GuessWhat : classificação de áudio IoT para deficientes auditivos

Thaís Bezerra de Menezes Benício de Sousa [1],

[1]Universidade Federal do Ceará - Departamento de Engenharia de Teleinformática(DETI)
Fortaleza – CE – Brazil

{thais.ufc}@alu.ufc.br

1. Escopo

1.1. Apresentação

O projeto GuessWhat consiste em um sistema distribuído de classificação de sons captados pelo microfone da placa BitDogLab. O público-alvo consiste principalmente de deficientes auditivos, os quais se beneficiam da descrição do som ambiente. A BitDogLab é um kit de desenvolvimento que contém uma Raspberry Pi Pico W, ou RP2040. O sistema GuessWhat é, portanto, capaz de captar 3s de áudio por vez, enviar via Wi-fi ao servidor, o qual pré-processa o som e retorna a classificação: por exemplo, se é fala, música, choro, barulho de animal.

1.2. Título

GuessWhat : classificação de áudio IoT para deficientes auditivos

1.3. Objetivos

- Proporcionar a deficientes auditivos uma descrição sonora do ambiente
- Prover uma interface intuitiva no próprio sistema embarcado, para os deficientes auditivos
- Desenvolver uma aplicação em Python para o servidor

1.4. Funcionalidades

- Gravação do som ambiente, a cada cerca de 30s: 3s de áudio são gravados, então esse som é enviado ao servidor para classificá-lo
- Envio do áudio via Wi-fi ao servidor
- Classificação do som presente no áudio
- Interação com o usuário através de LEDs e de mensagens no display OLED
- Input do usuário através de um botão; o botão inicia a gravação contínua. Para encerrar o sistema, basta apertar o botão de novo.

1.5. Justificativa

A perda de audição é um problema com uma variedade de causas, que afeta todas as idades, mas, especialmente, os idosos [Karunaratna 2021]. Essa deficiência pode trazer sérias consequências para o indivíduo, devido à perda de sinais importantes, como alarme de incêndio, barulho de eletrodomésticos, o som da chuva etc [Karunaratna 2021]. Além disso, segundo a Organização Mundial da Saúde(OMS), cerca de 430 milhões de pessoas no mundo sofrem com perda auditiva [Mojado 2024]. Tendo isso em vista, o desenvolvimento de um sistema como o GuessWhat é essencial para fornecer a esses indivíduos um meio de inclusão que complemente seu tratamento profissional.

1.6. Originalidade

A seguir, serão apresentados alguns trabalhos relacionados ao sistema GuessWhat. Deve-se observar que a proposta do GuessWhat é original e se diferencia em alguns aspectos das pesquisas a seguir.

[Karunaratna 2021] utilizam uma Raspberry Pi 3 com um microfone para captar continuamente dados de áudio; se determinadas classes de eventos forem reconhecidas, o sistema envia notificações via Wi-fi aos clientes (App mobile e também um wearable), portados pelo deficiente auditivo.

[An et al. 2022] captam áudio pelo smartphone e enviam a um servidor que utiliza deep-learning, com técnicas de imagem, para classificar o som do áudio. Assim, ele envia de volta ao App, que transmite também via bluetooth a notificação para um bracelete vibrável, equipado com um Arduino nano.

[Salem et al. 2023] aplicam diversos modelos de classificação para detectar sons específicos durante o trânsito, a fim de auxiliar os surdos que dirigem. Assim, eles testam o sistema utilizando uma Raspberry pi 4, equipada com microfone e display.

[Mojado 2024] utilizam um Arduino Nano BLE com modelo de redes neurais convolucionais para classificação de áudio a fim de auxiliar surdos no ambiente doméstico. Para embarcar o modelo, utilizaram a ferramente Edge Impulse. Além disso, acoplam uma ESP32 ao sistema para fazer a interação com o usuário via IoT, tanto enviando mensagens para um App, como enviando sinais para uma fita de LEDs RGB(cada padrão de cor indica uma classe diferente de áudio).

Após essa revisão bibliográfica, percebeu-se a falta de trabalhos que utilizem uma Raspberry pi Pico W na captação de áudio, junto com o modelo pré-treinado YAMNET para a classificação.

2. Especificação do hardware

2.1. Diagrama de blocos

Na figura 1, observa-se o diagrama em blocos dos componentes físicos do projeto.

O microfone tem a função de captar o som ambiente, amostrado a uma taxa de 22.05 kHz, por meio do ADC (conversor analógico digital) da Raspberry.

O botão deve ser pressionado pelo usuário a fim de iniciar o sistema. Depois, o botão pode ser pressionado a qualquer momento para interromper a gravação contínua. Ele funciona como chave liga/desliga, e seu estado é monitorado por meio de interrupção no GPIO.

A raspberry é o microcontrolador do sistema, responsável pela recepção de comando e envio de dados de áudio ao servidor. Vale ressaltar que, como forma de feedback visual, a Raspberry também aciona o LED azul, indicando que não está gravando, e aciona o LED verde, durante os 3s de gravação. Além disso, o display OLED tem a função de fornecer informações úteis, como indicar o envio de dados e mostrar a classe do áudio retornada pelo servidor.

Por fim, o tem-se o servidor em Python. Ele irá receber o vetor de áudio, realizar um processamento de redução de ruído e salvá-lo em um arquivo .wav. Depois, utilizando

o modelo YAMNET pré-treinado do TensorFlow, ele irá determinar qual a classe a que pertence o áudio do som. Vale ressaltar que esse modelo reconhece mais de 500 classes, como fala, música, animal etc. Assim, o servidor retorna a classe para a Raspberry. Vale ressaltar que o nome das classes está em inglês.

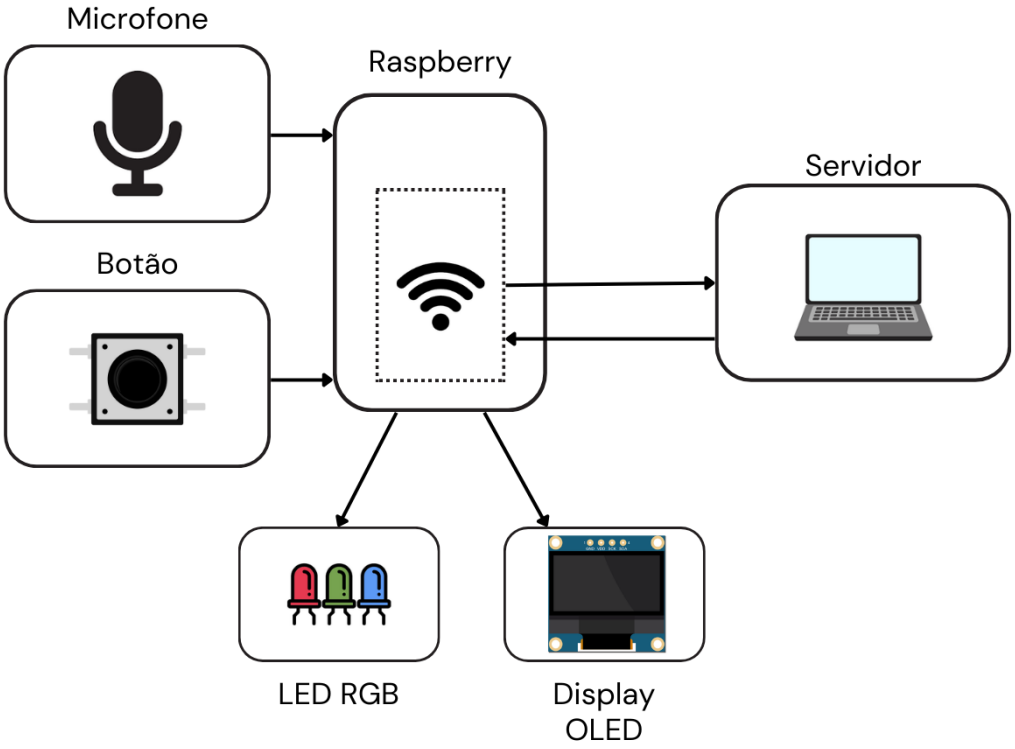

Figura 1. Diagrama de blocos do sistema GuessWhat

2.2. Esquemático

Na figura 2, pode-se ver o esquemático do hardware. Esse circuito foi feito através do simulador Wokwi; como o mesmo não possui microfone, utilizou-se um potenciômetro na ilustração, no lugar do microfone. Vale ressaltar que o potenciômetro também retorna valores analógicos, o que é válido para fins de ilustração.

A pinagem utilizada foi:

- Pino 37 - Vcc($3.3V$) : Anodo comum do LED RGB, Vcc do potenciômetro(microfone) e Vcc do display OLED
- Pino 3 - GND 1 : GND do pushbutton, que é configurado como pull-up via software; GND do display OLED
- Pino 23 - GND 5 : Ground do potenciômetro(microfone)
- GP5 : GPIO do pushbutton, configurada como entrada pull-up
- GP11 e GP12 : GPIO dos leds verde e azul, respectivamente, configurados como GPIO de saída
- GP14 e GP15 : SDA e SCL do display OLED, configurados como i2c
- GP28 : ADC do microfone, representado pelo sinal do potenciômetro

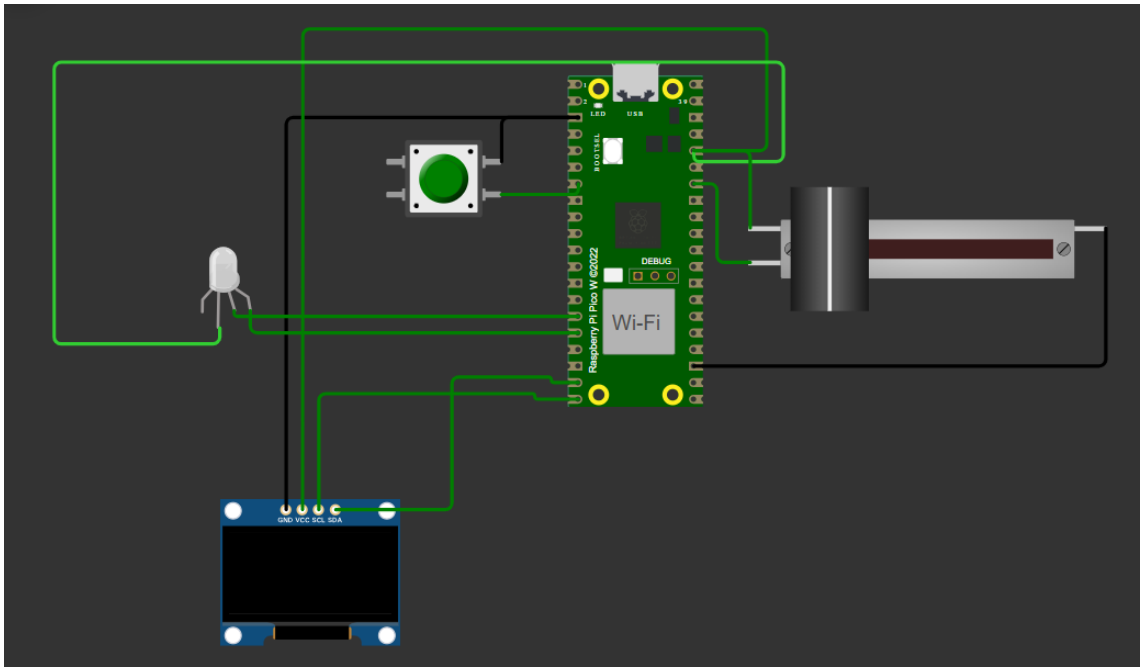

Figura 2. Esquemático do circuito do sistema GuessWhat

3. Especificação do firmware

3.1. Blocos funcionais

Na figura 3, observa-se a organização em camadas tanto do firmware quanto do software da aplicação. A seta à direita indica o fluxo dos dados desde a captura pelo microfone, até a sua classificação, no servidor. A seguir, será dada uma explicação mais detalhada sobre o código de cada camada:

- Captura do microfone

```
void sample_mic_no_dma()
{

    sample_count = 0;
    sample_period = time_us_64() + (1000000 / WAV_SAMPLE_RATE);

    while (sample_count < SAMPLES)
    {

        while(time_us_64() < sample_period);
        sample_period = time_us_64() + (1000000 / WAV_SAMPLE_RATE);
        adc_buffer[sample_count++] = (int16_t)((adc_read() * 32) -
            32768);

    }
}
```

Listing 1. Captura de áudio pelo microfone

A variável sample_count é responsável por contar o número de amostras coletadas, que foram definidas no cabeçalho do código como sendo 3 ∗

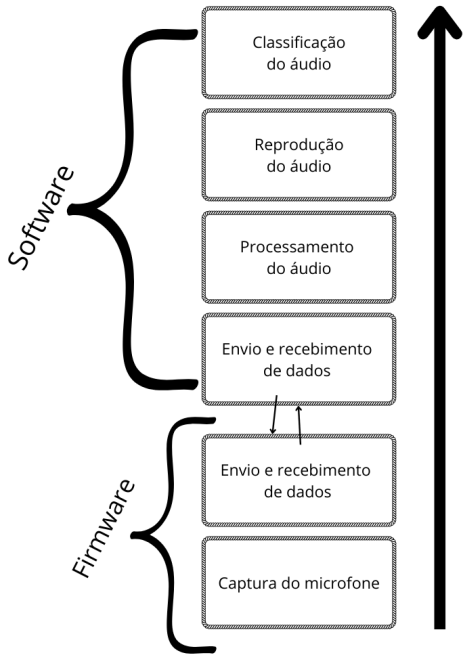

Figura 3. Diagrama em camadas do sistema GuessWhat

WAV_SAMPLE_RATE, já que são 3s de áudio captado. Foram escolhidos 3s de áudio devido à limitação da memória da Raspberry. Também foi escolhida a taxa de amostragem como $WAV\_SAMPLE\_RATE = 22050Hz$, padrão muito usado para arquivos .wav.

Já o $sample\_period$ refere-se ao próximo instante em que será feita amostragem, a partir do cálculo do instante atual em microssegundos, com a função $time\_us\_64$, somada ao período de amostragem. Então , o while principal fica preso na linha 10, esperando a hora certa de realizar a próxima captura.

Os dados são armazenados no adc_buffer. O valor lido, de 0 a 4095, passa por um escalamento(multiplicação por 32) e posterior centralização(subtração por 32768) para que fique na escala de áudio para int16, que é -32768 a 32767.

- Envio e recebimento de dados : firmware

Todos os dados são trocados através de sockets TCP. Assim, o cliente(Raspberry) precisa saber qual o endereço IP e porta do servidor. De toda forma, o protocolo TCP é um protocolo da camada de transporte na pilha TCP/IP. Ele é orientado à conexão, o que significa que antes da transmissão de dados, os dispositivos estabelecem uma conexão por meio de um processo chamado three-way handshake.

A nível de aplicação, os pacotes de áudio enviados ao servidor contêm apenas o dado bruto.

```
1 void send_to_server(uint16_t *data, size_t len) {
2     ip_addr_t server_ip;
3     IP4_ADDR(&server_ip, x, x, x, x); // IP do servidor
4 
5     tcp_arg(client_pcb, data); // Passar dados para o callback
```

```
    if(is_connected)
    {
        printf("envio de dados\n");


        // Calcular o tamanho total em bytes dos dados de audio (2 bytes por amostra)
        size_t data_size = len * sizeof(uint16_t);


        // Copiar os dados de audio para o buffer
        memcpy(send_buffer , data, data_size);

        // Verificar espaco disponivel no buffer TCP
        uint16_t available_space = tcp_sndbuf(client_pcb);
        if ((data_size) > available_space) {
            printf("Espaco insuficiente no buffer TCP: %d bytes disponiveis, %lu bytes necessarios\n",
             available_space,  data_size);

            return; // Retornar para evitar tentar o envio
        }


        // Enviar os dados
        err_t write_err = tcp_write(client_pcb, send_buffer, data_size, TCP_WRITE_FLAG_COPY);
        if (write_err != ERR_OK) {
            printf("Erro ao enviar dados: %d\n", write_err);

            return;
        }

        printf("Dados enviados (%lu bytes).\n", data_size);


    }

}
```

Listing 2. Envio de dados pela Raspberry

As libs do SDK utilizadas para a conexão Wi-fi e socket TCP foram $cyw43\_arch.h$ e $lwip/tcp.h$. A lib lwip(Light-weight IP) fornece uma abordagem leve para sistemas embarcados; no arquivo de configuração, foi selecionada a opção $NO\_SYS = 1$, isto é, a Raspberry irá executar sem sistema operacional, a fim de simplificar o desenvolvimento. Dessa forma, o processo de comunicação ocorre em modo pooling, aliado a algumas funções de callback relativas ao envio e recebimento de dados. Vale destacar que a escolha por esse modo de programação, apesar de simplificar a codificação, impacta no desempenho do sistema.

A respeito do código, tem-se que a struct tcp_pcb (Protocol Control Block) é uma estrutura da biblioteca lwIP que representa um bloco de controle de protocolo TCP. Além disso, os dados são enviados em um buffer $send\_buffer$, que é um vetor de char definido como variável global. Os dados são finalmente envia-

dos através da função tcp_write. Vale destacar que o tamanho de $send\_buffer$ é 2400; escolheu-se esse tamanho após alguns testes, e com base no tamanho máximo suportado pelo pacote TCP.

```
1
2  static err_t recv_callback(void *arg, struct tcp_pcb *tpcb,
       struct pbuf *p, err_t err) {
3      if (p == NULL) {
4
5          printf("Conexao encerrada.\n");
6          tcp_close(tpcb);
7          return ERR_OK;
8      }
9
10     if (err != ERR_OK) {
11         printf("Erro no callback de recepcao: %d\n", err);
12         pbuf_free(p);
13         return err;
14     }
15
16     // Verifica o tamanho do buffer recebido
17     size_t len = p->len;
18     if (len > 0) {
19         if ((recv_buffer_len + len) < sizeof(recv_buffer)) {
20             // Copia os dados para o buffer global
21             memcpy(recv_buffer + recv_buffer_len, p->payload,
                   len);
22             recv_buffer_len += len;
23             printf("Recebi %zu bytes. Total acumulado: %zu
                   bytes.\n", len, recv_buffer_len);
24         } else {
25             printf("Buffer de recepcao esta cheio. Dados
                   descartados.\n");
26         }
27     }
28
29     // Libera o pbuf apos processar os dados
30     pbuf_free(p);
31
32     // Confirma o recebimento ao servidor
33     tcp_recved(tpcb, len);
34
35     return ERR_OK;
36 }
37
38
39 char* get_received_data(size_t *data_len) {
40     if (recv_buffer_len > 0) {
41         *data_len = recv_buffer_len;
42         recv_buffer_len = 0; // Reseta o comprimento apos
               retornar os dados
43         return recv_buffer;
44     } else {
45         *data_len = 0;
46         return NULL;
47     }
```

```
}
```

Listing 3. Recebimento de dados pela Raspberry

Já para o recebimento de dados, a maior parte da lógica ficou encapsulada na função de callback. Após fazer uma checagem de exceções, a função acessa o buffer de recebimento (uma struct pbuf) e copia os dados recebidos ($p- > payload$) para o vetor global $recv\_buffer$. Assim, a Raspberry recebe o resultado da classificação do áudio. Na função $get\_received\_data$, é feita uma checagem para ver se de fato houve dados recebidos, e, então, é retornado o ponteiro do vetor $recv\_buffer$.

- Envio e recebimento de dados : software

  A comunicação implementada no servidor Python foi feita com a biblioteca $socket$. O socket TCP do servidor é instanciado a partir do argumento $SOCK\_STREAM$, indicando que é um socket orientado à conexão.

- Processamento e reprodução do áudio

```python
def process_audio(full_audio_data, sample_rate):
    # Concatena os dados de audio acumulados
    full_audio_array = np.concatenate(full_audio_data)

    audio_no_noise = nr.reduce_noise(y = full_audio_array,
        sr=sample_rate)


    # Salva o audio como arquivo WAV
    audio_path = "audios/received_audio.wav"
    save_audio_as_wav(audio_no_noise, sample_rate, audio_path)


    # Classifica o audio
    audio_class = classify_audio(audio_path)

    return audio_class
```

Listing 4. Processamento e reprodução do áudio

A função de processamento do áudio no servidor concatena os pacotes recebidos, que eram armazenados na lista $full\_audio\_array$. Depois, é feita uma remoção de ruído com a lib $noisereduce$. O som é salvo como arquivo .wav, e, por fim, utilizado para a classificação.

- Classificação do áudio

```python

model = hub.load('https://tfhub.dev/google/yamnet/1')

class_map_path = model.class_map_path().numpy()
class_names = class_names_from_csv(class_map_path)

def classify_audio(audio_path):
    wav_file_name = audio_path
    sample_rate, wav_data = wavfile.read(wav_file_name, 'rb')
    sample_rate, wav_data = ensure_sample_rate(sample_rate,
        wav_data)

```

```
12      waveform = wav_data / tf.int16.max
13
14
15      scores, embeddings, spectrogram = model(waveform)
16
17      scores_np = scores.numpy()
18      spectrogram_np = spectrogram.numpy()
19      infered_class = class_names[scores_np.mean(axis=0).argmax()]
20      print(f'The main sound is: {infered_class}')
21      return infered_class
```

Listing 5. Classificação do áudio

O modelo de classificação utilizado é o YAMNET, pré-treinado e disponível via TensorFlowHub. Os dados são convertidos para uma taxa de amostragem de 16 kHz, exigida pelo modelo. Então, os dados são normalizados e passados para o modelo. O retorno mais importante do modelo é *scores*, contendo o grau de pertencimento a cada classe de áudio. Então, a classe inferida é aquela com a maior média de pontuação (linha 19). Por fim, essa classe será enviada de volta à Raspberry, no formato string, para ser mostrada no display oled.

3.2. Fluxograma

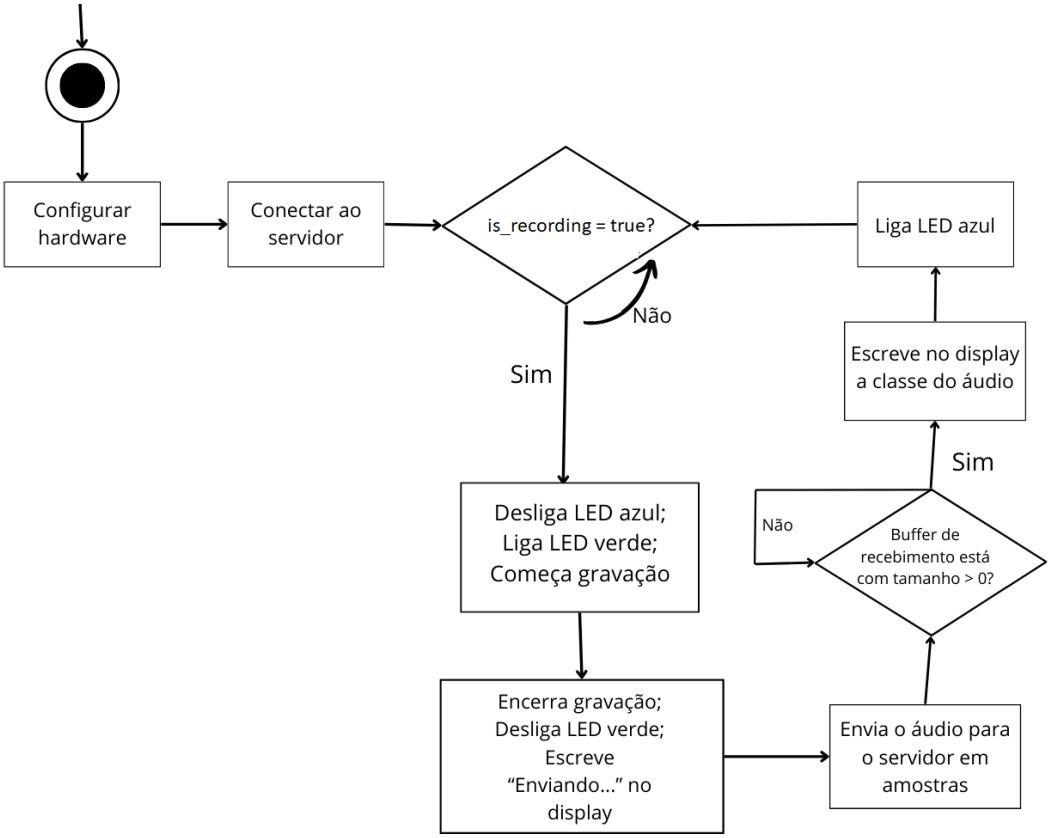

Figura 4. Fluxograma firmware GuessWhat

Na figura 4, pode-se observar o fluxograma do firmware, que, inicialmente, configura periféricos do hardware(gpio, i2c, adc); então, conecta-se à rede Wi-fi e também ao servidor. Após isso, o programa entra em loop aguardando até que a flag is_recording torne-se verdadeira. Essa flag é inicialmente falsa, e é invertida toda vez que o botão é pressionado(através da interrupção). Assim, quando o botão é ligado, o LED azul, que estava ligado pela configuração do hardware, é desligado, e o LED verde é ativado. Ao fim da gravação, nenhum LED está aceso; é escrita uma mensagem de "Enviando áudio"no display OLED. Após isso, o áudio é dividido em amostras dentro de um loop interno, e cada pedaço é enviado ao servidor pela função $send\_to\_server$. Dessa forma, com o término do envio, a Raspberry entra em loop, esperando pela resposta do servidor, que é a classe do áudio. Portanto, ao perceber que o tamanho do buffer $data\_len$ é maior que zero, o microcontrolador escreve no display oled a classe do áudio. Então, ele pisca o LED azul e começa uma nova gravação, em ciclo até que o botão seja pressionado.

O código a seguir é a implementação do fluxograma.

```
while (true) {


        if(is_recording)
        {
            printf("Iniciando gravacao...\n");
            gpio_put(BLUE_LED_PIN,0);

            sleep_ms(1000); //para nao gravar o barulho do botao
            gpio_put(GREEN_LED_PIN,1);

            sample_mic_no_dma();

            gpio_put(GREEN_LED_PIN,0);

            printf("termino da gravacao");

            // mandando ao servidor
            memset(ssd, 0, ssd1306_buffer_length);
            ssd1306_draw_string(ssd, 0, 0, "Enviando audio");

            render_on_display(ssd, &frame_area);

            for (int i = 0; i < SAMPLES; i += CHUNK_SIZE) {
                // Calcular o tamanho do chunk atual (o ultimo pode
                    ser menor que CHUNK_SIZE)
                size_t current_chunk_size = ((i + CHUNK_SIZE) <=
                    SAMPLES)
                                            ? CHUNK_SIZE
                                            : (SAMPLES - i);

                // Passar o ponteiro deslocado para a funcao
                    send_to_server
                send_to_server(&adc_buffer[i], current_chunk_size);
                sleep_ms(500);
            }

            sleep_ms(1000);
```

```
36
37              send_to_server(&listen_flag, 1); // Envia a flag ao
                    servidor
38
39              memset(ssd, 0, ssd1306_buffer_length);
40              render_on_display(ssd, &frame_area);
41
42              while(true){
43
44                  data = get_received_data(&data_len);
45
46                  printf("%d",data_len);
47
48                  if (data_len > 0)
49                  {
50                      uint x = 5;  // Posicao inicial x
51                      uint y = 0;  // Posicao inicial y
52
53                      for (uint i = 0; i < data_len; i++)
54                      {
55                          if (x + CHAR_WIDTH > MAX_WIDTH)
56                          {
57                              // Quebra de linha: Reinicia x e move para
                                    a proxima linha
58                              x = 5;
59                              y += LINE_HEIGHT;
60                          }
61
62                          printf("%c", data[i]);
63                          ssd1306_draw_char(ssd, x, y, data[i]);
64                          x += CHAR_WIDTH; // Avanca para a proxima
                                posicao horizontal
65                      }
66
67                      render_on_display(ssd, &frame_area);
68                      break;
69                  }
70              sleep_ms(100);
71              }
72
73
74              sleep_ms(8000);
75
76
77
78              gpio_put(BLUE_LED_PIN,1);
79          }
80
81
82
83      sleep_ms(100);
84
85  }
```

Listing 6. Loop principal da Raspberry

Na figura 5, observa-se o fluxograma do software da aplicação. Inicialmente, o servidor fica travado no “listen”, à espera da conexão do cliente. Então, após conectar, ele entra em loop, aguardando o recebimento de dados. O final do recebimento dos 3s é indicado por meio do envio de um pacote unitário, apenas com uma flag ”1”. Assim, o servidor sabe que é o momento de processar o áudio e enviar a classe obtida de volta à Raspberry. Depois, ele entra no loop novamente.

A listagem a seguir demonstra a implementação desse fluxograma.

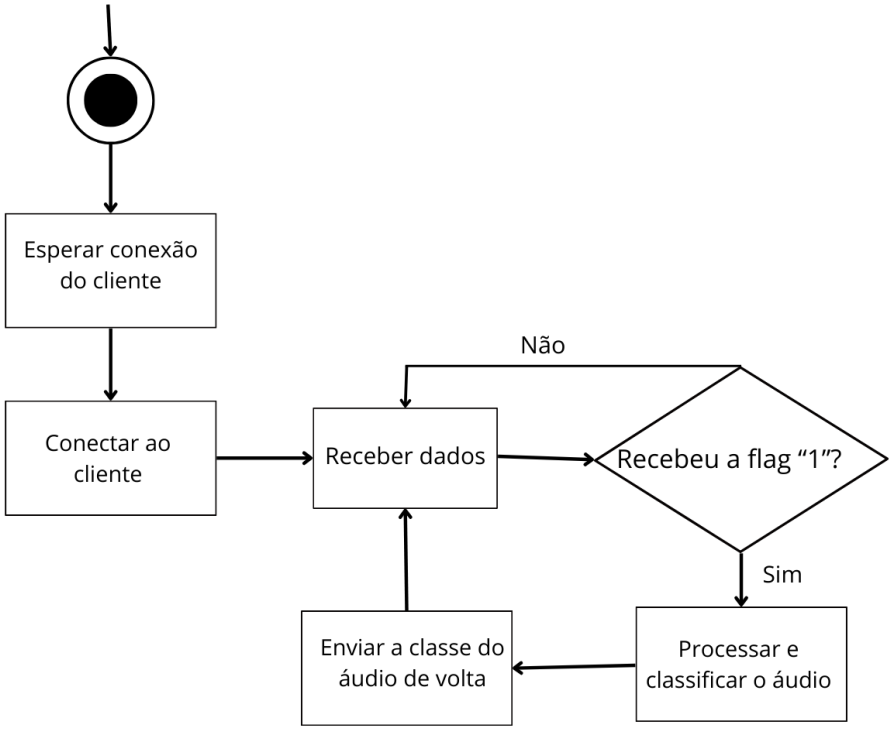

Figura 5. Fluxograma: software/servidor GuessWhat

```
def receive_data():
    full_audio_data = []  # Armazenar os dados de audio completos
    total_samples = 0

    client_socket, client_address = start_server()
    print(f"Conexao estabelecida com o cliente {client_address}.")

    data_buffer = b""

    while True:

        try:
              # Buffer para armazenar os pacotes recebidos

            packet = client_socket.recv(BUFFER_SIZE)

            if len(packet) == 2:
```

```
18
19                  print("recebi listen")
20
21                  listen_flag2 = packet[0] + (packet[1] << 8)
22                  print(f"Flag manualmente convertida: {listen_flag2}")
23
24                  if listen_flag2 == 1:
25                      print("Flag LISTEN recebida, processando o audio
                            acumulado...")
26
27                      audio_data = np.frombuffer(data_buffer,
                            dtype=np.uint16)
28                      full_audio_data.append(audio_data)
29                      total_samples += len(full_audio_data)
30
31                      print(f"Dados acumulados: {total_samples}
                            amostras")
32
33
34
35                      transcription = process_audio(full_audio_data,
                            SAMPLE_RATE)
36                      client_socket.sendall(transcription[:15].encode
37                      ("utf-8"))
38
39
40                      full_audio_data = []
41                      total_samples = 0
42
43                  data_buffer = b""
44              if not packet:
45                  raise ConnectionError("Conex o encerrada pelo
                        cliente.")
46
47              data_buffer += packet  # Acumula os pacotes recebidos
48
49
50              print(f"dados recebidos: {len(packet)}")
51              print(f"dados acumulados: {len(data_buffer)}")
52
53
54          except Exception as e:
55              print(f"Erro: {e}")
```

Listing 7. Loop principal do servidor

3.3. Inicialização

Na listagem abaixo, tem-se o processo de inicialização do firmware. Primeiro, iniciam-se as entradas e saídas na linha 3, a fim de permitir a depuração. Depois, é feito o processo de conexão à rede Wi-fi escolhida, por meio das funções disponibilizadas pela lib $cyw43\_arch.h$.

Após isso, é configurado o ADC, responsável pela captura de áudio do microfone. O ADC do microfone é o pino GP28, e o canal de entrada escolhido é o 2. Então, os pinos

de GPIO de entrada e saída são configurados; no caso, os LEDs azul e verde são setados como saída, e o botão em GP5 é setado como entrada pull-up. Além disso, é adicionada uma função de callback para o botão, que irá checar se este foi pressionado.

Dessa forma, a Raspberry se conecta ao servidor. Após a conexão bem-sucedida, ela acende o LED azul. Por fim, o display OLED, conectado pela interface i2c, é inicializado. Os pinos de SDA e SCL são setados como i2c e, além disso, configurados como pull-up, a fim de manter os sinais em nível alto, quando não estão sendo acionados pelo barramento. Depois, a área total de renderização é calculada, com o auxílio das constantes disponíveis no driver do display $ssd1306$. Por fim, o display é zerado com a função $memset$.

```
int main() {
  stdio_init_all();

  // Delay para o usuario abrir o monitor serial...
  sleep_ms(1000);

  // Configurar Wi-Fi
  if (cyw43_arch_init()) {
      printf("Falha ao inicializar Wi-Fi.\n");
      return -1;
  }
  cyw43_arch_enable_sta_mode();

  const char *ssid = "nome sua rede wi-fi";
  const char *password = "senha da rede";
  printf("Conectando ao Wi-Fi...\n");
  if (cyw43_arch_wifi_connect_timeout_ms(ssid, password,
     CYW43_AUTH_WPA2_AES_PSK, 10000)) {
      printf("Falha ao conectar ao Wi-Fi.\n");
      return -1;
  }
  printf("Wi-Fi conectado!\n");

  // Preparacao do ADC.
  printf("Preparando ADC...\n");

  adc_gpio_init(MIC_PIN);
  adc_init();
  adc_select_input(MIC_CHANNEL);
  adc_set_clkdiv(ADC_CLOCK_DIV);

  printf("ADC Configurado!\n\n");

  gpio_init(GREEN_LED_PIN);
  gpio_set_dir(GREEN_LED_PIN, GPIO_OUT);
  gpio_put(GREEN_LED_PIN, 0);

  gpio_init(BLUE_LED_PIN);
  gpio_set_dir(BLUE_LED_PIN, GPIO_OUT);
  gpio_put(BLUE_LED_PIN, 0);


  gpio_init(RECORD_BTN);
```

```
  gpio_set_dir(RECORD_BTN,GPIO_IN);
  gpio_pull_up(RECORD_BTN);

// Configurar interrupcao no botao (queda de borda)
  gpio_set_irq_enabled_with_callback(RECORD_BTN, GPIO_IRQ_EDGE_FALL,
     true, &gpio_callback);


  sleep_ms(5000);

  // conectando ao servidor
  connect_to_server();


  gpio_put(BLUE_LED_PIN,1);

  // Inicializacao do i2c
    i2c_init(i2c1, ssd1306_i2c_clock * 1000);
    gpio_set_function(I2C_SDA, GPIO_FUNC_I2C);
    gpio_set_function(I2C_SCL, GPIO_FUNC_I2C);
    gpio_pull_up(I2C_SDA);
    gpio_pull_up(I2C_SCL);

    // Processo de inicializacao completo do OLED
    ssd1306_init();

    // Preparar area de renderizacao para o display
    struct render_area frame_area = {
        start_column : 0,
        end_column : ssd1306_width - 1,
        start_page : 0,
        end_page : ssd1306_n_pages - 1
    };

    calculate_render_area_buffer_length(&frame_area);

    // zera o display inteiro
    uint8_t ssd[ssd1306_buffer_length];
    memset(ssd, 0, ssd1306_buffer_length);
    render_on_display(ssd, &frame_area);

   while(true){
   /*...*/
   }
cyw43_arch_deinit();
return 0;
}
```

Listing 8. Inicialização do firmware

4. Execução do projeto

4.1. Metodologia

Para definir o escopo do projeto, foi levada em consideração a capacidade do hardware disponibilizado pelo programa (isto é, o kit BitDogLab). Assim, buscou-se implementar

uma ideia envolvendo vários dos componentes presentes na placa, inclusive o módulo Wi-fi próprio da Raspberry Pi Pico W.

Em relação às escolhas de software, utilizaram-se as libs disponíveis no SDK em C para o firmware. Já para o servidor, tendo em vista a necessidade de utilizar um modelo de IA, escolheu-se Python para sua implementação, devido à existência da biblioteca Tensorflow.

Sobre a revisão bibliográfica realizada, foi utilizada como base de publicações a plataforma Google Scholar. As palavras-chave de busca foram : "sound classification deaf hearing impaired iot".

Acerca do desenvolvimento do projeto, foi utilizada a IDE Visual Studio Code. O monitor serial do próprio VsCode foi empregado para depurar a Raspberry. Além disso, a plataforma github foi utilizada para o controle de versionamento do projeto.

4.2. Resultados

No link `https://drive.google.com/file/d/1nKu7EKa3laN9QvD6QBHCi9zgaqqu2Leo/view?usp=sharing`, há um vídeo contendo uma demonstração do projeto. Pode-se perceber que, no vídeo, apesar do ruído de fundo, o modelo pôde identificar o som principal, que é a fala. Além disso, o nome "silence"no display refere-se ao último som percebido, que foi silêncio.

Foram feitos diversos testes de validação da precisão do classificador, com diferentes áudios, por exemplo: fala humana, música, animal, gato, buzina, assovio, choro. Essas classes foram todas corretamente retornadas. Particularmente, percebeu-se que a ação "cantar"é classificada em geral como fala ou música; essa limitação pode ser atribuída à curta duração do áudio(3s).

Em relação à latência do sistema, o tempo total, incluindo envio e classificação do áudio, é de cerca de 30s. É uma performance não muito satisfatória, e o gargalo está no envio de dados pela Raspberry, que dura cerca de 28s . De fato, para simplificar o desenvolvimento, programou-se a comunicação em modo pooling, de forma que os pacotes são enviados sequencialmente, com um delay de 500ms entre cada um, para evitar falhas no buffer e garantir a correta recepção no servidor. Além disso, o buffer de envio tem um tamanho de apenas 2400 amostras, devido aos limites do tamanho do pacote TCP. Esses fatores contribuem para aumentar o delay.

5. Conclusão

O sistema GuessWhat é um protótipo de auxílio aos deficientes auditivos no seu dia a dia. Ele fornece uma classificação satisfatória de sons comums do dia a dia, o que pode auxiliar na vida de seus usuários.

Para melhorar a latência do sistema, trabalhos futuros podem explorar outras abordagens de comunicação, utilizando o modo de sistema operacional fornecido pelo RP2040, bem como middlewares a exemplo do nanopb. Além disso, pode-se desenvolver uma aplicação mobile para se comunicar com o servidor e receber as notificações dos eventos de áudio; assim, eventos importantes, como campainha, podem receber um alerta especial.

Referências

An, J.-H., Koo, N.-K., Son, J.-H., Joo, H.-M., and Jeong, S. (2022). Development on deaf support application based on daily sound classification using image-based deep learning. *JOIV: International Journal on Informatics Visualization*, 6(1-2):250–255.

Karunaratna, S. (2021). *IoT for elderly care: Design and development of sound event recognition system for hearing-impaired people*. PhD thesis, General Sir John Kotelawala Defence University.

Mojado, C. V. (2024). Deaf and hard of hearing (dhh) home assistant with iot application and decision support. *Explorer*, 9:2.

Salem, O., Mehaoua, A., and Boutaba, R. (2023). The sight for hearing: An iot-based system to assist drivers with hearing disability. In *2023 IEEE Symposium on Computers and Communications (ISCC)*, pages 1305–1310.