



RAFAEL BASTOS ANDRADE

**CRIAÇÃO DE MÓDULOS DE CONVERSAÇÃO E EMOÇÃO
PARA OPENTERA: APLICAÇÃO COM CHATGPT**

LAVRAS – MG

2023

RAFAEL BASTOS ANDRADE

**CRIAÇÃO DE MÓDULOS DE CONVERSÇÃO E EMOÇÃO PARA OPENTERA:
APLICAÇÃO COM CHATGPT**

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do curso de Engenharia de Controle e Automação, para a obtenção do título de Bacharel.

Prof. Dr. André de Lima Salgado

Orientador

LAVRAS – MG

2023

RAFAEL BASTOS ANDRADE

**CRIAÇÃO DE MÓDULOS DE CONVERSAÇÃO E EMOÇÃO PARA OPENTERA:
APLICAÇÃO COM CHATGPT**

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do curso de Engenharia de Controle e Automação, para a obtenção do título de Bacharel.

APROVADA em 25 de Julho de 2023.

Prof. Dr. Danilo Alves de Lima UFLA
Felipe Silva Dias Copybase

Prof. Dr. André de Lima Salgado
Orientador

**LAVRAS – MG
2023**

Dedico este trabalho aos meus pais, meus irmãos, meus avós, minha namorada e toda a minha família por sempre terem acreditado em mim.

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus por ter me dado vida, saúde e muita força para chegar até aqui. Aos meus pais, Junior e Maria Amélia, por todo apoio emocional e esforço incessante para me manter todos esses anos em Lavras para que eu conseguisse alcançar meus objetivos e sonhos. Aos meus irmãos, Lucas e Maria Aíssa pelo incentivo durante o meu período de estudo. À minha namorada Letícia por me motivar e apoiar sempre. Aos meus avós, Ronaldo, Dirlinha e Zelinda(*in memoriam*) que se fizeram presente me apoiando em todos esses anos que estive estudando. As minhas tias Adélia, Aline, Liliane, Maéve, Melissa, Neuza e Raquel por sempre acreditarem em mim e me ajudarem em todas as circunstâncias. Ao meu orientador André por me ajudar sem medir esforços ao longo do desenvolvimento deste trabalho. Aos membros da banca, professor Danilo Alves e Felipe Silva da empresa Copybase, por estarem presente neste momento tão importante da minha graduação. A UFLA por proporcionar um ensino de qualidade e gratuito. Agradeço aos amigos que fiz ao longo desses anos na universidade e a todos que fizeram parte dessa trajetória.

*O insucesso é apenas uma oportunidade para recomeçar com mais inteligência.
(Henry Ford)*

RESUMO

A robótica assistiva social (SARs) se refere a robôs que fornecem ajuda a usuários humanos. Com os avanços tecnológicos e estudos em Inteligência Artificial Generativa novos estudos e projetos em SARs estão sendo desenvolvidos. O *framework* OpenTera atualmente não contempla o potencial dessas tecnologias para facilitar o desenvolvimento de SARs. Além disso, não inclui módulos específicos para conversação, que são essenciais para as funções sociais do robô. O objetivo deste estudo é realizar adaptações no *framework* OpenTera a fim de incorporar os benefícios do uso da API do ChatGPT no desenvolvimento de SARs. O estudo e desenvolvimento foi feito por meio de um aplicativo móvel em Flutter, utilizando-se bibliotecas para reconhecimento de fala automático e produção de fala, que se comunica com a API do ChatGPT para responder a questionamentos feitos pelo usuário. Os resultados do estudo levaram ao desenvolvimento de um novo módulo para adaptação do *framework* OpenTera. Diante dos resultados, o novo módulo pode ser adicionado ao *framework*, tornando-o mais robusto, com um módulo de SARs integrado a uma inteligência artificial generativa.

Palavras-chave: Robô. Inteligência Artificial. Interação humano-robô.

ABSTRACT

Assistive social robotics (SARs) refers to robots that provide assistance to 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 in order to incorporate the benefits of using the ChatGPT API in SARs' development. 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. In light of the results, the new module can be added to the framework, making it more robust, with an integrated SARs module powered by generative artificial intelligence.

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

LISTA DE FIGURAS

Figura 3.1 – Pacote Flutter extraído.	19
Figura 3.2 – Acessando as variáveis de ambiente.	19
Figura 3.3 – Editando a variável Path.	20
Figura 3.4 – Adicionando o caminho do flutter/bin.	20
Figura 3.5 – Boas-vindas Android Studio.	21
Figura 3.6 – Escolhendo componentes de instalação do Android Studio.	21
Figura 3.7 – Escolhendo local de instalação Android Studio.	22
Figura 3.8 – Iniciando a instalação do Android Studio.	22
Figura 3.9 – Instalação do Android Studio finalizada.	23
Figura 3.10 – Instalação de componentes do Android Studio.	23
Figura 3.11 – Acessando SDK Manager no Android Studio.	24
Figura 3.12 – Selevionando SDK Command-line Tools no Android Studio.	25
Figura 3.13 – Adicionando variável de ambiente ANDROID_HOME.	25
Figura 3.14 – Adicionando variável de ambiente JAVA_HOME.	25
Figura 3.15 – Instalação do Visual Studio Community com C++.	26
Figura 3.16 – Termos de instalação Visual Studio Code.	27
Figura 3.17 – Ferramentas do Visual Studio Code.	27
Figura 3.18 – Extensão do Flutter no Visual Studio Code.	28
Figura 3.19 – Acessando as API keys do ChatGPT	28
Figura 3.20 – Criando uma nova API key do ChatGPT.	29
Figura 3.21 – Obtendo a URL da API do ChatGPT.	29
Figura 4.1 – Módulo Social Assistive Robots	31
Figura 4.2 – Fluxograma do funcionamento do aplicativo.	32
Figura 4.3 – Aplicativo exibindo <i>gif</i> enquanto ouve	37
Figura 4.4 – Aplicativo exibindo <i>gif</i> enquanto fala	38

SUMÁRIO

1	INTRODUÇÃO	9
1.1	Objetivos do estudo	10
2	REVISÃO DE LITERATURA	11
2.1	Reconhecimento de fala automático	11
2.2	Produção de fala	12
2.3	Emoção na interação	13
2.4	IA Generativa	15
3	MATERIAIS E MÉTODOS	17
3.1	Métodos	17
3.2	Materiais	18
3.2.1	Instalação e configuração do Flutter no Windows	18
3.2.2	Instalação e configuração do Android Studio	21
3.2.3	Instalação do Microsoft Visual Studio Community 2022 com C++	26
3.2.4	Instalação do editor de código Visual Studio Code	26
3.2.5	Configuração do Flutter no Visual Studio Code	28
3.2.6	Obtenção da <i>key</i> e da URL da API do ChatGPT	28
4	RESULTADOS E DISCUSSÕES	30
4.1	Visão arquitetônica OpenTera++	30
4.2	Módulos de Conversação OpenTera++	31
4.3	Módulo de emoção do OpenTera++	37
4.4	Realização de testes	39
5	CONCLUSÃO	42
	REFERÊNCIAS	43

1 INTRODUÇÃO

A definição de SARs (Robótica Assistiva Social) proposta por Feil-Seifer e Mataric (2005) envolve a interseção entre a Robótica Assistiva (AR) e a Robótica Interativa Social (SIR). Nela, AR se refere a robôs que fornecem ajuda ou suporte a usuários humanos, enquanto a SIR diz respeito a robôs que interagem com os usuários. A partir da definição apresentada por Feil-Seifer e Mataric (2005), entendemos que o objetivo de SAR é prover assistência aos usuários humanos por meio da interação social, assim como a robótica assistiva. No entanto, SAR se diferencia de SIR por ter um foco mais específico na interação social. Enquanto em SIR o robô busca desenvolver interações próximas e eficazes com o humano pelo valor intrínseco da própria interação, em SAR o robô visa criar interações próximas e eficazes com um usuário humano para oferecer assistência e promover progresso mensurável em aspectos como recuperação, reabilitação, aprendizado, etc.

O OpenTera é um *framework* de arquitetura de microsserviços que foi desenvolvido para permitir que engenheiros de software e robótica prototipem e implantem aplicativos H-IoT (*Human-Internet of Things*) sem a necessidade de iniciar do zero a cada vez. Essa solução abrangente emprega uma série de módulos independentes para tarefas como gerenciamento de dados e sessões, telemedicina e tarefas/ações assistivas diárias, além de dispositivos e ambientes inteligentes, todos conectados por meio do *framework* (PANCHEA et al., 2022).

Após os avanços nos estudos sobre IA (Inteligência Artificial) Generativa e o lançamento do ChatGPT, grandes empresas começaram a investir em estudos e ferramentas nessa área. O Google lançou o Bard, um chatbot capaz de responder a questionamentos feitos pelos usuários (COLLINS, 2023). A Microsoft, em parceria com a OpenAI, lançou o Bing Chat, que utiliza o ChatGPT e também é capaz de responder a perguntas (MICROSOFT, 2023b). Além disso, existem, também, sistemas generativos de imagens a partir de descrições, como o Dall-E 2 e o Midjourney (OPENAI, 2023b; MIDJOURNEY, 2023). É importante ressaltar que, além desses exemplos mencionados, diversas outras IAs generativas já foram lançadas e muitos estudos e projetos estão em andamento nessa área. Com o ritmo acelerado de inovação, é provável que novas IAs sejam lançadas em breve, oferecendo recursos e oportunidades ainda mais amplas para diversos setores.

Apesar dos benefícios oferecidos pela IA generativa, o OpenTera atualmente não contempla o potencial dessas tecnologias para facilitar o desenvolvimento de SARs. Além disso, o OpenTera não inclui módulos específicos para conversação e design de emoções, que são es-

senciais para as funções sociais na interação humano-robô (BARTNECK et al., 2020). Diante desse contexto, o objetivo deste estudo é realizar adaptações no framework OpenTera a fim de incorporar os benefícios do uso da API(Interface de Programação de Aplicação) do ChatGPT no desenvolvimento de SARs.

1.1 Objetivos do estudo

Este estudo visa aprimorar a qualidade de SAR por meio do desenvolvimento de novas tecnologias para prototipagem rápida dos robôs. Para isso, propomos os seguintes objetivos específicos (*OE*s):

- OE1:** Desenvolver um módulo de escuta para o OpenTera em tecnologia de desenvolvimento móvel que possa transcrever as falas do ambiente em cadeias de caracteres.
- OE2:** Implementar um módulo de conexão com a API do ChatGPT para enviar as cadeias de caracteres transcritas.
- OE3:** Integrar um módulo de consulta à API do ChatGPT para receber respostas aos prompts que contenham as cadeias de caracteres transcritas.
- OE4:** Construir um módulo de fala que possa transformar as respostas do ChatGPT em áudio para o robô pronunciar.
- OE5:** Projetar um módulo de exibição de gifs animados que possa representar emoções durante a escuta e a fala do robô.

Considerando que as tecnologias empregadas neste trabalho são recentes no mercado¹, nossos objetivos possuem maior caráter tecnológico, pois visam explorar possibilidades de desenvolvimento móvel para prototipagem rápida e de baixo custo de SAR, a fim de subsidiar futuras pesquisas na área que demandem tal prototipagem.

¹ Trabalho desenvolvido e escrito entre Maio e Julho de 2023

2 REVISÃO DE LITERATURA

Neste capítulo serão apresentados estudos anteriores em tópicos abordados neste trabalho, sendo eles o reconhecimento de fala automático, produção de fala, emoções na interação e inteligência artificial generativa. Ele fornecer uma visão panorâmica do estado atual do conhecimento sobre os temas e identifica as principais teorias, afim de estabelecer uma base sólida para o desenvolvimento do presente estudo.

2.1 Reconhecimento de fala automático

"O reconhecimento de fala, também conhecido como reconhecimento automático de fala (ASR), reconhecimento de fala por computador ou conversão de fala em texto, é um recurso que permite que um programa processe a fala humana em um formato escrito." IBM (2023).

Aldarmaki et al. (2022, p. 3) define o reconhecimento de fala automático, do qual compreendemos e traduzimos como:

É o processo de identificar automaticamente padrões em uma forma de onda de fala. Padrões que podem ser detectados a partir da fala incluem a identidade do falante, idioma, emoção e a transcrição textual da enunciação falada. Esta última é geralmente o que se busca no ASR.

No ASR tradicional, de acordo com Aldarmaki et al. (2022) os modelos são compostos por três componentes principais: um modelo acústico, um dicionário de pronúncia e um modelo de linguagem. O modelo acústico calcula a probabilidade de unidades acústicas, por exemplo fonemas que são a menor unidade reconhecível da fala, que podem ser modeladas usando Modelos de Mistura Gaussianas (GMMs) e Modelos de Markov Ocultos (HMMs). O modelo de linguagem faz o cálculo da probabilidade de uma sequência de palavras. O dicionário de pronúncia é utilizado para mapear a sequência de fonemas em palavras, geradas da combinação dos dois modelos.

Aldarmaki et al. (2022) explica que no ASR moderno os sistemas para o reconhecimento são *end-to-end*. Arquiteturas de decodificador-codificador, o uso de redes neurais convolucionais (CNNs) e redes neurais recorrentes (RNNs), são utilizadas para processar o áudio de entrada e gerar uma sequência de caracteres como saída.

Para se trabalhar com o reconhecimento de fala automático em Flutter existem diversas bibliotecas disponíveis para isso. A biblioteca *azure_speech_recognition_null_safety* que implementa Azure Cognitive Speech Recognition para Flutter, isso permite que o usuário utilize

os recursos da Azure para reconhecimento de fala e transcrição para texto, sua documentação pode ser encontrada no repositório público de Bregant (2020). A *google_speech* implementa o plugin Speech-to-Text API do Google para o mesmo objetivo que pode ser encontrada no repositório Junghans (2021). Existe também a *speech-to-text* nativa do próprio Flutter que também permite atingir esse objetivo, sendo mantida pela comunidade, e sua documentação pode ser acessada em Software (2021). É possível encontrar diversas outras no site oficial de gerenciador de pacotes Dart para Flutter Google (2017).

A biblioteca escolhida para o desenvolvimento deste trabalho foi a *speech-to-text* por diversos fatores. Além de oferecer suporte para diversos idiomas, o desenvolvedor desse trabalho a considera simples e de fácil implementação. Considera que não há necessidade de qualquer configuração adicional, permitindo a transcrição da fala em texto em tempo real. As demais bibliotecas necessitam configurações adicionais, tais como a criação de uma conta Google e Microsoft para obtenção de chaves de API. Isso pode gerar custos adicionais caso ultrapasse um limite de transcrições (GOOGLE, 2023b; MICROSOFT, 2023c).

A biblioteca *speech-to-text* nativa do Flutter é utilizada para o reconhecimento de voz através do microfone do dispositivo, transformando o áudio da fala captada do microfone em texto (SOFTWARE, 2021).

No aplicativo desenvolvido, foram utilizados três métodos da classe *SpeechToText* presente na biblioteca. O primeiro método utilizado foi o *initialize()* que inicializa o serviço de reconhecimento de voz e retorna *true* (verdadeiro) caso a inicialização seja bem sucedida e *false* (falso) caso ocorra algum problema. O segundo método é o *listen()* que inicializa o modo de escuta e permite converter cada palavra que foi reconhecida em texto. Por fim, o método *stop()* que interrompe a escuta caso ela esteja ativa (SOFTWARE, 2021).

2.2 Produção de fala

Como define Ning et al. (2019), a produção de fala, também conhecida como *text-to-speech* em inglês, é uma tecnologia que envolve diversas áreas como acústica, linguística, processamento de sinais digitais e estatística com o objetivo principal de converter uma entrada de texto em fala.

"Modelos TTS (Text-to-Speech) baseados em redes neurais de ponta a ponta geralmente primeiro convertem o texto em características acústicas (por exemplo, espectrogramas de mel) e, em seguida, transformam os mel-espectrogramas em amostras de áudio." (REN et al., 2019).

É possível trabalhar com a produção de fala em Flutter fazendo o uso de alguma das bibliotecas disponíveis no site oficial de pacotes Google (2017). Lá é possível encontrar uma diversidade de opções que permitem trabalhar transformando um texto qualquer em áudio de fala. A *flutter_azure_tts*, por exemplo, implementa o plugin Microsoft Azure Cognitive Text-To-Speech API que permite transformar um texto em fala humana sintetizada em diversas línguas (FEL, 2021). Ao passo que a *flutter_tts* utiliza de recursos do próprio aparelho para produzir o mesmo resultado (LUTTON, 2018).

Para o desenvolvimento deste trabalho foi escolhida a *flutter_tts* pelos mesmos motivos da escolha da *speech_to_text*. Além de ser robusta e com capacidade de produzir fala em vários idiomas, é de fácil implementação de acordo com o desenvolvedor desse trabalho, ao passo que a *flutter_azure_tts* pode gerar configurações adicionais como a necessidade de se criar uma conta Microsoft e obtenção de chaves de API para uso. Além disso, a *flutter_azure_tts* pode gerar custos adicionais a partir de um limite de produção (MICROSOFT, 2023c).

A biblioteca *flutter_tts* faz o caminho inverso à biblioteca *speech-to-text*, ou seja, ela é utilizada para transformar um texto qualquer em fala. Essa biblioteca também possui algumas classes e vários métodos para atingir o principal objetivo (LUTTON, 2018).

Para o desenvolvimento utilizou-se três métodos da classe *FlutterTts*. O primeiro é o método *setLanguage()* usado para setar em qual linguagem as palavras serão faladas, neste caso a linguagem 'pt-BR', que representa o Português do Brasil, é passada como parâmetro para o método. O segundo é o *speak()* que é utilizado para reproduzir o texto em fala, esse método espera como parâmetro o texto que deve ser reproduzido. E por último o método *setCompletionHandler()* que espera uma função de retorno como parâmetro, o seu objetivo é chamar essa função após o término do método *speak()*, ou seja, enquanto todo o texto não é reproduzido totalmente a função de retorno não é chamada (LUTTON, 2018).

2.3 Emoção na interação

Bartneck et al. (2020, p. 118) explica que nas interações humano-robô as emoções são consideradas um importante canal de comunicação. Nesse sentido, mesmo que um robô não tenha sido projetado principalmente para expressar emoções, as pessoas ainda assim tendem a interpretá-los como se fossem movidos por estados emocionais.

De acordo com Bartneck et al. (2020, p. 120): "Normalmente, as pessoas projetam robôs que transmitem emoções por meio de expressões faciais. A abordagem mais comum aqui é imitar a maneira como as pessoas demonstram emoções."

Em uma interação humano-robô, onde emoções estão envolvidas, a satisfação do usuário aumenta de acordo com Dohsaka et al. (2014), conforme citado por Chaves e Gerosa (2021, p. 28) onde:

[...] propuseram um chatbot que usa expressões emocionais empáticas e auto-orientadas para manter os usuários envolvidos em um diálogo estilo questionário. Os resultados da pesquisa revelaram que expressões empáticas melhoraram significativamente a satisfação do usuário. Além disso, as expressões empáticas também melhoraram as avaliações do usuário do agente de pares em relação à intimidade, compaixão, amabilidade e encorajamento.

Apesar de serem emoções simplificadas, *gifs* animados também podem transmiti-las de maneira eficaz. Eles se traduzem em uma linguagem visual que pode ser facilmente compreendida e interpretada por quem interage com eles. Por exemplo, um *gif* animado de uma pessoa sorrindo transmite alegria e felicidade, enquanto um *gif* de alguém franzindo a testa e parecendo preocupado pode evocar uma sensação de ansiedade ou tensão. Além disso, a natureza repetitiva dos *gifs* animados permite que as emoções sejam prolongadas e enfatizadas. Ao repetir uma expressão facial várias vezes, por exemplo, os *gifs* podem criar uma sensação de intensidade emocional, reforçando a mensagem emocional transmitida (YANG; ZHANG; LUO, 2019).

Foram escolhidos dois *gifs* animados para simular o funcionamento do aplicativo desenvolvido ao longo do trabalho, tanto no modo de escuta (*speech-to-text*) quanto no modo de fala (*text-to-speech*).

No modo de escuta, o *gif* é representado por dois olhos, simbolizando a atenção do aplicativo em relação à pessoa que está falando (ROBOTICS, 2021). Além disso, esses olhos podem transmitir emoções como curiosidade, interesse ou empatia, reforçando a capacidade do aplicativo de se conectar emocionalmente com o usuário. Já o *gif* no modo de fala é representado por um emoji de uma boca se movendo, como se estivesse falando, podendo demonstrar variados tipos de emoções como alegria, surpresa, ênfase ou até mesmo preocupação, dependendo do contexto das palavras ditas (CHUWY'S, 2019).

Os *gifs* foram retirados do site giphy, que disponibiliza *gifs* gratuitos para uso e estão disponíveis em Robotics (2021), Chuwy's (2019), utilizados para o modo escuta e fala, respectivamente.

2.4 IA Generativa

Zhang et al. (2023) explica que uma Inteligência Artificial Generativa (AIGC) é um conjunto de tarefas ou aplicações que geram conteúdo novo por meio de métodos de IA a partir de dados existentes. "A IA generativa moderna depende de vários fundamentos técnicos, desde arquitetura de modelo e pré-treinamento auto-supervisionado até métodos de modelagem generativa (como GAN e modelos de difusão)." (ZHANG et al., 2023, p. 2).

Depois de avanços nos estudos sobre IA Generativa e o lançamento do ChatGPT, grandes empresas começaram a se movimentar para lançar estudos e ferramentas na área. O Google lançou o Bard, um chatbot que responde a questionamentos feitos pelos usuário (COLLINS, 2023). A Microsoft lançou em parceria com a OpenAI, o Bing Chat que utiliza o ChatGPT, que também responde perguntas feitas a ele (MICROSOFT, 2023b). Existem também as generativas de imagem a partir de uma descrição introduzida, como o Dall-E 2 e o Midjourney OpenAI (2023b), Midjourney (2023), respectivamente. Vale ressaltar ainda que além das mencionadas, existem várias outras que já foram lançadas e muitos estudos e projetos estão em desenvolvimento nesse sentido, e com o ritmo acelerado de inovação, é provável que sejam lançadas novas IAs em breve, oferecendo ainda mais recursos e oportunidades para diversos setores.

O ChatGPT é um modelo de inteligência artificial generativa desenvolvido pela OpenAI. Ele foi construído sobre a arquitetura GPT (*Generative Pre-trained Transformer*), que utiliza redes neurais profundas e técnicas de aprendizado de máquina para entender e gerar texto (HETLER, 2023).

A versão utilizada para o desenvolvimento deste trabalho é a GPT-3.5 uma das versões mais avançadas da OpenAI, treinada com uma grande variedade de dados e conhecimentos até o fim do ano de 2021 (NATALIE, 2023; OPENAI, 2023c). Possui ainda uma API bastante acessível e de fácil integração e uso em várias plataformas (OPENAI, 2023c). Por esses motivos foi escolhido para o desenvolvimento do trabalho.

Para a comunicação entre o aplicativo e a API da OpenAI foi utilizada a biblioteca http nativa (DART, 2018). No site oficial da OpenAI foi obtida a chave e a url(Localizador Uniforme de Recursos) da API para comunicação entre as partes OpenAI (2023c). Utilizando a biblioteca foi feita uma requisição utilizando-se o método *POST*, passando no *header* da requisição a chave obtida através da API e no *body* a fala dita pelo usuário que inicialmente foi convertida em texto utilizando-se a biblioteca *speech-to-text*. Depois de feita a requisição, é aguardada a

resposta da API e caso haja sucesso, o texto gerado é convertido em fala, utilizando agora a biblioteca *text-to-speech*.

3 MATERIAIS E MÉTODOS

Este capítulo descreve os métodos e técnicas utilizados na realização deste estudo, bem como os recursos necessários para sua execução. A escolha dos procedimentos metodológicos foi baseada nos objetivos e na natureza da pesquisa.

3.1 Métodos

O objetivo deste estudo consiste em investigar as possibilidades tecnológicas de adaptação do framework OpenTera para incorporar os benefícios do uso de Inteligências Artificiais generativas na prototipagem de SARs. É importante ressaltar que a busca por melhorias nas interações humano-robô não se restringe apenas a projetos que visam explorar necessidades ou preferências humanas, mas também abrange a exploração e o avanço técnico nessa área (BARTNECK et al., 2020, p. 57).

Portanto, este estudo está fundamentado nos passos metodológicos tradicionais da área (BARTNECK et al., 2020, p. 57):

- 1. Definição do problema ou questão de pesquisa.
- 2. Construção da interação.
- 3. Realização de testes.
- 4. Análise dos resultados.
- 5. Repetição dos passos 2 a 4 até que o problema ou questão seja satisfatoriamente abordado.

A definição do problema de pesquisa encontra-se exposta na seção introdutória deste estudo: o framework OpenTera não possui módulos para facilitação da prototipagem de conversação e emoções em robôs sociais. As seções subsequentes descrevem a construção da interação do SAR pela utilização da API da OpenAI, desenvolvimento de aplicativo em Flutter, bibliotecas e APIs de reconhecimento de fala e tradução de *strings* de caracteres em fala (áudio) para o robô. Nas referidas seções, são detalhados os testes realizados e as análises efetuadas para identificar a forma como a adaptação do OpenTera poderia ser executada, viabilizando a incorporação dos benefícios provenientes da API. Todos os testes e análises foram feitos com base na experiência do autor do trabalho enquanto desenvolvedor de software. Seguimos os passos de Sejnowski (2023) para realizar teste de Turing com a nova aplicação.

O teste de Turing é um processo avaliativo para verificar a habilidade de uma máquina de manifestar comportamento inteligente equivalente ou indiscernível de um ser humano. O

teste consiste em um avaliador humano fazer questões a um interlocutor humano e a um interlocutor máquina, ignorando qual é qual, e tentar reconhecer quem é a máquina com base nas respostas. Se o avaliador não puder diferenciar a máquina do humano, a máquina satisfaz o teste (SEJNOWSKI, 2023, p. 317).

Ressaltamos que o autor possui um ano e dois meses de experiência como desenvolvedor no mercado, trabalhando para a empresa MGCode (<<https://mgcode.com.br/>>). Durante os três primeiros meses, atuou como estagiário em desenvolvimento. Os demais nove meses foram de trabalho como desenvolvedor júnior, atuando com tecnologias de desenvolvimento Web (Javascript, PHP, Angular, React, Java e React Native).

3.2 Materiais

Esta seção apresenta todos os materiais e ferramentas utilizadas para o desenvolvimento do trabalho. É detalhada a instalação de ferramentas necessárias e configurações que precisam ser feitas, além da obtenção de recursos úteis para se utilizar a API do ChatGPT.

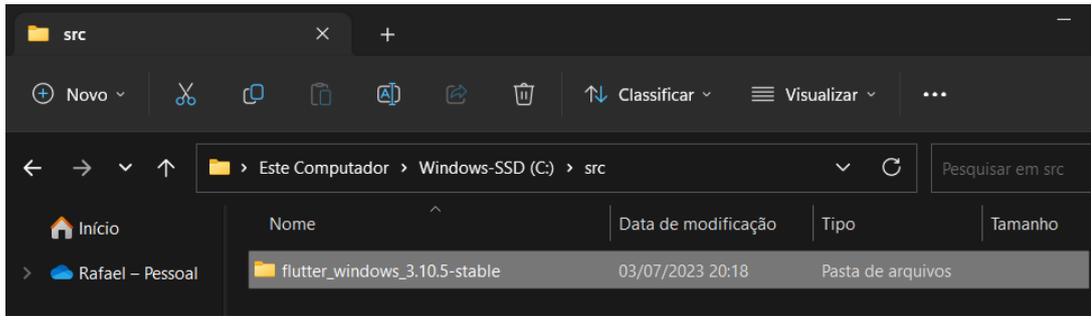
3.2.1 Instalação e configuração do Flutter no Windows

A primeira coisa a se fazer para a instalação do Flutter é verificar os requisitos exigidos para o ambiente de desenvolvimento. Os requisitos são:

- Sistema operacional Windows 10 ou superior.
- 1,64GB de espaço em disco, excluindo-se as IDEs.
- Windows PowerShell 5.0 ou mais recente.
- Git para Windows 2.x, com as opções de usar comandos git no prompt.

Após conferir se a máquina é compatível com os requisitos basta baixar o pacote mais recente do Flutter SDK, que pode ser encontrado em (FLUTTER, 2023) e extraí-lo para o local desejado para a instalação, no caso deste projeto foi utilizado seguinte caminho `C:\src`. É aconselhável que o caminho de instalação não contenha caracteres especiais ou espaço.

Figura 3.1 – Pacote Flutter extraído.



Após a instalação é necessário acessar as variáveis de ambiente do Windows e adicionar à variável Path o caminho até a pasta *bin* do Flutter, conforme mostrado na Figura 3.2, Figura 3.3 e Figura 3.4.

Figura 3.2 – Acessando as variáveis de ambiente.

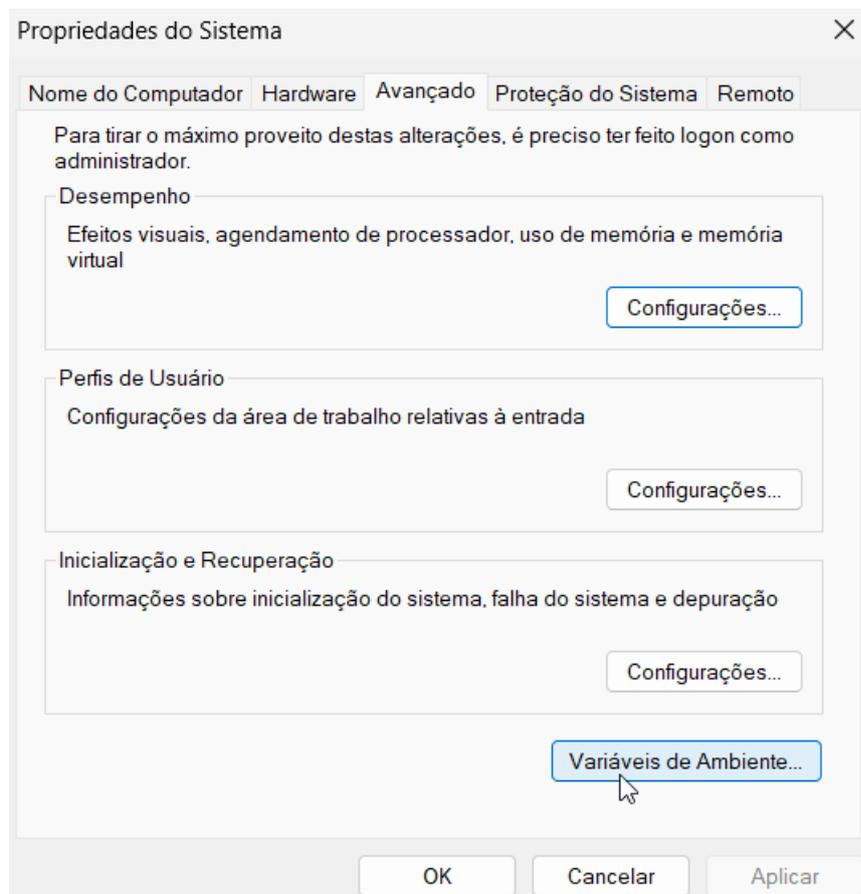


Figura 3.3 – Editando a variável Path.

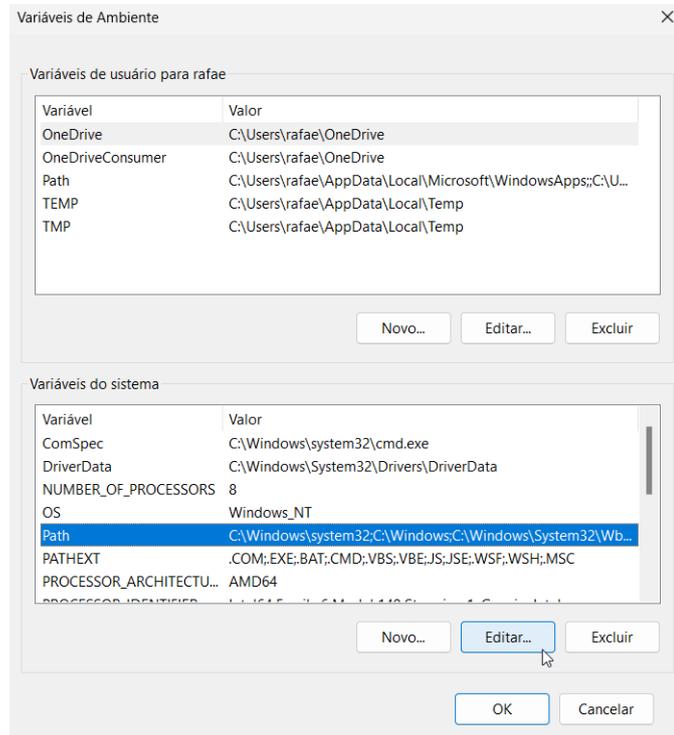
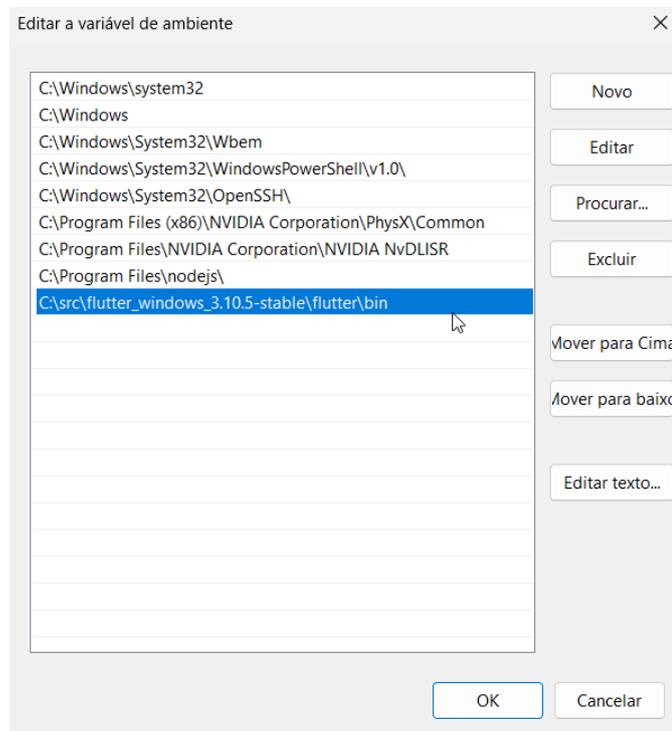


Figura 3.4 – Adicionando o caminho do flutter/bin.

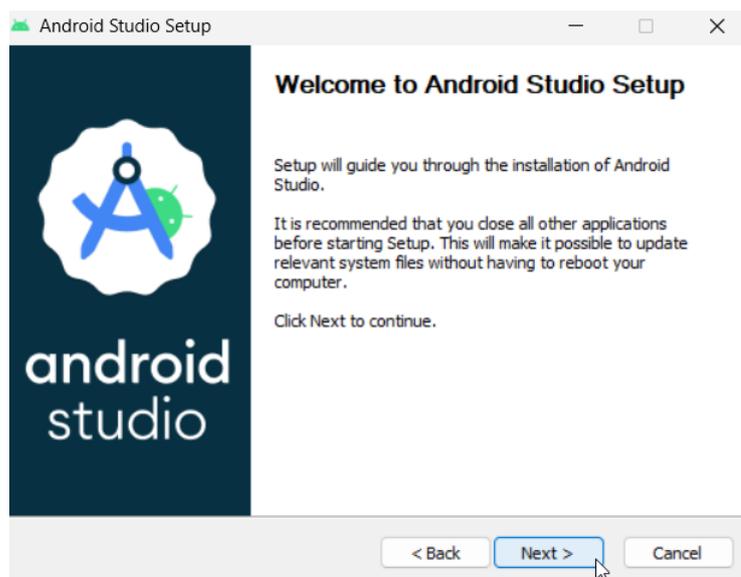


Isso permitirá rodar comandos Flutter diretamente no prompt de comandos e a instalação estará completa.

3.2.2 Instalação e configuração do Android Studio

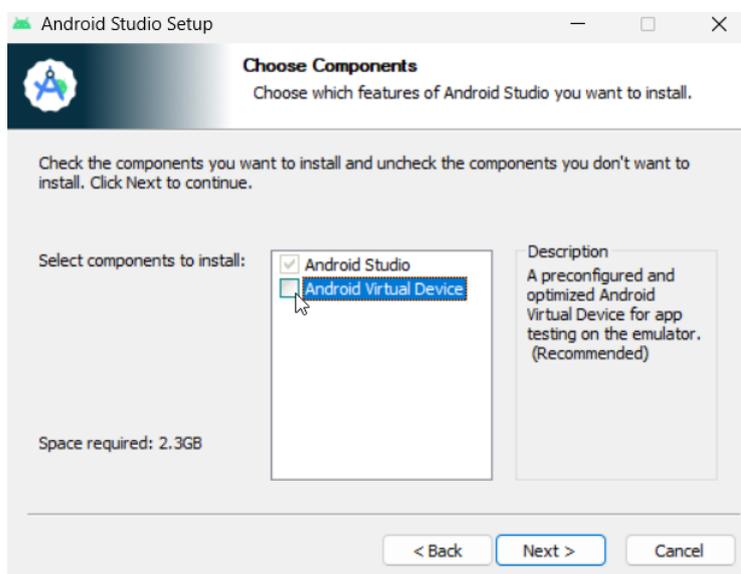
A instalação e configuração do Android Studio é bem simples, basta baixar o executável e executá-lo (GOOGLE, 2023a). Logo após executá-lo a tela de boas-vindas será exibida, basta clicar em *Next* para prosseguir, conforme mostrado na Figura 3.5

Figura 3.5 – Boas-vindas Android Studio.



Depois é necessário escolher os componentes de instalação. Como para o presente trabalho o aplicativo foi executado em um dispositivo físico a opção de criar um emulador foi desmarcada como mostra a Figura 3.6.

Figura 3.6 – Escolhendo componentes de instalação do Android Studio.



Em seguida, é necessário escolher o local onde será feita a instalação e prosseguir para instalá-lo, conforme demonstrado na Figura 3.7 e Figura 3.8, respectivamente.

Figura 3.7 – Escolhendo local de instalação Android Studio.

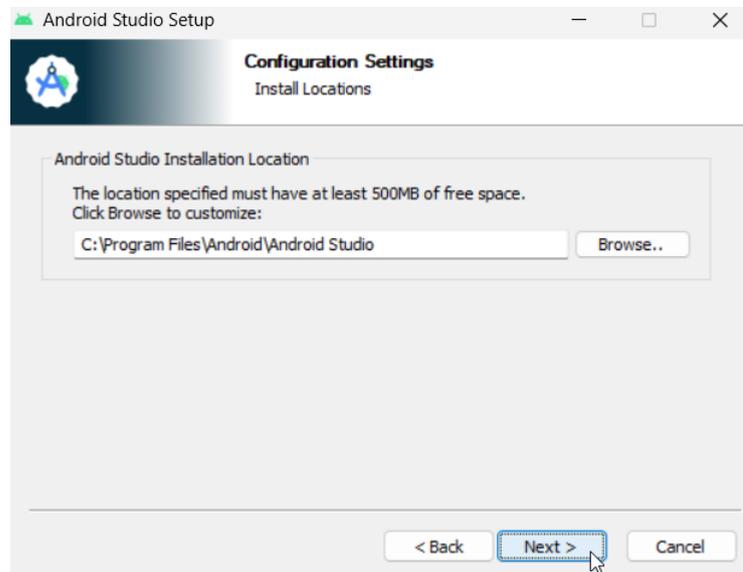
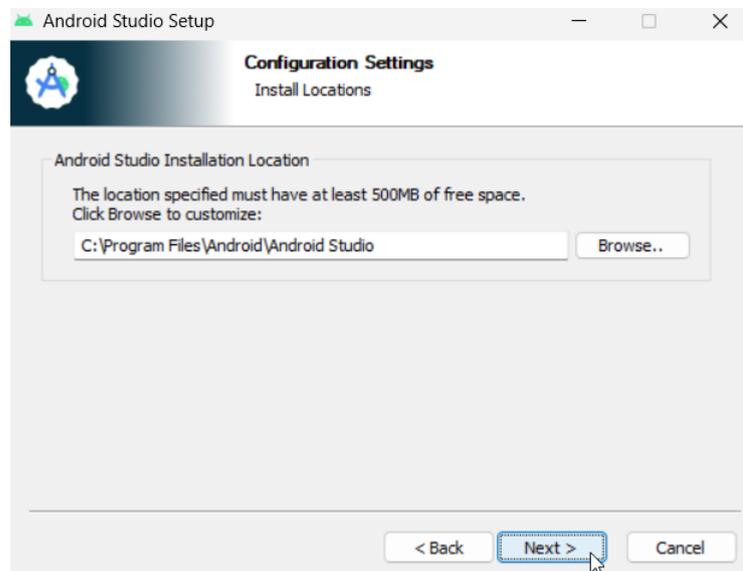
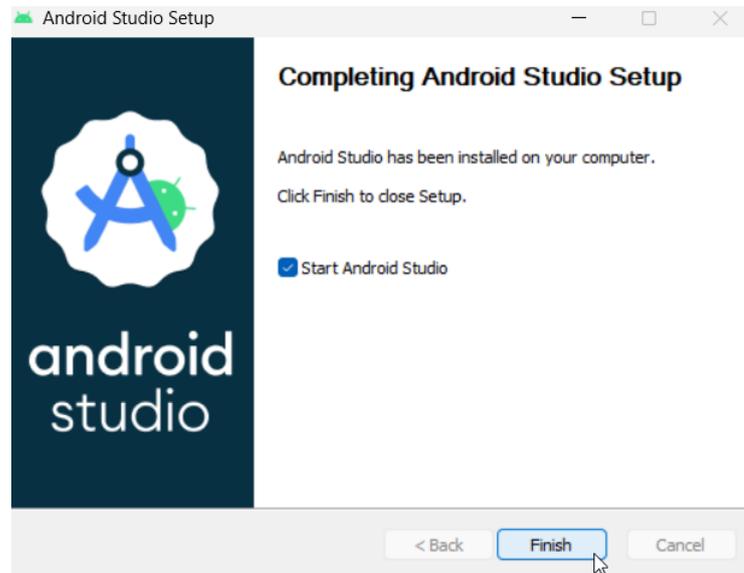


Figura 3.8 – Iniciando a instalação do Android Studio.



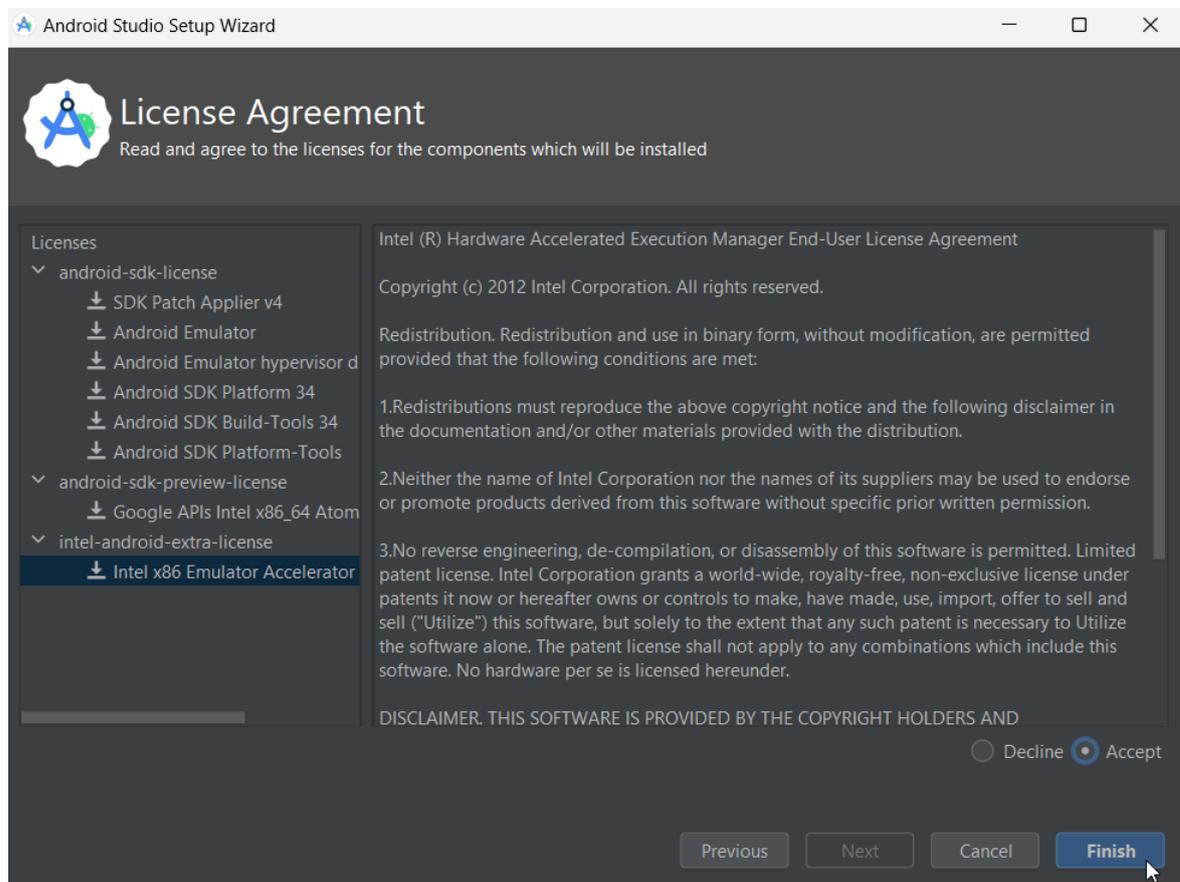
Assim que a instalação for concluída, a opção para iniciar o AndroidStudio pode ser marcada, conforme a Figura 3.9.

Figura 3.9 – Instalação do Android Studio finalizada.



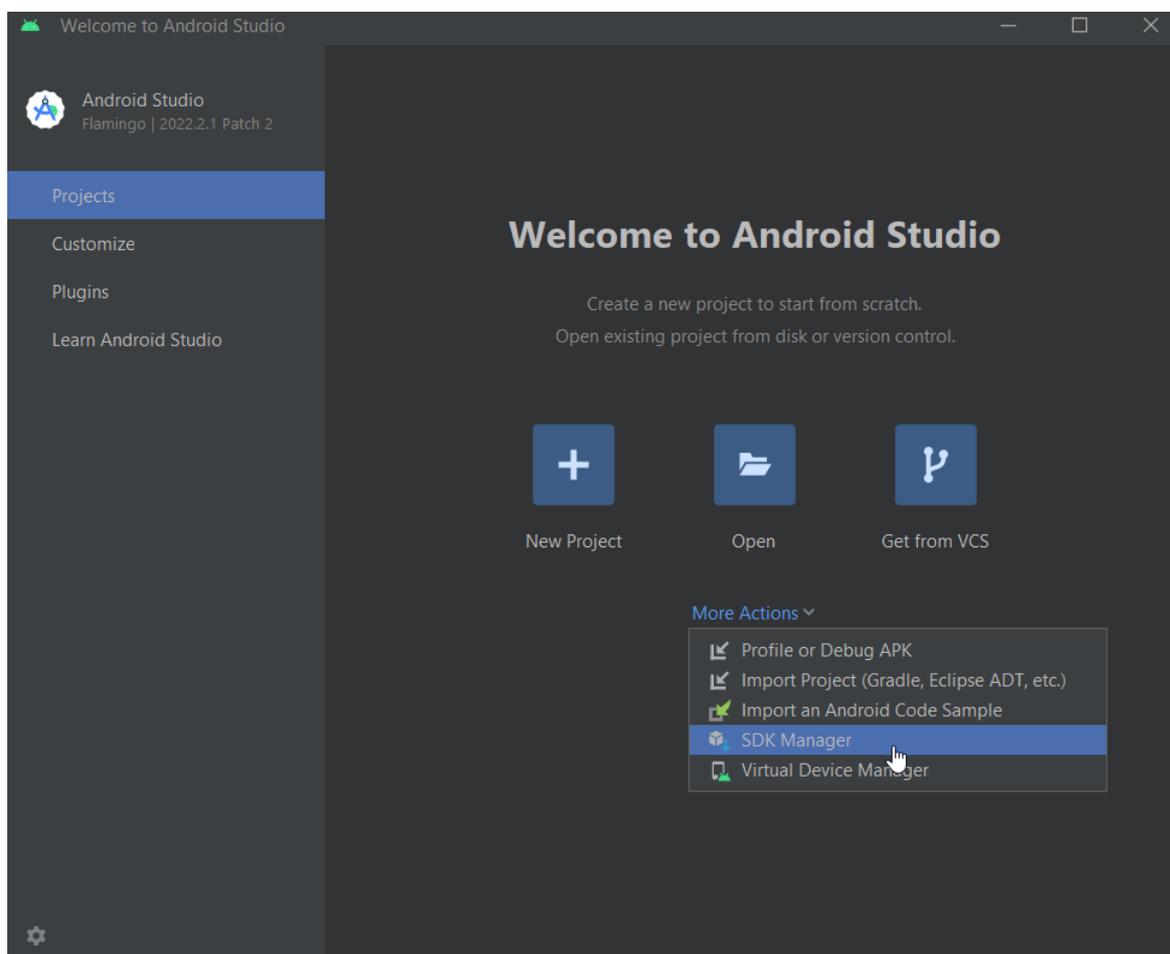
Com o Android Studio já iniciado é necessário aceitar as políticas de licença antes de prosseguir para a instalação dos componentes, como indicado na Figura 3.10

Figura 3.10 – Instalação de componentes do Android Studio.



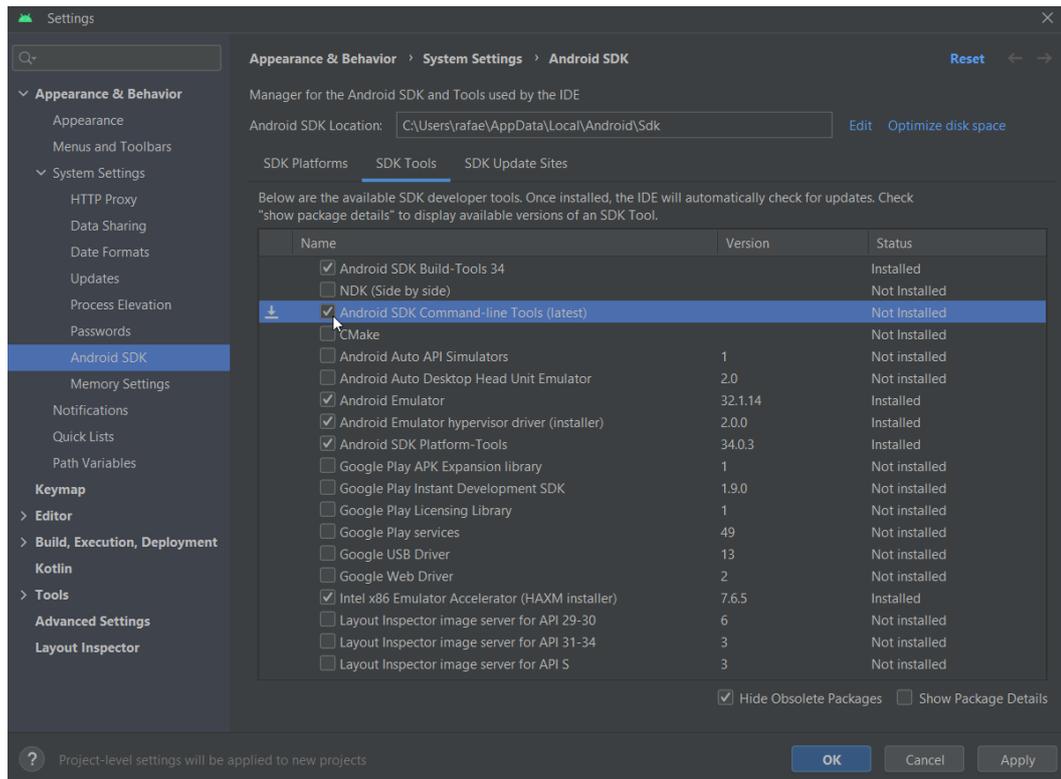
Após a instalação dos componentes finalizar é necessário baixar o *SDK Command-line Tools*. São apenas dois passos para realizar a instalação. Primeiro, deve-se abrir o AndroidStudio e ir até a opção *SDK Manager*, conforme mostrado na Figura 3.11.

Figura 3.11 – Acessando SDK Manager no Android Studio.

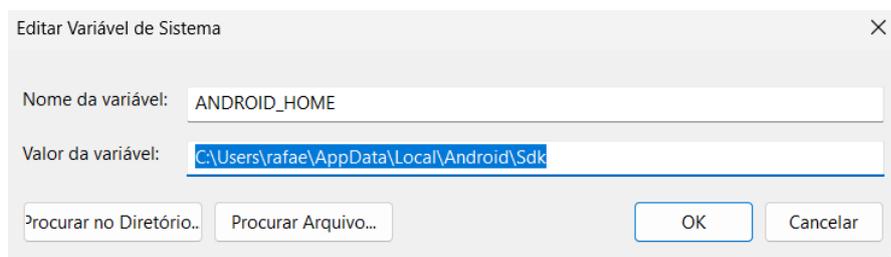
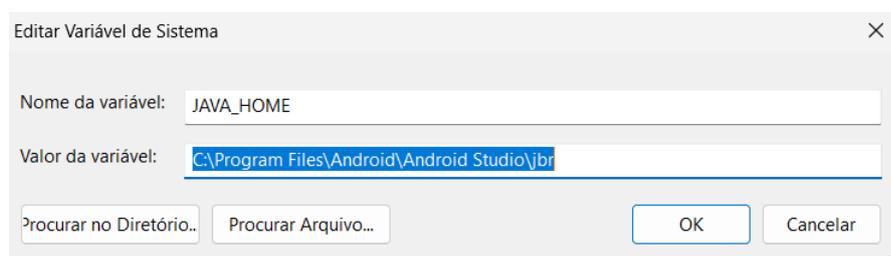


Depois de ter acessado *SDK Manager* deve-se navegar até a aba *SDK Tools* e selecionar a opção *Android SDK Command-line Tools* e clicar em OK para instalá-la, de acordo com a Figura 3.12.

Figura 3.12 – Selecionando SDK Command-line Tools no Android Studio.



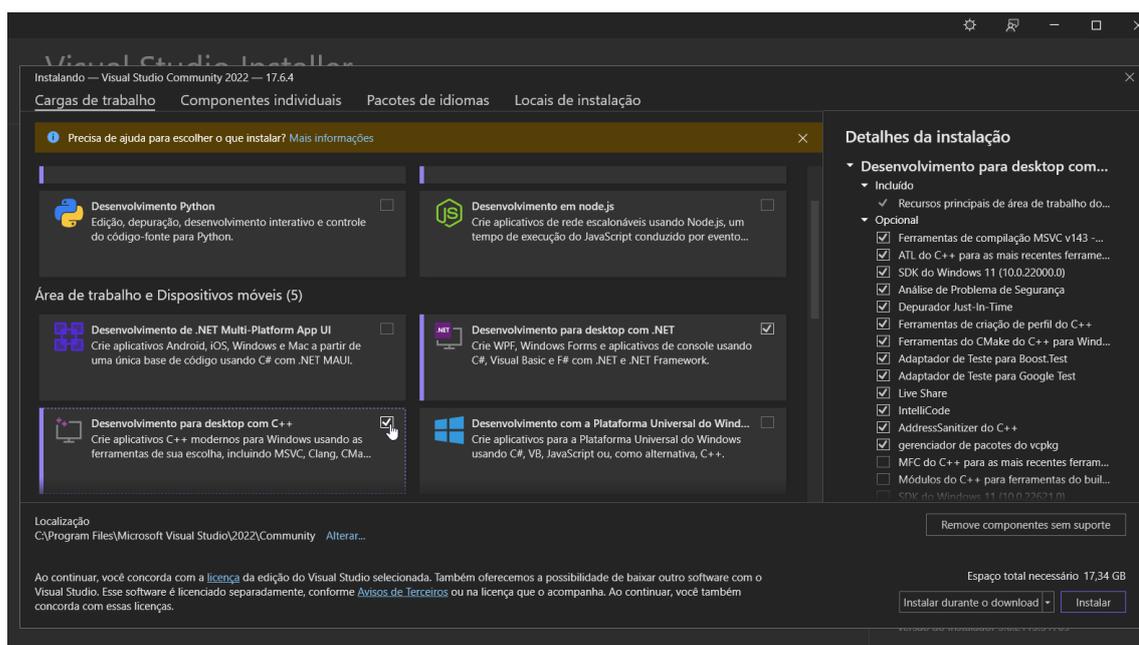
Por fim, é necessário adicionar duas novas variáveis de ambiente nomeadas `ANDROID_HOME` e `JAVA_HOME` com os caminhos do Android SDK (Kit de Desenvolvimento de Software) e jbr do Android Studio como mostrado na Figura 3.13 e Figura 3.14, respectivamente.

Figura 3.13 – Adicionando variável de ambiente `ANDROID_HOME`.Figura 3.14 – Adicionando variável de ambiente `JAVA_HOME`.

3.2.3 Instalação do Microsoft Visual Studio Community 2022 com C++

O Microsoft Visual Studio deve ser instalado com o componente do C++. Logo, é necessário baixar o instalador, executá-lo e depois selecionar a opção Desenvolvimento para desktop com C++ e iniciar a instalação, conforme Figura 3.15. Após a instalação, o Visual Studio com o C++ já estará configurado.

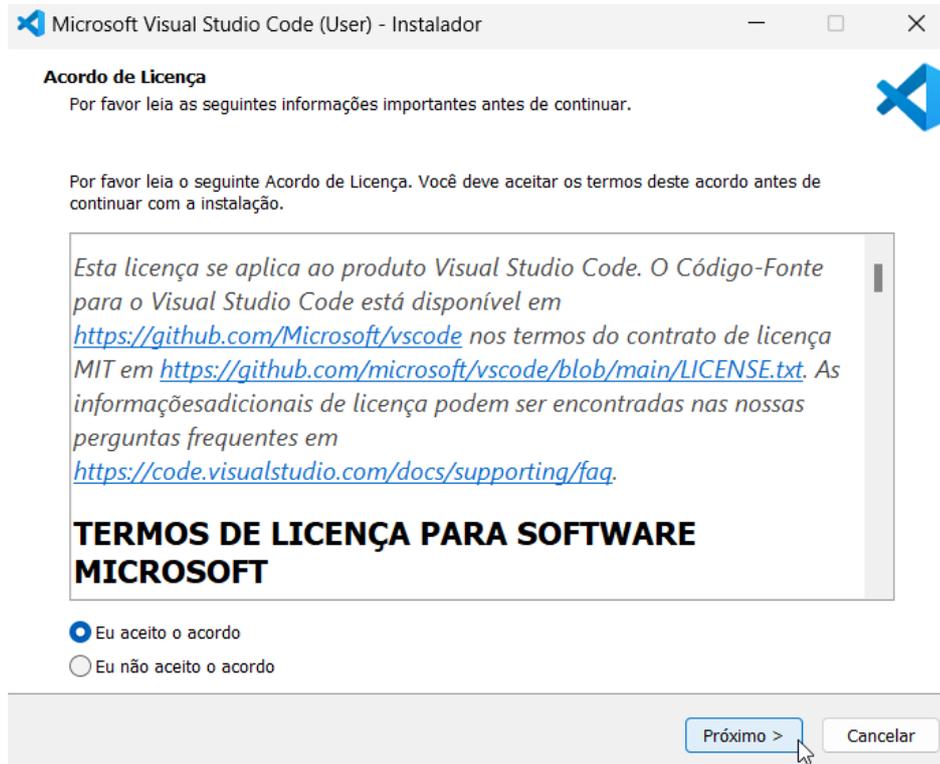
Figura 3.15 – Instalação do Visual Studio Community com C++.



3.2.4 Instalação do editor de código Visual Studio Code

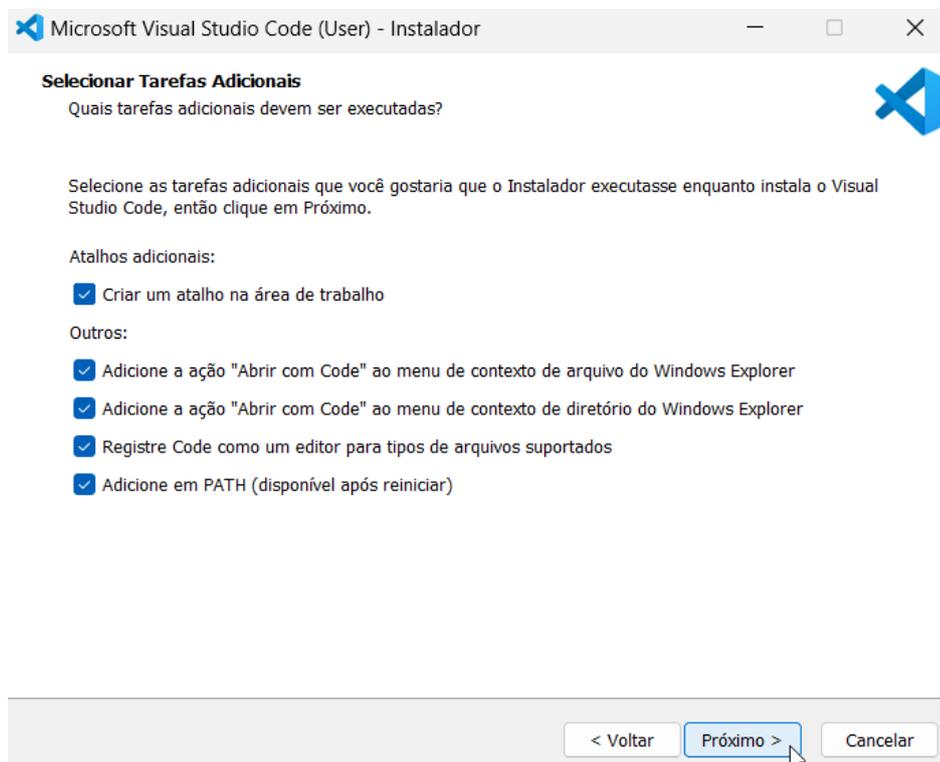
O Visual Studio Code, popular VSCode, será utilizado para o desenvolvimento do código do aplicativo por ser de fácil uso e completo para se criar um aplicativo Flutter segundo o autor do presente trabalho. Para sua instalação é necessário baixar seu executável (MICROSOFT, 2023a). Após executá-lo, logo no início da instalação os termos devem ser aceitos para prosseguir, como exibido na Figura 3.16.

Figura 3.16 – Termos de instalação Visual Studio Code.



Prosseguindo, deve-se marcar todas as opções de ferramentas para melhor uso do VS-Code e prosseguir para a instalação, conforme Figura 3.17.

Figura 3.17 – Ferramentas do Visual Studio Code.

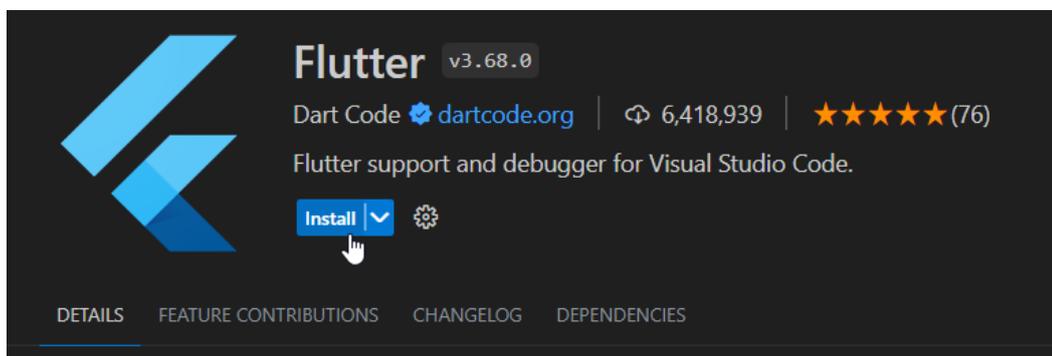


Com a instalação finalizada e o VSCode pronto para uso, deve ser configurada a extensão do Flutter que é tema da próxima subsecção.

3.2.5 Configuração do Flutter no Visual Studio Code

A configuração do Flutter no VSCode é rápida e não tem demais complicações. É necessário apenas ir até Extensões no editor de código e pesquisar por Flutter, a extensão necessária é a primeira que aparece e pode ser conferida na Figura 3.18.

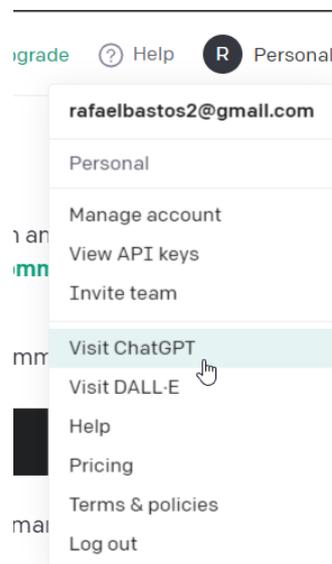
Figura 3.18 – Extensão do Flutter no Visual Studio Code.



3.2.6 Obtenção da key e da URL da API do ChatGPT

A obtenção da API key do ChatGPT é simples. É necessário acessar o site da OpenAI e navegar até a documentação da API e fazer o login (OPENAI, 2023a). Logo após o login, basta clicar em cima do perfil no canto superior direito e ir em *View API keys*, como mostrado na Figura 3.19.

Figura 3.19 – Acessando as API keys do ChatGPT



Já na página das *API key* é necessário criar uma nova, clicando no botão *Create new secret key*, como ilustrado na Figura 3.20. Então, basta dar um nome, salva-lá e guarda-lá para usa-lá posteriormente.

Figura 3.20 – Criando uma nova API key do ChatGPT.

API keys

Your secret API keys are listed below. Please note that we do not display your secret API keys again after you generate them.

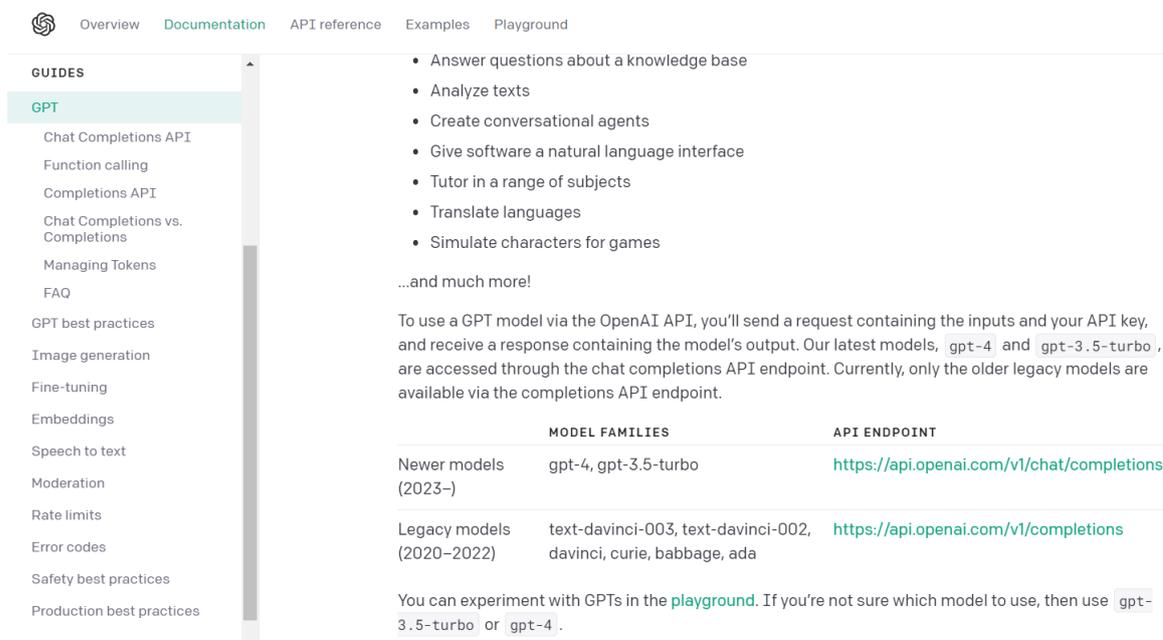
Do not share your API key with others, or expose it in the browser or other client-side code. In order to protect the security of your account, OpenAI may also automatically rotate any API key that we've found has leaked publicly.

NAME KEY CREATED LAST USED ⓘ

+ Create new secret key

Para obter a URL da API é necessário navegar até a documentação e no menu lateral esquerdo dentro de *Guides* selecionar GPT. A URL da API está presente em uma tabela com os modelos da OpenAI, como mostrado na Figura 3.21.

Figura 3.21 – Obtendo a URL da API do ChatGPT.



Overview Documentation API reference Examples Playground

GUIDES

- GPT
- Chat Completions API
- Function calling
- Completions API
- Chat Completions vs. Completions
- Managing Tokens
- FAQ
- GPT best practices
- Image generation
- Fine-tuning
- Embeddings
- Speech to text
- Moderation
- Rate limits
- Error codes
- Safety best practices
- Production best practices

- Answer questions about a knowledge base
- Analyze texts
- Create conversational agents
- Give software a natural language interface
- Tutor in a range of subjects
- Translate languages
- Simulate characters for games

...and much more!

To use a GPT model via the OpenAI API, you'll send a request containing the inputs and your API key, and receive a response containing the model's output. Our latest models, `gpt-4` and `gpt-3.5-turbo`, are accessed through the chat completions API endpoint. Currently, only the older legacy models are available via the completions API endpoint.

	MODEL FAMILIES	API ENDPOINT
Newer models (2023-)	<code>gpt-4</code> , <code>gpt-3.5-turbo</code>	https://api.openai.com/v1/chat/completions
Legacy models (2020-2022)	<code>text-davinci-003</code> , <code>text-davinci-002</code> , <code>davinci</code> , <code>curie</code> , <code>babbage</code> , <code>ada</code>	https://api.openai.com/v1/completions

You can experiment with GPTs in the [playground](#). If you're not sure which model to use, then use `gpt-3.5-turbo` or `gpt-4`.

Para o presente trabalho é usado o modelo `gpt-3.5-turbo`.

4 RESULTADOS E DISCUSSÕES

Esta seção apresenta de forma clara como o projeto foi construído e os resultados obtidos. É demonstrado a visão arquitetônica do novo módulo que surge como proposta para integrar o *framework* OpenTera, denominado OpenTera++. Ainda, detalha como se deu a construção do aplicativo móvel em Flutter, seu funcionamento e limitações.

4.1 Visão arquitetônica OpenTera++

O OpenTera++ foi desenvolvido para ser parte integrante do OpenTera, funcionando como um robô assistivo social. Ele segue o padrão do *framework* original, contando com uma parte no *backend* e outra no *frontend*, além de uma camada intermediária do transporte seguro HTTPS.

O *backend* é composto pela API do ChatGPT que recebe as requisições vindas do *frontend* as processa e devolve a resposta com base em uma pergunta feita. O *frontend* é composto pela aplicação móvel Flutter, com as bibliotecas utilizadas *speech-to-text* e *text-to-speech* e os métodos criados no desenvolvimento do aplicativo *_listen()*, *_play()*, *getChatGptResponse()* e *_startInteraction()*. De forma resumida a aplicação recebe perguntas do usuário por meio de fala, trancreve-as para texto utilizando a biblioteca *speech-to-text*, envia para o *backend* que processa a pergunta e devolve uma resposta que é transmitida ao usuário por meio da biblioteca *flutter_tts*.

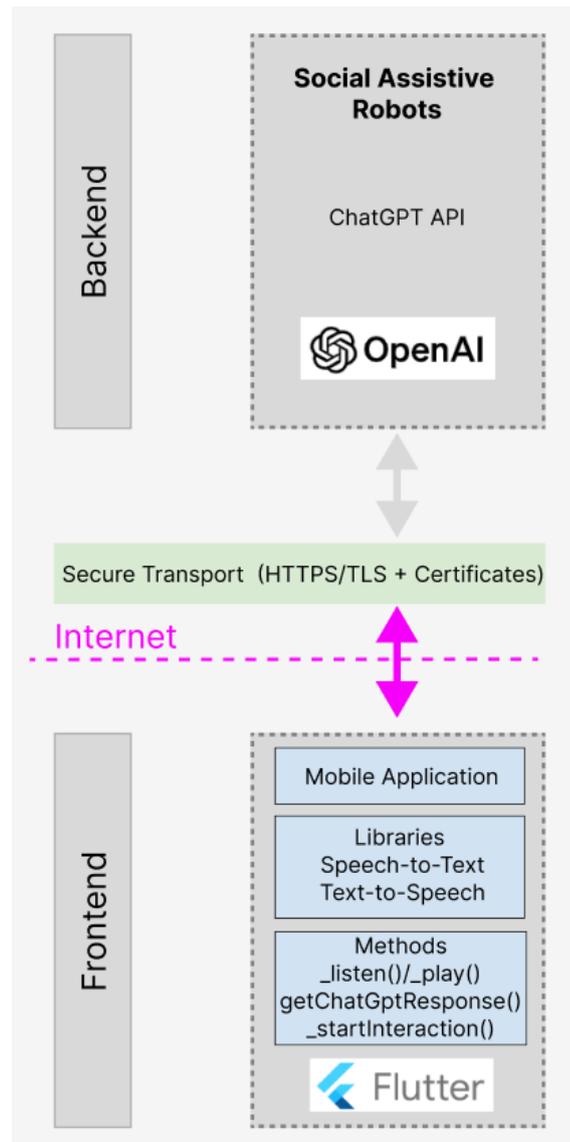
Portanto, novo módulo pode lidar com as interações com os usuários, processar suas perguntas e obter respostas do ChatGPT. Isso permite que o OpenTera ofereça um assistente virtual inteligente como parte integrante de sua funcionalidade.

Ele ainda pode ser integrado a outros módulos do OpenTera, como gerenciamento de dados e sessões. Isso permite que o robô social utilize informações relevantes e contextuais para fornecer respostas mais precisas e personalizadas aos usuários.

Em resumo, a incorporação deste novo módulo, um robô assistivo social baseado na API do ChatGPT, ao *framework* OpenTera pode enriquecer a funcionalidade do framework, oferecendo aos usuários um assistente virtual inteligente e interativo.

O módulo proposto foi nomeado OpenTera++ e seu objetivo é ser um robô social assistivo, ele é mostrado na Figura 4.1.

Figura 4.1 – Módulo Social Assistive Robots



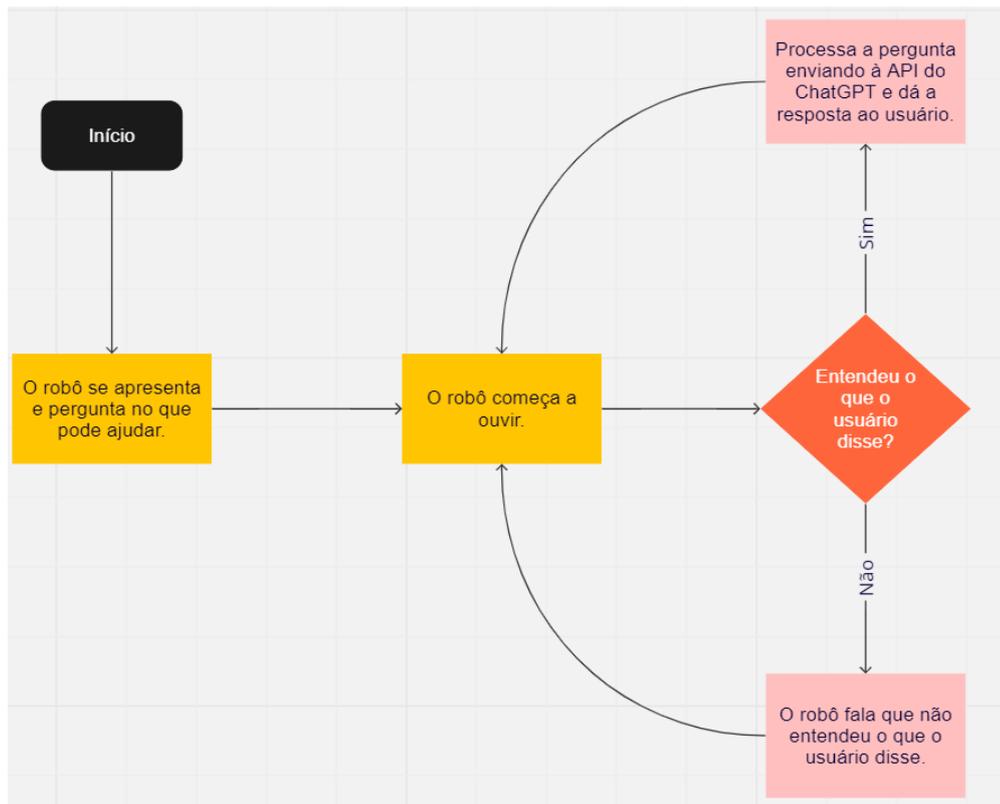
4.2 Módulos de Conversação OpenTera++

Desenvolvido em Flutter, o aplicativo construído neste trabalho foi projetado para ser um robô social inteligente onde o usuário pode interagir com ele e fazer perguntas (por meio de voz) dos mais variados temas que são processadas e respondidas pelo robô por meio do ChatGPT. É como ter um parceiro de conversa sempre disponível para sanar suas curiosidades e ajudá-lo com informações.

Quando o aplicativo é iniciado o robô dá boas-vindas e pergunta como pode ajudar. Logo em seguida ele começa a ouvir por meio do microfone do dispositivo, caso ele entenda o que o usuário tenha dito ele enviará um requisição à API do ChatGPT e aguardará a resposta. Com

a resposta pronta, ele retorna ao usuário e por meio de fala responde o questionamento feito anteriormente. Caso não entenda o que o usuário disse ou se nada for dito o robô irá falar que não entendeu e questiona se a pergunta pode ser repetida. Para exemplificar, o funcionamento é exibido no fluxograma da Figura 4.2.

Figura 4.2 – Fluxograma do funcionamento do aplicativo.



Para o módulo de conversação do OpenTera++ foram desenvolvidos quatro métodos que foram denominados `_startInteraction()`, `_listen()`, `_play()` e `getChatGptResponse()`. Eles serão explicados e mostrados a seguir.

Sendo um dos métodos mais importantes, o `_startInteraction()` recebe como parâmetro o status atual da interação. Ele é utilizado para controlar os tipos de interação do robô com o usuário que podem ser: `start`, `error` e `repeat`. O método é chamado com status `start` quando o aplicativo inicia, e basicamente ajusta o texto de resposta para dar as boas vindas e transmite isso por meio de fala ao usuário chamando o método `speak` da biblioteca `flutter_tts`. O status `error` faz a mesma coisa que o status `start`, porém o texto de resposta ao usuário muda. E por fim, o status `repeat` é utilizado para quando a resposta da API é retornada e já foi transmitida ao usuário. Todos os três status após cumprirem sua função chamam o método `_listen()` que logo será explicado. O código do método `_startInteraction()` é exibido a seguir.

```

1 Future<void> _startInteraction(String status) async {
2     if (status == 'start') {
3         _text = 'Ola. Como posso te ajudar?';
4     } else if (status == 'error') {
5         _text = 'Nao entendi o que voce disse, poderia repetir?';
6         _interaction = 0;
7     }
8
9     if ((status == 'start' || status == 'error') && _interaction ==
10    0) {
11         setState(() {
12             _isListening = false;
13             _isSpeaking = true;
14         });
15
16         await Future.delayed(const Duration(seconds: 1));
17         await flutterTts.setLanguage('pt-BR');
18         await flutterTts.speak(_text);
19
20         flutterTts.setCompletionHandler(() {
21             setState(() {
22                 _isSpeaking = false;
23             });
24             _listen();
25         });
26     } else if (status == 'repeat') {
27         _isListening = false;
28         await Future.delayed(const Duration(seconds: 1));
29         _listen();
30     }
31 }

```

O método `_listen()` tem o objetivo principal de ouvir, através do microfone do dispositivo, e transcrever tudo o que foi dito para texto. Ele possui duas saídas principais. Se o usuário não disse nada ou disse algo que não pôde ser compreendido, o método `_startInteraction()` é

chamado passando como parâmetro o status *error*, que foi explicado anteriormente. Caso contrário, o método *listen()* da biblioteca *speech_to_text*, introduzida anteriormente, é chamado para transcrever as palavras ditas pelo usuário em texto. Quando o usuário terminar de falar, o método *_play()* é chamado. O método *_listen()* criado é mostrado a seguir.

```
1 void _listen() async {
2     if (!_isListening) {
3         bool available = await _speech.initialize(
4             onStatus: (val) => print('onStatus: $val'),
5             onError: (val) {
6                 if (val.errorMsg == 'error_speech_timeout' ||
7                     val.errorMsg == 'error_no_match') {
8                     _startInteraction('error');
9                 }
10            },
11        );
12        if (available) {
13            setState(() => _isListening = true);
14            _speech.listen(
15                onResult: (val) async {
16                    setState(() {
17                        _text = val.recognizedWords;
18                    });
19                    if (val.finalResult) {
20                        _play();
21                    }
22                },
23            );
24        }
25    } else {
26        setState(() => _isListening = false);
27        _speech.stop();
28    }
29 }
```

No método `_play()` o principal objetivo é pegar o texto que foi transcrito a partir da fala no método anterior e passá-lo como parâmetro para o método `_getChatGptResponse()`, que é explicado na sequência, e que processa a requisição à API do ChatGPT. O método `_play()` então aguarda o retorno do `_getChatGptResponse()` que retorna a resposta da API e chama o método `speak` da biblioteca `flutter_tts` para responder ao usuário por meio de fala. Quando todo o método `speak` for finalizado, o método `_startInteraction()` será chamado recebendo como parâmetro o status `repeat`, iniciando o fluxo novamente conforme pode ser visto na Figura 4.2. O código é mostrado na sequência.

```

1 void _play() async {
2   _voice = await getChatGptResponse(_text);
3
4   setState(() {
5     _isListening = false;
6     _isSpeaking = true;
7   });
8
9   await flutterTts.setLanguage('pt-BR');
10  List<int> latin1Bytes = latin1.encode(_voice);
11  String textUtf8 = utf8.decode(latin1Bytes);
12  await flutterTts.speak(textUtf8);
13
14  flutterTts.setCompletionHandler(() {
15    setState(() async {
16      _isSpeaking = false;
17      _isListening = true;
18      _interaction++;
19      _startInteraction('repeat');
20    });
21  });
22 }

```

Por último, o método `_getChatGptResponse()` é responsável por fazer uma requisição utilizando o protocolo de comunicação web HTTP e o método POST por meio da biblioteca `http` do Flutter para a API do ChatGPT. Ele consiste em basicamente obter o texto por parâmetro

e realizar a requisição. Se a resposta da API for bem sucedida o método retorna o texto gerado na resposta, caso contrário uma exceção é lançada. Nota-se que no objeto *headers* o atributo *Authorization* contém a *API key*, obtida em seção anterior, após a palavra *Bearer*. Vale ressaltar ainda, que na variável *response*, o método *parse* recebe como parâmetro a URL da API também obtida em seção anterior, conforme mostrado abaixo no código do método.

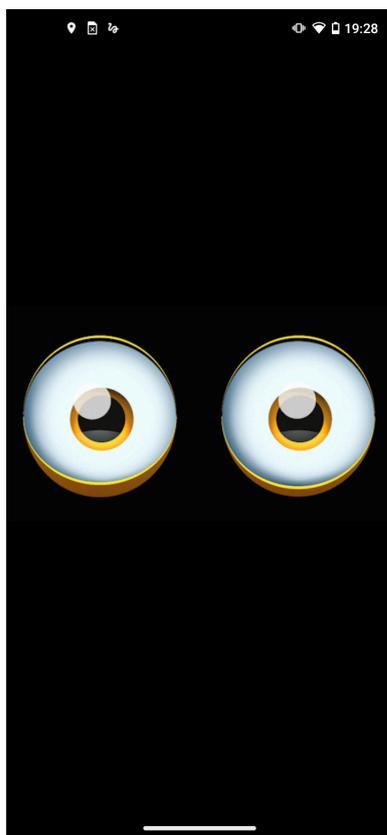
```
1 Future<String> getChatGptResponse(String message) async {
2     final headers = {
3         'Content-Type': 'application/json',
4         'Authorization':
5             'Bearer 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
6     },
7
8     final data = {
9         'model': 'gpt-3.5-turbo',
10        'messages': [
11            {"role": "user", "content": message}
12        ],
13        'max_tokens': 50,
14    };
15    try {
16        final response = await http.post(Uri.parse(apiUrl),
17            headers: headers, body: jsonEncode(data));
18        if (response.statusCode == 200) {
19            final jsonResponse = jsonDecode(response.body);
20            final chatGptResponse = jsonResponse['choices'][0]['message']
21            ['content'];
22            return chatGptResponse;
23        } else {
24            throw Exception('Failed to get ChatGPT response');
25        }
26    } catch (e) {
27        return "Nao pode ser processado";
28    }
```

4.3 Módulo de emoção do OpenTera++

O módulo de emoção do OpenTera++ completa o módulo de conversação com a exibição de *gifs* animados para transmitir as emoções do robô social, tornando a experiência mais envolvente. Quando o robô estiver ouvindo a pergunta, o usuário verá um *gif* com dois olhos atentos, transmitindo a sensação de estar sendo ouvido com atenção. E, enquanto o robô estiver processando a resposta, um *gif* de uma boca falando será exibido, para representar a resposta da pergunta feita.

No momento em que o robô está ouvindo um *gif* animado de dois olhos é exibido, transmitindo a sensação ao usuário de que ele está prestando atenção no que está sendo dito. A tela do aplicativo nesse momento fica como ilustrado na Figura 4.3.

Figura 4.3 – Aplicativo exibindo *gif* enquanto ouve



No momento em que o robô começa a falar, seja dando as boas-vindas ou respondendo ao que o usuário perguntou, um *gif* animado de uma boca falando é exibido na tela, para transmitir que o robô está falando algo, como pode ser visto na Figura 4.4.

Figura 4.4 – Aplicativo exibindo *gif* enquanto fala

Para exibição e manipulação dos *gifs* animados, enquanto o aplicativo ouve fala, foram utilizadas duas variáveis booleanas nomeadas *_isListening* e *_isSpeaking*, respectivamente. Elas são controladas e tem seus valores alterados dentro dos métodos *_startInteraction()*, *_listen()* e *_play()* de acordo com o contexto de cada método. Quando a variável *_isListening* for igual a *true*(verdadeiro) o aplicativo estará ouvindo e será exibido o *gif* dos olhos. E quando *_isSpeaking* for *true*(verdadeiro) o aplicativo estará falando sendo exibido o *gif* da boca. As duas variáveis nunca coincidem de estarem como *true*(verdadeiro) ao mesmo tempo.

O código do *widget* responsável pela interface de usuário do aplicativo é mostrado a seguir.

```

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 }
```

Nota-se que dentro de *children* na linha 11 do código existem dois widgets *Visibility* que exibe os *gifs* de acordo com as variáveis *_isListening* e *_isSpeaking*. Isso é controlado pelo atributo *visible* deixando que o *Visibility* seja mostrado ou não na tela.

4.4 Realização de testes

Para a realização de testes no aplicativo, seguimos os passos de Sejnowski (2023) para realizar teste de Turing com a aplicação recém desenvolvida. Primeiramente foram conduzidos testes funcionais para verificar se todas as funcionalidades do aplicativo estão operando como esperado. Isso envolveu testar o reconhecimento de fala para garantir que as perguntas do usuário fossem interpretadas e transcritas para texto corretamente, e se o sistema pôde com-

preender a entrada em diferentes situações e contextos. Um exemplo desses testes foi gravado e disponibilizado em <https://youtu.be/F_soHSmHa7c>.

O exemplo gravado mostra a interação do autor do trabalho com o aplicativo desenvolvido. Com base nesses testes pode-se aprimorar as funcionalidades do aplicativo, fazendo o refinamento até chegar em uma versão final satisfatória. Conforme testamos, identificamos que o aplicativo ainda possui algumas limitações em seu funcionamento. Uma delas é que seu modo de escuta é contínuo, ou seja, sempre depois de responder o usuário ele começa a ouvir novamente e tentar transcrever a fala em texto, independente do que aconteça. O prompt a seguir demonstra uma dessas interações do autor do trabalho com a aplicação desenvolvida. Nessa interação observa-se que logo após cada resposta o robô social começa a ouvir e tenta transcrever a fala em texto sempre dando uma resposta ao usuário, mesmo que nada tenha sido dito.

Robô: Olá. Como posso te ajudar?

Prompt: Olá, boa tarde.

Robô: Olá! Boa tarde! Como posso ajudar você hoje?

Prompt: Quem foi o primeiro ser humano a pisar na Lua?

Robô: O primeiro ser humano a pisar na Lua foi Neil Armstrong, um astronauta da NASA (Administração Nacional da Aeronáutica e Espaço dos Estados Unidos). Ele fez esse histórico feito em 20 de julho de 1969, durante a missão Apollo 11. Ao descer do módulo lunar chamado "Eagle", Neil Armstrong pronunciou a famosa frase: "Um pequeno passo para o homem, um salto gigante para a humanidade".

Prompt: ...

Robô: Não entendi o que você disse, poderia repetir?

Prompt: ...

Robô: Não entendi o que você disse, poderia repetir?

Prompt: Muito obrigado.

Robô: De nada! Se tiver mais alguma pergunta ou precisar de ajuda, sinta-se à vontade para perguntar. Tenha um ótimo dia!

...

O ideal seria que ele respondesse e parasse de tentar transcrever e só voltasse a ouvir com a transcrição de fala em texto ativada caso o usuário dissesse alguma palavra-chave que ativasse esse modo, por exemplo. Isso é um comportamento comum entre os assistentes virtuais comandados por voz atualmente, e seria um comportamento padrão esperado para esse tipo de robô social.

De acordo com Panchea et al. (2022), a camada do servidor NGINX é um *proxy* reverso para controlar e proteger o acesso à rede onde os módulos *backends* estão, sendo a porta de entrada para o *frontend* lidando com a criptografia de dados e certificados TLS. Ela tem por objetivo de encaminhar as solicitações HTTPS para o serviço de *backend* correto. Essa camada não foi implementada neste trabalho por não ter sido testada, e fica como sugestão para trabalhos futuros. Assim, a comunicação do aplicativo(*frontend*) passaria por esse servidor antes de chegar efetivamente na API do ChatGPT(*backend*) proporcionando ainda mais segurança na comunicação da aplicação.

5 CONCLUSÃO

A arquitetura de microserviços OpenTera atualmente não possui um módulo que facilite o desenvolvimento de robótica social assistiva (SARs) integrada a uma inteligência artificial generativa. O *framework* também não contempla módulos de conversação entre humano-robô, como reconhecimento e produção de fala. Perante o exposto, este estudo buscou abordar adaptações ao *framework*, com o intuito de incorporar os benefícios de utilizar uma IA generativa, neste caso o ChatGPT, no desenvolvimento de SARs.

Ao longo do estudo e desenvolvimento do presente trabalho construiu-se um novo módulo que pode ser incorporado à arquitetura do OpenTera, ao qual denominamos OpenTera++ (vide Figura 4.1). Seguindo o padrão do *framework*, o módulo proposto possui um componente no *backend*, uma camada intermediária e outro componente no *frontend*. O *backend* é composto pela API do ChatGPT, da OpenAI. A camada intermediária é responsável pela comunicação entre as partes por meio do transporte seguro HTTPS. E o *frontend* é construído a partir de um aplicativo móvel Flutter, utilizando bibliotecas para reconhecimento e produção de fala, *speech-to-text* e *flutter_tts* respectivamente, além de métodos desenvolvidos para que o funcionamento se fizesse possível.

Neste trabalho, foi desenvolvido um protótipo para avaliar as possibilidades de integração da API da OpenAI com o OpenTera. O código-fonte do aplicativo pode ser acessado em: <<https://github.com/rafout/opentera-pp>>. Os cinco objetivos específicos propostos foram atingidos com a realização deste estudo. No entanto, também foram identificadas algumas limitações nos resultados obtidos. O aplicativo desenvolvido apresenta algumas restrições, como por exemplo, ele transcreve de forma contínua o áudio captado pelo microfone, sem considerar o contexto da interação com o usuário. O módulo desenvolvido para o OpenTera não utiliza a camada de *proxy* reverso com servidor NGINX original do *framework*, pois não foi testado. Para trabalhos futuros, sugere-se abordar esses dois aspectos que se configuram como limitações.

No aplicativo, uma sugestão seria implementar uma palavra-chave, por exemplo, quando o usuário a proferisse, o aplicativo ativaria o modo escuta e transcreveria as palavras ditas, sendo interrompido após o usuário parar de falar. Por parte do módulo do *framework* a proposta seria a implementação de um servidor NGINX que intermediaria a comunicação entre *frontend* e *backend* encaminhando as requisições feitas pelo *frontend* ao serviço correto do *backend*, isso proporcionaria uma camada de segurança a mais no módulo apresentado.

REFERÊNCIAS

- ALDARMAKI, H. et al. Unsupervised automatic speech recognition: A review. **Speech Communication**, v. 139, p. 76–91, 2022. ISSN 0167-6393. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0167639322000292>>.
- BARTNECK, C. et al. **Human-robot interaction: An introduction**. [S.l.]: Cambridge University Press, 2020.
- BREGANT, C. **GitHub - cristianbregant/azure_speech_recognition: Implementation of Azure Cognitive Speech Recognition for Flutter** — **github.com**. 2020. <https://github.com/cristianbregant/azure_speech_recognition>. [Accessed 20-Jun-2023].
- CHAVES, A. P.; GEROSA, M. A. How should my chatbot interact? a survey on social characteristics in human–chatbot interaction design. **International Journal of Human–Computer Interaction**, Taylor Francis, v. 37, n. 8, p. 729–758, 2021. Disponível em: <<https://doi.org/10.1080/10447318.2020.1841438>>.
- CHUWY’S. **Chuwy’s GIF**. — **giphy.com**. 2019. <<https://giphy.com/gifs/XAyRXqH8LZwtyI8BU7>>. [Accessed 23-Jun-2023].
- COLLINS, S. H. E. **Try Bard and share your feedback**. — **blog.google**. 2023. <<https://blog.google/technology/ai/try-bard/>>. [Accessed 28-Jun-2023].
- DART. **GitHub - http: A composable, Future-based library for making HTTP requests**. 2018. <<https://github.com/dart-lang/http>>. [Accessed 27-Jun-2023].
- FEIL-SEIFER, D.; MATARIC, M. J. Defining socially assistive robotics. In: IEEE. **Proceedings of the 2005 IEEE 9th International Conference on Rehabilitation Robotics**. [S.l.], 2005. p. 465–468.
- FEL. **GitHub - ghuyfel/flutter_azure_tts: Flutter implementation of Microsoft Azure Cognitive Text-To-Speech API**. — **github.com**. 2021. <https://github.com/ghuyfel/flutter_azure_tts>. [Accessed 21-Jun-2023].
- FLUTTER. **Flutter Install** — **docs.flutter.dev**. 2023. <<https://docs.flutter.dev/get-started/install/windows>>. [Accessed 28-Jun-2023].
- GOOGLE. **The official package repository for Dart and Flutter apps**. — **pub.dev**. 2017. <<https://pub.dev/>>. [Accessed 27-Jun-2023].
- GOOGLE. **Download Android Studio** — **developer.android.com**. 2023. <<https://developer.android.com/studio>>. [Accessed 28-Jun-2023].
- GOOGLE. **Preços do Speech-to-Text**. — **cloud.google.com**. 2023. <<https://cloud.google.com/speech-to-text/pricing?hl=pt-br>>. [Accessed 21-Jun-2023].
- HETLER, A. **What is ChatGPT?** — **techtarget.com/whatis/definition/ChatGPT**. 2023. <<https://www.techtarget.com/whatis/definition/ChatGPT>>. [Accessed 26-Jun-2023].
- IBM. **What is speech recognition?** — **ibm.com/topics/speech-recognition**. 2023. <<https://www.ibm.com/topics/speech-recognition>>. [Accessed 26-Jun-2023].

- JUNGHANS, F. **GitHub - felixjunghans/google_speech: Google Speech API for Flutter** — **github.com**. 2021. <https://github.com/felixjunghans/google_speech>. [Accessed 20-Jun-2023].
- LUTTON, D. **GitHub - dlutton/flutter_tts: Flutter Text to Speech package.** — **github.com**. 2018. <https://github.com/dlutton/flutter_tts>. [Accessed 21-Jun-2023].
- MICROSOFT. **Download Visual Studio Code** — **code.visualstudio.com**. 2023. <<https://code.visualstudio.com/download>>. [Accessed 28-Jun-2023].
- MICROSOFT. **Seu copiloto habilitado por IA para a web.** — **microsoft.com**. 2023. <<https://www.microsoft.com/pt-br/bing?form=MA13FJ>>. [Accessed 28-Jun-2023].
- MICROSOFT. **Speech Services pricing.** — **azure.microsoft.com**. 2023. <<https://azure.microsoft.com/en-us/pricing/details/cognitive-services/speech-services/>>. [Accessed 21-Jun-2023].
- MIDJOURNEY. **Midjourney** — **openai.com**. 2023. <<https://www.midjourney.com/>>. [Accessed 28-Jun-2023].
- NATALIE. **What is ChatGPT?** — **openai.com/en/articles**. 2023. <<https://help.openai.com/en/articles/6783457-what-is-chatgpt>>. [Accessed 27-Jun-2023].
- NING, Y. et al. A review of deep learning based speech synthesis. **Applied Sciences**, v. 9, n. 19, 2019. ISSN 2076-3417. Disponível em: <<https://www.mdpi.com/2076-3417/9/19/4050>>.
- OPENAI. **API Reference** — **openai.com/docs/api-reference**. 2023. <<https://platform.openai.com/docs/api-reference>>. [Accessed 28-Jun-2023].
- OPENAI. **DALL-E 2.** — **openai.com**. 2023. <<https://openai.com/dall-e-2>>. [Accessed 28-Jun-2023].
- OPENAI. **Documentation ChatGPT** — **openai.com/docs/introduction/overview**. 2023. <<https://platform.openai.com/docs/introduction/overview>>. [Accessed 27-Jun-2023].
- PANCHEA, A. M. et al. Opentera: A microservice architecture solution for rapid prototyping of robotic solutions to covid-19 challenges in care facilities. **Health and Technology**, Springer Science and Business Media Deutschland GmbH, v. 12, p. 583–596, 3 2022. ISSN 21907196.
- REN, Y. et al. Fastspeech: Fast, robust and controllable text to speech. **Advances in neural information processing systems**, v. 32, 2019.
- ROBOTICS, S. **Shape Robotics GIF.** — **giphy.com**. 2021. <<https://giphy.com/gifs/oNvolApeRjCKIfIjcb>>. [Accessed 23-Jun-2023].
- SEJNOWSKI, T. J. Large language models and the reverse turing test. **Neural computation**, MIT Press, v. 35, n. 3, p. 309–342, 2023.
- SOFTWARE, C. **GitHub - csdcorp/speech_to_text: A Flutter plugin that exposes device specific text to speech recognition capability.** — **github.com**. 2021. <https://github.com/csdcorp/speech_to_text>. [Accessed 20-Jun-2023].

YANG, Z.; ZHANG, Y.; LUO, J. Human-centered emotion recognition in animated gifs. In: **2019 IEEE International Conference on Multimedia and Expo (ICME)**. [S.l.: s.n.], 2019. p. 1090–1095.

ZHANG, C. et al. A complete survey on generative ai (aigc): Is chatgpt from gpt-4 to gpt-5 all you need? **arXiv preprint arXiv:2303.11717**, 2023.