South American Development Society Journal

Vol.: 10 | Nº.: 30 | **Ano:** 2024 | **ISSN:** 2446-5763 | **DOI:** 10.24325/issn.2446-5763.v10i30p196-228
Data de Submissão: 19/10/2024 | **Data de Publicação:** 22/12/2024

INTEGRAÇÃO DO CHAT GPT A DISPOSITIVOS IOT PARA TRAZER UMA MELHOR EXPERIÊNCIA DE USABILIDADE

Nathan Bitu de Oliveira
Faculdade Engenheiro Salvador Arena
081200004@faculdade.cefsa.edu.br

Gustavo Correia Scarabelli
Faculdade Engenheiro Salvador Arena
082210030@faculdade.cefsa.edu.br

Victor Inácio de Oliveira
Faculdade Engenheiro Salvador Arena
victorif@hotmail.com

Resumo

O projeto aborda a integração de dispositivos IoT com inteligência artificial generativa, utilizando o microcontrolador ESP32. O objetivo principal é aprimorar a interação dos usuários com esses dispositivos, permitindo o acesso a funcionalidades por meio de comandos de voz ou texto de forma natural e acessível. Inicialmente, foi realizada uma revisão bibliográfica para compreender melhor o estado da arte do IoT e das tecnologias de IA, como o Chat GPT. O projeto propõe a criação de um sistema que utiliza o sensor de umidade e temperatura DHT11 para coletar dados e, com o auxílio do Chat GPT, permitir que o usuário faça perguntas diretas, como "Em que dia da semana a temperatura esteve acima da média?". Os resultados preliminares com a simulação de um chatbot em Python, baseado no modelo de linguagem MariTalk da Maritaca AI, indicam que essa integração facilita a usabilidade e inclui pessoas com diferentes níveis de conhecimento. Apesar dos desafios técnicos, o projeto se mostra promissor por explorar novas possibilidades na interação entre IoT e IA.

Palavras-chave: Internet das Coisas (IoT). Inteligência Artificial (IA). Análise de Dados. Usabilidade.

Abstract

The project addresses the integration of IoT devices with generative artificial intelligence using the ESP32 microcontroller. The main goal is to enhance user interaction with these devices, allowing access to functionalities through voice or text commands in a natural and accessible way. Initially, a literature review was conducted to better understand the state of the art in IoT and AI technologies, such as Chat GPT. The project proposes the creation of a system that uses a DHT11 humidity and temperature sensor to collect data, and with the help of Chat GPT, enable users to ask direct questions, such as "On which day of the week was the temperature above average?". Preliminary results from a Python-based chatbot simulation, using the MariTalk language model from Maritaca AI, indicate that this integration improves usability and includes people with different levels of expertise. Despite technical challenges, the project appears promising as it explores new possibilities in the interaction between IoT and AI.

Keywords: Internet of Things (IoT). Artificial Intelligence (AI). Data Analysis. Usability.

1 Introdução

O ESP32, criado pela Espressif Systems, é uma placa de desenvolvimento "System on a Chip" (SoC), similar ao Arduino UNO. Destinada principalmente para aplicações em IoT (Internet das Coisas), a placa oferece conectividade WiFi (2,4 GHz) e Bluetooth, destacando-se por seu baixo custo, baixo consumo de energia, dois núcleos de microprocessador, múltiplas entradas e saídas, além de ser compatível com a linguagem de programação do Arduino. Essas características fazem do ESP32 uma opção acessível e poderosa para projetos complexos.

A inteligência artificial (IA) permite que computadores realizem tarefas tipicamente associadas a seres inteligentes, como manter conversas, reconhecer padrões e executar ações, sem a necessidade de programação específica. Existem diferentes formas de aprendizado em IA, como o aprendizado supervisionado, não supervisionado, por reforço e auto supervisionado, cada uma com métodos distintos de

processamento de dados. Ao ser integrada à IoT, a IA proporciona sistemas mais acessíveis e interativos, com grande potencial para automação.

A IoT, relacionada à Indústria 4.0 e à automação industrial, refere-se a uma rede de dispositivos conectados que se comunicam entre si e com a nuvem para transmitir e receber dados. Sensores e softwares facilitam essa interação, permitindo o monitoramento e controle de dispositivos como lâmpadas, cafeteiras, portões e máquinas industriais. Com a crescente geração de dados e o avanço nas telecomunicações, a IoT se expandiu para diversas áreas, sendo projetada para gerar mais de 73 trilhões de gigabytes de dados globalmente até 2025.

Neste contexto, o projeto explora a integração da IA com a IoT, utilizando o ESP32 para coletar dados de sensores e o modelo de linguagem MariTalk da Maritaca AI para desenvolver um chatbot inteligente. Esse chatbot permite interações por voz, fornecendo informações relevantes sobre o ambiente monitorado. A integração da IA com IoT abre novas possibilidades para aplicações em automação residencial, industrial e outras áreas, tornando os sistemas mais acessíveis e interativos.

2 Referencial teórico

Este tópico tem como objetivo apresentar as ferramentas, conceitos e recursos utilizados na construção do projeto, entre eles estão os conceitos de IoT, Inteligência Artificial e Chatbots.

2.1 ESP 32

Criada pela empresa, *Espressif Systems*, o ESP32 é uma placa de desenvolvimento, assim como o Arduino UNO, do tipo “SoC”, *System on a Chip* ou Sistema em um Chip (ESPRESSIF, 2024).

Figura 1 – ESP32

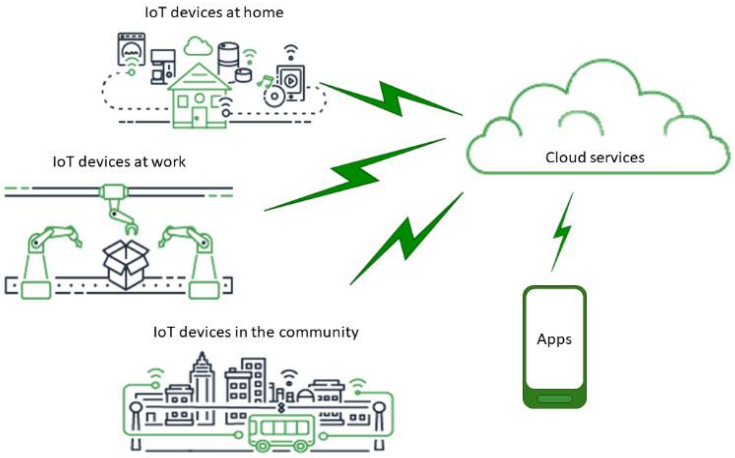
Fonte: (ESPRESSIF, 2024)

Sua criação foi focada especialmente para ser usada no desenvolvimento de sistemas com IoT, internet das coisas, isso porque ela possui conectividade com o WiFi (2,4 GHz) e o Bluetooth. Além disso, a placa se destacou por possuir um preço acessível de até 6 dólares, consumir pouquíssima energia, possuir dois núcleos em seus microprocessadores, possuir várias entradas e saídas em sua placa e ser compatível com a linguagem de programação do Arduino, ou seja, ela é acessível e ainda assim muito poderosa, capaz de ser a protagonista em circuitos muito robustos e complexos (RANDOM NERD TUTORIALS, 2024).

2.2 IoT

Muito associada às automações de processos industriais e sinônimo da Indústria 4.0, IoT, *Internet of Things* ou ainda, Internet das Coisas (THE INTERNET OF THINGS WITH ESP32, 2024), se refere a uma rede coletiva onde diversos dispositivos são conectados à internet e podem interagir entre si e com a nuvem ao receber e transmitir dados, e a tecnologia empregada para essa comunicação funcionar, como softwares e/ou sensores.

Figura 2 – Gráfico para Explicação do IoT



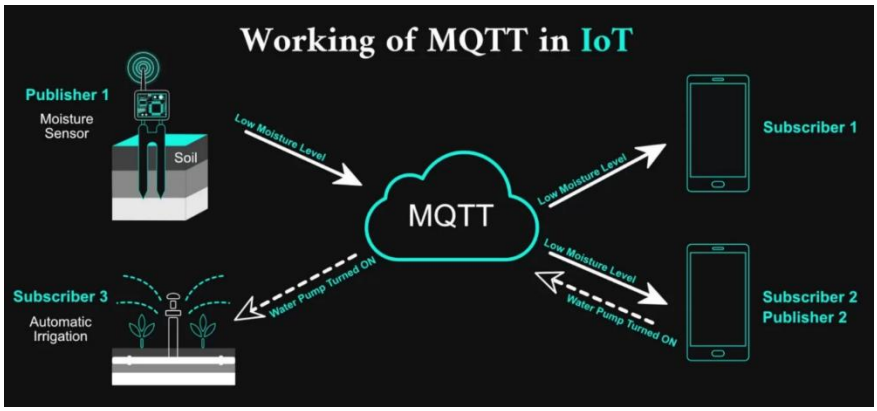
Fonte: (AWS, 2024)

Com a quantidade de dados gerados aumentando de forma exponencial, surgiu a necessidade de automatizar os processos e assim, monitorá-los de uma forma mais eficiente, somados ao advento de chips de computador baratos, menores, mais rápidos e mais inteligentes, e telecomunicações de alta largura de banda vários dispositivos foram sendo conectados à internet e assim o IoT surgiu. É importante destacar que o custo para integrar a capacidade computacional a pequenos objetos diminuiu consideravelmente ao longo do tempo. Ademais, até 2025, a IDC, *International Data Corporation*, prevê uma geração de dados global maior que 73 trilhões de gigabytes, como exemplo de dispositivos que podem ser conectados tem-se lâmpadas, cafeteiras, portões, máquinas industriais, televisores etc. (SAP, 2024)

2.3 Protocolo MQTT

Desenvolvido pela IBM (GUPTA, 2021) no final dos anos 90, MQTT, (*Message Queuing Telemetry Transport*) é um protocolo usado principalmente para conectar dispositivos entre si na internet através de serviços de *backend* e, diferente do protocolo HTTP, suporta uma comunicação assíncrona entre duas partes.

Figura 3 – Exemplo de Lógica de Funcionamento do MQTT



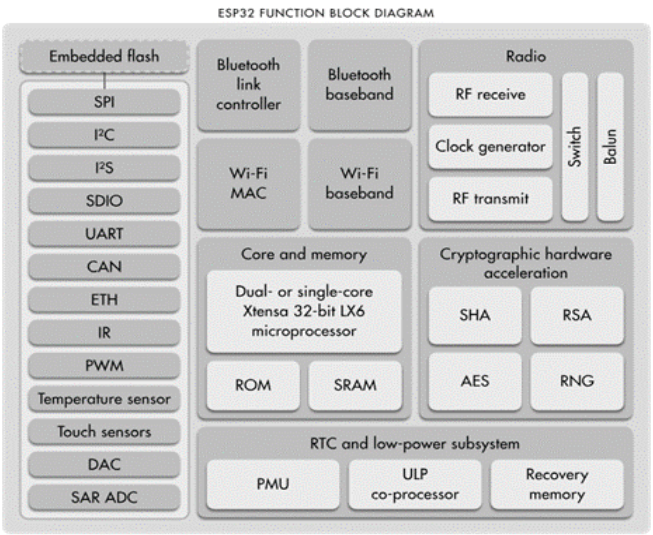
Fonte: (GUPTA, 2021)

A arquitetura do protocolo MQTT usa o modelo *publish-subscribe*, ou seja, o dispositivo que envia a mensagem e o dispositivo que a recebe não se comunicam diretamente, ao invés disso o *publisher* se conecta nesse intermediário, transmitindo a mensagem para os dispositivos que estão inscritos.

2.4 Sistemas com ESP32 e IoT

O ESP32 oferece um grande suporte para conectar um dispositivo à internet, seu microprocessador *Tensilica Xtensa* LX6 opera em uma frequência de 2,4 GHz, portanto, mais dispositivos podem interagir através de uma rede já existente, mas se não houver um WiFi, ele cria sua própria rede de conexão, permitindo uma conectividade ainda maior (VIEIRA FRANZINI, SILVA MAMEDE, & DE MELO, 2020).

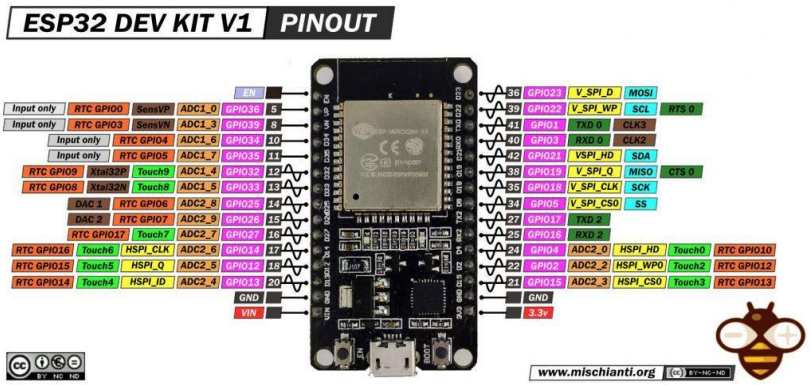
Figura 4 – Diagrama das funções de bloco do ESP32



Fonte: (THE INTERNET OF THINGS WITH ESP32, 2024)

Ademais, muitos dos seus pinos são capazes de fazer leituras analógicas e digitais, o que também possibilita a integração de dispositivos analógicos ao sistema, o que é crucial quando se pensa em automatizar algum processo residencial por exemplo (VIEIRA FRANZINI, SILVA MAMEDE, & DE MELO, 2020).

Figura 5 – Imagem dos tipos de pinos do ESP32



Fonte: (THE INTERNET OF THINGS WITH ESP32, 2024)

2.5 Inteligência Artificial

Inteligência artificial é a habilidade de um computador digital performar ações que normalmente são associadas a seres inteligentes, sem que eles sejam especificamente programados para tal. Exemplos de habilidades que podem ser desenvolvidas por esse tipo de modelo são: desenvolver conversas, reconhecer padrões e executar ações (COPELAND, 2024).

Existem diversas formas de aprendizado, exemplos são: Aprendizado supervisionado: O agente aprende com seus erros em comparação a rótulos de dados factuais. Aprendizado não-supervisionado: O agente encontra padrões matemáticos nos dados sem a necessidade de rótulos factuais. Aprendizado por reforço: O agente aprende com base em recompensas por suas ações em ambientes específicos. Aprendizado auto supervisionado: É uma forma de aprendizado não-supervisionado onde o agente cria seus próprios rótulos fazendo uma máscara dos dados factuais, tenta prever esses dados e, em seguida, compara suas respostas com os dados originalmente mascarados (SAH, 2020).

2.6 Aprendizagem Profunda

Seguindo a ideia de mímica do aprendizado humano (ROSENBLATT, 1958) desenvolveu o Perceptron, que tenta imitar a forma como o neurônio humano funciona, onde são recebidas entradas, que são processadas e resultam em uma saída. (RUMELHART, HINTON, & WILLIAMS, 1986) avançaram a pesquisa desenvolvendo o algoritmo de backpropagation, o que fez possível o treinamento efetivo do algoritmo de multi-layer perceptron, também conhecido como rede neural, pois tenta imitar a forma como o cérebro humano conecta neurônios entre si para desenvolver o pensamento e processamento de dados.

Algoritmos de redes neurais quando desenvolvidos com mais de duas camadas são comumente chamados de redes neurais profundas, esse tipo de algoritmo mostram os melhores resultados em dados não estruturados quando comparados a modelos de aprendizado de máquina mais simples. Isso muitas vezes é causado pelo baixo viés de modelos de aprendizagem profunda, que lhes permite aprender padrões sutis em dados.

2.7 Processamento de linguagem natural (PLN)

O estudo do processamento de linguagem natural tem como objetivo desenvolver modelos que consigam processar e entender a linguagem humana. Tipos de tarefas que podem ser desenvolvidas são: Classificação de texto: A partir de entendimento de linguagem separar dados em diferentes classes, Named-entity recognition: baseado no texto encontrar entidades dentro do texto, Tradução de texto e Geração de resumo de textos. (GUPTA, MAJUMBER, & VAJJALA, 2020)

Entendimento de linguagem é uma tarefa bastante complicada, tendo em vista que o sentido do texto pode mudar completamente baseado no contexto e na estrutura, o uso de técnicas como Bag of words, TF-IDF e Word2Vec tentam criar representações numéricas de texto que podem ser usadas com modelos de IA para executar tarefas de PLN. (GUPTA, MAJUMBER, & VAJJALA, 2020)

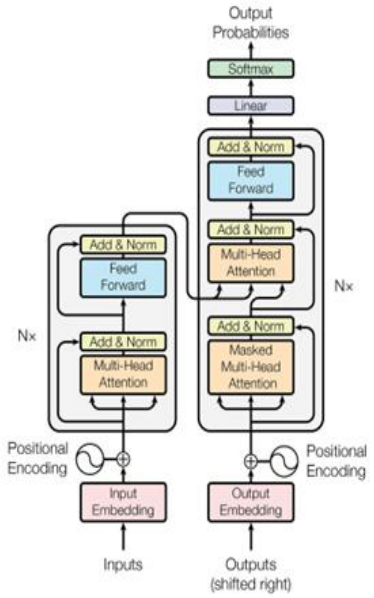
Até 2016, projetos de PLN normalmente levavam algumas das seguintes arquiteturas de rede neural: Redes neurais recorrentes: O valor de saída de um neurônio é alimentado de forma autoregressiva para ele mesmo, isso serve para dar uma ideia de sequência dos dados e de propagação de informação de forma autoregressiva, esses modelos carregam como principal problema o fenômeno de gradientes explodindo ou sumindo, o que dificultava o ajuste do modelo; Redes neurais LSTM: Utiliza de portões de memória de curto e longo prazo usados para definir a quantidade de informação propagada entre cada iteração, embora esses modelos consigam resolver o problema de gradientes explodindo ou sumindo, ele tem dificuldade em guardar informações por grandes períodos de tempo, fazendo com que informações sejam perdidas na sequência. (DIRAC, 2019)

2.8 Transformers

Para resolver os problemas citados anteriormente, foi desenvolvida a arquitetura transformer de redes neurais, essa arquitetura tem como objetivo criar representações de espaço latente (Embeddings) para a sequência de palavras, e então, utilizar os embeddings para prever a próxima palavra em uma sequência. A arquitetura conta com dois grandes diferenciais quando comparada a outras arquiteturas existentes: Mecanismo de atenção: Gera pesos para cada palavra na sequência quando inseridas no contexto. Positional Encoding: Gera códigos que representam a posição de cada palavra na sequência. (VASWANI, et al., 2017)

No artigo oficial a arquitetura é apresentada com dois grandes componentes, o Encoder: Gera representações numéricas de espaço latente para o texto, e o Decoder: Utiliza representações numéricas para fazer classificação textual de forma autorregressiva. (VASWANI, et al., 2017)

Figura 6 – Arquitetura Transformer



Fonte: (VASWANI, et al., 2017)

2.9 Large Language Models (LLMs)

Em (RADFORD, NARASIMHAN, SALIMANS, & SUTSKEVER, 2018) a OpenAI desenvolveu o Generative Pre-trained Transformer ou GPT, onde um modelo Transformer foi treinado de forma semi-supervisionada que consiste em duas partes: Pré-treino: O modelo foi treinado de forma massiva auto supervisionada em corpos textuais diversos e massivos; Fine-tuning: O modelo passa por um treinamento de forma supervisionada, onde os dados são curados e passam por rotulação humana. A primeira versão do GPT tinha 117 milhões de parâmetros treináveis e foi treinado gastando 0.96 petaflop-dias de poder computacional. Após o treinamento foi notado que o modelo tinha a habilidade de zero-shot reasoning onde o modelo demonstrou habilidade que superaram o estado da arte em diversas tasks como classificação, resposta a perguntas, avaliação de similaridade semântica, geração de resumos textuais, entre diversas outras tasks. Futuras versões deste modelo foram usadas para a criação do ChatGPT. (BROWN, et al., 2020)

2.10 In-context learning

In-context learning é uma técnica que permite que modelos de linguagem como o GPT-3 gerem respostas aprimoradas através de ajustes nos prompts enviados ao modelo. Conforme apresentado por (BROWN, et al., 2020), o in-context learning pode aumentar significativamente a assertividade desses modelos em diversas tarefas, incluindo classificação, tradução e geração de textos informados. Essa técnica pode ser implementada de várias formas: em uma abordagem few-shot, o modelo recebe múltiplos exemplos durante a inferência; em one-shot, apenas um exemplo é fornecido; e em zero-shot, nenhuma resposta exemplo é utilizada, confiando-se unicamente no conhecimento adquirido durante o treino. Além disso, o in-context learning permite a inserção dinâmica de informações durante a inferência, enriquecendo a capacidade de resposta do modelo.

2.11 Tool Augmented LLMs

Normalmente, os LLMs não conseguem gerar informações que ultrapassem a data de seu último treinamento, uma limitação devido ao modelo ser treinado até um determinado momento e depois mantido estático para inferência. Uma estratégia conhecida para mitigar esse problema, conforme discutido em (SCHICK, et al., 2023) envolve permitir que, durante a inferência, os modelos acessem ferramentas externas, como calculadoras, ferramentas de busca ou sensores, para enriquecer seu contexto e, consequentemente, fornecer respostas mais precisas e atualizadas aos usuários.

2.12 Conceito de *ChatBots*

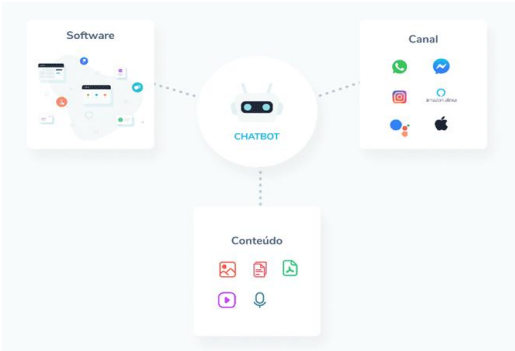
Um *chatbot* é desenvolvido com o objetivo de se comunicar com usuários, muitos deles possuem em seu desenvolvimento Inteligência Artificial e, somada ao *Machine Learning*, os *chatbots* chegaram a um outro nível, sendo capazes de interpretar a intenção e até mesmo o sentimento dos usuários para fornecer soluções assertivas para as mais diversas necessidades (O QUE É UM CHATBOT?, 2024). Hoje em dia, o uso dos *chatbots* se destaca no atendimento a clientes, e suas interfaces de

conversação podem ser por meio de aplicativos de mensagens, sites, e outras plataformas digitais (SALESFORCE, 2024).

2.13 Sistemas com *ChatBots*

Os *chatbots* são compostos por 3 elementos, um canal, meio por onde se realizará a interação com o *chatbot*; um conteúdo, os recursos utilizados como texto, imagens, emoji; e por fim, um software, o algoritmo onde o *chatbot* foi construído.

Figura 7 – Modelo de funcionamento de um *chatbot*



Fonte: (BLIPBLOG, 2021)

Os *chatbots* podem responder e interagir com os usuários 24 horas por dia, 7 dias por semana, sem limitações de tempo ou local físico. Isso os torna atraentes para empresas que podem não ter a capacidade ou os recursos para manter funcionários trabalhando o tempo todo. Os *chatbots* podem ser de três tipos, baseado em regras, que responde apenas com respostas pré-determinadas; com processamento de linguagem natural, que usa IA para aprender com as respostas dos usuários, e híbrido, que combina regras com o processamento de linguagem natural (ESPRESSIF, 2024; BLIPBLOG, 2021).

Figura 8 – Funcionamento de um *chatbot*

Fonte: (BLIPBLOG, 2021)

3 Metodologia

3.1 Inteligência Artificial e IoT

No mundo atual, a inteligência artificial (IA) é onipresente, presente em dispositivos eletrônicos de uso diário, como caixas eletrônicos, celulares e televisões. Os *chatbots* com IA, como o ChatGPT, têm ganhado destaque desde fevereiro de 2023, oferecendo uma nova forma de interação e assistência aos usuários. No entanto, muitas pessoas ainda não compreendem totalmente como esses *chatbots* funcionam e como podem ser utilizados de maneira eficaz.

A integração do ChatGPT a dispositivos IoT representa uma oportunidade de melhorar a usabilidade e a experiência do usuário. Essa integração permite que os *chatbots* sejam incorporados a uma variedade de dispositivos, como assistentes domésticos inteligentes e sistemas de automação residencial, facilitando o controle e acesso a funcionalidades por meio de comandos de voz ou texto.

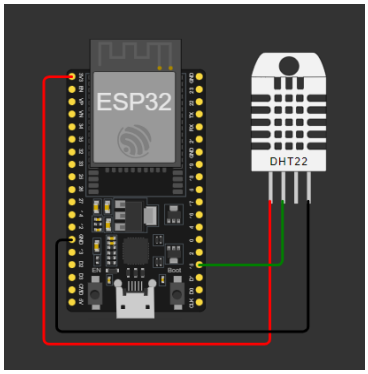
Para explorar essa integração, serão realizados experimentos práticos com o ESP32, nos quais os *chatbots* serão utilizados em cenários específicos. Serão avaliados aspectos como a precisão na compreensão das solicitações do usuário, a velocidade de resposta e a facilidade de interação, fornecendo insights valiosos para o desenvolvimento futuro dessa tecnologia.

Em resumo, a integração do ChatGPT a dispositivos IoT promete simplificar e otimizar as interações entre humanos e máquinas. Ao compreender melhor como essa integração funciona e explorar seu potencial, pode-se aproveitar os benefícios de uma experiência de usuário mais intuitiva e eficiente em um mundo cada vez mais conectado e automatizado.

3.2 Simulação do Protótipo

Antes de iniciar a construção do circuito físico, fez-se necessário a simulação do mesmo através do Software Wokwi, o circuito foi montado como mostra na figura 9:

Figura 9 – Circuito ESP32 + DHT22



Fonte: (WOKWI, 2024)

Nesta figura, pode-se identificar o microcontrolador ESP32, que será responsável por alimentar e se comunicar com todo o circuito e o DHT22, (*Digital Humidity and Temperature sensor*), que é o sensor responsável por medir a temperatura e a umidade de um determinado local. Para ligar o sensor a placa, liga-se seu pino de alimentação, (*VCC*), à entrada de alimentação do ESP32, (*3V3*), após isso o pino do sinal do DHT 22, (*SDA*), é conectado na entrada digital do ESP32, (*15*), por fim, liga-se o pino de aterramento do sensor com a entrada para aterramento do microcontrolador, (*GND*).

Para testar esse circuito foi criado um código que envia seus dados para o MQTT Broker através da plataforma HiveMQ que utiliza o protocolo MQTT para realizar a comunicação IoT dos componentes.

Figura 10 – Código de Teste parte 1

```
20  import network
21  import time
22  from machine import Pin
23  import dht
24  import ujson
25  from umqtt.simple import MQTTClient
26
27  # MQTT Server Parameters
28  MQTT_CLIENT_ID = "micropython-weather-demo"
29  MQTT_BROKER    = "broker.mqttdashboard.com"
30  MQTT_USER      = ""
31  MQTT_PASSWORD  = ""
32  MQTT_TOPIC     = "wokwi-weather"
33
34  sensor = dht.DHT22(Pin(15))
```

Fonte: (WOKWI, 2024)

O código se inicia importando todos os módulos que utilizaremos, seguido dos parâmetros com valores que será enviado ao MQTT, na linha 34, o sensor é criado através da leitura do sinal do pino 15 do ESP32.

Figura 11 – Código de Teste parte 2

```
36  print("Connecting to WiFi", end="")
37  sta_if = network.WLAN(network.STA_IF)
38  sta_if.active(True)
39  sta_if.connect('Wokwi-GUEST', '')
40  while not sta_if.isconnected():
41    print(".", end="")
42    time.sleep(0.1)
43  print(" Connected!")
44
45  print("Connecting to MQTT server... ", end="")
46  client = MQTTClient(MQTT_CLIENT_ID, MQTT_BROKER, user=MQTT_USER, password=MQTT_PASSWORD)
47  client.connect()
48
49  print("Connected!")
```

Fonte: (WOKWI, 2024)

A partir da linha 36 até a linha 43, é realizado a conexão do ESP32 com a rede Wi-Fi e a partir da linha 45 é realizada a conexão com o MQTT.

Figura 12 – Código de Teste parte 3

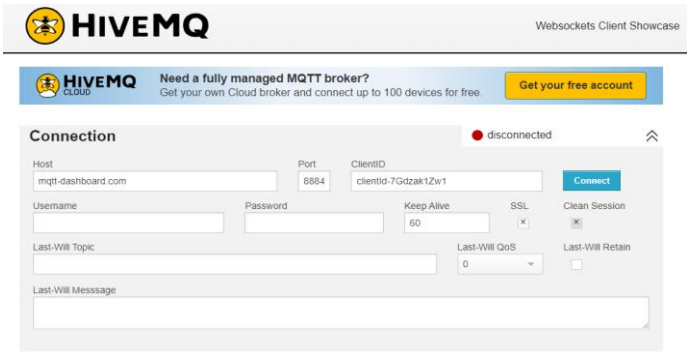
```
51  prev_weather = ""
52  while True:
53    print("Measuring weather conditions... ", end="")
54    sensor.measure()
55    message = ujson.dumps({
56      "temp": sensor.temperature(),
57      "humidity": sensor.humidity(),
58    })
59    if message != prev_weather:
60      print("Updated!")
61      print("Reporting to MQTT topic {}: {}".format(MQTT_TOPIC, message))
62      client.publish(MQTT_TOPIC, message)
63      prev_weather = message
64    else:
65      print("No change")
66    time.sleep(1)
```

Fonte: (WOKWI, 2024)

Nesta figura, é possível visualizar o loop principal do código, o qual se houver mudança no valor da temperatura ou da umidade detectada pelo sensor o código enviará os novos valores para o MQTT e irá mostrar na saída do console.

Agora, para visualizar os valores recebidos pelo MQTT Broker, o HiveMQ será responsável por informar as atualizações em tempo real. Para isso, primeiro será necessário entrar no site *https://www.hivemq.com/demos/websocket-client/* e se pressionar o botão "*Connect*", como mostrado na imagem a seguir.

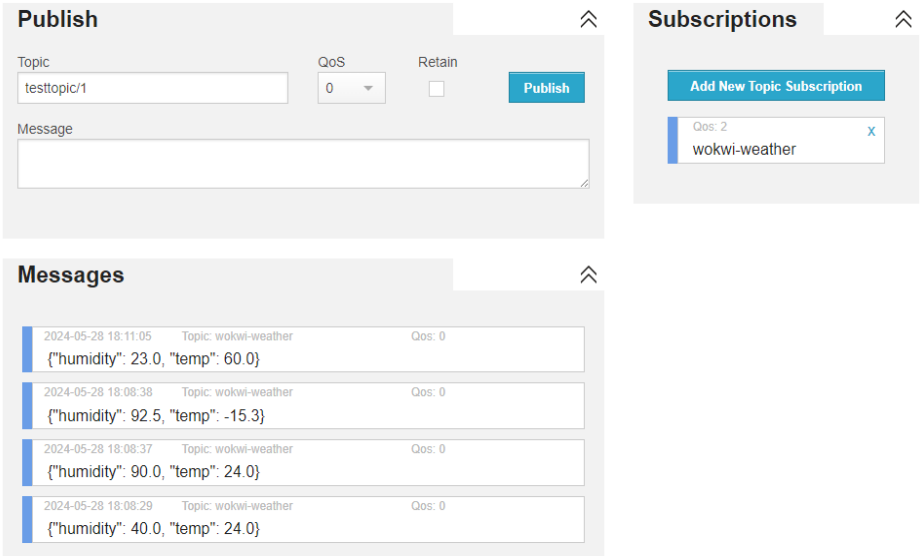
Figura 13 – Configuração do HiveMQ parte 1



Fonte: (HIVEMQ WEBSOCKET, 2024)

Após isso, deve-ser cadastrar um novo "*Topic*" em "*Subscription*" chamado "*wokwi-weather*" e então, com o circuito em execução, já será possível visualizar os valores sendo publicados no MQTT em tempo real.

Figura 14 – Configuração do HiveMQ parte 2

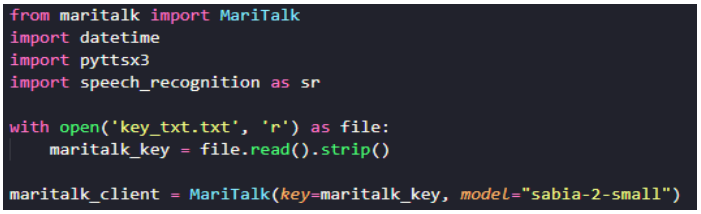


Fonte: (HIVEMQ WEBSOCKET, 2024)

3.3 Desenvolvimento Chatbot com IAG

O código apresentado implementa um chatbot inteligente, capaz de interagir com o usuário por voz, utilizando tecnologias de reconhecimento de fala, síntese de voz e um modelo de linguagem.

Figura 15 – Código do Chatbot inteligente em Python parte 1



```python
from maritalk import MariTalk
import datetime
import pyttsx3
import speech_recognition as sr

with open('key_txt.txt', 'r') as file:
    maritalk_key = file.read().strip()

maritalk_client = MariTalk(key=maritalk_key, model="sabia-2-small")
```

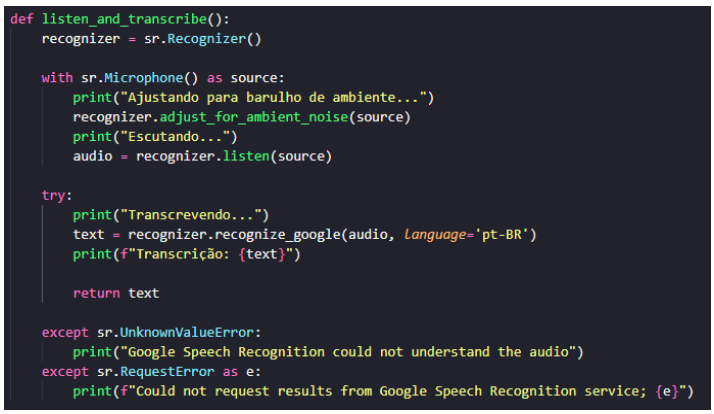
Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

A primeira etapa consiste em importar as bibliotecas necessárias para o funcionamento do chatbot, sendo elas:

- maritalk: Biblioteca que fornece acesso à API do MariTalk, o modelo de linguagem que dará inteligência ao chatbot.
- datetime: Usada para obter informações sobre a data e hora atuais.
- pyttsx3: Permite a conversão de texto em fala, ou seja, a voz do chatbot.
- speech_recognition: Responsável por reconhecer a fala do usuário e convertê-la em texto.

Em seguida, o código carrega a chave de API do MariTalk a partir de um arquivo de texto e inicializa o cliente do MariTalk, especificando o modelo de linguagem a ser utilizado (sabia-2-small).

Figura 16 – Código do Chatbot inteligente em Python parte 2



```python
def listen_and_transcribe():
    recognizer = sr.Recognizer()

    with sr.Microphone() as source:
        print("Ajustando para barulho de ambiente...")
        recognizer.adjust_for_ambient_noise(source)
        print("Escutando...")
        audio = recognizer.listen(source)

    try:
        print("Transcrevendo...")
        text = recognizer.recognize_google(audio, language='pt-BR')
        print(f"Transcrição: {text}")

        return text

    except sr.UnknownValueError:
        print("Google Speech Recognition could not understand the audio")
    except sr.RequestError as e:
        print(f"Could not request results from Google Speech Recognition service; {e}")
```

Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

A função *listen_and_transcribe* é responsável pela captura, ajuste de ruído e transcrição de fala do usuário. A função utiliza o Google Speech Recognition para transcrição de áudio.

Figura 17 – Código do Chatbot inteligente em Python parte 3

```python
def text_to_speech(text, lang='pt'):
    engine = pyttsx3.init()

    engine.setProperty('rate', 200)  # Speed percent
    engine.setProperty('volume', 1)  # Volume 0-1

    voices = engine.getProperty('voices')
    engine.setProperty('voice', voices[0].id)

    engine.say(text)

    engine.runAndWait()
```

Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

A função *texto_to_speech* recebe um texto e o converte em fala utilizando o mecanismo pyttsx3, configurando a velocidade e o volume da voz.

Figura 18 – Código do Chatbot inteligente em Python parte 4

```python
def update_sensor_data(sensor_data):
    context_message = {
        "role": "system",
        "content": f"Responda o usuário com um bom dia/tarde/noite baseado no horario atual, além disso sempre que o usuário" +
        f" quiser saber a temperatura ou a hora atual cheque as informações a seguir. Contexto de sensores: {sensor_data}" +
        f", não faça formatação no texto de resposta."
    }
    return context_message
```

Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

A função *update_sensor_data* atualiza a mensagem inicial de sistema com os dados do sensor, no momento os dados de sensor vêm do próprio código, porém no futuro virão dos sensores em campo.

Figura 19 – Código do Chatbot inteligente em Python parte 5

```python
def receive_message(history, message):
    history.append({"role": "user", "content": message})
    return history
```

Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

A função *receive_message* adiciona a mensagem do usuário ao histórico de chat

Figura 20 – Código do Chatbot inteligente em Python parte 6

```python
def generate_response(history):
    print_history(history)
    response = maritalk_client.generate(
        history,
        do_sample=True,
        max_tokens=200,
        temperature=0.7,
        top_p=0.95
    )["answer"]

    history.append({"role": "assistant", "content": response})
    return response, history
```

Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

A função *generate_response* envia o histórico de chat ao modelo e recebe a resposta.

Figura 21 – Código do Chatbot inteligente em Python parte 7

```python
def print_history(history):
    for entry in history:
        if entry["role"] == "system":
            print(f"System: {entry['content']}")
        elif entry["role"] == "user":
            print(f"You: {entry['content']}")
        elif entry["role"] == "assistant":
            print(f"Bot: {entry['content']}")
```

Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

A função *print_history* serve apenas para imprimir o histórico de conversa de forma estruturada.

Figura 22 – Código do Chatbot inteligente em Python parte 8

```python
def chatbot_loop():
    time = datetime.datetime.now().strftime("%H:%M")
    sensor_data = {'temperatura_quarto': 23, 'temperatura_sala': 24, 'horario_agora': time}

    history = [update_sensor_data(sensor_data)]

    while 1:
        print('')
        user_message = listen_and_transcribe()
        if 'ASSISTENTE' in user_message.upper():
            history = receive_message(history, user_message)
            response, history = generate_response(history)
            print(f"Bot: {response}")
            text_to_speech(response)

            time = datetime.datetime.now().strftime("%H:%M")
            sensor_data = {'temperatura_quarto': 24, 'temperatura_sala': 26, 'horario_agora': time}
            context_message = update_sensor_data(sensor_data)

            history = [context_message] + history[1:]
        elif 'PARAR CONVERSA' in user_message.upper():
            break
        else:
            continue
```

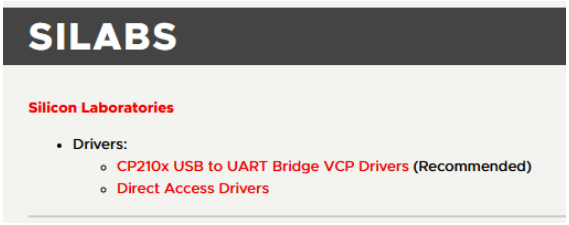
Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

A função *chatbot_loop* é a função principal do código, nela o chat é iniciado e toda a lógica é encadeada. No loop o chat espera o usuário dizer “assistente” para enviar as perguntas ao modelo, se o usuário disser “parar conversa” o loop é quebrado e o chat para.

3.4 Montagem do Circuito Físico

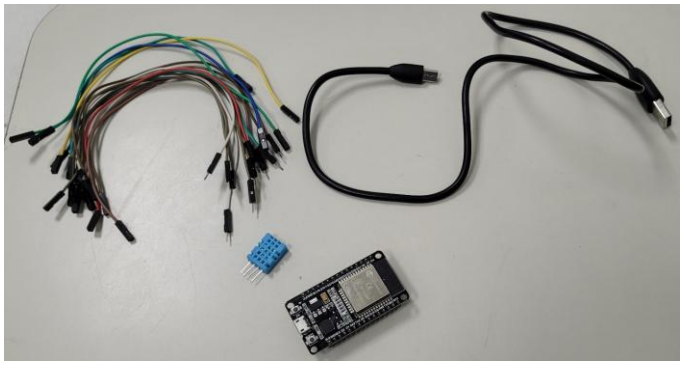
Para montar o circuito físico, utilizaremos os seguintes componentes: o ESP32, o sensor DHT-11, um cabo USB para conexão ao computador, um cabo microUSB v8 para conexão ao ESP32, e cabos Jumper, que são pequenos condutores usados para conectar dois pontos em um circuito eletrônico. O código será desenvolvido e executado utilizando o Arduino IDE. Além disso, é importante verificar se o computador onde o Arduino IDE está instalado possui o driver necessário para que a porta (COM) à qual o microcontrolador está conectado seja reconhecida.

Figura 23: Instalação do driver para reconhecimento de porta (COM)



Fonte: (ESP32.NET. USB-UART Interface, 2024)

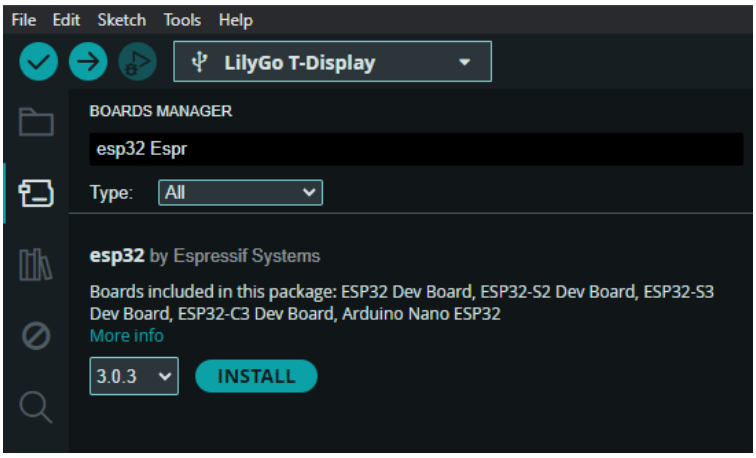
Figura 24: Materiais utilizados para montagem do circuito



Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

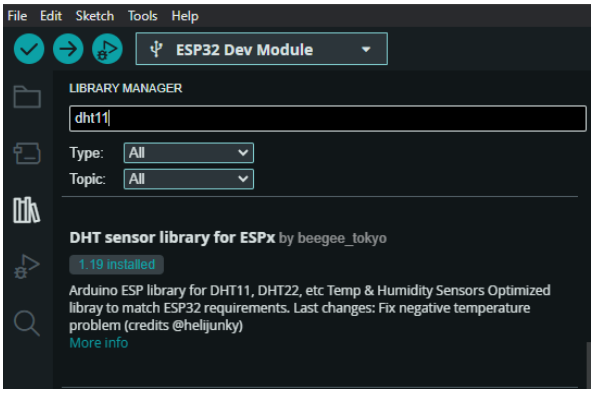
Para realização do código será necessário instalar a placa do ESP32 e a biblioteca que possibilitará a conexão do ESP32 com o sensor DHT-11 no Arduino IDE.

Figura 25: Instalação da placa ESP32



Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

Figura 26: Instalação da biblioteca do sensor DHT-11



Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

Com o circuito conectado e as configurações do ambiente concluídas, o próximo passo é carregar o código e monitorar os resultados fornecidos pelo microcontrolador e pelo sensor, que irão medir a umidade e a temperatura do ambiente.

Figura 27: Código para ESP32 e DHT-11 parte 1

```
#include "DHT.h"

#define DHTPIN 15
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

void setup(){
  Serial.begin(9600);
  dht.begin();
}
```

Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

Figura 28: Código para ESP32 e DHT-11 parte 2

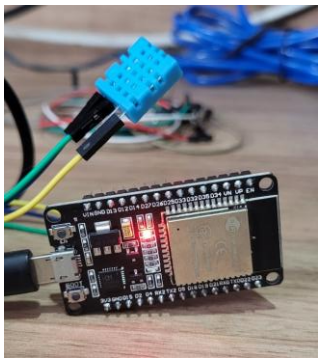
```
void loop(){

  float h = dht.readHumidity();
  float t = dht.readTemperature();

  if(isnan(t) || isnan(h))
  {
    Serial.println("Failure while reading the DHT 11 sensor");
  }
  else
  {
    Serial.print("Humidity: ");
    Serial.print(h);
    Serial.print("  ");
    Serial.print("Temp: ");
    Serial.print(t);
    Serial.println(" °C");
  }
  delay(1000);
}
```

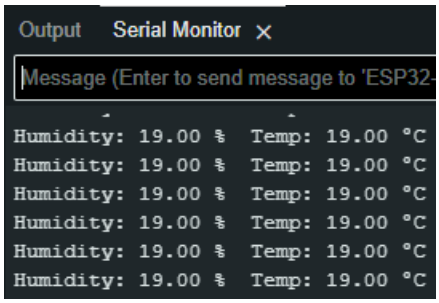
Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

Figura 29: Circuito físico montado



Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

Figura 29: Resultado apresentado



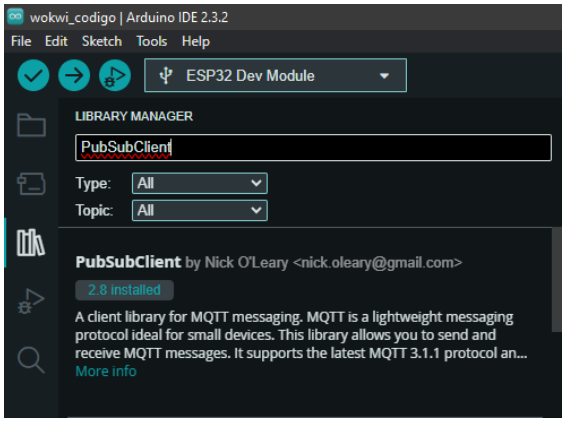
Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

3.5 Envio dos dados para nuvem através do HiveMQ

Com o circuito pronto, o próximo passo é enviar os dados capturados do ambiente monitorado para a nuvem, para tal, o código do circuito foi alterado, mas sua montagem física se manteve a mesma.

Agora, o circuito importa novas bibliotecas, a WiFi, para conexão da placa do ESP32 com o WiFi, o WiFiClientSecure que serve para adicionar uma validação SSL na conexão com o broker do MQTT e a PubSubClient, para conexões com MQTT, as duas primeiras já estão inclusas na placa ESP32, porém a última deve ser instalada através do Arduino IDE.

Figura 30: Instalação das novas bibliotecas



Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

A primeira parte do código é referente a importação das bibliotecas, além da configuração de conexão com o WiFi e com o MQTT, ele também inicializa as bibliotecas do WiFiClientSecure e do PubSubClient, configura o sensor DHT11 conectado na porta 15 do ESP32 e cria uma variável que servirá como um histórico para capturar a leitura anterior dos dados.

Figura 31: Código para envio dos dados na nuvem parte 1

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include <DHT.h>

// Wi-Fi parameters
const char* ssid = "WIFI NAME";
const char* password = "WIFI PASSWORD";

// MQTT Server Parameters
const char* mqtt_server = "graybumble-q1imoe.a02.usw2.aws.hivemq.cloud";
const int mqtt_port = 8883;
const char* mqtt_user = "icec24-publisher";
const char* mqtt_password = "Icec2409";
const char* mqtt_topic = "sensores-ic";

// MQTT and WiFi Client
WiFiClientSecure espClient;
PubSubClient client(espClient);

// DHT11 Settings
#define DHTPIN 15
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

String prev_weather = "";
```

Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

No método, setup, o serial e o sensor serão inicializados, o sistema irá realizar a conexão com o WiFi, o certificado para o SSL será ignorado, para fins de testes, o client irá configurar o servidor, e por fim, ele se conectará ao MQTT.

Figura 32: Código para envio dos dados na nuvem parte 2

```
void setup() {

  Serial.begin(115200);

  dht.begin();

  connectToWiFi();

  espClient.setInsecure();

  // Set MQTT client server
  client.setServer(mqtt_server, mqtt_port);

  connectToMQTT();
}
```

Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

Em seguida no método, loop, o sistema irá conferir a conexão com o cliente, se ele estiver conectado ele fará a leitura do sensor a cada 5 segundos, se houver falha na leitura ele identificará e reportará ao usuário que houve erro no serial, senão o sensor verificará se houve alguma mudança em relação a leitura anterior, se a leitura for igual a anterior o sistema irá avisar que não há alterações, porém se houver alterações, ele publicará os novos dados colhidos no tópico criado dentro servidor do MQTT e os enviará para o serial.

Figura 33: Código para envio dos dados na nuvem parte 3

```
void loop() {
  if (!client.connected()) {
    connectToMQTT();
  }
  client.loop();

  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Fail while reading DHT sensor!");
    return;
  }

  String weather = "{\"temp\": " + String(temperature) + ", \"humidity\": " + String(humidity) + "}";

  if (weather != prev_weather) {
    Serial.println("Updated!");
    Serial.println("Publishing in topic " + String(mqtt_topic) + ": " + weather);

    if (client.publish(mqtt_topic, weather.c_str())) {
      Serial.println("Successfully published!");
    } else {
      Serial.println("Fail while publishing the message.");
    }

    prev_weather = weather;
  } else {
    Serial.println("No modifications in the data.");
  }

  // 5 seconds of delay between the readings
  delay(5000);
}
```

Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

Os dois métodos auxiliares utilizados no método setup foram criados para realizar a conexão com o WiFi e o MQTT e eles funcionam da seguinte maneira. Para a conexão com o WiFi o sistema avisa ao usuário pelo serial que ele iniciará a conexão e ele inicia, enquanto a conexão não é finalizada ele envia ao serial ".", que serve como identificadores de espera ao usuário, e ao finalizar a conexão ele envia ao serial a mensagem "Connected". Para a conexão com o MQTT ele inicia verificando se o client já está conectado, enquanto ele não estiver conectado ele avisa o usuário pelo serial que a conexão irá iniciar e tenta se conectar ao cliente usando o client id, o usuário e a senha do MQTT fornecidas nas primeiras linhas do código, se houver algum erro de conexão ele enviará o serial e tentará realizar a conexão novamente a cada 5 segundos, senão ele só enviará ao usuário a mensagem "Connected in the MQTT Server!", identificando que a conexão foi estabelecida com sucesso.

Figura 34: Código para envio dos dados na nuvem parte 4

```
void connectToWiFi() {
  Serial.print("Connecting in the WiFi");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(100);
  }
  Serial.println(" Connected!");
}

void connectToMQTT() {
  while (!client.connected()) {
    Serial.print("Connecting in the MQTT server...");
    if (client.connect("graybumble-q1imoe", mqtt_user, mqtt_password)) {
      Serial.println("Connected in the MQTT Server!");
    } else {
      Serial.print("Fail while connecting. Error code: ");
      Serial.println(client.state());
      delay(5000);
    }
  }
}
```

Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

Figura 35: Resultado obtido a partir do novo código

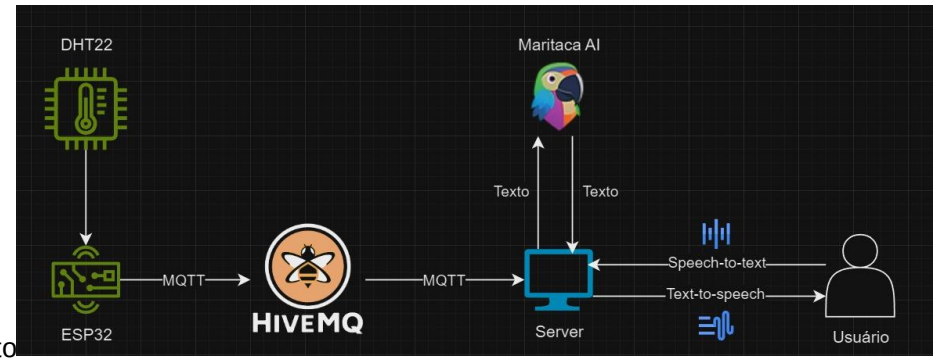
```
14:41:39.984 -> No modifications in the data.
14:41:45.024 -> Updated!
14:41:45.024 -> Publishing in topic sensores-ic: {"temp": 26.00, "humidity": 16.00}
14:41:45.024 -> Successfully published!
14:41:50.059 -> No modifications in the data.
14:41:55.073 -> No modifications in the data.
14:42:00.088 -> No modifications in the data.
14:42:05.129 -> Updated!
14:42:05.129 -> Publishing in topic sensores-ic: {"temp": 25.00, "humidity": 16.00}
14:42:05.129 -> Successfully published!
14:42:10.156 -> Updated!
14:42:10.156 -> Publishing in topic sensores-ic: {"temp": 26.00, "humidity": 16.00}
14:42:10.156 -> Successfully published!
14:42:15.172 -> Updated!
14:42:15.173 -> Publishing in topic sensores-ic: {"temp": 25.00, "humidity": 16.00}
14:42:15.173 -> Successfully published!
14:42:20.173 -> No modifications in the data.
14:42:25.240 -> No modifications in the data.
14:42:30.263 -> No modifications in the data.
```

Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

3.6 Integração do chat com o ESP-32

A integração dos dados enviados pelo ESP-32 foi feita através da conexão MQTT do chat com o HiveMQ, para tal foram adicionadas 3 novas funções ao *chatbot*: *setup_mqtt:* Faz a conexão com o broker MQTT da HiveMQ e inicia o loop de leitura de mensagens, *stop_mqtt:* Termina a conexão com o broker e finaliza o loop de leitura de mensagens, *on_message:* Executada a cada nova mensagem recebida pelo *broker,* atualiza o contexto dos sensores enviado ao LLM com os dados mais atualizados, já na função principal, é iniciado uma thread para o recebimento de mensagens pelo MQTT e outra thread para processamento e recebimento de mensagens do *chatbot.*

Figura 36: Arquitetura de funcionamento



Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

Figura 37: Função setup_mqtt

```python
def setup_mqtt():
    global mqtt_client
    mqtt_client = mqtt.Client(client_id="", userdata=None)
    mqtt_client.on_message = on_message
    mqtt_client.tls_set(tls_version=mqtt.ssl.PROTOCOL_TLS)
    mqtt_client.username_pw_set("icec24", "Icec2409")
    mqtt_client.connect(MQTT_BROKER, 8883)
    mqtt_client.subscribe(MQTT_TOPIC)
    mqtt_client.loop_start()
    print('connected')
```

Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

Figura 38: Função on_message

```python
def on_message(client, userdata, message):
    global sensor_data
    payload = message.payload.decode('utf-8')
    data = json.loads(payload)
    sensor_data = {
        'temperatura_quarto': data.get('temp', 0),
        'temperatura_sala': data.get('temp', 0),    # Usando a mesma temperatura do quarto
        'horario_agora': datetime.datetime.now().strftime("%H:%M")
    }
    print(sensor_data)
```

Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

Figura 39: Função stop_mqtt

```python
def stop_mqtt():
    global mqtt_running
    mqtt_running = False
    mqtt_client.loop_stop()
    mqtt_client.disconnect()
```

Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

Figura 40: Função principal

```python
if __name__ == "__main__":
    mqtt_thread_instance = threading.Thread(target=mqtt_thread)
    mqtt_thread_instance.start()

    try:
        chatbot_loop()
    finally:
        stop_mqtt()
        mqtt_thread_instance.join()
```

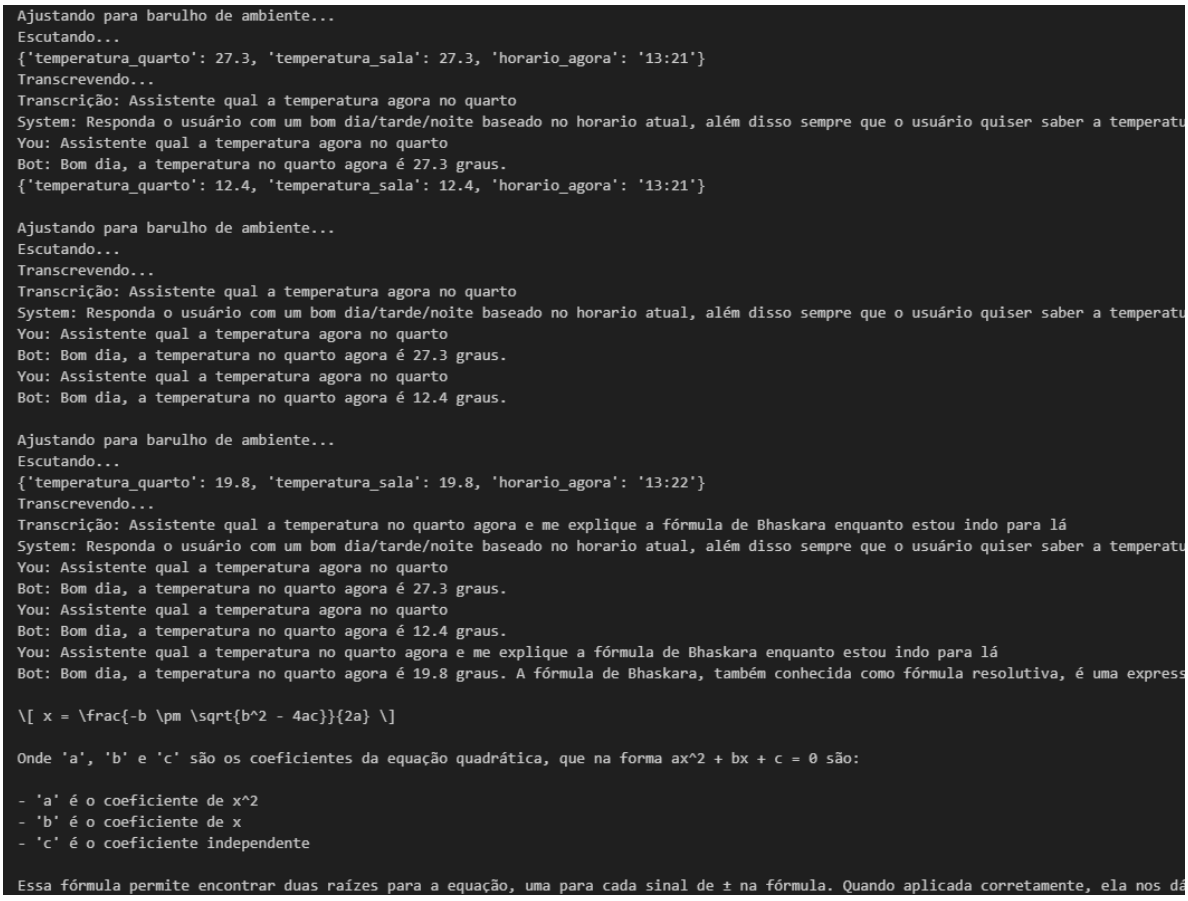
Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

4 Resultados e discussão

Dentro dos resultados obtidos, foi evidenciado que a inclusão do LLM em um sistema de monitoramento com IoT e ESP32 proporcionou uma significativa melhoria na interação com o usuário. Ao invés de apenas oferecer respostas predefinidas, o *chatbot* é capaz de processar os dados retornados pelas requisições do usuário e apresentá-los de forma acessível e inclusiva, diretamente no chat. Isso permite que qualquer pessoa, independentemente de seu nível de familiaridade com o sistema, possa compreender as informações solicitadas e estender a conversa para além dos limites do monitoramento em questão.

É importante ressaltar que, além de responder a consultas relacionadas ao sistema monitorado, o *chatbot* também se mostrou capaz de lidar com perguntas mais gerais e abrangentes, como questões sobre o clima ou calendário. Essa flexibilidade na interação amplia consideravelmente a utilidade e a versatilidade do sistema, permitindo que os usuários explorem uma variedade de tópicos e solicitações, mesmo que não diretamente relacionados ao monitoramento em si.

Figura X: Exemplo de funcionamento do chatbot



Fonte: (DESENVOLVIMENTO PRÓPRIO, 2024)

Referências

AWS. (2024). **O que é IoT**. Disponível em: https://aws.amazon.com/pt/what-is/iot/. Acesso em: 23 mar. 2024.

BLIPBLOG. (16 de setembro de 2021). **Chatbot**. Disponível em: https://www.blip.ai/blog/chatbots/chatbot/. Acesso em: 23 mar. 2024.

COSTA, M. (4 de dezembro de 2024). **Quais são os tipos de inteligência artificial (IA)? Exemplos e características**. Disponível em: https://www.alura.com.br/artigos/tipos-inteligencia-artificial-ia#quais-sao-os-tipos-de-inteligencia-artificial. Acesso em: 23 mar. 2024.

DIAS, M. (2024). **O que é ESP32? Pra que serve? Quando usar?** Disponível em: LOBO DA ROBÓTICA: https://lobodarobotica.com/blog/o-que-e-esp32-pra-que-serve-quando-usar/. Acesso em: 23 mar. 2024.

ESP32.NET. (2024). USB-UART Interface. Disponível em: http://esp32.net/usb-uart/. Acesso em: 22 ago. 2024.

ESPRESSIF. (2024). **ESP-IDF Get Started Guide.** Disponível em: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html. Acesso em: 24 mar. 2024.

GUPYA, V. (20 de maio de 2021). **Vantagens de usar MQTT para dispositivos IoT**. Disponível em: https://psiborg.in/advantages-of-using-mqtt-for-iot-devices/. Acesso em: 24 mar. 2024.

HiveMQ. (2024). **HiveMQ**. Disponível em: https://www.hivemq.com. Acesso em: 28 mai. 2024.

TUDO SOBRE IOT, **"IoT Feito Fácil": Brincando com o ESP32 no Arduino IDE**. (29 de novembro de 2022). Tudo sobre IoT. Disponível em:

https://www.tudosobreiot.com.br/blog/1098-iot-feito-facil_-brincando-com-o-esp32-no-arduino-ide. Acesso em: 29 mar. 2024.

NASCIMENTO, E. F., COSTA JÚNIOR, F. L., LIRA, F. A., COUTINHO, J. P., MOTA, J. B., BEZERRA, G. P. B., & JUCÁ, S. S. C. (18 de março de 2024). **Sistemas IoT para alerta de emergência em ambientes internos.** In: Anais da VII Escola Regional de Computação Aplicada à Saúde, pp. 282-287.

MISCHIANTI, R. (17 de fevereiro de 2021). **ESP32 Dev Kit V1: High-Resolution Pinout and Specs.** Disponível em: https://mischianti.org/doit-esp32-dev-kit-v1-high-resolution-pinout-and-specs/. Acesso em: 06 abr. 2024.

OLIVEIRA, V. I. de; FREIRE, F. L.; ZANATTA, A. R.. **Optical properties of Er and Er+ Yb doped hydrogenated amorphous silicon films**. *Journal of Physics: Condensed Matter*, v. 18, n. 32, p. 7709, 31 jul. 2006

ORACLE**, O que é um chatbot?** (2024). Oracle. Disponível em: https://www.oracle.com/br/chatbots/what-is-a-chatbot/. Acesso em: 06 abr. 2024.

ORACLE. (2024). **Internet das Coisas.** Disponível em: https://www.oracle.com/br/internet-of-things/what-is-iot/. Acesso em: 06 abr. 2024.

RANDOM NERD TUTORIALS. (2024). **Getting Started with ESP32**. Disponível em: https://randomnerdtutorials.com/getting-started-with-esp32/. Acesso em: 06 abr. 2024.

SALESFORCE. (2024). **Chatbot**. Disponível em: https://www.salesforce.com/br/atendimento-ao-cliente/chatbot/. Acesso em: 13 abr. 2024.

SAP. (2024). **O que é IoT?** Disponível em: https://www.sap.com/brazil/products/artificial-intelligence/what-is-iot.html. Acesso em: 13 abr. 2024.

SPADINI, A. S. (4 de dezembro de 2024). **O que é inteligência artificial? Como funciona uma IA, quais os tipos e exemplos**. Disponível em: https://www.alura.com.br/artigos/inteligencia-artificial-ia. Acesso em: 20 abr. 2024.

THE INTERNET OF THINGS WITH ESP32. (2024). Disponível em: http://esp32.net. Acesso em: 20 abr. 2024.

Vieira Franzini, L., Silva Mamede, R., & De Melo, W. C. (27 de novembro de 2020). **Internet das Coisas com ESP32 e Amazon Alexa**, pp. 1-6.

WOKWI. (2024). **ESP32 Guide.** Disponível em: https://docs.wokwi.com/guides/esp32. Acesso em: 28 mai. 2024.

WOKWI. (2024). **DHT22 Guide**. Disponível em: https://docs.wokwi.com/parts/wokwi-dht22. Acesso em: 28 mai. 2024.

WOKWI. (2024). **Project 322577683855704658.** Disponível em: https://wokwi.com/projects/322577683855704658. Acesso em: 28 mai. 2024.

WEBSOCKET CLIENT. (2024). **HiveMQ.** Disponível em: https://www.hivemq.com/demos/websocket-client/. Acesso em: 28 mai. 2024.

BROWN, T. B., MANN, B., RYDER, N., SUBBIAH, M., KAPLAN, J., DHARIWAL, P., RAMES, A. **Language Models are Few-Shot Learners.** 34th Conference on Neural Information Processing Systems, 2020.

COPELAND, B. **Artificial Intelligence.** Retrieved from Enciclopaedia Brittanica: https://www.britannica.com/technology/artificial-intelligence acesso em março de 2024.
DIRAC, L. **LSTM is dead. Long Live Transformers!** Seattle Applied Deep Learning. Seattle, Washington, Estados Unidos da América, 2019.

GUPTA, A., MAJUMBER, B., & VAJJALA, S. **Practical Natural Language Processing.** O'Reilly, 2020.

RADFORD, A., NARASIMHAN, K., SALIMANS, T., & SUTSKEVER, I. **Improving Language Understanding By Generative Pre-Training.** OpenAI, 2018.

ROSENBLATT, F. **The perceptron: A probabilistic model for information storage and organization in the brain.** American Psychological Association, 1958.

RUMELHART, D. E., HINTON, G. E., & WILLIAMS, R. J. **Learning representations by back-propagating errors.** Nature, 1986.

SAH, S.. **Machine Learning: A Review of Learning Types.** mlrelated, 2020.

SCHICK, T., DWIVEDI-YU, J., DESSI, R., RAILEANU, R., LOMELI, M., HAMBRO, E., . . . SCIALOM, T. **Toolformer: Language Models Can Teach Themselves to Use Tools.** arXiv, 2023.

VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., POLOSUKHIN, I. **Attention Is All You Need.** 31st Conference on Neural Information Processing System - NeurIPS, 2017.