UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
CURSO DE GRADUAÇÃO EM ENGENHARIA ELETRÔNICA

João Paulo Vieira

Desenvolvimento de um sistema de aquisição de áudio via ESP32 para classificação de cenas acústicas

Florianópolis
2024

João Paulo Vieira

Desenvolvimento de um sistema de aquisição de áudio via ESP32 para classificação de cenas acústicas

Trabalho de Conclusão de Curso do Curso de Graduação em Engenharia Eletrônica do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Bacharel em Engenharia Eletrônica.
Orientador: Prof. Richard Demo Souza, Dr.
Coorientador: Prof. Walter Antonio Gontijo, Me.

Florianópolis
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Vieira, João Paulo
Desenvolvimento de um sistema de aquisição de áudio via ESP32 para classificação de cenas acústicas / João Paulo Vieira ; orientador, Richard Demo Souza, coorientador, Walter Antônio Gontijo, 2024.
51 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Eletrônica, Florianópolis, 2024.

Inclui referências.

1. Engenharia Eletrônica. 2. Aquisição de áudio. 3. Sistemas embarcados. 4. Cenas acústicas. 5. Aprendizado de máquina. I. Demo Souza, Richard. II. Gontijo, Walter Antônio. III. Universidade Federal de Santa Catarina. Graduação em Engenharia Eletrônica. IV. Título.

João Paulo Vieira

Desenvolvimento de um sistema de aquisição de áudio via ESP32 para classificação de cenas acústicas

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Engenharia Eletrônica" e aprovado em sua forma final pelo Curso de Graduação em Engenharia Eletrônica.

Florianópolis, 15 de Outubro de 2024.

Documento assinado digitalmente
Daniela Ota Hisayasu Suzuki
Data: 21/10/2024 10:18:10-0300
CPF: ***.397.389-**
Verifique as assinaturas em https://v.ufsc.br

Profa. Daniela Ota Hisayasu Suzuki, Dra.
Coordenadora do Curso

Banca Examinadora:

Documento assinado digitalmente
WALTER ANTONIO GONTIJO
Data: 21/10/2024 10:44:50-0300
CPF: ***.287.996-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Walter Antônio Gontijo, Me
Coorientador

Documento assinado digitalmente
Eduardo Augusto Bezerra
Data: 20/10/2024 14:05:31-0300
CPF: ***.851.577-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Eduardo Augusto Bezerra, Dr.
Avaliador

Documento assinado digitalmente
Eduardo Luiz Ortiz Batista
Data: 20/10/2024 19:25:14-0300
CPF: ***.521.889-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Eduardo Luiz Ortiz Batista, Dr.
Avaliador

Este trabalho é dedicado a minha namorada, Sarah, aos meus queridos amigos Brenda e Vitor Hugo e a toda minha família.

AGRADECIMENTOS

Gostaria de agradecer, inicialmente, o meu coorientador Walter Gontijo, que me ensinou muito nos dois anos que estive dentro do LINSE. Esse trabalho foi desenvolvido ao longo de um ano inteiro sob a orientação dele, e chegamos num resultado muito bom! Agradeço também o meu orientador Richard, um professor que admiro e me inspira muito a continuar na academia, uma referência para mim no Ensino em Engenharia. A meu pai e mãe, Afonso e Silvia, que estiveram me apoiando nessa jornada da graduação por esses cinco anos, obrigado por acreditarem em mim! A minha companheira, Sarah, obrigado por compartilhar a vida comigo e me acompanhar em todos os processos! Por fim, aos meus amigos: eu sou feliz porque vivo ao lado de vocês, obrigado por tudo!

RESUMO

Este trabalho de conclusão de curso descreve o desenvolvimento de um sistema embarcado para aquisição e transmissão de áudio, com o objetivo de classificar cenas acústicas utilizando um modelo de aprendizado profundo. O sistema utiliza a ESP32 como microcontrolador, microfones digitais MEMS para capturar áudio e um cartão SD para armazenamento. Os arquivos de áudio gravados são transmitidos para um servidor web via WiFi e, posteriormente, classificados por uma aplicação em Python que interage com o modelo de aprendizado profundo SED-LINSE, treinado com a arquitetura PaSST (Patchout Fast Spectrogram Transformer). O trabalho detalha o desenvolvimento do sistema, incluindo a configuração dos componentes, a implementação dos protocolos de comunicação e a criação das aplicações web para gerenciamento de dados e classificação. Os resultados demonstraram o bom funcionamento do sistema em testes práticos, com a ESP32 registrando cada etapa por meio de logs na porta serial. A qualidade do áudio gravado foi considerada boa, com taxa de amostragem de 16 kHz e espectrogramas mostrando continuidade nos dados. O trabalho destaca os desafios de integrar aprendizado de máquina em sistemas embarcados com recursos limitados de memória e processamento. A transmissão de dados via WiFi, combinada com o processamento na nuvem, surge como uma solução eficiente para superar essas limitações, permitindo maior flexibilidade e capacidade de processamento sem sobrecarregar o hardware local.

Palavras-chave: Aquisição de áudio. Sistemas embarcados. Aprendizado Profundo. Cenas acústicas.

ABSTRACT

This final project describes the development of an embedded system for audio acquisition and transmission, with the goal of classifying acoustic scenes using a deep learning model. The system utilizes the ESP32 as the microcontroller, digital MEMS microphones to capture audio, and an SD card for storage. The recorded audio files are transmitted to a web server via Wi-Fi and subsequently classified by a Python application that interacts with the SED-LINSE deep learning model, trained with the PaSST (Patchout Fast Spectrogram Transformer) architecture. The project details the system's development, including the configuration of components, implementation of communication protocols, and the creation of web applications for data management and classification. The results demonstrated the system's proper functioning during practical tests, with the ESP32 logging each step via the serial port. The audio quality was considered good, with a sampling rate of 16 kHz, and spectrograms showed continuity in the data. The project highlights the challenges of integrating machine learning in embedded systems with limited memory and processing resources. The use of Wi-Fi data transmission, combined with cloud processing, emerges as an efficient solution to overcome these limitations, allowing greater flexibility and processing capacity without overloading the local hardware.

Keywords: Audio acquisition. Embedded systems. Deep learning. Acoustic scenes.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CHIPEN	Chip Enable
DMA	Direct Memory Access
ESP32	Espressif Systems 32-bit
FPGA	Field-Programmable Gate Array
GND	Ground
GPU	Graphics Processing Unit
HTTP	Hypertext Transfer Protocol
I^2S	Inter-IC Sound Bus
ID	Identification
IoT	Internet of Things
L/R	Left/Right
MCU	Microcontroller Unit
MEMS	Micro-Electro-Mechanical Systems
PC	Personal Computer
ROM	Read-Only Memory
SCK	Serial Clock
SD	Serial Data
SED	Sound Event Detection
SNR	Signal-to-Noise Ratio
SRAM	Static Random Access Memory
TSMC	Taiwan Semiconductor Manufacturing Company
URL	Uniform Resource Locator
USB	Universal Serial Bus
VDD	Voltage Drain Drain
WiFi	Wireless Fidelity
WS	Word Select

LISTA DE SÍMBOLOS

Hz	Hertz
dBA	Decibél ponderado pela curva A
V	Volt
m	Metro
B	Byte
s	Segundo

SUMÁRIO

1 INTRODUÇÃO

Na última década, a Internet das Coisas (IoT) emergiu como um tópico central na tecnologia, impulsionada por aplicações que se tornaram viáveis através de avanços em conectividade, miniaturização eletrônica e processamento computacional. A criação de sistemas embarcados portáteis, de baixo consumo, capazes de realizar comunicação sem fio por protocolos como Zigbee, Bluetooth e WiFi, permitiu o desenvolvimento e a implementação em larga escala de aplicações voltadas para sensoriamento (SWAMY; KOTA, 2020).

Com o crescimento exponencial no volume de dados gerados por milhões de dispositivos, surgiu a necessidade de soluções que realizem algum tipo de processamento automático, como o aprendizado de máquina. Esses métodos possibilitam realizar classificações, segmentações, detecções de irregularidades, entre outros tipos de processamento em tempo real, facilitando aplicações de monitoramento contínuo (SAMIE; BAUER; HENKEL, 2019).

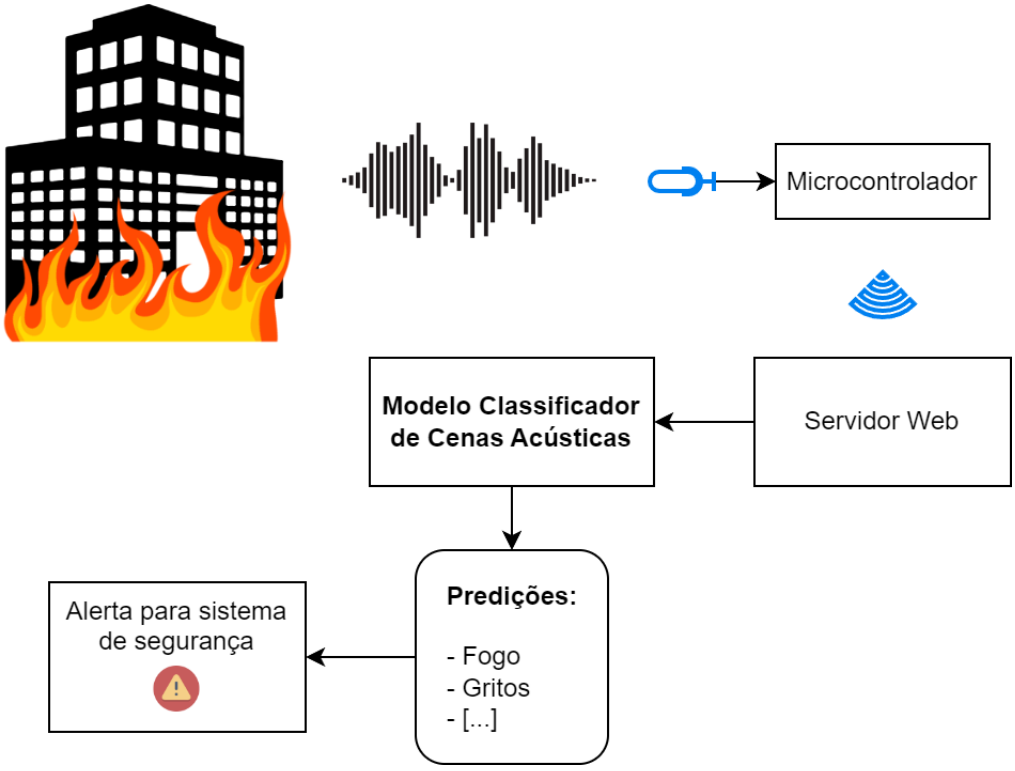
A possibilidade de embarcar diretamente modelos pré-treinados de aprendizado de máquina em sistemas IoT também tem ganhado destaque. Quando tais modelos são confiáveis, o processamento dos dados pode ser feito diretamente no dispositivo, reduzindo a necessidade de enviá-los para processamento na nuvem (BRANCO; FERREIRA; CABRAL, 2019). No entanto, há um desafio em criar sistemas de baixo consumo, portáteis e produzíveis em larga escala que tenham poder computacional para execução em tempo real. Para modelos de aprendizado raso, como a Regressão Logística, por exemplo, já há uma série de implementações em sistemas embarcados, considerando a simplicidade destes modelos (SILVA; ALVES DE SOUZA; BATISTA, 2019). Já para modelos de aprendizado profundo, que atualmente apresentam os melhores desempenhos, a implementação passa a ser mais complexa, devido ao grande número de parâmetros que compõem estes modelos (PUANGPONTIP; HEWETT, 2022).

Há uma série de trabalhos focados na implementação de Redes Neurais Convolucionais (CNNs) em *hardwares* embarcados, como FPGAs ou GPUs (QIU *et al.*, 2016). No entanto, embarcar esses modelos em microcontroladores, como ESP32, é mais desafiador devido às suas limitações de memória, processamento e consumo de energia (SVOBODA *et al.*, 2022). Dessa forma, uma solução viável atualmente consiste em manter os sistemas IoT de baixo consumo transmitindo dados brutos, evitando a necessidade de um pré-processamento complexo (YE *et al.*, 2023).

Entre as várias aplicações que se beneficiam desse tipo de solução, destaca-se a detecção de cenas acústicas (SED) (DANG; VU; WANG, 2017). Modelos de aprendizado profundo, como os baseados em arquiteturas Transformers (PARK; JEONG; LEE, 2021), têm sido amplamente utilizados para esse tipo de tarefa, oferecendo alta precisão em cenários como segurança no trabalho, vigilância e detecção de falhas em ambientes industriais

(HEITTOLA *et al.*, 2013). Quando aplicada em vigilância, por exemplo, a detecção automática de cenas acústicas pode ser utilizada para identificar e alertar eventos que apontam algum risco a segurança das pessoas, como incêndios, tiros, gritos e etc. Já no contexto industrial, pode se identificar algum tipo de falha crítica e/ou anomalia no maquinário que ponha em risco a saúde dos trabalhadores. Isso se deve à capacidade dos modelos de aprendizado profundo de extrair representações de alta qualidade dos sinais de áudio (ZHANG *et al.*, 2022), resultando em um desempenho superior na detecção desses eventos acústicos (KIM; CHUNG, 2022). A Figura 1 exemplifica um caso de uso em que se utiliza um modelo para alertar um incêndio, com base na identificação do som associado ao fogo.

Figura 1 – Exemplo de um caso de uso da detecção automática de cenas acústicas.



Fonte: Autor.

Dentro desse contexto, este trabalho propõe o desenvolvimento de um sistema embarcado com microcontrolador capaz de adquirir áudio em alta qualidade e se comunicar com um servidor na nuvem, de maneira similar a um sistema IoT. O processamento dos dados será realizado externamente, utilizando um modelo de aprendizado profundo para detecção de cenas acústicas, garantindo um sistema eficiente em termos de consumo de energia e capaz de lidar com grandes volumes de dados de forma eficaz.

1.1 OBJETIVOS

Nas seções abaixo estão descritos o objetivo geral e os objetivos específicos deste TCC.

1.1.1 Objetivo Geral

Este trabalho tem como objetivo geral desenvolver um sistema embarcado que viabilize transmissões de sinais de áudio periodicamente para um servidor Web, e criar uma aplicação que se integre a esse servidor e faça classificação de cenas acústicas utilizando um modelo de aprendizado profundo.

1.1.2 Objetivos Específicos

a) Integrar microfones digitais MEMS com a ESP32 para realizar aquisições de áudio via protocolo I2S, e que essas sejam contínuas e de boa qualidade.

b) Integrar um cartão SD com a ESP32 usando o protocolo SPI e paralelizar gravação/aquisição de modo a conseguir gravar o áudio adquirido em memória externa.

c) Conectar a ESP32 com um servidor Web, usando WiFi. Com isso, enviar os arquivos de áudio adquiridos para o servidor.

d) Criar uma aplicação em um servidor Web que processe as requisições realizadas pela ESP32 e salve os arquivos na Cloud.

e) Criar uma aplicação em Python que se integre com um modelo de aprendizado profundo (SED - LINSE) que faça classificações de cenas acústicas.

2 FUNDAMENTOS

Neste capítulo será dada a fundamentação teórica utilizada para o desenvolvimento do sistema proposto. Especificamente, serão abordados tópicos essenciais para sua implementação, cobrindo a tecnologia utilizada para aquisição de áudio (MEMS), os protocolos de comunicação que foram utilizados (I^2S, SPI, HTTP) e os conceitos de cena acústica e modelos de aprendizado profundo.

2.1 MICRO-ELECTRO-MECHANICAL SYSTEMS (MEMS)

A tecnologia MEMS consiste em sistemas micro-eletro-mecânicos, que conectam estruturas mecânicas miniaturizadas a circuitos integrados. Isso fornece uma alternativa interessante a uma série de aplicações envolvendo sensoriamento e instrumentação de modo a permitir que fabricantes possam desenvolver sistemas cada vez menores, incorporando sensores diretamente aos circuitos em seu processo de fabricação, sem que esses elementos sejam discretizados na placa (KEIM, 2020).

No contexto deste trabalho, utilizaram-se microfones digitais tipo MEMS, que funcionam de forma semelhante aos microfones de eletreto, consistindo em um elemento capacitivo que varia conforme a pressão sonora. Os microfones já possuem integrados um circuito de pré-amplificação e conversor analógico-digital, facilitando ao usuário a interface com sistemas microcontrolados (KEIM, 2017).

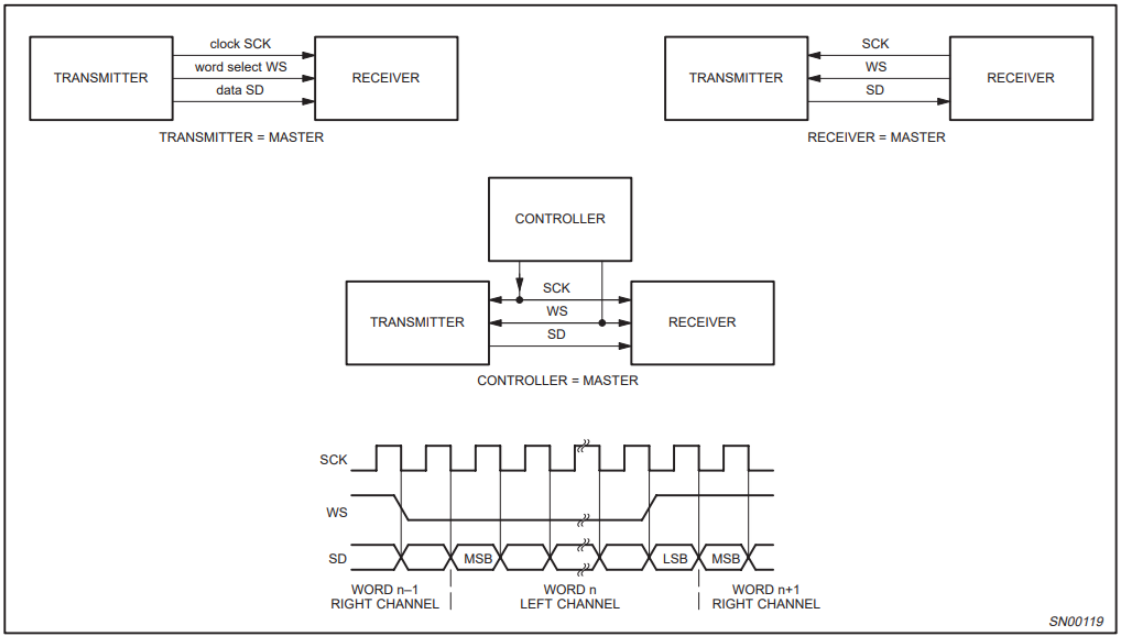
2.2 INTER-IC SOUND (I^2S)

O Inter-IC Sound (I^2S) é um protocolo de comunicação desenvolvido pela Philips Semiconductors, com enfoque em tranmissão de áudio digital entre dispositivos eletrônicos (PHILIPS SEMICONDUCTORS, 1986). O protocolo separa um canal de tranmissão de dados dos canais de controle, permitindo múltiplas formas de interação entre o dispositivo transmissor e o receptor. Três vias são utilizadas, a Serial Data (SD), para transmissão dos dados, a Serial Clock (SCK) para o *clock* de operação e Word Select (WS) para escolha de qual dispositivo enviará dados (utilizado em um sistema estéreo). Na Figura 2, exibe-se um exemplo das múltiplas formas de configuração que podem ser realizadas nesse protocolo, identificando-se que os sinais de controle podem ser enviados pelo transmissor, receptor ou algum circuito externo de controle.

2.3 SERIAL PERIPHERAL INTERFACE (SPI)

O Serial Peripheral Interface é um dos protocolos mais utilizados para realizar interface entre microcontroladores e ICs periféricos, como sensores, ADCs, DACs e outras aplicações (DHAKER, 2018). É um protocolo síncrono, que realiza sincronização dos dados transmitidos baseado na subida ou descida do *clock*. O SPI viabiliza a troca simultânea

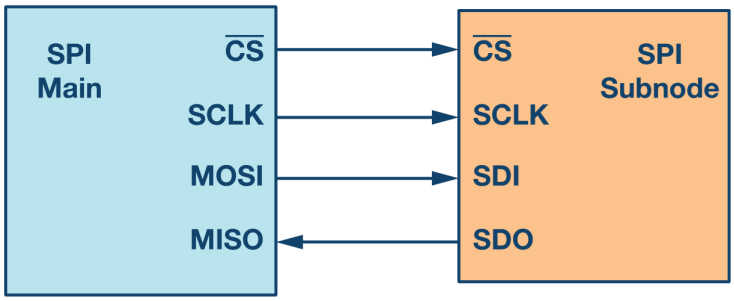
Figura 2 – Possíveis configurações de comunicação I²S e temporização do protocolo.



Fonte: (PHILIPS SEMICONDUCTORS, 1986).

de dados entre dois dispositivos (por exemplo, entre um microcontrolador e uma memória externa), sendo implementado principalmente em uma estrutura de quatro canais: Clock (SCK), Chip Select (CS), Main Out, Subnode In (MOSI) e Main In, Subnode Out (MISO). Na Figura 3, é dado um exemplo de comunicação SPI entre dois dispositivos, o principal e um *subnode* (subnó).

Figura 3 – Comunicação entre dois dispositivos utilizando o SPI.



Fonte: (DHAKER, 2018).

Na Figura, identifica-se que os sinais de controle como SCK e CS são enviados exclusivamente pelo dispositivo principal, enquanto os dados podem ser transmitidos de um dispositivo para outro pelas vias MISO e MOSI. Para o subnó, o pino MOSI é mapeado em Serial Data In (SDI), e o pino MISO em Serial Data Out (SDO).

2.4 HYPERTEXT TRANSFER PROTOCOL (HTTP)

O Hypertext Transfer Protocol (HTTP) é o principal protocolo utiizado por aplicações para realizar a comunicação via Internet, tendo uma série de usos, sendo o mais famoso a troca de informação entre navegadores *web* e servidores *web* (GOURLEY *et al.*, 2002). A Tabela 1 apresenta os métodos mais utilizados que o protocolo possui.

Tabela 1 – Alguns métodos HTTP comuns

Método HTTP	**Descrição**
GET	Enviar recurso nomeado do servidor para o cliente.
PUT	Armazenar dados do cliente em um recurso nomeado do servidor.
DELETE	Deletar o recurso nomeado de um servidor.
POST	Enviar dados do cliente para uma aplicação gateway no servidor.
HEAD	Enviar apenas os cabeçalhos HTTP da resposta para o recurso nomeado.

Fonte: Adaptado de (GOURLEY *et al.*, 2002).

No contexto deste trabalho, especificamente, utilizam-se os métodos GET e POST. O método GET permite que a aplicação solicite algum tipo de dado do servidor. Quando acessa-se uma página *web*, por exemplo, é feito um GET, e a página retorna um código em HTML. O método GET também permite, via URL, enviar uma pequena quantia de dados. O método POST, por sua vez, existe especificamente para o envio de dados da aplicação ao servidor. No método, cria-se um cabeçalho que informa qual é a estrutura de dado enviado seguindo a formatação MIME (Multipurpose Internet Mail Extensions). Para o envio de dados binários, por exemplo, utiliza-se a estrutura *application/octet-stream*. Em sequência, são enviados os dados. Assim, o servidor *web* pode utilizar a estrutura MIME recebida para decodificar os dados.

2.5 CENAS ACÚSTICAS

Uma cena acústica é um rótulo que pessoas utilizam para descrever algum tipo de evento reconhecível pelo som. Esse rótulo permite que as pessoas compreendam o conceito por trás dele e associem essa cena (evento) a outras conhecidas (HEITTOLA *et al.*, 2013). As cenas acústicas podem representar diretamente algum evento, como o som de uma fala, sirene ou chuva, por exemplo.

Existe um campo de pesquisa voltado a analisar como pode ser realizada uma análise computacional de cenas acústicas, realizando algum tipo de processamento que identifique a cena com base em sons (ROUAT, 2008). A esse processo, dá-se o nome de Detecção Automática de Cenas Acústicas, comumente abordado na literatura como Sound Event Detection (SED). Há uma série de aplicações para a SED, como vigilância, monitoramento não invasivo em saúde, detecção de falha em ambiente industrial (HEITTOLA *et al.*, 2013),

segurança do trabalho (NERI *et al.*, 2022), detecção de anomalias (MNASRI; ROVETTA; MASULLI, 2022), dentre outros.

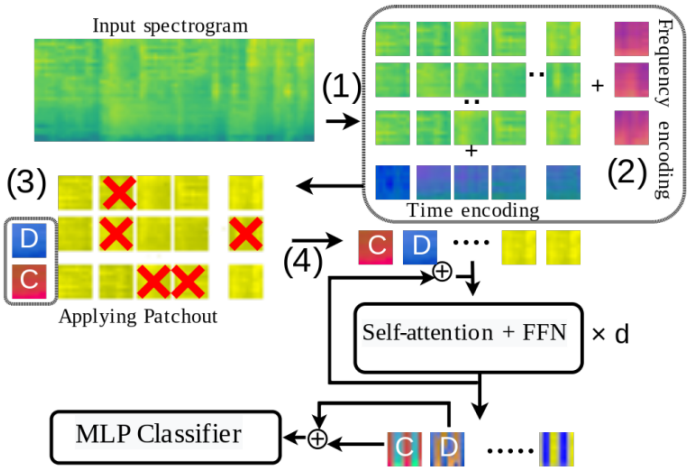
Atualmente, as técnicas que possuem melhor performance para detecção de cenas acústicas se baseiam em modelos de aprendizado profundo (NERI *et al.*, 2022), sendo descritas de forma detalhada em (MESAROS *et al.*, 2021).

2.6 MODELOS DE APRENDIZADO PROFUNDO

O aprendizado profundo é uma sub-área do aprendizado de máquina que extrai uma representação dos dados de entrada através de camadas sucessivas de transformação nos dados e utiliza Redes Neurais Artificais para encontrar a melhor relação entre a representação obtida (MISHRA; REDDY; PATHAK, 2021). Os modelos de aprendizado profundo se diferenciam dos métodos clássicos por sua profundidade e alta complexidade. Com esses modelos, busca-se realizar algum tipo específico de tarefa, seja classificação, segmentação, detecção de objetos, processamento de linguagem natural, reconhecimento de padrões, dentre outros.

No contexto deste trabalho, utiliza-se um modelo de aprendizado profundo com arquitetura desenvolvida diretamente para extração de representações de espectrogramas, a Patchout faSt Spectrogram Transformer (PaSST) (KOUTINI *et al.*, 2022). A PaSST baseia-se em trabalhos notórios da literatura (GONG; CHUNG; GLASS, 2021), que adaptaram arquiteturas da visão computacional para o processamento de áudio, utilizando espectrogramas como entrada das redes. Na Figura 4, é dado um diagrama que descreve o processamento do espectrograma de áudio pela arquitetura.

Figura 4 – Diagrama que descreve a arquitetura PaSST.



Fonte: (KOUTINI *et al.*, 2022).

Na figura, destaca-se a aplicação do Patchout, técnica que consiste em retirar

partes do espectrograma para reduzir o custo computacional no treinamento, e que ao mesmo tempo atua como regularização, ajudando o modelo a melhorar sua capacidade de generalização. É feita, também, uma codificação posicional - que informa a ordem dos elementos para o modelo - separada para tempo e frequência. Antes dos dados serem enviados para um classificador, passam por camadas de Auto-Atenção, que permitem que modelos processem informações contextualmente dentro da sequência. Uma descrição mais detalhada da arquitetura é feita em (KOUTINI *et al.*, 2022).

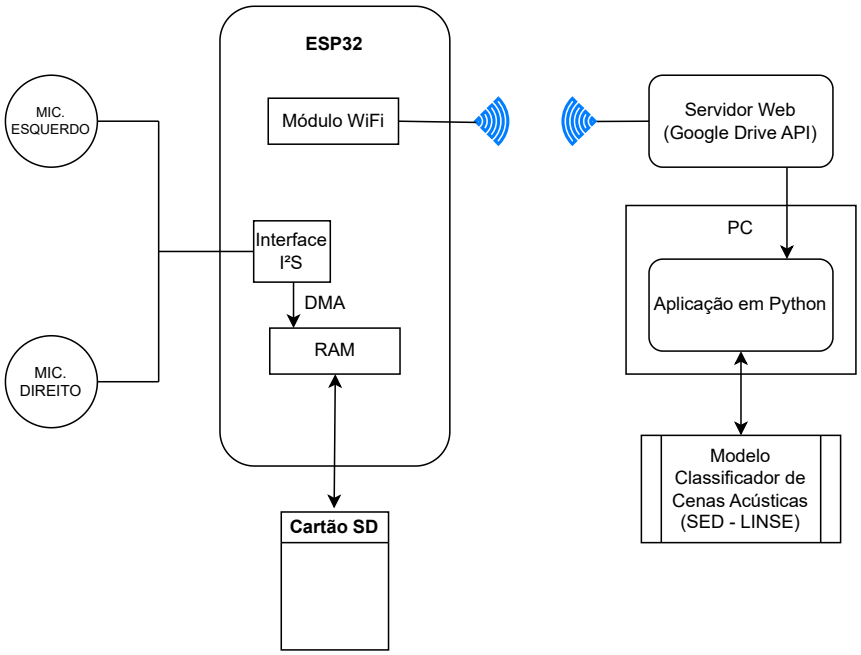
3 MATERIAIS E MÉTODOS

Neste capítulo serão apresentadas a visão geral e específica do sistema proposto, destacando seus componentes, blocos construtivos, configurações, especificações e limitações. Além disso, no Anexo A está disponível a Tabela 3, contendo as principais funções que serão expostas nas Seções 3.2.2, 3.2.3, 3.2.4 e 3.2.5 a seguir. Por fim, o código fonte do projeto estará disponibilizado na plataforma Github[1].

3.1 VISÃO GERAL DO SISTEMA

O sistema proposto consiste, numa descrição simples, em gravar e transmitir sons utilizando *hardware* de baixo consumo e classificá-los com um modelo de aprendizado profundo. O processo de gravação e transmissão é iniciado pela ESP32, que lida com a captura do áudio estéreo por meio de microfones digitais, armazena esses dados em um cartão SD e, em seguida, envia-os para a nuvem via um servidor *Web*. Posteriormente, um computador acessa esses dados armazenados e realiza a classificação de cenas acústicas do áudio utilizando uma aplicação que se integra à API do modelo SED - LINSE. Como o foco desse trabalho está no desenvolvimento de aplicações voltadas à aquisição e transmissão de áudio, não será feita uma descrição em detalhes do modelo utilizado (SED), que foi desenvolvido em outro escopo. A Figura 5 apresenta o diagrama de blocos do sistema.

Figura 5 – Diagrama de blocos do sistema.



Fonte: Autor.

[1] Código fonte: `https://github.com/jpvieir/audiorecording-esp32`

Na Figura 5, observa-se a conexão entre ambos os microfones ao ESP32, que utiliza de sua interface I^2S para fazer a aquisição de áudio, enviando as amostras diretamente à memória (DMA).

3.2 VISÃO ESPECÍFICA DO SISTEMA

A seguir, serão descritos os componentes e explicados os blocos construtivos que compõem o sistema, estes últimos dividindo-se em Configuração, Aquisição, Gravação, Transmissão e Classificação.

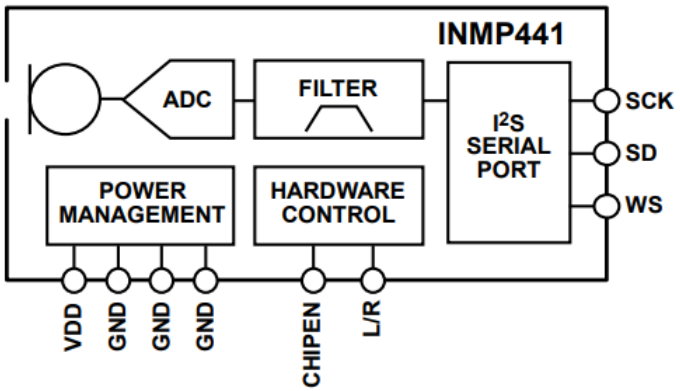
3.2.1 Componentes

Nesta seção serão descritos os componentes eletrônicos que compõem o sistema, sendo eles: dois microfones digitais tipo MEMS INMP441, módulo adaptador de cartão SD MH-SD, cartão SD, microcontrolador ESP32 WROOM e computador.

3.2.1.1 Microfones INMP441

O INMP441 é um microfone digital do tipo MEMS, de baixa potência, alta performance, omnidirecional e saída digital. Ele consiste num sensor MEMS, um bloco para tratamento do sinal, um conversor analógico-digital, filtros *anti-aliasing* e uma interface I^2S de 24 bits, permitindo fácil comunicação com microcontroladores. De acordo com o *datasheet* do microfone, ele possui uma resposta em frequência plana numa faixa de 60 Hz - 15 kHz, além de uma SNR de 61 dBA, o que viabiliza gravação de áudio em alta qualidade, captando a maior parte do espectro audível, com pouca interferência de ruído. A seguir, na Figura 6, é apresentado o diagrama de blocos do microfone.

Figura 6 – Diagrama de blocos do microfone INMP441.



Fonte: (INVENSENSE, 2014).

Na Figura 6, identifica-se a conversão e pré-processamento do sinal, que é enviado para a porta *Serial Data* (SD). As portas *Serial Clock* (SCK), *Word Select* (WS) e *Left/Right* (L/R) viabilizam a utilização do protocolo, tendo seus sinais recebidos do dispositivo que está realizando a comunicação (ESP32). Há também portas de alimentação e uma *Chip Enable* (CHIPEN), que permite ao usuário ligar ou desligar o microfone. Por padrão, essa porta já vem ativa. Na Figura 7, é exposto o modelo físico do microfone, em que é possível identificar os pinos de alimentação e comunicação.

Figura 7 – Microfone INMP441.

Fonte: (ROBU.IN, 2024).

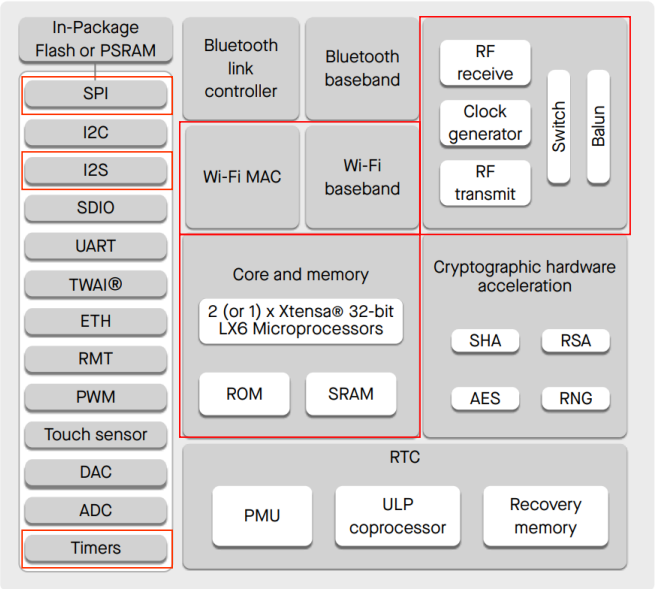
3.2.1.2 Módulo adaptador de Cartão SD

Para realizar a leitura de um cartão SD, utilizou-se o MH-SD Card Module, disponível em (BAÚ DA ELETRÔNICA, 2024). Esse módulo possui 8 pinos, sendo dois deles para o GND, um para alimentação de 3.3 V e outro para 5 V. Os pinos restantes cumprem a função de realizar a interface do cartão SD com a ESP32, sendo eles os pinos MISO, MOSI, SCK e CS, associados ao protocolo SPI. Os pinos MISO e MOSI dizem respeito aos sinais enviados do cartão SD para a ESP32, e da ESP32 para o cartão, respectivamente. O pino SCK, assim como no protocolo I^2S, representa o sinal de clock enviado da ESP32 para o cartão SD, enquanto o CS é um pino de seleção (utilizado quando há múltiplos dispositivos a serem selecionados).

3.2.1.3 ESP32

A ESP32 é uma MCU (*Microcontroller Unit*) integrada com módulos WiFi e Bluetooth que operam em 2.4 GHz, projetada com tecnologia TSMC de litografia 40 nm. A placa viabiliza o desenvolvimento de uma série de aplicações em cenários de sistemas embarcados, permitindo a aquisição de sinais analógicos e digitais, comunicação wireless com outros dispositivos, ao mesmo tempo que um baixo consumo de energia (ESPRESSIF SYSTEMS, 2023). Escolheu-se a ESP32 para esse projeto por possuir suportes aos protocolos de comunicação SPI, I^2S, além de conexão à Internet via Wifi. Na Figura 8, é dado o diagrama de blocos que compõe a MCU, destacando-se o que foi utilizado no sistema proposto.

Figura 8 – Diagrama de blocos da ESP32.



Fonte: Adaptado de (ESPRESSIF SYSTEMS, 2023).

O modelo escolhido para este sistema é o ESP-WROOM-32, que se destaca por sua memória ROM de 448 kB e SRAM de 520 kB. Embora essas especificações sejam suficientes para diversas aplicações, elas representam um desafio significativo ao lidar com gravações de áudio, devido ao grande volume de dados gerado. Com 520 kB de SRAM, utilizando uma taxa de amostragem de 16 kHz, dois canais e amostras de 32 bits, é possível armazenar apenas cerca de meio segundo de áudio, por exemplo. Isso ressalta a necessidade de um gerenciamento eficiente da memória no dispositivo para permitir a captura e o processamento contínuo dos dados.

3.2.1.4 Computador

Por fim, o último componente que integra este sistema é um computador (ou *laptop*), que possui duas funções: alimentar a ESP32 via conexão USB e executar uma aplicação em Python. Essa aplicação é responsável por fazer o *download* de arquivos disponíveis no Google Drive e enviá-los para a API do modelo SED - LINSE, que recebe arquivos de áudio e devolve classificações de cenas acústicas.

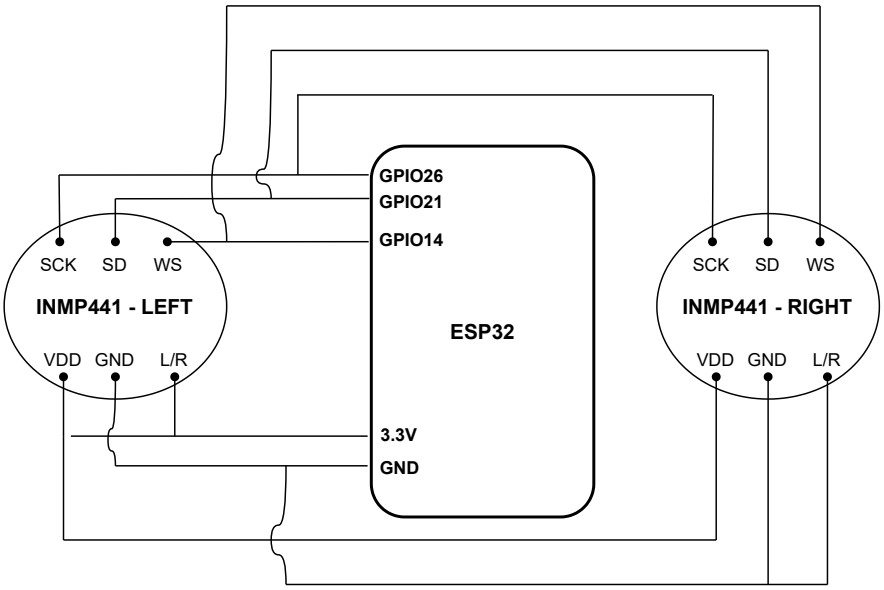
3.2.2 Configuração

Ao inicializar-se a placa, passa-se por um processo de configuração, que envolve montar o cartão SD (o que é feito através das bibliotecas `FS`, `SPI` e `SD`, disponibilizadas para a ESP32), conectar-se a uma rede WiFi (com credenciais configuradas no código) e configurar o módulo I^2S da placa para aquisição.

3.2.3 Aquisição

A aquisição de áudio é feita através dos microfones digitais INMP441, que se comunicam com a ESP32 via protocolo I^2S. Para utilizar o microfone, conectam-se os pinos de alimentação (VDD e GND) e de transmissão de dados (L/R, SCK, SD, WS) a ESP32, conforme mostra a Figura 9.

Figura 9 – Conexões dos microfones INMP441 esquerdo e direito com a ESP32.



Fonte: Autor.

Na Figura, identifica-se que ambos os microfones estão conectados aos mesmos pinos de clock (SCK), transmissão de dados (SD), e seleção da palavra (WS), diferindo pelos pinos L/R. O microfone esquerdo tem seu terminal L/R conectado ao VDD (indicando que está no canal esquerdo) e o direito ao GND (indicando que está no canal direito).

Para adquirir os sinais, é necessário configurar a ESP32 com uma biblioteca que dê suporte ao protocolo utilizado (I^2S). Neste projeto, utilizou-se a *esp32-audio* (GREENING, 2020), que facilita a interface, permitindo configurar diversos parâmetros da aquisição: frequência de amostragem, número de bits por amostra, número de *buffers*, tamanho de *buffer*, se a ESP32 está recebendo/transmitindo e etc.

Dada a importância de fornecer entradas ricas em informação ao modelo utilizado (SED - LINSE), que foi treinado com faixas de áudio amostradas a 32 kHz, buscou-se maximizar a taxa de amostragem neste projeto. Optou-se por utilizar uma frequência de 16 kHz, com palavras de 32 bits e 32 *buffers* de tamanho 1024 bits. Esses parâmetros foram definidos a partir de testes exaustivos para garantir a maior taxa de transmissão de dados possível, assegurando o funcionamento contínuo da aquisição e gravação, sem *overflow* na memória RAM da ESP32.

Um detalhe a se considerar é que o microfone INMP441, de acordo com seu *datasheet*, fornece até 24 bits por amostra. No entanto, testando diferentes configurações da biblioteca suporte à I²S, como palavras de 24 ou 16 bits, obtiveram-se resultados inconsistentes. Em ambos os casos citados, ao se realizar a leitura dos dados, o *buffer* não era preenchido. O único resultado consistente, e confiável, foi obtido ao se configurar palavras de 32 bits, que na prática fez com que cada vetor associado a uma amostra (de tamanho 32 bits) tivesse seus 16 bits mais significativos preenchidos. Isso fez necessário um deslocamento nos bits pra gravação adequada do áudio, o que será explicado na Seção 3.2.3.1.

A Tabela 2 resume os parâmetros configurados para comunicação I²S entre os dois microfones e a ESP32.

Tabela 2 – Parâmetros da configuração do protocolo I²S.

Parâmetro	**Valor**
Modo de operação	RX - Receiver
Taxa de amostragem	16 kHz
Número de bits por amostra	32
Número de buffers	32
Tamanho do buffer	1024 bits
Formato do canal	Left/Right - Estéreo

Fonte: Autor.

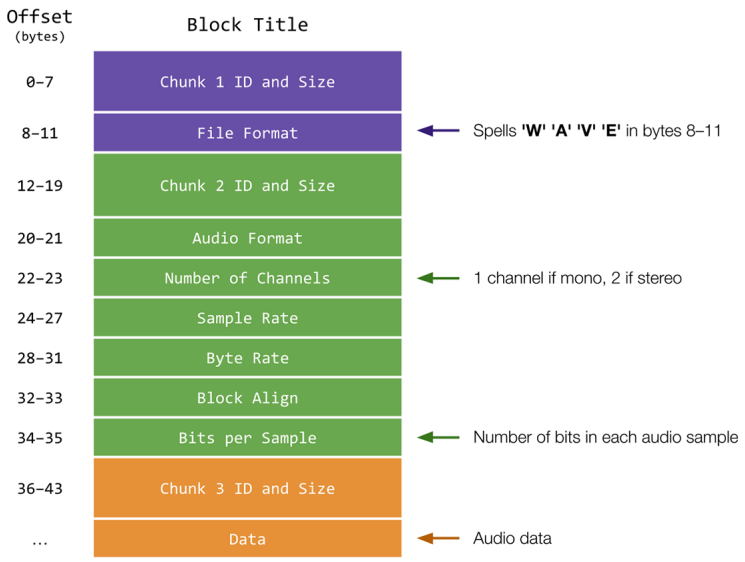
3.2.3.1 Gravação

Para gravar o áudio, aloca-se dinamicamente na memória um *buffer* de 1024 posições de 64 bits (totalizando 8 kiB), permitindo que a cada leitura sejam lidas 2048 amostras (1024 para cada canal). Após a alocação do *buffer*, cria-se um arquivo no cartão SD e é escrita um cabeçalho no formato Wave, de modo que o arquivo possa ser lido posteriormente. Para escrita do cabeçalho, foi criada uma estrutura que contém todos os dados que definem um arquivo Wave, conforme exemplifica a Figura 10.

Para realizar a leitura da porta I²S, utiliza-se a função de leitura (`read`) definida na biblioteca I²S. Nessa função, todos os dados lidos são salvos no *buffer* de amostras. Para gravá-lo no cartão SD, executa-se a função `read` e são realizados deslocamentos nos bits das amostras originais, pois, apesar de configurado para ter palavras de 32 bits, o microfone envia apenas 16 bits para cada amostra. Assim, os dados são escritos no cartão. O processo descrito é ilustrado na Figura 11.

Após a gravação, os *buffers* alocados são liberados da memória (de modo a garantir que não haja *overflow*). Observou-se que durante a gravação, é necessário manter o WiFi desligado, pois há conflito entre as funções de interrupção (I²S e WiFi).

Figura 10 – Cabeçalho de um arquivo Wave.



Fonte: (HARVARD UNIVERSITY, 2023).

3.2.3.2 Temporização

Permite-se ao usuário configurar parâmetros como o tempo de cada gravação e o tempo entre aquisições, alterando variáveis antes da compilação do código. Para correta identificação da data e hora exatas em que é gravado, utiliza-se a biblioteca `NTPClient`. A biblioteca viabiliza, dado que a placa esteja conectada à Internet, que seja adquirida a data atual, o que é utilizado para nomear os arquivos gravados. Desse modo, tem-se, para cada arquivo gravado, uma estrutura como: "D-Ano-Mês-Dia-T-Hora-Minuto-Segundo.wav".

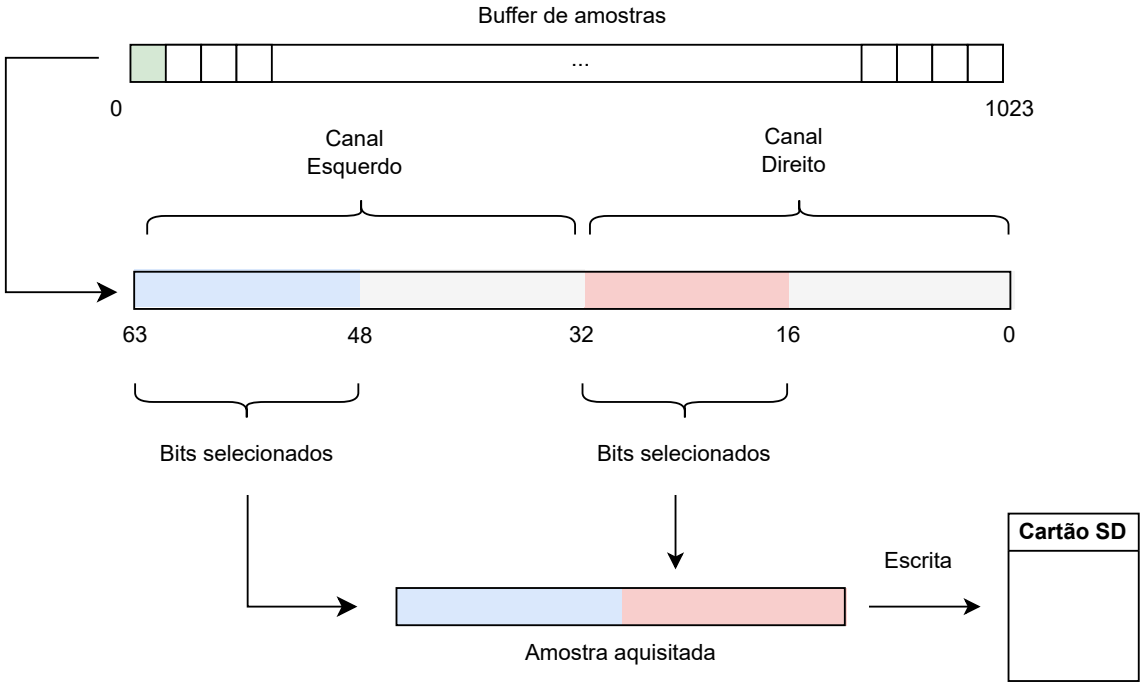
3.2.4 Transmissão do áudio

Para realizar o envio das gravações para um servidor Web, utiliza-se a ESP32 e o Google AppScript, plataforma integrada à API do Google Drive. Através da AppScript, gera-se um endereço (denominado URL de implantação) para o qual a ESP32 pode enviar HTTP Gets e POSTs, sendo assim possível realizar a transmissão dos dados. A seguir, serão descritos os algoritmos desenvolvidos para as duas plataformas (ESP32 e AppScript), explicando os detalhes da comunicação realizada.

3.2.4.1 ESP32 - Transmissão dos dados

Para o envio do arquivo, desenvolveu-se a função `sendBinaryDataToAppScript`. Nela, inicialmente é feito um HTTP Get, enviando ao servidor o nome do arquivo e seu tamanho, para que posteriormente seja checado se o arquivo recebido equivale (em tamanho) ao que foi enviado. Para garantir um resultado positivo no GET, são feitas até

Figura 11 – Processo de aquisição e escrita das amostras no cartão SD.



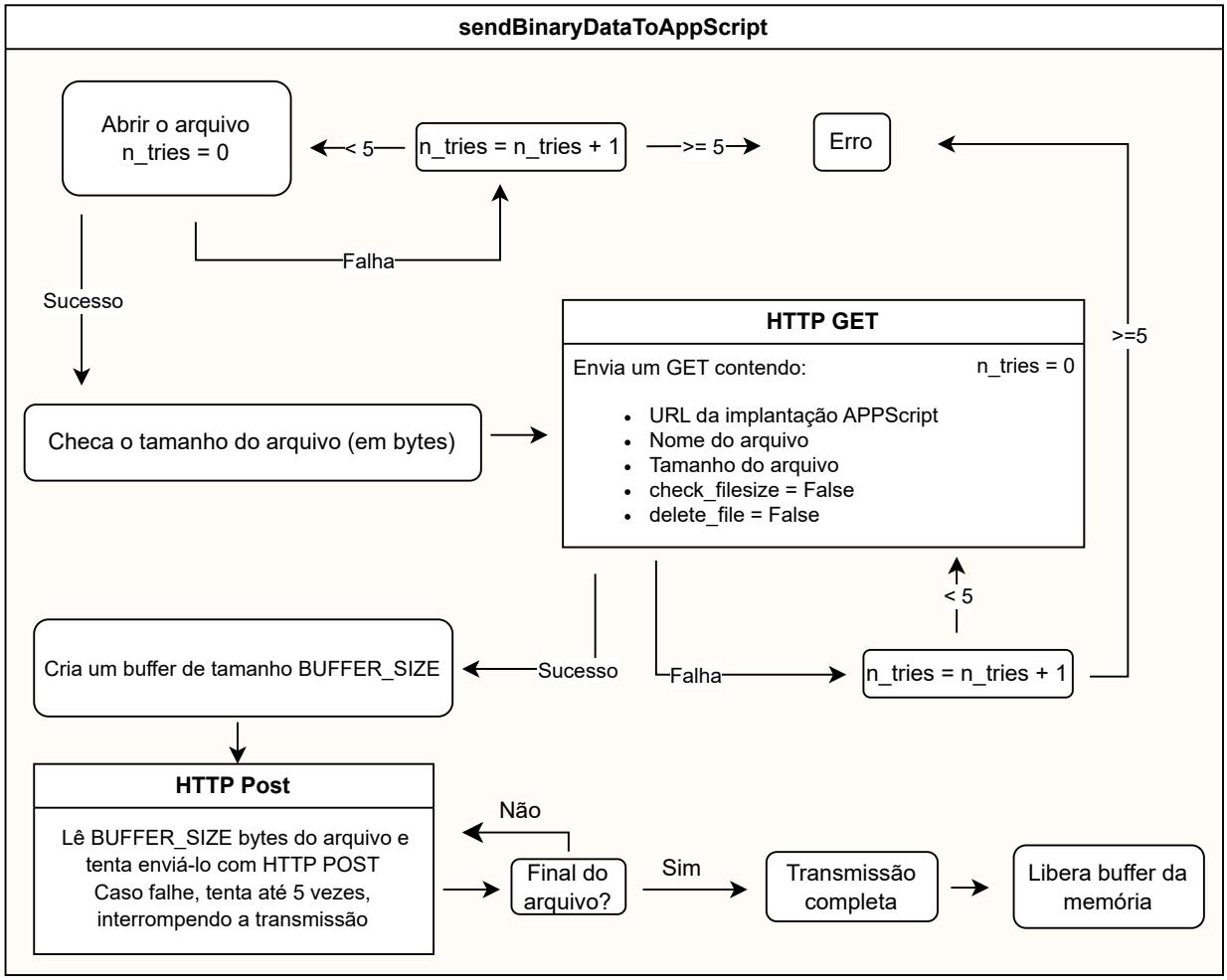
Fonte: Autor.

cinco tentativas, e caso não seja possível enviar o *filename*, é encerrada a operação. Esse mesmo procedimento é também realizado para outras funções que dependem do WiFi, como os HTTP POSTs, por onde são enviadas as amostras, e as funções de controle, como `checkFileSize` e `deleteFile`.

Feito o envio das informações básicas do arquivo ao servidor, é definido um *buffer* (utilizado para armazenar os dados lidos do cartão SD) e o tipo de dado a ser enviado (binário, de formato `octet-stream`). Após isso, são enviados os dados em pacotes de tamanho BUFFER_SIZE = 4 kB, através de HTTP POSTs. Realiza-se, para cada pacote lido, a tentativa de envio ao servidor. Esse processo é repetido até que o arquivo tenha sido enviado por completo. Caso o servidor retorne um erro na recepção de um *buffer*, são feitas até cinco tentativas. Por fim, libera-se o *buffer* da memória, fecha-se o arquivo, e desliga-se a conexão HTTP. Todo o procedimento descrito é ilustrado no diagrama da Figura 12.

Após o envio completo do arquivo, executa-se a função `checkFileSize`, que envia um HTTP GET para o servidor de modo a identificar se o tamanho do arquivo salvo no servidor é igual ao tamanho do arquivo no cartão SD, realizando-se uma checagem de equivalência entre ambos. Nessa função, envia-se o parâmetro *check_filesize* (presente na URL) como *true*, o que fará com o que o servidor execute uma função para comparar os tamanhos dos arquivos. Caso o servidor retorne *false*, a ESP32 chama a função `deleteFile`, que

Figura 12 – Diagrama ilustrando o algoritmo da função `sendBinaryDataToAppScript`.



Fonte: Autor.

realiza outro GET, agora com um comando para deletar o arquivo (parâmetro *delete_file* = true). Desse modo, o programa tenta novamente enviar o arquivo, até um limite máximo de tentativas (cinco). Nos Quadros 1 e 2, respectivamente, são definidas as variáveis de controle utilizadas e é dado um exemplo da estrutura da URL utilizada no HTTP GET. Além disso, na Na Figura 13, ilustra-se o algoritmo de checagem.

3.2.4.2 AppScript - Recepção dos dados

A AppScript é uma plataforma da Google com um ambiente de desenvolvimento que permite a criação de aplicações em JavaScript que se integram com o Google Workspace (Google Drive, G-Mail e etc.). Nesse trabalho, desenvolveu-se um código na AppScript que implementa duas funções: `doGet` e `doPost`, associadas às operações de GET e POST

Quadro 1 – Variáveis de controle.

Variáveis de controle	
check_filesize	**Tipo:** booleano **Função:** Identificar se a aplicação Web deve comparar o tamanho da variável fileSize (enviada no início da função sendBinaryDataToAppScript) com o tamanho do arquivo salvo no Google Drive.
delete_file	**Tipo:** booleano **Função:** Informar à aplicação Web se deve deletar o arquivo salvo, indicando que houve alguma falha na transmissão que comprometeu os dados.

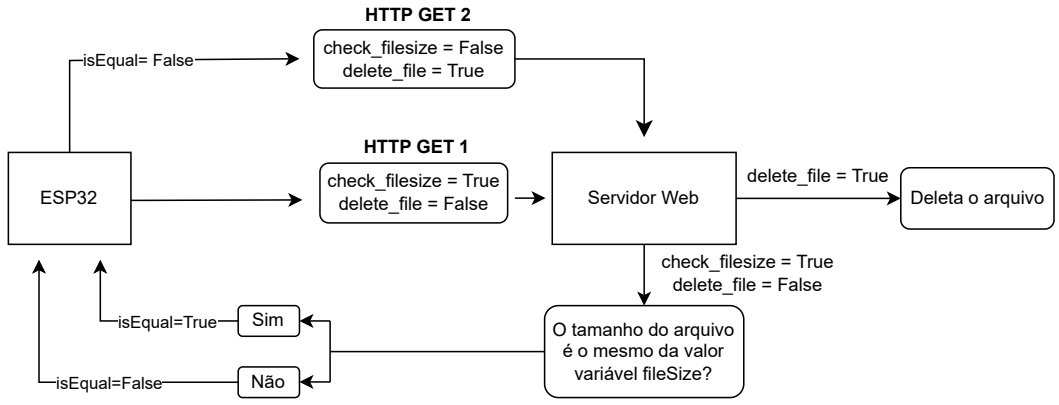
Fonte: Autor.

Quadro 2 – Estrutura da URL - HTTP GET.

Estrutura do HTTP GET
url_implantação + ?filename=NomeDoArquivo + &fileSize=TamanhoDoArquivo + &check_filesize=Booleano + &delete_file=Booleano

Fonte: Autor.

Figura 13 – Algoritmo das funções `checkFileSize` e `deleteFile`.



Fonte: Autor.

realizadas pela ESP32. Essas funções viabilizam a comunicação entre a ESP32 e o Google, e vice-versa, permitindo o envio e checagem dos arquivos de áudio.

A função `doGet` permite com que a ESP32 realize uma requisição para a AppScript, enviando parâmetros e recebendo uma resposta do servidor. Nessa função, os parâmetros passados são: nome do arquivo e seu tamanho (*filename* e *fileSize*), e variáveis condicionais (*checkFileSize* e *deleteFile*) para realizar a checagem do tamanho ou deletar o arquivo (na situação em que os tamanhos diferem). As variáveis *filename* e *fileSize* são declaradas de forma global, de modo que possam ser acessadas na função `doPost`. Após a declaração, são checadas as variáveis condicionais mencionadas anteriormente, conferindo se o código

deve ou não comparar os tamanhos ou deletar um arquivo já salvo. No Quadro 3, é dado um pseudocódigo que exemplifica o procedimento realizado pela função.

Quadro 3 – Pseudocódigo da função doGet.

AppScript - `doGet`

```
def doGet(x):

    global fileSize = x.parametros.filesize
    global filename = x.parametros.filename
    bool checkFileSize = x.parametros.check_filesize
    bool deleteFile = x.parametros.delete_file

    if check_filesize == True:
        compara tamanho de fileSize com o arquivo filename
        if fileSize == tamanho_arquivo:
            return True
        else:
            return False

    if deleteFile == True:
        deleta arquivo de nome filename
```

Fonte: Autor.

A função `doPost` é responsável por receber os dados enviados pela ESP32, em *chunks* de $BUFFER\_SIZE = 4kB$. No envio do primeiro chunk, cria-se o arquivo com o filename enviado, e após isso os próximos *chunks* são adicionados ao arquivo. No Quadro 4 é dado um pseudocódigo que exemplifica o funcionamento da função.

Foram desenvolvidas também, outras funções auxiliares: `concatUint8Arrays`, `getFileIdByName`, `getFileSize` e `deleteFileById`. A função `concatUint8Arrays` concatena os dados recebidos via HTTP POST ao arquivo já existente, e as funções `getFileIdByName`, `getFileSize` e `deleteFileById`, obtêm o ID do arquivo, o tamanho, e deleta o arquivo, respectivamente. Todas essas funções são executadas internamente dentro de `doGet` e `doPost` .

3.2.5 Classificação

Após o envio dos arquivos de áudio ao servidor, utiliza-se uma aplicação para fazer *download* deles para o PC/*laptop* e posteriormente enviá-los para um modelo de classificação. A seguir, será descrito qual foi o modelo utilizado (SED - LINSE) e o algoritmo da aplicação desenvolvida.

Quadro 4 – Pseudocódigo da função doPost.

AppScript - doPost

```
def doPost(x):

    dados_binarios = x.DadosPost.obterBytes()
    lista_de_arquivos = DriveApp.ObterListaDeArquivos()

    if filename not in lista_de_arquivos:
        DriveApp.CriarArquivo(filename)
        DriveApp.AdicionarBytes(dados_binarios)
    else:
        DriveApp.AdicionarDadosAoArquivo(filename,
  dados_binarios)
```

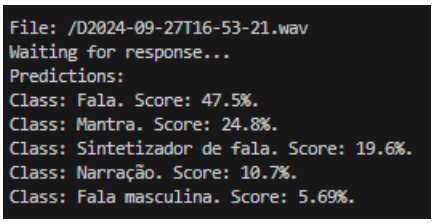
Fonte: Autor.

3.2.5.1 SED - LINSE

O SED é um modelo treinado no Laboratório de Circuitos e Processamento de Sinais (LINSE) da Universidade Federal de Santa Catarina, utilizado para classificação de cenas acústicas. Possui cerca de 500 classes, que variam entre diferentes cenas. O modelo utiliza a arquitetura PaSST (Patchout faSt Spectrogram Transformer) (KOUTINI *et al.*, 2022) e retorna ao usuário um vetor de predições ordenado pela maior probabilidade. Para acessar o modelo, deve-se estar conectado à rede WiFi do LINSE, e realizar um HTTP POST com o arquivo de áudio que deseja-se classificar, sendo retornado pelo servidor as classes preditas e suas probabilidades. O modelo possui uma alta acurácia, além de viabilizar a identificação de mais de uma cena acústica por faixa, considerando que retorna o vetor completo das probabilidades associadas a cada classe.

A Figura 14 mostra um exemplo de classificação utilizando o SED, em que é enviado um arquivo de áudio com uma fala.

Figura 14 – Exemplo de predição retornada pelo modelo SED - LINSE.

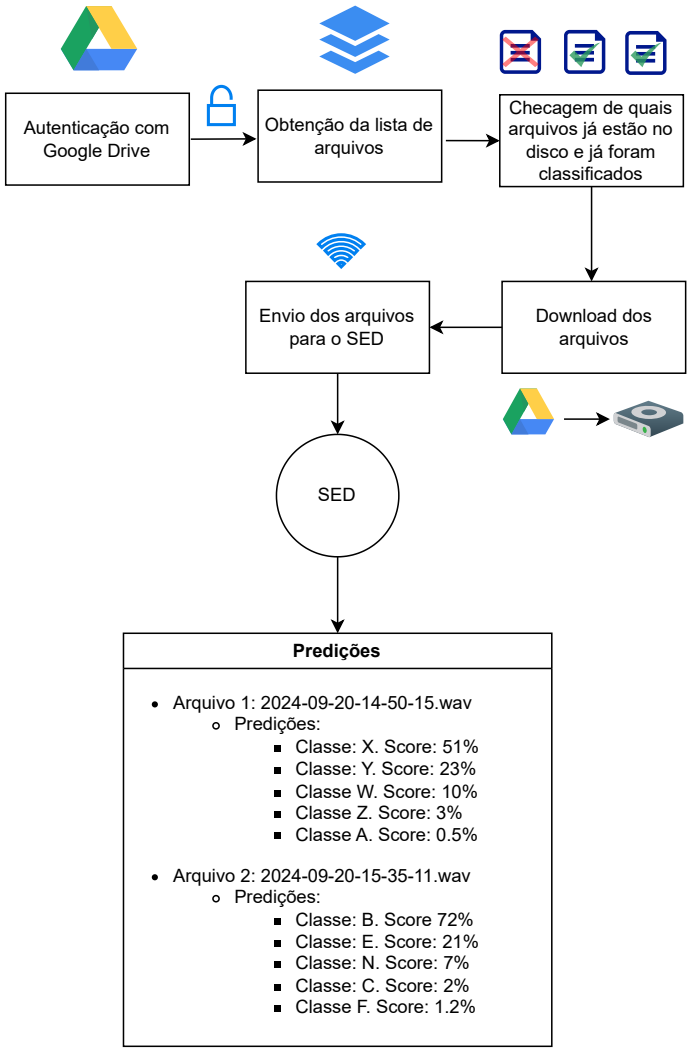


Fonte: Autor.

3.2.5.2 Aplicação

A aplicação responsável por obter as predições foi desenvolvida em Python. Seu funcionamento consiste em realizar uma autenticação com a Google Drive API, baixar os arquivos de áudio que foram enviados pela ESP32 (caso já não estejam no disco), identificar quais foram os arquivos em que já foi feita uma predição, e então enviar uma lista de arquivos para a API do modelo SED, que retorna as predições. Por fim, são salvas as predições em arquivo texto para posterior checagem do usuário (ao mesmo tempo em que exibe-se as na tela). A seguir, na Figura 15, é dado um diagrama que ilustra os passos realizados.

Figura 15 – Diagrama da aplicação em Python.



Fonte: Autor.

Para fazer a autenticação com a Google, utiliza-se a biblioteca `google.auth` disponível para o Python. É necessário, antes da execução do código, baixar um arquivo de autenticação gerado pela Google Drive API (no portal Google Cloud). Esse arquivo

será acessado por funções da biblioteca, e permitirá ao usuário realizar o *download* (ou *upload*) de arquivos do Drive. Observa-se que na primeira execução, abre-se uma janela no navegador pedindo permissão para o acesso. Após isso, é possível realizar a autenticação de forma automática sempre que se executa tal função.

Para obter o nome dos arquivos presentes na pasta do Drive ao qual se deseja acessar, realiza-se um HTTP GET para a Google API, contendo na URL a ID (código de identificação) da pasta e a chave da API disponível na Google Cloud. Desse modo, é possível verificar todos os arquivos em formato Wave que deseja-se baixar, o que será necessário para identificar se há algum arquivo novo (em relação aos já presentes no disco).

Para verificar se é necessário fazer o *download* dos arquivos, compara-se a lista de arquivos presentes no Drive com o que está salvo no disco, através de duas funções: `getFilelistFromDisk` e `compareAndDelete`. Na primeira função, utiliza-se a biblioteca `os` para acessar os arquivos em disco. Na segunda função, compara-se os valores das duas listas de arquivos, removendo os que estão em comum.

Na função `DownloadWavFilesFromFolder`, entrega-se a entrada um dicionário contendo os nomes dos arquivos e suas respectivas IDs (do Drive). A função possui outras duas entradas, sendo elas o diretório no qual deseja-se salvar os arquivos, e uma variável que é retornada pelo processo de autenticação (necessária para o *download*). Desse modo, os arquivos são baixados através da função `googleapiclient.http.MediaIoBaseDownload`.

Na função `Predict`, faz-se um HTTP POST para a API do SED-LINSE, enviando os arquivos Wave do computador. Por fim, o servidor retorna uma predição que permite ao usuário identificar as cenas acústicas mais prováveis para o modelo.

4 RESULTADOS

Nesta seção será demonstrado o funcionamento do sistema, abrangendo os principais passos realizados, sendo eles: montagem do cartão SD, conexão à Internet, aquisição e gravação, envio dos dados à Web, *download* dos arquivos via PC e predição. Observa-se que, para fins de avaliação do correto funcionamento do sistema, utilizou-se de *logs* impressos na porta serial pela ESP32, de modo que o usuário possa acompanhar o passo-a-passo do que está sendo executado, verificando se houve sucesso ou falha na ação. Observa-se que numa implementação prática desse projeto, em um contexto em que a ESP32 fosse alimentada via bateria, não haveria a necessidade de tais *logs* (ou poderia ser realizada uma adaptação que enviasse *logs* via WiFi). Serão expostas, também, as formas de onda e metadados associados aos arquivos gravados, de modo a demonstrar o resultado obtido com a gravação.

4.1 PASSO-A-PASSO: ESP32

Ao se iniciar o sistema, realiza-se a montagem do cartão SD. Posteriormente, conecta-se à Internet para adquirir a data atual (de modo a nomear o arquivo a ser salvo), e então passa-se a aquisição e gravação. Os *logs* associados a esses passos são dados no Quadro 5.

Quadro 5 – *Logs* associados à montagem do cartão SD, aquisição da data e gravação do sinal.

```
[WiFi] Connecting to LINSE-WIFI
[WiFi] WiFi is disconnected
[WiFi] WiFi is connected!
[WiFi] IP address: 192.168.xx.xx

[RECORD] Filename: /D2024-09-27T14-23-46.wav
[SDCARD] SD Card Type: SDHC
[RECORD] Starting to record in 3...
[RECORD] 2...
[RECORD] 1...
[RECORD] Recording!
[RECORD] Wrote: 188416 bytes.
```

Fonte: Autor.

No quadro, mostra-se ao usuário uma contagem de três segundos, indicando que será iniciado o processo de gravação. Com ela completa, imprime-se o tamanho total do arquivo.

Após isso, reconecta-se à Internet e abre-se o arquivo que a ser enviado. Realiza-se inicialmente um HTTP Get para enviar os parâmetros necessários à AppScript, como nome e tamanho do arquivo. Os passos descritos são registrados pelos *logs* do Quadro 6.

Quadro 6 – *Logs* associados ao envio do tamanho e nome do arquivo.

```
[WiFi] Connecting to LINSE-WIFI
[WiFi] WiFi is disconnected
[WiFi] WiFi Status: 3
[WiFi] WiFi is connected!
[WiFi] IP address: 192.168.xxx.xx
[SDCARD] Opening file..
[SDCARD] File succesfully opened!
Sending file: /D2024-09-27T14-23-46.wav

[GET] Sending filename and filesize...
[GET] Result code: 302
[GET] Filename sent!
```

Fonte: Autor.

Após isso, é feito o envio completo do arquivo via HTTP Post. Como os arquivos não cabem na memória da ESP32, o envio é realizado através de *chunks* de 4096 bytes. Ao término do processo, a ESP32 envia outro Get de modo a identificar equivalência entre os arquivos e, caso não haja, pede ao AppScript para que seja deletado e repete o envio. Os passos descritos são registrados pelos logs do Quadro 7.

4.1.1 Limitações do envio

Em uma série de testes realizados, buscou-se maximizar o tamanho dos pacotes enviados via HTTP Post. Inicialmente, trabalhou-se com pacotes de 32 kB, o que permitia um envio em até 15 segundos de arquivos com aquisições de três segundos. No entanto, utilizar um pacote desse tamanho causou uma série de instabilidades. Uma delas era o próprio servidor (da Google) impedir o envio, retornando a mensagem "*Request Denied*". Para solucionar isso, limitou-se os pacotes a 4 kiB, o que desacelerou consideravelmente a transmissão, mas garantiu um envio seguro dos dados. Na prática, isso impacta diretamente o tempo entre gravações que o usuário pode escolher, considerando que, para um áudio de três segundos, o sistema toma dois minutos para realizar o envio completo. Dentro da taxa de transmissão viabilizada pelo protocolo WiFi, esse tempo de transmissão foi considerado alto.

Quadro 7 – *Logs* associados ao envio do arquivo.

```
[POST] Reading and sending file...
[POST] Bytes read: 4096
[POST] Buffer size: 4096
[POST] Sending chunk 0 - 4096 bytes...
[POST] HTTP Response code: 302
[POST] Sent chunk successfully.
                .
                .
                .
[CHECK] Checking file size on AppScript...
[CHECK] File Size Comparison Result: false
[DELETE] Wrong size. Deleting file on
   AppScript...
[DELETE] File delete result: true
```

Fonte: Autor.

4.2 ARQUIVOS NO GOOGLE DRIVE E FORMA DE ONDA

Com o envio completo, é possível identificar a presença dos arquivos na pasta definida pelo usuário. Na Figura 16, é dada uma lista de arquivos associados a gravações realizadas em diferentes datas com este sistema.

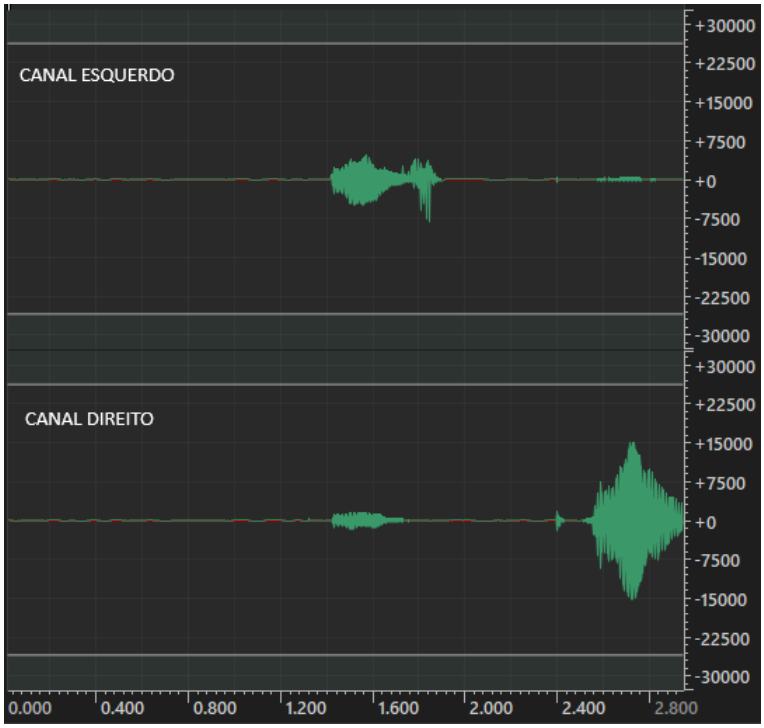
Figura 16 – Arquivos enviados ao Google Drive.

Fonte: Autor.

Para checagem da qualidade do áudio, utilizou-se o aplicativo *open-source* OcenAudio, que permite a visualização de formas de onda, espectros e realiza uma série de funções básicas de processamento de sinais (OCENAUDIO, 2024). Na Figura 17, exibe-se a forma

de onda associada ao arquivo "D2024-09-27T14-23-46.wav", enquanto na Figura 18, seus metadados.

Figura 17 – Forma de onda do arquivo "D2024-09-27T14-23-46.wav".



Fonte: Autor.

Observa-se que o tamanho do arquivo (em segundos) corresponde ao configurado em código (3 s). Além disso, validou-se que foram gravados dois canais (estéreo), sendo possível visualizar diferentes formas de onda em cada canal. Fez-se, também, uma análise do espectrograma do sinal, de modo a identificar se haviam descontinuidades e se o conteúdo espectral de fato vai até 8 kHz (considerando a frequência de amostragem escolhida). Na Figura 19, é dado o espectrograma do mesmo arquivo, em que foi possível identificar que a gravação ocorreu da forma esperada.

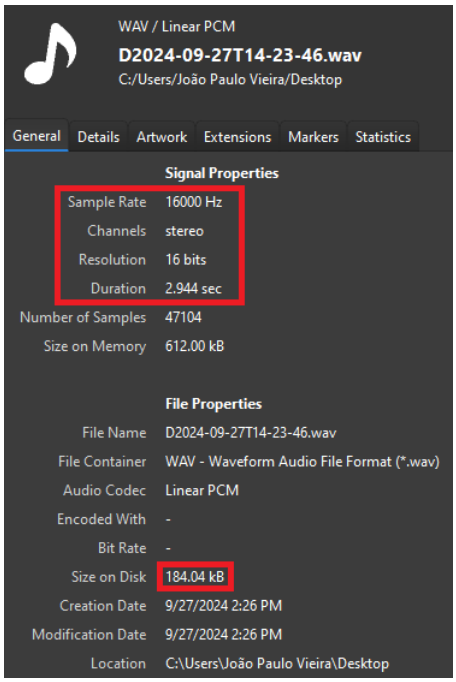
4.3 APLICAÇÃO EM PYTHON

Por fim, na aplicação em Python, temos *logs* associados ao download e escolha de quais arquivos serão classificados, dados no Quadro 8.

No Quadro, é possível identificar os arquivos presentes na Figura 16 (que estão no Google Drive). Com isso, é feita a classificação através do modelo SED - LINSE. Para exemplificar diferentes casos de uso do modelo, testaram-se sons de sirene[1], tiro[2] e

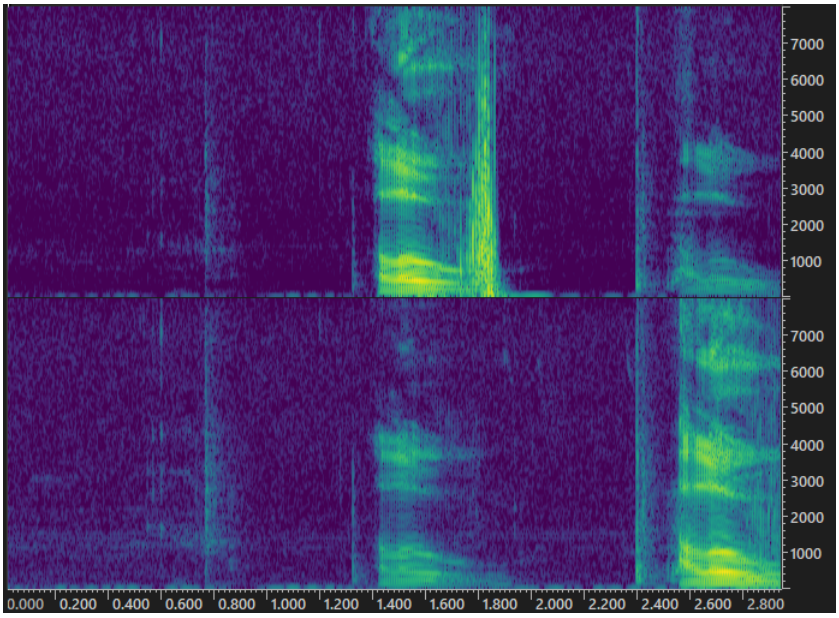
[1] Disponível em: `https://youtu.be/VOSTEarWrqQ`.
[2] Disponível em: `https://youtu.be/Nyi_vSzPcWw`.

Figura 18 – Metadados associados ao arquivo "D2024-09-27T14-23-46.wav".



Fonte: Autor.

Figura 19 – Espectrograma do arquivo "D2024-09-27T14-23-46.wav".



Fonte: Autor.

fogo/incêndio[3], encontrados na Internet, e o som de uma fala do autor[4] . A seguir, nas

[3] Disponível em: `https://youtu.be/mz9ftphTWTM`.

[4] Áudio da voz do autor. Disponível em: `https://github.com/jpvieir/audiorecording-esp32/blob/main/test_audio_files/_D2024-10-02T17-33-33.wav`

Quadro 8 – *Logs* da aplicação em Python.

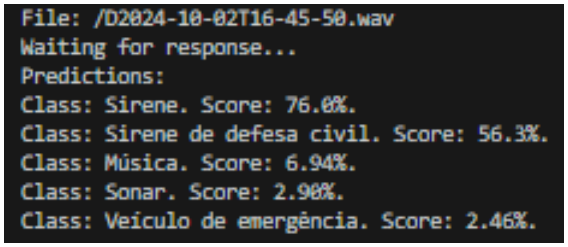
```
[AUTH] Complete.

Downloading file D2024-09-27T15-04-49.wav.
   Progress:  100%
----------------------------------------
Downloading file D2024-09-27T14-56-29.wav.
   Progress:  100%
----------------------------------------
Downloading file D2024-09-27T14-32-07.wav.
   Progress:  100%
----------------------------------------
Downloading file D2024-09-27T14-23-46.wav.
   Progress:  100%
----------------------------------------
Downloading file D2024-09-27T14-14-50.wav.
   Progress:  100%
----------------------------------------
Downloading file D2024-09-27T14-06-30.wav.
   Progress:  100%
----------------------------------------
Downloading file D2024-09-27T13-58-09.wav.
   Progress:  100%
----------------------------------------
Downloading file D2024-09-27T13-46-13.wav.
   Progress:  100%
----------------------------------------
Downloading file D2024-09-27T13-25-52.wav.
   Progress:  100%
----------------------------------------
Files to classify:
 ['D2024-09-27T13-25-52.wav'
   'D2024-09-27T13-46-13.wav'
 'D2024-09-27T13-58-09.wav'
   'D2024-09-27T14-06-30.wav'
 'D2024-09-27T14-14-50.wav'
   'D2024-09-27T14-23-46.wav'
 'D2024-09-27T14-32-07.wav'
   'D2024-09-27T14-56-29.wav'
 'D2024-09-27T15-04-49.wav']
```

Fonte: Autor.

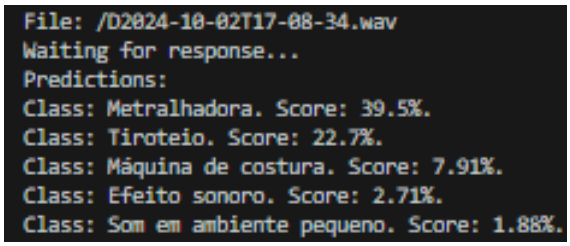
Figuras 20, 21, 22 e 23, são dados exemplos de resultados obtidos com o modelo para os testes realizados, nos quais é possível verificar que as cenas acústicas retornadas conferem com os sons gravados.

Figura 20 – Cenas acústicas mais prováveis retornadas pelo SED - LINSE para um som de sirene.



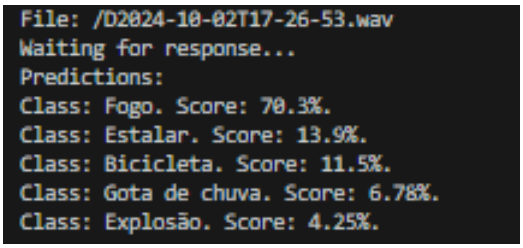
Fonte: Autor.

Figura 21 – Cenas acústicas mais prováveis retornadas pelo SED - LINSE para um som de tiro.



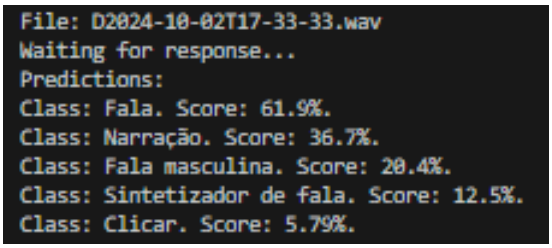
Fonte: Autor.

Figura 22 – Cenas acústicas mais prováveis retornadas pelo SED - LINSE para um som de incêndio/fogo.



Fonte: Autor.

Figura 23 – Cenas acústicas mais prováveis retornadas pelo SED - LINSE para um som de voz masculina.



Fonte: Autor.

5 CONCLUSÃO

Neste trabalho, desenvolveu-se um sistema de aquisição e transmissão de áudio via ESP32 e aplicações web para tratamento dos dados recebidos e classificação de cenas acústicas por meio de um modelo de aprendizado profundo. Consideram-se os objetivos gerais e específicos cumpridos, com cada etapa validada por meio de testes práticos. Obteve-se uma boa qualidade nas aquisições de áudio (estéreo, amostrado a 16 kHz), essencial para a melhor identificação das cenas acústicas. Além disso, a aquisição e gravação de áudio foram paralelizadas, permitindo a obtenção de faixas contínuas. Com o acesso à Internet (via WiFi), realizou-se a transmissão das faixas de áudio gravadas, enviando-as em pacotes de 4 kB devido a limitações na memória do dispositivo, bem como na API do AppScript. Desenvolveu-se uma aplicação em JavaScript, na plataforma Google AppScript, que realiza a recepção de requisições HTTP GET e POST, salvando os arquivos enviados no Google Drive. Por fim, desenvolveu-se uma aplicação em Python, que se conecta à API do Google Drive para realizar o download dos arquivos de áudio, e posteriormente enviá-los para um modelo de classificação de cenas acústicas treinado no LINSE.

Para a execução deste trabalho, estudaram-se os protocolos I^2S, SPI, HTTP, e as linguagens de programação C/C++, JavaScript e Python. Além disso, consultou-se a documentação da ESP32 fornecida por sua fabricante, Espressif, para um maior entendimento de como programar na plataforma.

5.1 SUGESTÕES PARA TRABALHOS FUTUROS

Compreende-se que o projeto, em seu estado atual de desenvolvimento, não contempla a possibilidade de uma implementação em larga escala e portátil, considerando que a ESP32 é alimentada via conexão USB e depende de conexão WiFi para a transmissão dos dados, o que limita o alcance do sistema.

Dessa forma, um possível aprimoramento do projeto pode ser feito através de um levantamento do consumo energético do sistema, identificando-se o que poderia ser feito para reduzir o gasto, explorando quais funções da ESP32 poderiam ser desligadas, ou alterando o modo de operação do dispositivo. Além disso, sugere-se avaliar a utilização de outro protocolo de comunicação, como o Bluetooth, que possui um menor custo energético e taxa de transferência de dados. Em sistemas IoT, é comum que os sensores se comuniquem com um *gateway* por meio desse protocolo, e o *gateway* poderia realizar a transmissão de diversas faixas de áudio para um servidor web.

REFERÊNCIAS

BAÚ DA ELETRÔNICA. **Módulo Cartão SD Card**. [*S.l.*: *s.n.*], 2024. https://www.baudaeletronica.com.br/produto/modulo-cartao-sd-card.html. Acessado em: 29-set-2024.

BRANCO, Sérgio; FERREIRA, André G.; CABRAL, Jorge. Machine Learning in Resource-Scarce Embedded Systems, FPGAs, and End-Devices: A Survey. **Electronics**, Multidisciplinary Digital Publishing Institute, v. 8, n. 11, p. 1289, nov. 2019. ISSN 2079-9292. DOI: 10.3390/electronics8111289. Acesso em: 29 set. 2024.

DANG, An; VU, Toan; WANG, Jia-Ching. A Survey of Deep Learning for Polyphonic Sound Event Detection. *In:* INTERNATIONAL Conference on Orange Technologies. [*S.l.*: *s.n.*], dez. 2017. P. 75–78. DOI: 10.1109/ICOT.2017.8336092.

DHAKER, Piyu. Introduction to SPI Interface. **Analog Dialogue**, 2018. Acessado em: 27-09-2024. Disponível em: https://www.analog.com/en/resources/analog-dialogue/articles/introduction-to-spi-interface.html.

ESPRESSIF SYSTEMS. **ESP32 Series Datasheet, Version 4.7**. [*S.l.*], 2023. Accessado em: 20-09-2024. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

GONG, Yuan; CHUNG, Yu-An; GLASS, James R. AST: Audio Spectrogram Transformer. **CoRR**, abs/2104.01778, 2021. arXiv: 2104.01778. Disponível em: https://arxiv.org/abs/2104.01778.

GOURLEY, David *et al.* **HTTP: The Definitive Guide**. 1ª edição. Beijing Köln: O'Reilly Media, nov. 2002. ISBN 978-1-56592-509-0.

GREENING, Chris. **ESP32 Audio**. [*S.l.*: *s.n.*], 2020. https://github.com/atomic14/esp32_audio/tree/master. Accessado em: 27-09-2024.

HARVARD UNIVERSITY. **WAV Header Image**. [*S.l.*: *s.n.*], 2023. Accessado em: 20-09-2024. Disponível em: https://cs50.harvard.edu/indonesia/2023/psets/4/reverse/.

HEITTOLA, Toni *et al.* Context-Dependent Sound Event Detection. **EURASIP Journal on Audio, Speech, and Music Processing**, v. 2013, n. 1, p. 1, jan. 2013. ISSN 1687-4722. DOI: 10.1186/1687-4722-2013-1.

INVENSENSE. **INMP441 Omnidirectional Microphone with Bottom Port and I²S Digital Output**. [*S.l.*], 2014. Datasheet. Disponível em: https://invensense.tdk.com/wp-content/uploads/2015/02/INMP441.pdf.

KEIM, Robert. Improving on the Electret: An Introduction to MEMS Microphones. **All About Circuits**, 2017. Acessado em: 27-09-2024. Disponível em: https://www.allaboutcircuits.com/technical-articles/improving-on-the-electret-an-introduction-to-mems-microphones/.

_____. Introduction to MEMS (Microelectromechanical Systems). **All About Circuits**, 2020. Acessado em: 27-09-2024. Disponível em: https://www.allaboutcircuits.com/technical-articles/introduction-to-mems-microelectromechanical-systems/.

KIM, Soo-Jong; CHUNG, Yong-Joo. A Transformer Model Based on Multi-scale Features to Improve the Performance of Sound Event Detection. **The Journal of Korean Institute of Information Technology**, v. 20, p. 93–100, jun. 2022. DOI: 10.14801/jkiit.2022.20.6.93.

KOUTINI, Khaled *et al.* Efficient Training of Audio Transformers with Patchout. *In:* INTERSPEECH 2022. [*S.l.*]: ISCA, set. 2022. (interspeech$_2$022). DOI: 10.21437/interspeech.2022-227. Disponível em: http://dx.doi.org/10.21437/Interspeech.2022-227.

MESAROS, Annamaria *et al.* Sound Event Detection: A tutorial. **IEEE Signal Processing Magazine**, Institute of Electrical e Electronics Engineers (IEEE), v. 38, n. 5, p. 67–83, set. 2021. ISSN 1558-0792. DOI: 10.1109/msp.2021.3090678. Disponível em: http://dx.doi.org/10.1109/MSP.2021.3090678.

MISHRA, Ranjan Kumar; REDDY, G. Y. Sandesh; PATHAK, Himanshu. The Understanding of Deep Learning: A Comprehensive Review. **Mathematical Problems in Engineering**, v. 2021, n. 1, p. 5548884, 2021. ISSN 1563-5147. DOI: 10.1155/2021/5548884. Acesso em: 1 out. 2024.

MNASRI, Zied; ROVETTA, Stefano; MASULLI, Francesco. Anomalous Sound Event Detection: A Survey of Machine Learning Based Methods and Applications. **Multimedia Tools and Applications**, v. 81, n. 4, p. 5537–5586, fev. 2022. ISSN 1573-7721. DOI: 10.1007/s11042-021-11817-9.

NERI, Michael *et al.* Sound Event Detection for Human Safety and Security in Noisy Environments. **IEEE Access**, v. 10, p. 134230–134240, 2022. ISSN 2169-3536. DOI: 10.1109/ACCESS.2022.3231681.

OCENAUDIO. **OcenAudio: An Easy, Fast and Powerful Audio Editor**. [*S.l.*: *s.n.*], 2024. https://www.ocenaudio.com/whatis. Accessado em: 27-09-2024.

PARK, Sooyoung; JEONG, Youngho; LEE, Taejin. MANY-TO-MANY AUDIO SPECTROGRAM TRANSFORMER: TRANSFORMER FOR SOUND EVENT LOCALIZATION AND DETECTION. *In:* DETECTION and Classification of Acoustic

Scenes and Events. [*S.l.*: *s.n.*], 2021. Disponível em: https://api.semanticscholar.org/CorpusID:245355723.

PHILIPS SEMICONDUCTORS. **I²S Bus Specification**. [*S.l.*], 1986. Datasheet. Disponível em: https://www.sparkfun.com/datasheets/BreakoutBoards/I2SBUS.pdf. Acesso em: 28 set. 2024.

PUANGPONTIP, Supadchaya; HEWETT, Rattikorn. On Using Deep Learning for Business Analytics: At What Cost? **Procedia Computer Science**, v. 207, p. 3738–3747, jan. 2022. ISSN 1877-0509. DOI: 10.1016/j.procs.2022.09.434. Acesso em: 29 set. 2024.

QIU, Jiantao *et al.* Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. *In:* ACM/SIGDA International Symposium. [*S.l.*: *s.n.*], fev. 2016. P. 26–35. DOI: 10.1145/2847263.2847265.

ROBU.IN. **INMP441 MEMS High Precision Omnidirectional Microphone Module I2S**. [*S.l.*: *s.n.*], 2024. Imagem da internet. Disponível em: https://robu.in/product/inmp441-mems-high-precision-omnidirectional-microphone-module-i2s/.

ROUAT, J. Computational Auditory Scene Analysis: Principles, Algorithms, and Applications (Wang, D. and Brown, G.J., Eds.; 2006) [Book review]. **IEEE Transactions on Neural Networks**, v. 19, n. 1, p. 199–199, jan. 2008. ISSN 1941-0093. DOI: 10.1109/TNN.2007.913988.

SAMIE, Farzad; BAUER, Lars; HENKEL, Jörg. From Cloud Down to Things: An Overview of Machine Learning in Internet of Things. **IEEE Internet of Things Journal**, v. 6, n. 3, p. 4921–4934, 2019. DOI: 10.1109/JIOT.2019.2893866.

SILVA, Lucas; ALVES DE SOUZA, Vinícius; BATISTA, Gustavo. Uma Ferramenta de Suporte Ao Uso de Classificadores Em Sistemas Embarcados. *In:* SIMPÓSIO Brasileiro de Automação Inteligente (SBAI). [*S.l.*: *s.n.*], nov. 2019. DOI: 10.17648/sbai-2019-111582.

SVOBODA, Filip *et al.* Deep Learning on Microcontrollers: A Study on Deployment Costs and Challenges. *In:* EUROSYS '22: Seventeenth European Conference on Computer Systems. [*S.l.*: *s.n.*], abr. 2022. P. 54–63. DOI: 10.1145/3517207.3526978.

SWAMY, S. Narasimha; KOTA, Solomon Raju. An Empirical Study on System Level Aspects of Internet of Things (IoT). **IEEE Access**, v. 8, p. 188082–188134, 2020. DOI: 10.1109/ACCESS.2020.3029847.

YE, Le *et al.* Research Progress on Low-Power Artificial Intelligence of Things (AIoT) Chip Design. **Science China Information Sciences**, v. 66, n. 10, p. 200407, set. 2023. ISSN 1869-1919. DOI: 10.1007/s11432-023-3813-8. Acesso em: 29 set. 2024.

ZHANG, Yixiao *et al.* Spectrogram Transformers for Audio Classification. *In:* 2022 IEEE International Conference on Imaging Systems and Techniques (IST). [*S.l.*: *s.n.*], jun. 2022. P. 1–6. DOI: 10.1109/IST55454.2022.9827729.

ANEXO A – PRINCIPAIS FUNÇÕES QUE COMPÕEM O SISTEMA

Tabela 3 – Principais funções associadas à ESP32, AppScript e Python e suas descrições.

Plataforma	Função	Descrição
ESP32	mountSDCard / unmoutSDCard	Monta/desmonta o cartão SD.
	configureWiFi	Se conecta à rede WiFi configurada no código.
	record	Inicia a aquisição e grava o áudio no cartão SD, com tamanho pré-configurado pelo usuário.
	sendBinaryDataToAppScript	Envia requisições HTTP Get (com informações básicas do arquivo, como tamanho e nome) e HTTP Post com os arquivos gravados.
	checkFileSizeWithAppScript	Envia HTTP Get para identificar se o arquivo salvo no Google Drive tem o tamanho correto.
	deleteFile	Envia HTTP Get para informar ao AppScript que deve deletar o arquivo, pois difere do original.
AppScript	doGet	Recebe parâmetros enviados pela ESP32 via HTTP GET e salva-os para uso posterior. Checa parâmetros de controle para executar alguma ação (verificar tamanho ou deletar arquivo).
	getFileIdByName	Com base no nome do arquivo, obtém o ID associado a ele no Google Drive.
	getFileSize	Obtém o tamanho do arquivo, utilizando o ID dado pela função getFileIdByName.
	deleteFileById	Deleta o arquivo, utilizando o ID dado pela função getFileIdByName.
	doPost	Recebe os pacotes de enviados pela ESP32 via HTTP POST e salvá-os no arquivo de áudio.
	concatUInt8Arrays	Concatena chunks recebidos ao arquivo original.
Python	authenticateWithDriveAPI	Autentica com a API do Google Drive com base num arquivo de autenticação e a chave da API.
	retrieveWavFilenamesFromFolder	Obtém os nomes dos arquivos de extensão Wave na pasta-alvo (diretório que contém as faixas de áudio).
	getFilelistFromDisk	Obtém os nomes dos arquivos de extensão Wave no disco (se existem).
	CompareAndDelete	Compara o resultado das duas funções anteriores, e identifica quais arquivos precisam ser baixados.
	DownloadWavFilesFromFolder	Faz o download dos arquivos escolhidos.
	Prediction	Envia os arquivos para o modelo SED, que retorna predições. Salva essas predições em um arquivo texto.

Fonte: Autor.