the *response* variable, the *parse* method receives as a parameter the API URL also obtained in the previous section, as shown below in the method code.

```
Future<String> getChatGptResponse(String message) async {
  final headers = {
```

```
    'Content-Type': 'application/json',
    'Authorization':
      'Bearer YOUR_TOKEN',
  };

  final data = {
    'model': 'gpt-3.5-turbo',
    'messages': [
      {"role": "user", "content": message}
    ],
    'max_tokens': 50,
  };
  try {
    final response = await http.post(Uri.parse(apiUrl),
      headers: headers, body: jsonEncode(data));
    if (response.statusCode == 200) {
      final jsonResponse = jsonDecode(response.body);
      final chatGptResponse = jsonResponse['choices'][0]['message']['content'];
      return chatGptResponse;
    } else {
      throw Exception('Failed to get ChatGPT response');
    }
  } catch (e) {
    return "Nao pode ser processado";
  }
}
```

C. OpenTera++ Emotion Modules

The OpenTera++ emotion module completes the conversation module by displaying animated *gifs* to convey the emotions of the social robot, making the experience more engaging. When the robot listens to the question, the user will see a *gif* with two attentive eyes, conveying the feeling of being listened to carefully. And, while the robot is processing the answer, a *gif* of a mouth talking will be displayed, to represent the answer to the question asked.

When the robot is listening, an animated *gif* of two eyes is displayed, giving the user the feeling that he is paying attention to what is being said. The application screen at this moment is as illustrated in Figure 4.3.

Figure 6 – Application displaying GIF while listening

Source: Authors (2024).

The moment the robot starts speaking, whether welcoming or responding to what the user has asked, an animated *gif* of a speaking mouth is displayed on the screen, to convey that the robot is saying something, as can be seen in Figure 4.4.

Figure 7 – Application displaying GIF while speaking

Source: Authors (2024).

While the application listens to speech, two Boolean variables named *_isListening* and *_isSpeaking* were used to (respectively) display and manipulate the animated *gifs*. They are controlled and have their values changed within the *_startInteraction()*, *_listen()* and *_play()* methods according to the context of each method. When the variable *_isListening* is equal to true, the application will be listening, and the gif of the eyes will be displayed. And when *_isSpeaking* is true, the application will be speaking, and the *gif* of the mouth will be displayed. The two variables never coincide with being true at the same time.

The widget code responsible for the application's user interface is shown below.

```
1. @override
2. Widget build(BuildContext context) {
3.    return Scaffold(
4.      backgroundColor: Colors.black,
5.      body: SizedBox(
6.        height: double.infinity,
7.        width: double.infinity,
8.        child: Center(
9.          child: Column(
10.           mainAxisAlignment: MainAxisAlignment.center,
11.           children: [
12.             Visibility(
13.               visible: _isSpeaking,
14.               child: const Image(
15.                 image: AssetImage("assets/talking.gif"),
16.               ),
17.             ),
18.             Visibility(
19.               visible: _isListening,
20.               child: const Image(
21.                 image: AssetImage("assets/listening.gif"),
22.               ),
23.             ),
24.           ],
25.         ),
26.       ),
27.     ),
28.   );
29. }
```

Note that within *children* on line 11 of the code, there are two *Visibility* widgets that display *gifs* according to the variables *_isListening* and *_isSpeaking*. This is controlled by the *visible* attribute, allowing *Visibility* to be shown or not on the screen.

D. Evaluation of the Modules

In the evaluation of our application, testing protocols were executed in adherence to the methodological framework outlined by Sejnowski (2023). The primary objective was to conduct Turing testing, gauging the capabilities of the newly developed application comprehensively. Our testing regimen commenced with a series of functional tests, aimed at validating the seamless operation of all application functionalities.

Specifically, considerable attention was devoted to scrutinizing the application's speech recognition system. This involved an intricate examination to ensure the accurate interpretation and transcription of user queries into text across diverse situational contexts. To provide a tangible representation of these tests, an illustrative example, demonstrating the interaction between the author and the developed application, was meticulously recorded and made accessible at https://youtu.be/F_soHSmHa7c.

Analyzing the results of these tests provided invaluable insights into the application's performance, serving as a foundation for targeted improvements. Notably, we identified certain limitations in the application's operation, with a key observation being the continuous listening mode. This functionality, whereby the application persistently attempts to transcribe speech into text after each user response, regardless of context, emerged as a noteworthy area for enhancement.

The ensuing prompt exemplifies this limitation:

Robot: Hello. How can I help you?
Prompt: Hello, good afternoon.
Robot: Hello! Good afternoon! How can I help you today?
Prompt: Who was the first human being to set foot on the Moon?
Robot: The first human being to set foot on the Moon was Neil Armstrong, a NASA astronaut...
Prompt: ...
Robot: I didn’t understand what you said, could you repeat?
Prompt: ...
Robot: I didn’t understand what you said, could you repeat?
Prompt: Thank you very much.
Robot: You’re welcome! If you have any further questions or need help, feel free to ask. Have a great day!

Considering contemporary standards observed in voice-controlled virtual assistants, it is recommended that the application should exhibit more nuanced responsiveness. An ideal enhancement involves the introduction of a keyword-activated mode, allowing the application to cease transcription and resume listening only upon explicit user indication. This proposed adjustment aligns with prevailing user expectations and contributes to an optimal user experience.

The evaluation of our testing procedures has not only identified areas for refinement but has also laid the groundwork for future enhancements, ensuring the continual evolution of our application toward an advanced and user-centric final version.

As elucidated by Panchea et al. (2022), the NGINX server layer operates as a pivotal component, functioning as a reverse proxy strategically positioned to exert control over and bolster the security of network access within the context of backend modules. Serving as an essential gateway to the front end, this layer assumes responsibility for managing data encryption and TLS certificates. Its primary objective is to effectively route HTTPS requests to the appropriate backend service, thereby optimizing the communication pathway.

It is imperative to note that, within the scope of our present study, the NGINX server layer was not integrated into the application's architecture. The decision to omit this element was deliberate, stemming from the fact that its functionality had not undergone testing in the context of our specific implementation. Consequently, the absence of the NGINX layer is proactively acknowledged as a facet left unexplored in the current work, thereby representing a prospective avenue for future research endeavors.

In conceptualizing the potential integration of the NGINX server layer in subsequent iterations, it is envisaged that the application's communication, originating from the front, would traverse this intermediary server before establishing a connection with the ChatGPT API in the backend. This architectural augmentation holds the promise of amplifying the overall security posture of the application's communicative processes, aligning with contemporary standards and best practices in the realm of AI-driven applications. As such, the NGINX server layer is an element for enhancing both the efficacy and resilience of the application's network infrastructure in future investigations.

E. Threat Modelling: Privacy and Data Protection Issues

In this section we present the results of our threat modeling process incremented with juridical discussion under the perspective of the Brazilian General Data Protection Regulation (BGDPR), and privacy. Overall, we review for programming methods: *_startInteraction(), _listen(), _play(), and getChatGptResponse()*.

In the *_startInteraction()* method, three vulnerabilities were identified. These include the absence of a logging mechanism posing a considerable risk for repudiation, the audible output of the _text variable raising concerns regarding information disclosure, and the potential for a Denial of Service (DoS) attack arising from the repeated invocation of the *_startInteraction()* function with an error status.

These vulnerabilities can also bring ethical and legal issues, especially with the BGDPR. First, the absence of a logging mechanism does not allow us to recognize who the user is and whose data is collected. According to The Right to Information Self-Determination (art. 2°, II, of BGDPR), the user might know what, when, and whose data is used by the company. This right ensures the owner controls all its information and decides what to do with it. Furthermore, in the absence of a logging mechanism you cannot create barriers to protect and ensure the security of those pieces of information. So, intimacy and privacy can be violated at any time, which is also an issue of information disclosure (art. 2°, IV, of BGDPR).

The potential of a DoS attack also can violate the user's right to access (art. 6°, IV, of BGDPR). The data subject has the right to easy access to information about the processing of their data, which must be made available in a clear, adequate, and conspicuous manner, among other characteristics provided for in regulations to comply with the principle of free access (art. 9°, caput, o of BGDPR). In other words, if a DoS attack is taking place, the service is not available to the user and holder of the data, and there is no way of accessing it to carry out the control as it can.

Similarly, the *_listen()* method exhibits vulnerabilities primarily related to repudiation and disclosure of information. The lack of a logging mechanism impedes the ability to track user interactions, while the storage of recognized words in the _text variable could inadvertently disclose sensitive information. Furthermore, the possibility of a DoS attack stems from repeated failures of the speech recognition service's initialization or listening functions.

The *_listen()* method is perhaps the most problematic. There are violations of privacy, intimacy, and data protection. It violates privacy by capturing information available in the environment. Information that has restricted access and can only be obtained with the express authorization of its owner. It violates privacy, as the API can be operating in a private environment and the user's most intimate moments without their consent.

Note: while privacy refers to the possibility of controlling information that enters and leaves the user’s physical space, intimacy is even more restricted: it is a restricted space within a restricted space. We will use the following situations to illustrate: a person inside their home is in a private moment, but a person during a sexual act is in an intimate moment. Thus, even if there are other people in the house, the act of intimacy is more restricted to its owner and those who can participate in it.

In this context, any information collected has probably not been expressly authorized by the user. Therefore, the legal basis for the collection, which is consent, has not been duly respected and thus the collection of data and information will be considered illegal, due to non-compliance with Art. 7, I, of the BGDPR.

Furthermore, it is clear that always-listening devices can affect privacy (Bohm, et. al., 2017) and intimacy (Lau, et. al., 2018). When capturing the audio of third parties who are in the same environment, their consent is not collected properly (free, informed, and conscious) (Cheng; Roedig, 2022). Data protection is also undermined by the lack of explicit consent, its legal and authorizing basis, under the terms of art. 7, I, of the BGDPR (Klein, 2020). People with disabilities can have their vulnerability aggravated by not being able to interact properly with the device, either due to lack of instruction or lack of control (Vieira, et. al., 2022), which directly affects the subject's informational self-determination (Mennard, et al., 2017).

In the *_play()* method, three vulnerabilities were identified. These include the absence of logging capabilities posing challenges for repudiation, the audible output of responses retrieved from external services, raising concerns about information disclosure, and repeated failures of functions within this method potentially resulting in a DoS condition.

Although the *_play()* method shows three vulnerabilities, they all violate one right: the right to explanation. The Right to Explanation consists of the company's duty to provide detailed information on how data is being collected, handled, and processed. For this detailing to be carried out correctly and compatible with what is being done, the log must exist and be faithful to all the acts carried out.

Thus, its absence undermines the realization of this right. However, if the data owner is protected by the Right to Explanation, there is also a legal duty for companies to collect, produce, receive, classify, archive, store, eliminate, and constantly evaluate this data. These actions and duties are usually designed in the Data Protection Impact Assessment. If it turns out that there is no way of carrying out these actions, mechanisms must be created for this purpose, as the controller and operator must keep a record of the personal data processing operations they carry out, especially when based on legitimate interest (art. 37 of BGDPR).

The getChatGptResponse(String message) method introduces a range of vulnerabilities, including spoofing, tampering, repudiation, information disclosure, and DoS risks. The utilization of bearer tokens for authorization creates an avenue for spoofing attacks if these tokens are compromised. Similarly, the inclusion of user-provided messages in HTTP requests opens the door for tampering. Furthermore, the lack of logging capabilities poses challenges for repudiation, the potential disclosure of sensitive information returned by the ChatGPT API raises information disclosure concerns, and the risk of DoS attacks due to repeated failures of the HTTP POST function further highlights the vulnerabilities within this method.

In conclusion, the state of the art regarding protecting rights in Always listening devices is divergent on the national and international scene but is still a constant concern for developers and jurists (Barrett; Liccardi, 2021). In the latter, human-robot-IA relations already have a solid definition (ISO 8373, 2012; Kim, 2024; Park, 2021; Hildt, 2021), and there are considerable points about the problems arising from this relationship, including the intelligibility of natural-computer language (Sheridan, 2016), exaggerated anthropomorphization (Złotowski, et al., 2015), privacy violations (Chatzimichali et. al., 2020), and protection of physical integrity (Haddadin, 2011). Furthermore, there are precedents from the Federal Trade Commission and the United States Department of Justice condemning Amazon for violating the privacy of children and adolescents by recording their voices on the company's device (FTC, 2023). As far as legislation is concerned, in the United States there are regulations on "unauthorized wiretaps" (Accidental Wiretraps), including the Federal Wiretap Act (18 U.S.C. §§ 2510-2522), which was used analogously in the situations that led to this research problem. There is also academic production on the subject, including Barrett and Liccardi (2022), in their approach to the aforementioned FTC regulation and its analogous application to the topic; Malkin, et. al., (2019), Liu and Malkin (2022) and Mhaidli, et. al., (2020), based on their research on the subject.

From the study carried out, it can be seen that there is no possibility of exhausting the theme. The main problem that arises from the results is: how to mitigate digital vulnerability and promote (or stimulate, ensure, or even protect) the authenticity of consent in human-robot interactions and human-artificial intelligence interactions on always-listening devices?

It is also almost impossible to filter conversations on these types of devices when they are active and connected (Bohm, et. al. 2017). This is a paradoxical limitation of the programming itself, as the devices need to be active in order to receive commands from the owner. However, they cannot be active enough to capture every type of conversation, as this violates the user's privacy and intimacy (Lau, et. al., 2019). In addition, there is extensive and current discussion about unauthorized facial recognition (Chow, 2024), the protection of informational self-determination in the face of invasive devices (Rouvroy, 2009; Buitelaar, 2017) and the use of techniques for behavioral induction, known as Nudges (Sunstein, 2017). It should be noted that various factual situations can lead to violations of legal situations and relationships. For this reason, the aim is to develop legal and computational methods and guidelines based on Privacy by Design (Schaar, 2010) to allow users not only to control but also to understand and be digitally literate to defend their rights. This is also a demand for the promotion of digital skills, which is part of the UN's SDGs (n. 4, education; and n. 10, reducing inequalities), to increase the proportion of people with basic and intermediate digital skills worldwide by 2030.

CONCLUSION

In the pursuit of augmenting the OpenTera microservices architecture to integrate assistive social robotics (SARs) with generative artificial intelligence, our study achieved several pivotal goals. These accomplishments are crucial steps towards enhancing human-robot interactions and leveraging the capabilities of generative AI, specifically ChatGPT. Let's examine the goals and their outcomes:

Our foremost goal was to develop a listening module for OpenTera in mobile development technology capable of transcribing environmental speech into strings. This goal was successfully realized, contributing a fundamental component to the OpenTera++ architecture. We implemented a module connecting to the ChatGPT API to send and receive transcribed string sequences represents a significant achievement. This integration enhances the system's capacity for generating intelligent responses. In addition, the successful integration of a module querying the ChatGPT API for responses to prompts containing transcribed string sequences enriches the OpenTera++ architecture, facilitating dynamic and context-aware interactions. The creation of a speech module capable of transforming ChatGPT responses into audio for articulation by the robot adds a crucial dimension to the SARs, enriching the user experience.

While the original goal of designing our own animated GIF was not fully met, a pragmatic approach was adopted by utilizing an existing GIF display module to represent emotions during the robot's listening and speech. This creative solution contributes to the overall expressive capabilities of the SARs.

The culmination of our efforts resulted in the development of OpenTera++, an extended architecture showcasing the successful incorporation of generative AI within the OpenTera framework. This prototype, available at https://github.com/rafout/opentera-pp, fulfills the specific objectives outlined in the study.

While celebrating these achievements, it is essential to acknowledge identified limitations. The continuous transcription of audio without contextual consideration and the absence of NGINX server integration within the OpenTera module are recognized constraints.

To address these limitations, future work is suggested. In the application realm, the implementation of a keyword-activated listening mode is proposed, enhancing contextual understanding. For the framework module, the incorporation of a NGINX server is recommended, providing an additional layer of security in communication between the front end and backend.

In conclusion, this study marks a stride towards the convergence of SARs and generative AI within the OpenTera framework. The achievements and identified limitations pave the way for continued refinement, fostering a more robust and sophisticated integration of cutting-edge technologies for enriched human-robot interactions. This is where this project aims to innovate and assist in the scientific development of Law and Computer Science by creating an appropriate thematic bibliography containing legal standards and guidelines to mitigate digital vulnerability and protect informed consent in human-robot and human-artificial intelligence relationships arising from always-listening devices using Privacy by Design techniques.

REFERENCES

Aldarmaki, H. et al., *Unsupervised Automatic Speech Recognition: A Review*, 139 Speech Commc'n 76 (2022).

Barrett, Lindsey & Ilaria Liccardi, *Accidental Wiretaps: The Implications of False Positives by Always-Listening Devices for Privacy Law & Policy*, 74 Okla. L. Rev. 79 (2021).

Bartneck, C. et al., *Human-Robot Interaction: An Introduction* (Cambridge Univ. Press 2020).

Bohm, Allison S. et al., *Privacy and Liberty in an Always-On, Always-Listening World*, 19 Colum. Sci. & Tech. L. Rev. 1 (2017).

Bregant, Cristian, *Azure Speech Recognition: Implementation of Azure Cognitive Speech Recognition for Flutter*, GitHub (2020), https://github.com/cristianbregant/azure_speech_recognition.

Buitelaar, J.C., *Post-Mortem Privacy and Informational Self-Determination*, 19 Ethics & Info. Tech. 129 (2017).

Chaves, A.P. & M.A. Gerosa, *How Should My Chatbot Interact? A Survey on Social Characteristics in Human–Chatbot Interaction Design*, 37 Int'l J. Hum.–Computer Interaction 729 (2021).

Chatzimichali, Anna, Ross Harrison & Dimitrios Chrysostomou, *Toward Privacy-Sensitive Human-Robot Interaction: Privacy Terms and Human-Data Interaction in the Personal Robot Era*, 12 Paladyn J. Behav. Robotics 160 (2020).

Cheng, Peng & Utz Roedig, *Personal Voice Assistant Security and Privacy: A Survey*, 110 Proc. IEEE 476 (2022).

Chow, Ka-Ho et al., *Personalized Privacy Protection Mask Against*

Unauthorized Facial Recognition, in European Conf. on Computer Vision 434 (Springer 2024).

Chuwy's, *Chuwy's GIF*, Giphy (2019), https://giphy.com/gifs/XAyRXqH8LZwtyI8BU7.

Collins, S.H.E., *Try Bard and Share Your Feedback*, Google Blog (2023), https://blog.google/technology/ai/try-bard/.

Dart, *HTTP: A Composable Future-Based Library for Making HTTP Requests*, GitHub (2018), https://github.com/dart-lang/http.

Feil-Seifer, David & Maja J. Mataric, *Defining Socially Assistive Robotics*, in Proc. IEEE Int'l Conf. on Rehabilitation Robotics 465 (2005).

Fel, *Flutter Azure TTS: Flutter Implementation of Microsoft Azure Cognitive Text-To-Speech API*, GitHub (2021), https://github.com/ghuyfel/flutter_azure_tts.

Flutter, *Flutter Install*, Docs.Flutter.dev (2023), https://docs.flutter.dev/get-started/install/windows.

FTC, *FTC and DOJ Charge Amazon with Violating Children's Privacy Law by Keeping Kids' Alexa Voice Recordings Forever*, Fed. Trade Comm'n (2023).

Google, *Pub.dev: Official Package Repository for Dart and Flutter Apps*, Google (2017), https://pub.dev/.

Google, *Download Android Studio*, Google (2023), https://developer.android.com/studio.

Google, *Speech-to-Text Pricing*, Google Cloud (2023), https://cloud.google.com/speech-to-text/pricing.

Haddadin, Sami, Alin Albu-Schäffer & Gerd Hirzinger, *Safe Physical Human-Robot Interaction: Measurements, Analysis and New Insights*, in Robotics Research: The 13th Int'l Symposium ISRR 395 (Springer 2011).

Hetler, Amanda, *What Is ChatGPT?*, TechTarget (2023), https://www.techtarget.com/whatis/definition/ChatGPT.

Hildt, Elisabeth, *What Sort of Robots Do We Want to Interact With? Reflecting on the Human Side of Human-Artificial Intelligence Interaction*, 3 Frontiers Comput. Sci. 671012 (2021).

IBM, *What Is Speech Recognition?*, IBM (2023), https://www.ibm.com/topics/speech-recognition.

Junghans, Felix, *Google Speech API for Flutter*, GitHub (2021), https://github.com/felixjunghans/google_speech.

Kim, Victoria, *Australia Has Barred Everyone Under 16 from Social Media. Will It Work?*, N.Y. Times (2024).

Klein, Andreas M. et al., *Exploring Voice Assistant Risks and Potential with Technology-Based Users*, in Proc. WEBIST 147 (2020).

Lau, Josephine, Benjamin Zimmerman & Florian Schaub, *Alexa, Are You Listening? Privacy Perceptions, Concerns and Privacy-Seeking*

Behaviors with Smart Speakers, 2 Proc. ACM Hum.-Computer Interaction 1 (2018).

Lutton, David, *Flutter Text to Speech Package*, GitHub (2018), https://github.com/dlutton/flutter_tts.

Menard, Philip, Gregory J. Bott & Robert E. Crossler, *User Motivations in Protecting Information Security: Protection Motivation Theory Versus Self-Determination Theory*, 34 J. Mgmt. Info. Sys. 1203 (2017).

Microsoft, *Download Visual Studio Code*, Microsoft (2023), https://code.visualstudio.com/download.

Microsoft, *Bing AI Copilot*, Microsoft (2023), https://www.microsoft.com/bing.

Microsoft, *Speech Services Pricing*, Microsoft Azure (2023), https://azure.microsoft.com/pricing/details/cognitive-services/speech-services.

Midjourney, *Midjourney*, Midjourney (2023), https://www.midjourney.com.

Mhaidli, Abraham et al., *Listen Only When Spoken To: Interpersonal Communication Cues as Smart Speaker Privacy Controls*, Proc. Privacy Enhancing Techs. (2020).

Natalie, *What Is ChatGPT?*, OpenAI Help Center (2023), https://help.openai.com.

Ning, Y. et al., *A Review of Deep Learning Based Speech Synthesis*, 9 Applied Sci. (2019).

OpenAI, *API Reference*, OpenAI (2023), https://platform.openai.com/docs/api-reference.

OpenAI, *DALL-E 2*, OpenAI (2023), https://openai.com/dall-e-2.

OpenAI, *ChatGPT Documentation*, OpenAI (2023), https://platform.openai.com/docs/introduction/overview.

Panchea, A.M. et al., *Opentera: A Microservice Architecture Solution for Rapid Prototyping of Robotic Solutions to COVID-19 Challenges in Care Facilities*, 12 Health & Tech. 583 (2022).

Park, Namkee et al., *Use of Offensive Language in Human-Artificial Intelligence Chatbot Interaction*, 121 Computers Hum. Behav. 106795 (2021).

Ren, Y. et al., *FastSpeech: Fast, Robust and Controllable Text to Speech*, 32 Advances Neural Info. Processing Sys. (2019).

Robotics, Shape Robotics, *Shape Robotics GIF*, Giphy (2021), https://giphy.com/gifs/oNvolApeRjCKIfIjcb.

Rouvroy, Antoinette & Yves Poullet, *The Right to Informational Self-Determination and the Value of Self-Development*, in Reinventing Data Protection? 45 (Springer 2009).

Schaar, Peter, *Privacy by Design*, 3 Identity Info. Soc'y 267 (2010).

Sejnowski, T.J., *Large Language Models and the Reverse Turing Test*, 35

Neural Computation 309 (2023).

Sheridan, Thomas B., *Human-Robot Interaction: Status and Challenges*, 58 Hum. Factors 525 (2016).

Software, CSD Corp., *Speech-to-Text Flutter Plugin*, GitHub (2021), https://github.com/csdcorp/speech_to_text.

Sunstein, Cass R., *Nudging: Um Guia (Muito) Resumido*, 3 Rev. Estud. Institucionais 1023 (2017).

Tabassum, Madiha et al., *Investigating Users' Preferences and Expectations for Always-Listening Voice Assistants*, 3 Proc. ACM Interactive Mobile Wearable Ubiquitous Techs. 1 (2019).

Vieira, Alessandro Diogo, Higor Leite & Ana Vitória Lachowski Volochtchuk, *The Impact of Voice Assistant Home Devices on People with Disabilities*, 184 Technological Forecasting & Soc. Change 121961 (2022).

Yang, Z., Y. Zhang & J. Luo, *Human-Centered Emotion Recognition in Animated GIFs*, in IEEE Int'l Conf. Multimedia & Expo 1090 (2019).

Zhang, C. et al., *A Complete Survey on Generative AI (AIGC): Is ChatGPT from GPT-4 to GPT-5 All You Need?*, arXiv:2303.11717 (2023).

Złotowski, Jakub et al., *Anthropomorphism: Opportunities and Challenges in Human-Robot Interaction*, 7 Int'l J. Soc. Robotics 347 (2015).

* * *

Sthéfano Bruno Santos Divino
PhD and Master's degrees in Private Law from the Pontifical Catholic University of Minas Gerais. Bachelor of Laws from the University Center of Lavras. Adjunct Professor in the Law Program at the University Center of Lavras. Former Substitute Professor of Private Law at the Federal University of Lavras (03/2019–03/2021). Attorney.
Email: sthefanoadv@hotmail.com
ORCID iD: https://orcid.org/0000-0002-9037-0405

André de Lima Salgado
Adjunct Professor in the Department of Computer Science at UFLA. He is Associate Editor of the International Journal of Advanced Robotic Systems (IJARS). His research focuses on Human-Artificial Intelligence Interaction, Human-Robot Interaction, Privacy and Agriculture 5.0.
Email: andre.salgado@ufla.br
ORCID iD: https://orcid.org/0000-0003-4396-1111

Danilo Chagas Clemente
Undergraduate student in Computer Science at the Federal University of Lavras.
Email: danilo.clemente@estudante.ufla.br
ORCID iD: https://orcid.org/0009-0001-4432-7449

Maria Eduarda Couto Grossi
She is currently an undergraduate student in the Computer Science course at the Federal University of Lavras (UFLA). She was awarded a PIBIC/CNPq Scientific Initiation scholarship in the project entitled "Benefits of web accessibility for the development of social robots", which began in 09/2023 and ends

at the end of February 2024. She continues to carry out research related to Artificial Intelligence and Human-Robot Interaction. Product Owner trainee since 03/2024.
Email: maria.grossi1@estudante.ufla.br
ORCID iD: https://orcid.org/0009-0008-2155-6389

Rafael Bastos Andrade
Undergraduate student in Computer Science at the Federal University of Lavras.
Email: rafaelbastos2@gmail.com
ORCID iD: https://orcid.org/0009-0008-6235-269X

10.59224/bjlti.v4i1.24-53
ISSN: 2965-1549