



Trabalho de Conclusão de Curso

**Controle de veículo através de palavras-chave
faladas utilizando *TinyML***

Victor Hugo Sampaio Lima

Orientador:

Prof. Dr. Erick de Andrade Barboza

Maceió, Maio de 2025

Victor Hugo Sampaio Lima

Controle de veículo através de palavras-chave faladas utilizando *TinyML*

Monografia apresentada como requisito parcial para
obtenção do grau de Bacharel em Engenharia de Com-
putação do Instituto de Computação da Universidade
Federal de Alagoas.

Orientador:

Prof. Dr. Erick de Andrade Barboza

Maceió, Maio de 2025

Catálogo na Fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico

Bibliotecário: Marcelino de Carvalho Freitas Neto – CRB-4 - 1767

L732c Lima, Victor Hugo Sampaio.
 Controle de veículo através de palavras-chave faladas utilizando
TinyML / Victor Hugo Sampaio Lima. – 2025.
 42 f. : il.

Orientador: Erick de Andrade Barboza.
Monografia (Trabalho de conclusão de curso em Engenharia de
Computação) - Universidade Federal de Alagoas, Instituto de Computação.
Maceió, 2025.

Bibliografia: f. 37.
Apêndices: f. 38-42.

1. Microcontroladores. 2. Dispositivo de borda. 3. *TinyML*. 4. *Firmware*.
5. *Hardware*. 6. Redes neurais (Computação). 7. *Edge Impulse*. 8. Veículos.
I. Título.

CDU: 004.032.26



Trabalho de Conclusão de Curso - TCC

Formulário de Avaliação

Nome do Aluno:

VICTOR HUGO SAMPAIO LIMA

Nº de Matrícula:

19211373

Título do TCC (Tema):

Controle de veículo através de palavras-chave faladas utilizando TinyML

Banca Examinadora

Erick de Andrade Barboza
Nome do Orientador

GLAUBER RODRIGUES LEITE
Nome do Professor

RODRIGO JOSÉ SARMENTO PEIXOTO
Nome do Professor

Documento assinado digitalmente



ERICK DE ANDRADE BARBOZA

Data: 27/05/2025 17:41:04-0300

Verifique em <https://validar.iti.gov.br>

Assinatura

Documento assinado digitalmente



GLAUBER RODRIGUES LEITE

Data: 02/06/2025 15:45:56-0300

Verifique em <https://validar.iti.gov.br>

Assinatura

Documento assinado digitalmente



RODRIGO JOSE SARMENTO PEIXOTO

Data: 29/05/2025 22:18:31-0300

Verifique em <https://validar.iti.gov.br>

Assinatura

Data da Defesa

27 / 05 / 2025

Nota Obtida

8,5 (oito e meio)

Observações

Coordenador do Curso
De Acordo

Documento assinado digitalmente



JOBSON DE ARAUJO NASCIMENTO

Data: 02/06/2025 15:58:42-0300

Verifique em <https://validar.iti.gov.br>

Assinatura

Agradecimentos

Inicialmente quero agradecer a Deus por tudo que aconteceu na minha vida para que eu conseguisse chegar até aqui. Também quero agradecer aos meus pais que, desde o princípio da minha educação, sempre me apoiaram imensuravelmente, proporcionando uma aprendizagem sem fronteiras. De maneira similar, gostaria de agradecer a minha companheira Júlia Montelares, que sempre me consolou nos momentos mais difíceis da graduação e sempre lembrou do meu potencial nos momentos de dúvidas.

Agradeço ao meu orientador, Prof. Dr. Erick de Andrade Barboza, por todo o apoio durante a elaboração deste trabalho, como também em outros momentos durante a graduação. Também estendo estes agradecimentos aos demais professores do curso de Engenharia de Computação, que não se limitaram apenas a ensinar, mas se preocuparam com o meu crescimento pessoal e profissional.

Agradeço ao time de Sistemas Embarcados do Centro de Inovação EDGE, onde tive a minha primeira experiência profissional. Atuando projetos reais, tive o privilégio de aprender com pessoas altamente capacitadas, que sempre me guiaram durante o meu trajeto. Ao José Gomes, que me apresentou o centro e me ajudou durante o processo seletivo. Ao Paulo Roberto e ao Matheus Tenório, que foram os primeiros a me acolher e que me guiaram nos meus primeiros passos. Ao Prof. Rodrigo Peixoto, que me aconselha como crescer profissionalmente, e a todos os demais companheiros do time, com quem aprendo diariamente.

Por fim, mas não menos importante, agradeço a todos os meus amigos de turma cuja amizade cultivei durante a graduação, que foram de extrema importância em todos os momentos ao longo do curso: Arthur Padilha, Arthur Ramos, Bernardo Brotelli, Eduardo Barros, Élisson Souza, Ítalo Oliveira, José Gomes, Kleber Vasconcelos, Leonardo Zaia, Mateus Batista, Monique Moreira, Paloma Lacerda, Rebeca Brandão, Rodrigo Ferreira, Túlio Cerqueira, Ulyanne Freire e Vinícius Marinho.

“Entrega o teu caminho ao Senhor; confia nele, e ele tudo fará.”

– Salmos 37:5

Resumo

O avanço tecnológico tem possibilitado o desenvolvimento de dispositivos com maior capacidade de processamento e armazenamento, aliados a custos de produção reduzidos, dimensões físicas menores e menor consumo energético. Dentre esses dispositivos, temos os microcontroladores, que são capazes de executar um *software* (nesse contexto é denominado *firmware*) e controlar um *hardware*. Paralelo ao avanço dos dispositivos eletrônicos, também está presente a aprendizagem de máquina, subárea da IA responsável por desenvolver dispositivos capazes de aprender e melhorar com a experiência.

Este trabalho tem como objetivo explorar e interligar essas duas grandes áreas da computação. Essa subárea é denominada de *TinyML*. Neste projeto foi elaborado um dispositivo de borda capaz de classificar palavras-chave utilizando uma rede neural para controlar um veículo. O modelo da rede neural é executada em um microcontrolador, que detecta a palavra-chave e envia um comando respectivo para um segundo microcontrolador, responsável por controlar os periféricos do veículo.

A execução do modelo de IA em um microcontrolador é possível através da plataforma *Edge Impulse*, onde é possível realizar o manejo do banco de dados, a criação, treinamento e teste do modelo, e por fim exportar o modelo em um formato compatível com o microcontrolador. Os resultados obtidos demonstraram que, embora com um *hardware* limitado, foi possível atingir um bom grau de acurácia, e que também foi possível estender o modelo para classificar mais palavras, mas sem interferir no funcionamento do veículo.

Palavras-chave: microcontroladores, dispositivo de borda, *TinyML*, *firmware*, *hardware*, *rede neural*, *Edge Impulse*, *veículo*.

Abstract

Technological evolution has enabled the development of devices with increasingly greater processing power and storage capacity while reducing their physical size and energy consumption. Among these devices are microcontrollers, components capable of executing software (referred to in this context as firmware) and controlling hardware. Alongside the evolution of electronic devices, machine learning, a subfield of AI that is responsible for developing devices capable of learning and improving from experience, has also significantly emerged.

This work aims to explore and interconnect these two major areas of computer science. This subfield is known as TinyML. In this project, an edge device was developed, capable of classifying keywords using a neural network in order to control a vehicle. The neural network model is executed on a microcontroller, which detects the keyword and sends the corresponding command to a second microcontroller responsible for controlling the vehicle's peripherals.

The execution of the AI model on a microcontroller is made possible through the Edge Impulse platform, where it is possible to manage the dataset, create, train, and test the model, and conclude by exporting the model in a format compatible with the microcontroller. The results obtained showed that, despite the hardware's limitations, it was possible to achieve a good level of accuracy, and the model could also be extended to classify more keywords without interfering with the vehicle's operation.

Keywords: *microcontrollers, edge device, TinyML, firmware, hardware, neural network, Edge Impulse, vehicle.*

Sumário

Lista de Abreviaturas e Siglas	viii
Lista de Figuras	x
Lista de Símbolos	xi
1 Introdução	1
1.1 Contexto e Motivação	1
1.2 Objetivo Geral	2
1.3 Objetivos Específicos	2
1.4 Estrutura do Trabalho	3
2 Fundamentação Teórica	4
2.1 Sistema Embarcado	4
2.1.1 Firmware	4
2.1.2 Hardware	5
2.1.3 Microcontrolador	6
2.2 Classificação de Palavras-Chave Usando IA	6
2.2.1 Problema de Detecção de Palavras-Chave (<i>Keywords Spotting</i>)	6
2.2.2 Rede Neural	7
2.2.3 Técnicas de Processamento de Áudio	7
2.3 TinyML	10
2.3.1 Vantagens e Desafios	10
2.3.2 Técnicas para Redução do Modelo	11
2.3.3 Frameworks Disponíveis	11
3 Metodologia	14
3.1 Edge Impulse	14
3.2 Firmware	15
3.3 Hardware	18
3.4 Banco de Dados	21
3.5 Modelo de IA	22

4	Resultados	27
4.1	Modelos de IA	27
4.2	Sistema Embarcado	32
4.3	Desafios, soluções e lições aprendidas	32
4.3.1	Banco de Dados	32
4.3.2	Firmware	33
4.3.3	Hardware	33
4.4	Síntese dos Resultados	33
5	Conclusão	35
5.1	Contribuições do Projeto	35
5.2	Trabalhos Futuros	36
A	PCI	38

Lista de Abreviaturas e Siglas

RTOS	<i>Real Time Operating System</i>
MCU	<i>Micro Controller Unity</i>
RAM	<i>Random Access Memory</i>
KWS	<i>Keyword Spotting</i>
PCI	<i>Placa de Circuito Impresso</i>
LDO	<i>Low Dropout Regulator</i>
LED	<i>Light Emitting Diode</i>
BOM	<i>Bill of Materials</i>
SPI	<i>Serial Peripheral Interface</i>
IC	<i>Inter-Integrated Circuit</i>
CI	<i>Circuito Integrado</i>
RTC	<i>Real-Time Clock</i>
IA	<i>Inteligência Artificial</i>
CNN	<i>Convolutional Neural Network</i>
MFCC	<i>Mel-Frequency Cepstral Coefficients</i>
DFT	<i>Discrete Fourier Transform</i>
IDFT	<i>Inverse Discrete Fourier Transform</i>
MFE	<i>Mel-Frequency Energy</i>
DCT	<i>Discrete Cosine Transform</i>
KDD	<i>Knowledge Discovery in Databases</i>

Lista de Figuras

1	Exemplo do processo de processamento de áudio utilizando MFCC	8
2	Exemplo de energia de um sinal em bandas de frequência distribuídas segundo a escala mel	9
3	Arquitetura do Software em Camadas	15
4	Cinemática do Driver de Motor	16
5	Diagrama de Features do Driver	16
6	Diagrama de Features da Infra	17
7	Placa Bluepill (STM32F103C8T6)	18
8	Placa Arduino Nano 33 BLE Sense	18
9	Hardware montado na <i>protoboard</i>	20
10	Molde do modelo de IA classificando seis palavras e ruído	22
11	Molde do modelo de IA classificando onze palavras e ruído	22
12	Coeficientes cepstrais de um dado gerado pelo Edge Impulse	23
13	Arquitetura do modelo da rede neural classificando seis palavras	24
14	Arquitetura do modelo da rede neural classificando onze palavras	25
15	Matriz de Confusão da rede neural do modelo detectando seis palavras após treinamento gerado pelo Edge Impulse	27
16	Resultados da rede neural do modelo detectando seis palavra após teste gerado pelo Edge Impulse	28
17	Matriz de Confusão da rede neural do modelo detectando onze palavras com alto ruído nos dados após treinamento gerado pelo Edge Impulse	28
18	Resultados da rede neural do modelo detectando onze palavras com alto ruído nos dados após teste gerado pelo Edge Impulse	29
19	Matriz de Confusão da rede neural do modelo detectando onze palavras sem ruído nos dados após treinamento gerado pelo Edge Impulse	29
20	Resultados da rede neural do modelo detectando onze palavras sem ruído nos dados após teste gerado pelo Edge Impulse	30
21	Matriz de Confusão da rede neural final do modelo detectando onze palavras após treinamento gerado pelo Edge Impulse	30
22	Resultado da rede neural final do modelo detectando onze palavras após teste gerado pelo Edge Impulse	31

23	Diagrama de Blocos do hardware	39
24	BOM	40
25	Esquemático da PCI	40
26	PCI	41
27	PCI em 3D	42

Lista de Símbolos

V	Volts
h	horas
min	minutos
s	segundos
m	mili
M	mega
Hz	Hertz

Introdução

1.1 Contexto e Motivação

Nas últimas décadas, o avanço exponencial da tecnologia permitiu que sistemas computacionais se tornassem mais rápidos, com maior poder de processamento e capacidade de armazenamento, tudo isso com o custo de produção reduzido, dimensões físicas cada vez menores e menor consumo energético. Dentre essas evoluções, especificamente dos circuitos integrados, surgiu os microcontroladores.

Uma unidade de microcontrolador (MCU) é essencialmente um pequeno computador em um único chip. Ele foi projetado para gerenciar tarefas específicas em um sistema incorporado sem a necessidade de um sistema operacional complexo (IBM, 2024a). Esses avanços permitiram que microcontroladores executassem desde algoritmos simples até sistemas sofisticados, como Sistemas Operacionais em Tempo Real (RTOS).

Inteligência Artificial (IA) é um campo da ciência que se concentra na criação de computadores e máquinas que podem raciocinar, aprender e atuar de maneira que normalmente exigiria inteligência humana ou que envolve dados com escala maior do que as pessoas podem analisar. A IA é um campo amplo que abrange muitas disciplinas diferentes, como ciência da computação, estatísticas e análises de dados, engenharia de hardware e software, linguística, neurociência e até mesmo filosofia e psicologia. Em um nível operacional para uso comercial, a IA é um conjunto de tecnologias baseadas principalmente em *machine learning* e aprendizado profundo, usada para análise de dados, previsões e previsão, categorização de objetos, processamento de linguagem natural, recomendações, recuperação inteligente de dados e muito mais. (Google Cloud, 2025). Hoje em dia existem diversos modelos de IA, desde mais complexos, como os modelos de IA generativa (ChatGPT, Gemini, etc) até os mais simples, como o que foi utilizado no projeto para reconhecimento de voz.

O avanço conjunto dessas duas grandes áreas da computação possibilitou a execução de modelos de inteligência artificial diretamente em microcontroladores, surgindo assim uma nova

área de aprendizagem de máquina, o *TinyML*. O *TinyML* está na intersecção de aplicações de Inteligência Artificial embarcada, algoritmos, hardware e software. O *TinyML* difere do aprendizado de máquina convencional (como por exemplo, servidor e nuvem) porque requer não apenas experiência em software, mas também experiência em hardware embarcado. (Harvard Professional and Lifelong Learning, 2024) (tradução nossa).¹

O problema resolvido neste trabalho é a detecção de palavras-chave, conhecido como *Keyword Spotting (KWS)*, que se trata em detectar uma ou mais palavras específicas. A motivação da resolução deste problema foi desenvolver um veículo que não necessitasse de um operador direto (como um controle remoto, por exemplo) e que possuísse processamento local.

No contexto do projeto, as palavras detectadas são os comandos que delegam a orientação em que o veículo deve se mover. Além da orientação, o tempo de duração do movimento também são definidos através de palavras-chave.

1.2 Objetivo Geral

O objetivo geral do projeto é desenvolver uma aplicação baseada em *TinyML* para detecção de palavras-chave faladas com o intuito de controlar um veículo e seus periféricos apenas com processamento local.

1.3 Objetivos Específicos

1. Criar um banco de dados com áudios das palavras-chave e ruídos;
2. Gerar um modelo de rede neural com acurácia acima de 60% após testes;
3. Utilizar dois microcontroladores: um para executar a rede neural e outro para controlar os periféricos do veículo; e definir um protocolo de comunicação para envio de comandos entre tais componentes;
4. Desenvolver um *firmware* baseado em RTOS para o microcontrolador que controla os periféricos do veículo;
5. Desenvolver um protótipo de *hardware* que suporte os microcontroladores e demais componentes do sistema;

¹No original: *TinyML is at the intersection of embedded Machine Learning (ML) applications, algorithms, hardware, and software. TinyML differs from mainstream machine learning (e.g., server and cloud) in that it requires not only software expertise, but also embedded-hardware expertise.*

1.4 Estrutura do Trabalho

O trabalho está organizado da seguinte forma. O segundo capítulo abordará conceitos que são fundamentais para o entendimento do funcionamento de um sistema embarcado, explicando detalhadamente cada um dos seus componentes. Também abordará conceitos de banco de dados, inteligência artificial e processamento de áudio, que são necessários para compreender o processo de detecção das palavras-chave.

O terceiro capítulo mostrará as etapas em que o projeto foi realizado e os componentes que foram utilizados, desde o desenvolvimento dos primeiros componentes de *firmware* até os testes finais no protótipo.

O quarto capítulo exibirá os resultados obtidos no projeto, tanto nos modelos de inteligência artificial como também no *firmware* e no *hardware* do sistema embarcado. Além disso, mostrará desafios que foram encontrados durante o desenvolvimento de cada etapa, como foram solucionados e quais lições foram aprendidas em cada desafio. Por fim, será apresentada uma síntese geral sobre os resultados adquiridos.

O quinto capítulo mostrará as contribuições do projeto para o desenvolvimento de aplicações baseadas em *TinyML* e destacará alguns pontos do projeto que poderão ser aprimorados para estender este trabalho.

2

Fundamentação Teórica

Neste capítulo, serão definidos os conceitos fundamentais para a compreensão da construção de um projeto de sistema embarcado, juntamente com um modelo de inteligência artificial.

2.1 Sistema Embarcado

Inicialmente, é importante definirmos o que é um sistema embarcado e qual a sua aplicabilidade. Existem diversas definições para explicar o que é um sistema embarcado, mas podemos resumir em um sistema de propósito específico, que muitas vezes possui limitação de hardware, consumo, custo, entre outras.

Um exemplo simples é pensar em um semáforo. Sua única função é ligar ou desligar cada lâmpada (vermelha, amarela ou verde) em uma sequência específica em um intervalo de tempo predefinido. Alguns semáforos podem variar o intervalo de tempo entre cada lâmpada de acordo com o horário do dia. Outros, mais complexos, já possuem lâmpadas menores na lateral para indicar o tempo restante para mudar seu estado. Já semáforos mais completos possuem um *display* indicando o intervalo. Independente das funcionalidades que cada modelo de semáforo tem, sua essência será sempre a mesma. Não faria sentido implementar um sistema complexo para um semáforo, pois por executar apenas uma tarefa simples, não é necessário um grande poder de processamento, com grande armazenamento e alto consumo. Para isso, é implementado um sistema embarcado, tornando o semáforo mais barato, mais robusto e consumindo menos energia, podendo funcionar ininterruptamente por longos períodos de tempo.

2.1.1 Firmware

O *Firmware* é responsável por executar todas as lógicas do sistema e controlar os componentes de *hardware*. No exemplo do semáforo, o *firmware* poderia ser responsável por:

- Definir o intervalo entre as mudanças de estado.

- Definir o tempo em que cada lâmpada permanecerá acesa.
- Controlar relés para ligar/desligar as lâmpadas.
- Enviar comandos para o CI (circuito integrado) que controla o *display*.
- Comunicar com outros semáforos (para sincronização, por exemplo).

Como também poderia ter outras responsabilidades, a depender das suas *features*.

2.1.1.1 Sistema Operacional em Tempo Real (RTOS)

Os RTOS são sistemas operacionais que facilitam o desenvolvimento de um sistema embarcado pois já possuem integrados funcionalidades como: gerenciamento de tarefas, gerenciamento de recursos, controle de temporização, entre outras.

Um exemplo prático de RTOS é o Zephyr², um sistema operacional de código aberto voltado para dispositivos embarcados com recursos limitados. O Zephyr oferece uma arquitetura modular, suporte a múltiplas plataformas de hardware e uma ampla gama de serviços prontos para uso, como os citados anteriormente. Devido às suas vantagens, ele é o mais utilizado na indústria.

2.1.2 Hardware

O conceito de *hardware* é muito amplo. Ele pode variar dependendo do contexto que está sendo utilizado. No caso de sistemas embarcados, o *hardware* é o conjunto de componentes eletrônicos que compõem o sistema. Geralmente, o *hardware* de um sistema embarcado é composto por:

- Microcontrolador ou microprocessador.
- Componentes elétricos (resistores, capacitores, diodos, CIs, transistores, entre outros).
- Sensores (luz, temperatura, movimento, distância, entre outros).
- Atuadores (motores, pistões, LEDs, *displays*, relés, entre outros).
- PCI (placa de circuito impresso).
- Energia (fontes de alimentação, baterias, painel solar, entre outros).

Esses são componentes bastante comuns em um *hardware* de um sistema embarcado. Porém, é um segmento muito complexo, podendo variar bastante de acordo com a aplicação. No exemplo do semáforo, seu *hardware* pode ser composto por:

²<https://zephyrproject.org/>

- Microcontrolador.
- Lâmpadas (LED (Light Emitting Diode), incandescente, etc).
- Relés, para ligar/desligar as lâmpadas.
- RTC (real-time clock), para saber a data e hora.
- Antena, caso haja comunicação externa.
- PCI.
- Fonte de alimentação.

2.1.3 Microcontrolador

O microcontrolador é a peça chave no projeto de um sistema embarcado. É ele que executa o *firmware* e controla o *hardware*. No mercado, existem diversos tipos de microcontroladores, cada um com suas características específicas (armazenamento, processamento, quantidade de pinos, conectividade, robustez, entre diversas outras).

2.2 Classificação de Palavras-Chave Usando IA

A introdução do *TinyML* viabilizou a resolução de uma série de problemas computacionais, principalmente em dispositivos embarcados. Entre esses desafios, destaca-se a classificação de palavras-chave.

2.2.1 Problema de Detecção de Palavras-Chave (*Keywords Spotting*)

O problema de classificação de palavras-chave é um problema de reconhecimento de fala (conhecido como *speech recognition*). Este problema consiste basicamente em detectar qual palavra está sendo falada dado um conjunto de palavras. Um exemplo prático deste problema são os assistentes pessoais, como por exemplo os dispositivos Echo, da Amazon. Estes dispositivos são ativados quando se é detectado a palavra de ativação (palavra-chave), que por padrão é "*Alexa*". Essa palavra pode ser alterada nas configurações do dispositivo para "*Amazon*" ou "*Echo*".

Com essa abordagem, os dados de voz são enviados e recebidos para o servidor apenas quando o dispositivo é ativado pela palavra chave. Sem essa abordagem, seria inviável possuir um sistema como esse, pois iria demandar bastante energia elétrica e largura de banda tanto do cliente como do servidor, pois as informações de áudio estariam sendo transmitidas a todo instante, inviabilizando seu funcionamento.

2.2.2 Rede Neural

Dentro da IA, temos a subárea de Aprendizagem de Máquina, que consiste em elaborar dispositivos capazes de aprender e melhorar com a experiência. Entre os modelos de aprendizagem de máquina, temos a *rede neural*, que foi o modelo utilizado no projeto para classificar as palavras.

Uma rede neural é um programa de aprendizado de máquina, ou modelo, que toma decisões de uma forma semelhante ao cérebro humano, utilizando processos que imitam a maneira como os neurônios biológicos trabalham juntos para identificar fenômenos, avaliar opções e chegar a conclusões. (IBM, 2024b).

As redes neurais são constantemente utilizadas como soluções em diversos problemas, como: problemas de regressão, como por exemplo prever o clima, ou prever tendências no mercado; problemas de classificação, como classificar e-mails como spam ou não, e no caso do projeto, classificar as palavras; processamento de linguagem natural, como análise de sentimentos de um texto (positivo, negativo ou neutro), entre diversos outros.

2.2.3 Técnicas de Processamento de Áudio

Diversos algoritmos têm sido desenvolvidos para o processamento de sinais de áudio, desempenhando um papel essencial tanto em tarefas de reconhecimento quanto na integração do áudio em modelos de inteligência artificial, como redes neurais. Tais algoritmos realizam a extração de características discriminativas do sinal sonoro, reduzindo sua dimensionalidade e mantendo apenas suas informações mais relevantes.

2.2.3.1 MFCC (Mel-Frequency Cepstral Coefficients)

O MFCC é uma técnica de extração de características amplamente utilizada no processamento de fala e áudio. Os MFCCs são usados para representar as características espectrais do som de uma forma adequada para diversas tarefas de aprendizado de máquina, como reconhecimento de fala e análise musical. (uday, 2023) (tradução nossa).³

A Figura 1 mostra o processo completo de processamento de áudio utilizando o MFCC desde o áudio bruto até a geração dos coeficientes cepstrais. O processo citado na referência foi resumido para facilitar o entendimento, mas o processo completo possui mais passos:

1. **Conversão A/D:** Converte o sinal de áudio analógico em um sinal digital, permitindo seu tratamento computacional;
2. **Pré-ênfase:** Amplifica as frequências mais altas do sinal de áudio, com o objetivo de compensar a atenuação natural dessas frequências durante a produção da fala e melhorar a relação sinal-ruído;

³No original: *MFCC is a feature extraction technique widely used in speech and audio processing. MFCCs are used to represent the spectral characteristics of sound in a way that is well-suited for various machine learning tasks, such as speech recognition and music analysis.*

3. **Janelamento:** Segmenta o sinal em pequenas janelas temporais, considerando que o sinal de fala apresenta características estacionárias apenas em curtos intervalos;
4. **DFT:** Aplica a Transformada Discreta de Fourier a cada janela, convertendo o sinal do domínio do tempo para o domínio da frequência, a fim de identificar seus componentes espectrais;
5. **Escala Mel:** Transforma a escala de frequência para a escala Mel, que se baseia na percepção auditiva humana, mais sensível a variações em baixas frequências;
6. **Função Logarítmica:** Aplica a função logarítmica aos espectros de energia, refletindo a forma como o sistema auditivo humano percebe variações de intensidade sonora;
7. **IDFT (ou DCT):** Utiliza a Transformada Discreta Inversa de Fourier ou, mais comumente, a Transformada Discreta do Cosseno (DCT), para extrair os coeficientes cepstrais, que representam as características acústicas do sinal de forma compacta e robusta.
8. **Características Dinâmicas:** Calcula as derivadas temporais dos coeficientes MFCC, conhecidas como delta (velocidade) e delta-delta (aceleração), com o objetivo de capturar as variações temporais do sinal de fala e enriquecer sua representação.

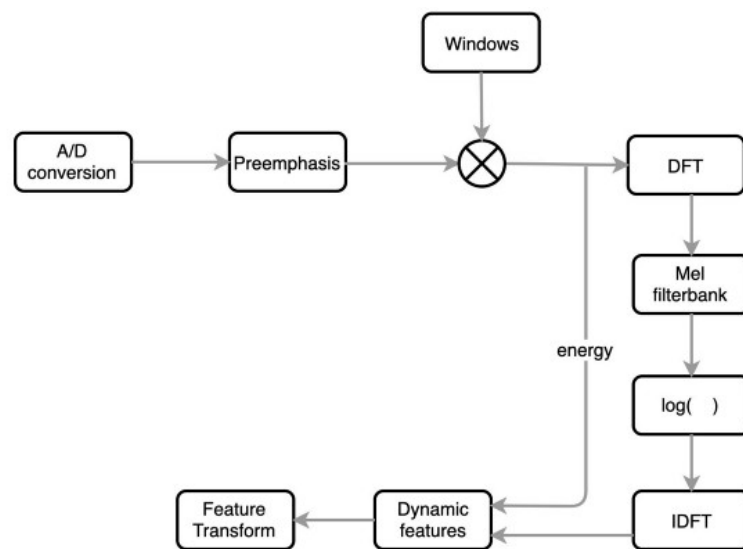


Figura 1: Exemplo do processo de processamento de áudio utilizando MFCC

Fonte: uday.

A DFT é dada pela fórmula:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi}{N}kn}, \quad \text{para } k = 0, 1, \dots, N-1$$

Onde:

- $x[n]$ é a sequência de entrada no domínio do tempo;
- $X[k]$ é o valor da DFT no índice k , ou seja, no domínio da frequência;
- N é o número total de amostras;
- j é a unidade imaginária, tal que $j^2 = -1$;
- $e^{-j\frac{2\pi}{N}kn}$ é o fator rotacional complexo que projeta a sequência no domínio da frequência.

2.2.3.2 MFE (Mel-Frequency Energy)

Similar ao MFCC, o MFE também é uma técnica de extração de características de áudio. No entanto, enquanto o MFCC aplica transformações adicionais como o logaritmo e a Transformada Discreta do Cosseno para gerar os coeficientes cepstrais, o MFE se concentra apenas no cálculo da energia do sinal em bandas de frequência distribuídas segundo a escala mel, resultando em uma representação mais direta e menos compactada do espectro sonoro.

O processo do MFE se dá em dividir o sinal de áudio em janelas curtas, aplicar a DFT para obter o espectro de frequência e em seguida calcular a energia em cada banda do banco de filtros mel. Enquanto o MFCC é mais utilizado em voz humana, o MFE pode ser utilizado tanto em voz humana como também em outros tipos de sons.

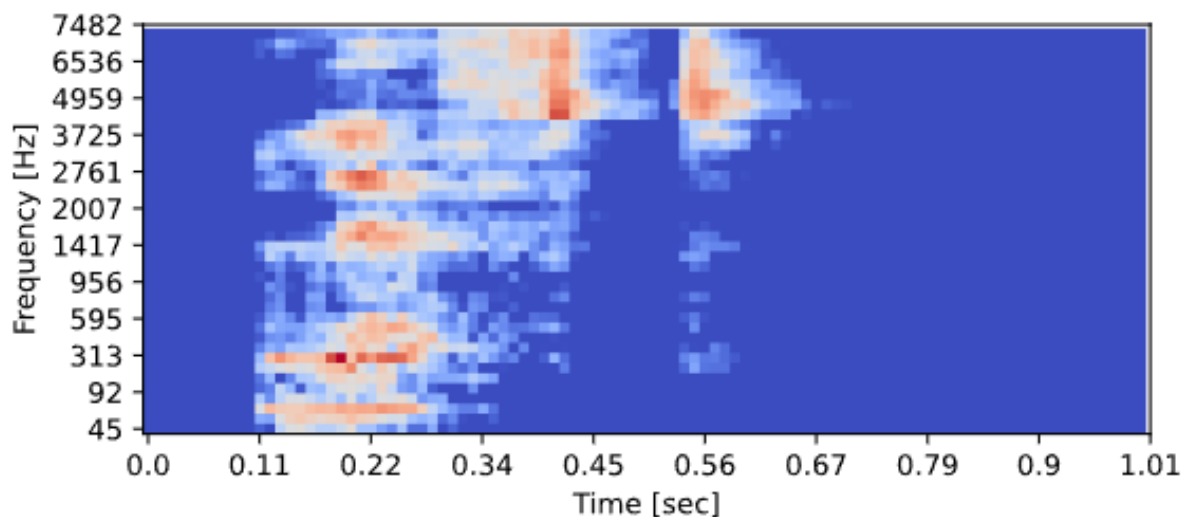


Figura 2: Exemplo de energia de um sinal em bandas de frequência distribuídas segundo a escala mel

Fonte: Edge Impulse.

A Figura 2 representa um exemplo da saída do MFE (energia de um sinal em bandas de frequência distribuídas segundo a escala mel) gerado pela plataforma do Edge Impulse.

2.3 TinyML

Um dispositivo pequeno com um impacto gigantesco. Este poderia ser um resumo para Tiny Machine Learning (ou TinyML), uma vertente emergente dentro da inteligência artificial. Neste modelo, algoritmos de Machine Learning são implementados em sistemas de baixo consumo de energia, como sensores ou microcontroladores, para executar tarefas automatizadas. (Nama, 2020).

2.3.1 Vantagens e Desafios

Ainda segundo o autor Nama, alguns dos principais diferenciais da tecnologia são:

- **Segurança de dados:** Como não há necessidade de transferir informações para ambientes externos, a privacidade dos dados é mais garantida;
- **Economia de energia:** A transferência de informações demanda uma grande infraestrutura de servidores. Quando não há transmissão de dados, economiza-se em energia e recursos, consequentemente em custos;
- **Sem dependência de conexão:** Se para funcionar o dispositivo depende de internet, e ela cai, será impossível enviar os dados para o servidor. É o caso de você tentar usar um assistente de voz e ele não responder porque está desconectado da internet;
- **Latência:** Transferir de dados leva tempo e muitas vezes ocasiona em um delay. Quando não envolve esse processo, o resultado é instantâneo;

Embora possua diversas vantagens, há algumas questões tanto de *software* quanto de *hardware* que devem ser levadas em consideração para o desenvolvimento de um sistema baseado em *TinyML*.

Por ser executado em um *hardware* com pouca memória disponível e com menor poder de processamento, o *software* acaba sendo limitado. Deve-se atentar a algoritmos de pré-processamento, como por exemplo o MFCC citado anteriormente. Algoritmos mais complexos podem demandar mais memória volátil, culminando na falta de memória para o modelo de inteligência artificial.

Outro fator importante ao tratar do *software* é o modelo de IA. Pelo mesmo motivo de limitação de memória, o modelo de IA também não pode exigir quantidade de memória volátil que venha a interferir na execução dos demais algoritmos do *firmware*. Como será demonstrado no capítulo de Resultados para o problema de classificação, mas também requerendo atenção para outros tipos de abordagens, embora aumentar a quantidade de *features* de saída não impacte tanto no tamanho do modelo, a sua acurácia sofre uma redução considerável, podendo interferir na sua funcionalidade.

Em relação ao *hardware*, além do poder de processamento e da quantidade de memória volátil e persistente serem suficientes para suprir as demandas do *firmware*, há outro ponto que requer mais atenção, que é a alimentação elétrica. A maioria das abordagens baseadas em *TinyML* são dispositivos que, ou não são fixos (como um relógio inteligente), ou não possuem uma fonte fixa de energia por perto (como sensores posicionados remotamente em troncos de árvores), sendo necessário assim a utilização de baterias.

Como a energia elétrica é um recurso limitado, o *hardware* precisa ser desenvolvido e utilizado de forma criteriosa, para que não interfira no funcionamento total do sistema.

2.3.2 Técnicas para Redução do Modelo

Por conta das limitações de *hardware* citadas na seção anterior, o modelo de IA utilizado em *TinyML* antes de ser executado no *hardware* é submetido à algumas técnicas de redução para que seu tamanho seja suportado pela quantidade de memória disponível. Existem diversas técnicas disponíveis, mas é possível citar as que são mais utilizadas para a redução de modelos de IA.

2.3.2.1 Quantização

A quantização se dá pela redução direta da precisão dos números que representam os parâmetros do modelo, como por exemplo de 32 para 8 bits. Essa operação reduz tanto o tamanho quanto a carga computacional exigida pelo modelo.

2.3.2.2 Poda

A poda consiste na remoção de conexões ou neurônios que possuem pouca contribuição no resultado final, diminuindo o modelo sem perda significativa na precisão. Um exemplo é eliminar conexões que possuem pesos muito próximos de zero.

2.3.2.3 Dessorção de Conhecimento

Conhecido como *Knowledge Distillation*, a técnica se dá no treinamento de um modelo menor (*student*) para imitar o comportamento de um modelo maior (*teacher*). Assim, o modelo menor aprende a fazer as previsões sem a necessidade de possuir o tamanho do modelo maior.

2.3.3 Frameworks Disponíveis

No contexto de *TinyML*, existem diversos *frameworks* que facilitam a implementação do modelo de IA em um sistema embarcado. Dentre os disponíveis pode-se citar:

2.3.3.1 TensorFlow Lite para Microcontroladores

“O TensorFlow Lite para microcontroladores foi desenvolvido para executar modelos de machine learning em microcontroladores e outros dispositivos usando apenas alguns kilobytes de memória. O ambiente de execução principal cabe em 16 KB em um Arm Cortex M3 e pode executar muitos modelos básicos. Ele não requer suporte a sistemas operacionais, bibliotecas C ou C++ padrão nem a alocação de memória dinâmica.

O TensorFlow Lite para microcontroladores foi projetado para as restrições específicas do desenvolvimento de microcontroladores. Se você estiver trabalhando em dispositivos mais potentes (por exemplo, um dispositivo Linux incorporado como o Raspberry Pi), o framework padrão do TensorFlow Lite pode ser mais fácil de integrar.” (TensorFlow, 2021).

Segundo o autor, o *framework* possui as seguintes limitações:

- Suporte para um subconjunto limitado de operações do TensorFlow;
- Suporte para um conjunto limitado de dispositivos;
- Biblioteca C++ de baixo nível que requer gerenciamento manual da memória;
- O treinamento no dispositivo não é compatível;

As etapas de implementação e execução de um modelo utilizando o *framework* são:

1. Gerar um modelo pequeno do TensorFlow que se ajuste ao dispositivo de destino e contenha operações do TensorFlow.
2. Converter o modelo criado utilizando o conversor do TensorFlow Lite.
3. Converter em uma matriz de bytes C utilizando ferramentas padrão para armazená-la em uma memória de programa somente leitura no dispositivo.
4. Executar inferência no dispositivo utilizando a biblioteca C++ e processar os resultados.

As operações do TensorFlow, dispositivos suportados, ferramentas padrão e a biblioteca C++ podem ser acessadas na página do TensorFlow Lite para microcontroladores, disponível na referência citada desta seção.

2.3.3.2 uTensor

uTensor é um framework de inferência de aprendizado de máquina extremamente leve, construído sobre o TensorFlow e otimizado para dispositivos Arm. Ele consiste em uma biblioteca de tempo de execução e uma ferramenta offline que cuida da maior parte do trabalho de conversão dos modelos. Seu repositório contém o núcleo do tempo de execução e algumas implementações de exemplo de operadores, gerenciadores/agendadores de memória e outros componentes,

sendo que o tamanho do núcleo do tempo de execução é de apenas cerca de 2 KB! (uTensor, 2025) (tradução nossa).⁴

O *framework* funciona utilizando um modelo construído e treinado no TensorFlow para gerar arquivos com extensão .cpp e .hpp que contêm o código necessário para realizar a inferência.

⁴No original: *uTensor is an extremely light-weight machine learning inference framework built on Tensorflow and optimized for Arm targets. It consists of a runtime library and an offline tool that handles most of the model translation work. This repo holds the core runtime and some example implementations of operators, memory managers/schedulers, and more, and the size of the core runtime is only 2KB!*

3

Metodologia

O desenvolvimento do projeto foi realizado por etapas, cada uma responsável por uma parte específica do sistema. Primeiramente, foi feita a montagem do protótipo do veículo, com a parte de *hardware* na *protoboard* e a criação do *firmware*. Durante o desenvolvimento individual de cada componente, tanto o seu *hardware* como o trecho de código do *firmware* responsável pelo seu controle foram testados simultaneamente. Após a finalização dessa etapa, foi iniciada a criação do banco de dados utilizada no modelo de inteligência artificial. Com o banco de dados concluído, foi iniciado a criação do modelo na plataforma *Edge Impulse*. Após concluído e refinado tanto o modelo como o banco de dados, foi feito o teste final no protótipo para validar o funcionamento total do sistema. Os arquivos de *firmware*, *hardware* e da PCI podem ser encontrados no GitHub do projeto.⁵

3.1 Edge Impulse

O *Edge Impulse* (Edgeimpulse, Inc, 2025) foi a plataforma escolhida para desenvolver o modelo de inteligência artificial. Ela é uma plataforma de desenvolvimento de aplicações de IA's embarcadas. Nessa plataforma, além da geração do modelo, é possível realizar outras atividades, como: coleta, seleção, edição, manipulação, entre outras operações com dados e treinamento, retreinamento, teste, refinamento, exportação para diferentes ambientes, entre outras operações com o modelo.

Além dessas importantes funcionalidades, a plataforma foi escolhida também por possuir vários *presets* padrões para diversos tipos de aplicação, entre elas a *Keyword Spotting*, ou *Deteção de Palavras-chave*, que é o propósito do projeto. A vantagem de ter um *presets*, é que a plataforma já sugere valores padrões para os diversos parâmetros que são utilizados para desenvolver o modelo, o que facilita e agiliza o desenvolvimento.

⁵Elaboração própria, *GitHub do Projeto*. Acessado em: 31 mar. 2025. Disponível em: <<https://github.com/victorhugosampaio/EchoCar/tree/main>>

3.2 Firmware

Foram utilizados dois códigos de *firmware* no projeto, um para cada microcontrolador. Foi utilizada a linguagem de programação C no desenvolvimentos de ambos.

O *firmware* do Arduino Nano 33 BLE Sense foi inicialmente gerado pela plataforma do Edge Impulse. Depois, foi adicionado o protocolo de comunicação *USART* pelo autor para o envio de comandos do Arduino para o STM32F103C8T6.

Para o *firmware* do STM32F103C8T6 foi utilizado o sistema operacional em tempo real *Zephyr*. Além das funcionalidades dos RTOS mencionadas, o *Zephyr* possui compatibilidade com diversos microcontroladores, como também possui integrado *drivers* de diversos periféricos, como *displays*, sensores e atuadores, protocolos de comunicação, etc, removendo assim a necessidade de implementar cada funcionalidade individualmente. Além disso, por ser um projeto *open source*, o *Zephyr* sempre está sendo atualizado de acordo com novas tendências que vão surgindo, não se tornando obsoleto. É possível encontrar todas as funcionalidades que o *Zephyr* oferece na página *About the Zephyr Project* (Zephyr Project, 2023).

A arquitetura de software do *firmware* foi baseada na arquitetura em camadas (representada na Figura 3), sendo dividida em duas: a camada de *driver* e a camada de *infra*. A camada de *driver* contém os *drivers* de controle do *display*, dos motores e do protocolo de comunicação *USART*.

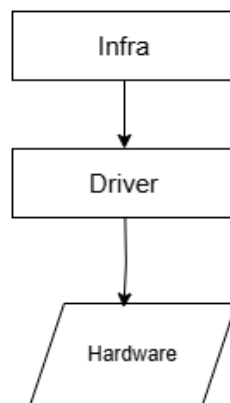


Figura 3: Arquitetura do Software em Camadas

Fonte: Autor.

A Figura 4 representa a cinemática do *driver* do módulo *driver* de motor MX1616H. Ela referencia os pinos de entrada do componente e a saída respectiva nos motores.

DC MOTOR	MODE	IN1	IN2	IN3	IN4
MOTOR-A	Forward	1/PWM	0		
	Reversion	0	1/PWM		
	Standby	0	0		
	Brake	1	1		
MOTOR-B	Forward			1/PWM	0
	Reversion			0	1/PWM
	Standby			0	0
	Brake			1	1
Note: 1. " 1 "represents a high level;" 0 "represents a low level;" PWM "on behalf of pulse width modulated wave, adjusts the duty cycle to change speed. 2. IN1, IN2 control MOTOR-A; IN3, IN4 control MOTOR-B; two are completely independent. 3. INx anti input common conduction function, pin floating is equivalent to low input.					

Figura 4: Cinemática do Driver de Motor
Fonte: Autodesk Instructables.

No diagrama de *features* do *driver* (representado na Figura 5), pode-se perceber que o *display* não é um componente obrigatório para o funcionamento do veículo, mas é essencial para que o usuário tenha um *feedback* das ações que o sistema está tomando. Para o projeto, foi escolhido o modelo *SPI*, devido à alta velocidade de transmissão de dados e a disponibilidade do protocolo no microcontrolador, mas também seria possível optar por um modelo *I²C*. O protocolo de comunicação entre os microcontroladores escolhido foi o *USART*, devido a sua facilidade de implementação. Embora possua uma menor capacidade de transmissão de dados, sua utilização é suficiente para o projeto. Seria possível também implementar outros protocolos, o *I²C* ou o *SPI*.

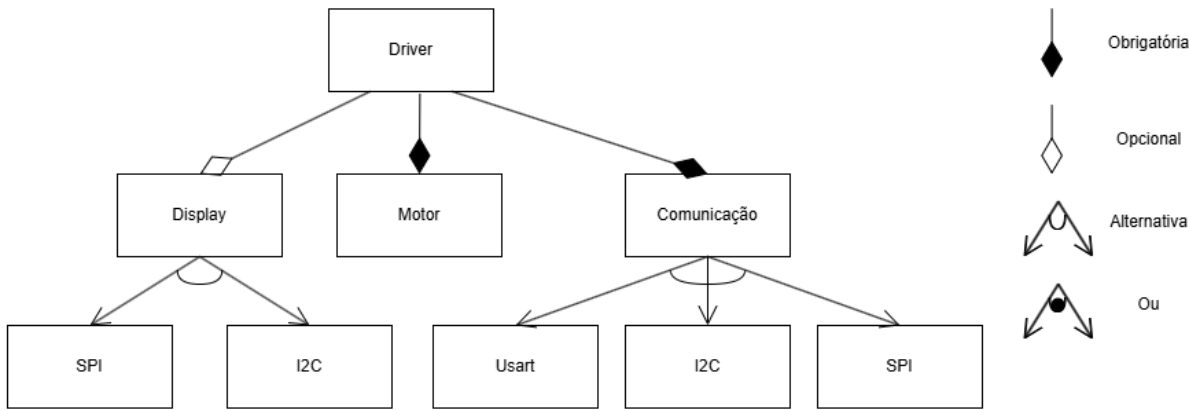


Figura 5: Diagrama de Features do Driver
Fonte: Autor.

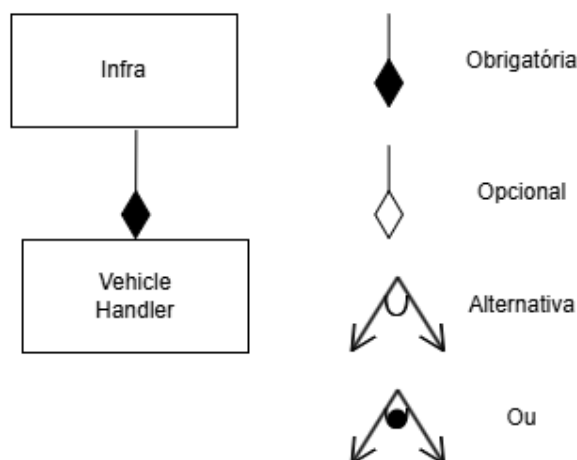


Figura 6: Diagrama de Features da Infra

Fonte: Autor.

A Figura 6 representa o diagrama de *features* da infraestrutura. A camada de *infra* contém o gerenciador do veículo, que é responsável por definir os movimentos que serão tomados de acordo com os comandos recebidos do Arduino. Primeiramente, é definida a direção que o veículo se movimentará através dos comandos *Left*, *Right*, *Forward*, *Backward*. Depois é definido o tempo de duração do movimento através dos comandos *One*, *Two*, *Three*, *Four* e *Five*. Caso durante a execução do movimento seja recebido o comando *Stop*, o veículo para, e só continua a execução quando recebe o comando *Go*. Os comandos são transmitidos com o tipo *char*. A tabela 3.1 mostra a relação entre o comando e o caractere relativo:

<i>Backward</i>	a
<i>Five</i>	b
<i>Forward</i>	c
<i>Four</i>	d
<i>Go</i>	e
<i>Left</i>	f
<i>Noise</i>	g
<i>One</i>	h
<i>Right</i>	i
<i>Stop</i>	j
<i>Three</i>	k
<i>Two</i>	l

Tabela 3.1: Relação entre Comando e Caractere

3.3 Hardware

Para o projeto, foram escolhidos dois microcontroladores, o *STM32F103C8T6* e o *Arduino Nano 33 BLE Sense*. A Figura 7 mostra a placa de desenvolvimento *Bluepill*, que contém o microcontrolador STM32F103C8T6, e a Figura 8 mostra o *devkit* do Arduino Nano.

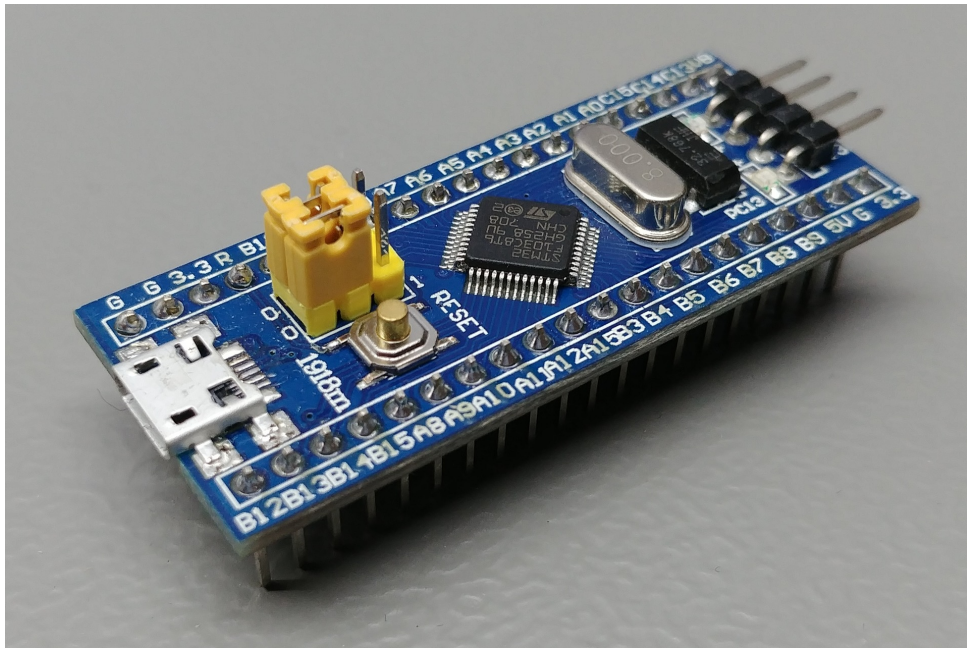


Figura 7: Placa Bluepill (STM32F103C8T6)

Fonte: Internet.

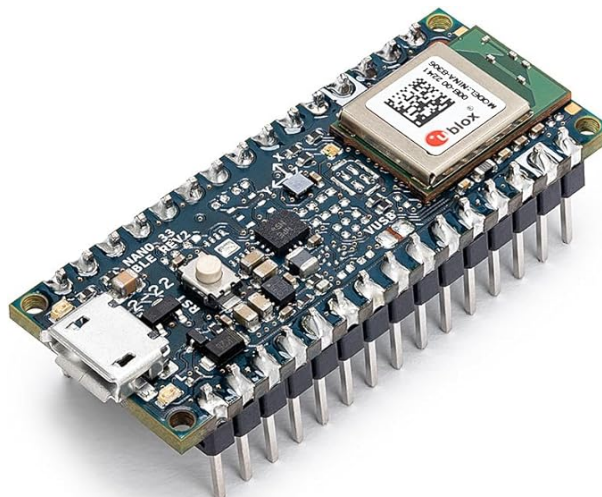


Figura 8: Placa Arduino Nano 33 BLE Sense

Fonte: Amazon.

Dentre suas diversas características, o STM32F103C8T6 foi escolhido por possuir um preço mais acessível, conter os protocolos de comunicação necessários para interagir com os outros

periféricos, armazenamento (RAM e Flash) suficiente para o *firmware* e por ter suporte no Zephyr. É possível encontrar todas as características do microcontrolador no seu *datasheet* (ST, 2023).

O Arduino Nano 33 BLE Sense foi escolhido por ter suporte na plataforma do *Edge Impulse* (plataforma utilizada para gerar o modelo de inteligência artificial), pela sua praticidade de uso e pela sua disponibilidade, concedido gentilmente a mim pelo orientador deste projeto, o que contribuiu para a agilidade do desenvolvimento.

Para a prototipagem e os testes iniciais, o *hardware* foi montado em uma *protoboard*, utilizando os componentes:

- Placa de desenvolvimento ARM STM32F103C8T6, para utilizar o microcontrolador.
- Módulo regulador de tensão AMS1117-3.3, para regular a tensão de entrada para 3.3V.
- Módulo driver de motor MX1616H, para controlar os motores.
- Quatro motores 5V com rodas (marca genérica).
- 2x baterias modelo 18650 (marca genérica).
- Suporte para 2x 18650 em paralelo.
- Interruptor (marca genérica).

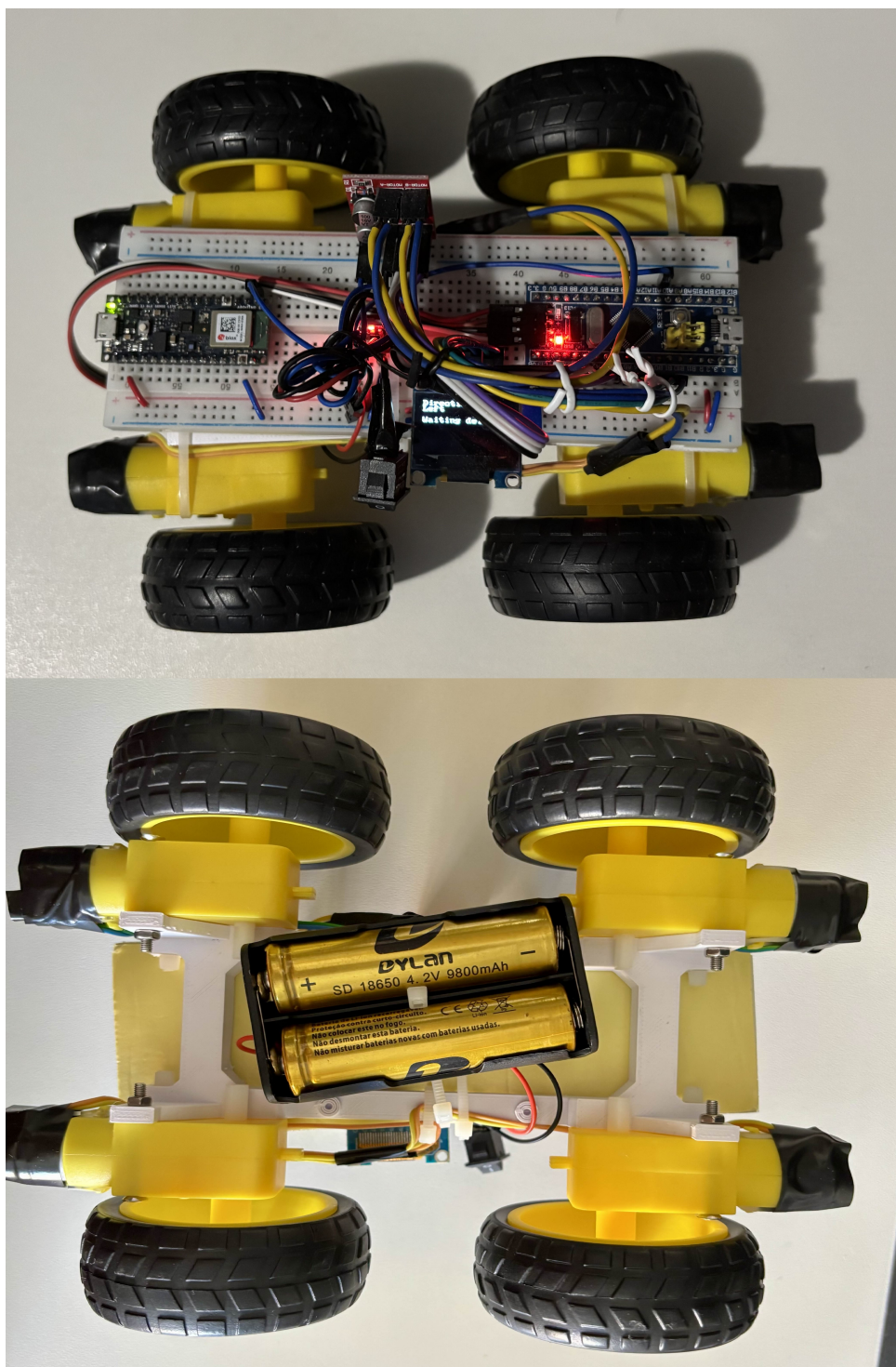


Figura 9: Hardware montado na *protoboard*

Fonte: Autor.

A Figura 9 demonstra o protótipo do *hardware* montado em uma *protoboard*. Na parte esquerda da *protoboard* está presente o *devkit* do microcontrolador Arduino Nano 33 BLE Sense e na parte direita está presente o *devkit* do microcontrolador STM32F103C8T6. O suporte com as baterias e o chassi foram dispostos na parte inferior do veículo.

Após os testes iniciais e validação do *firmware* e dos componentes de *hardware*, foi montada

a PCI (Placa de Circuito Impresso) no *software Kicad 8*, que é um *software open source* para desenvolvimento de PCI. Não foi possível montar a PCI e adquirir resultados utilizando-a, por isso seu desenvolvimento está demonstrado no apêndice A.

3.4 Banco de Dados

O banco de dados utilizado para treinamento e teste do modelo foi o *Speech Commands: A dataset for Limited-Vocabulary Speech Recognition* (WARDEN, 2018). É um *database* que contém áudios de palavras faladas de um vocabulário predefinido, onde cada *sample* possui uma janela de um segundo de duração, e também contém alguns áudios de ruídos. Dentre as palavras contidas no vocabulário, para os primeiros testes foram escolhidas: *Left*, *Right*, *Forward*, *Backward*, *Stop* e *Go*. Após obter bons resultados classificando seis palavras, foram adicionadas as palavras *One*, *Two*, *Three*, *Four* e *Five*. Existem duas versões do banco de dados, a versão utilizada foi a segunda, onde o autor incluiu mais palavras.

O manejo do banco de dados foi baseado no KDD (*Knowledge Discovery in Databases*), processo utilizado para extrair conhecimentos da base de dados em mineração de dados. A primeira parte do processo foi a seleção dos dados.

A quantidade de áudios por palavra disponível no banco de dados não é constante, então é necessário definir uma quantidade padrão de dados por palavra antes de dividi-lo entre treinamento e teste. Dentre as palavras utilizadas, a que continha a menor quantidade de dados foi a *Forward*, com 1.557 *samples*. Então, foi definido que seriam utilizadas 1.500 *samples* por palavra. As *samples* foram escolhidas aleatoriamente utilizando um algoritmo desenvolvido em *python*.

Além das *samples* de cada palavra, também foram carregados na plataforma do *Edge Impulse* os áudios de ruído *white_noise*, *pink_noise*, *exercise_bike* e *running_tap*. Posteriormente, foram adicionados mais áudios de ruído (*people-talking-289717*, *room-full-of-people-talking-223318*, *restaurant-talking-people-ambience-211724* e *group-talking-68823*), retirados do site *pixabay* (*pixabay*, 2025).

O tempo total de áudio do banco de dados final foi de 4h 40min 40s. Esses áudios foram divididos em 80% para treino e 20% para teste. A escolha dos arquivos para cada subconjunto foi feita automaticamente pela própria plataforma do *Edge Impulse*.

Na etapa de pré-processamento, foi selecionada a opção *Zero-pad data* na tela de configuração *Impulse Design*, que será explicada na próxima seção. Essa opção preenche com zeros os dados que tiverem menos de um segundo de duração.

Na etapa de enriquecimento dos dados, durante o desenvolvimento do modelo classificando onze palavras, foi selecionada a opção de adicionar pouca quantidade de ruído nos dados, o que melhorou a acurácia desse modelo. Essa opção será melhor discutida no capítulo de *Resultados*.

Normalmente, a última etapa do KDD é a mineração de dados (descobrir padrões, tendên-

cias, correlações, etc. entre os dados). Como o objetivo é delegar ao modelo a responsabilidade de achar os padrões e, então, classificar, esta etapa não foi realizada. A apresentação dos resultados também será demonstrada no capítulo de *Resultados*.

3.5 Modelo de IA

Os modelos de inteligência artificial para classificação de áudio foram desenvolvidos na própria plataforma. A criação de cada modelo segue algumas etapas.

A primeira etapa é a *Impulse Design*, onde se faz o primeiro molde do modelo de IA. No caso do projeto, foi feito um molde para classificação de áudio em ambos os modelos. A plataforma sugeriu utilizar uma Rede Neural Convolucional, conhecida como *Convolutional Neural Network* (CNN). Para o bloco de processamento de áudio, foi escolhido o *MFCC*, recomendado para extrair *features* de voz humana. Para o bloco de aprendizagem foi escolhida uma rede neural, recomendada para classificação.

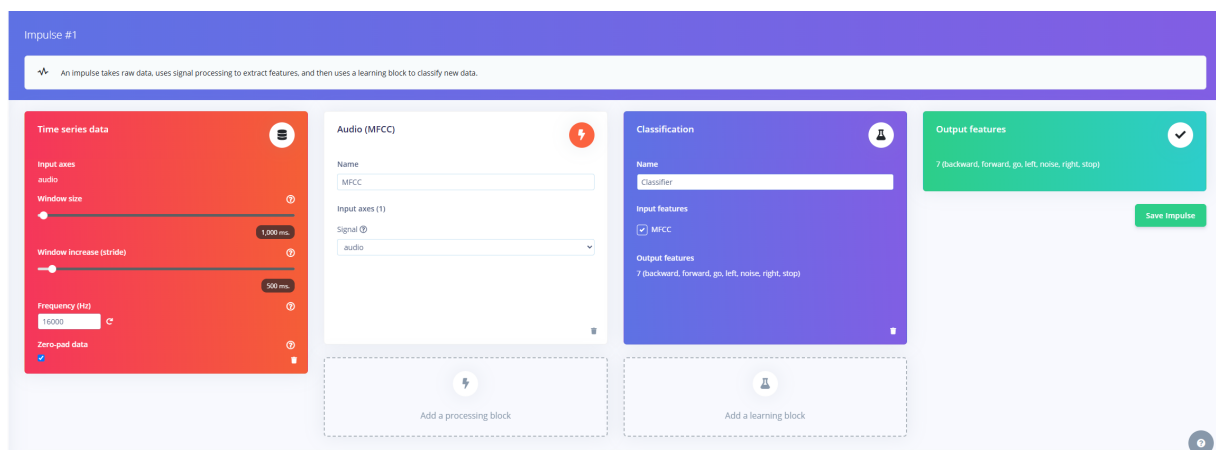


Figura 10: Molde do modelo de IA classificando seis palavras e ruído

Fonte: Edge Impulse.

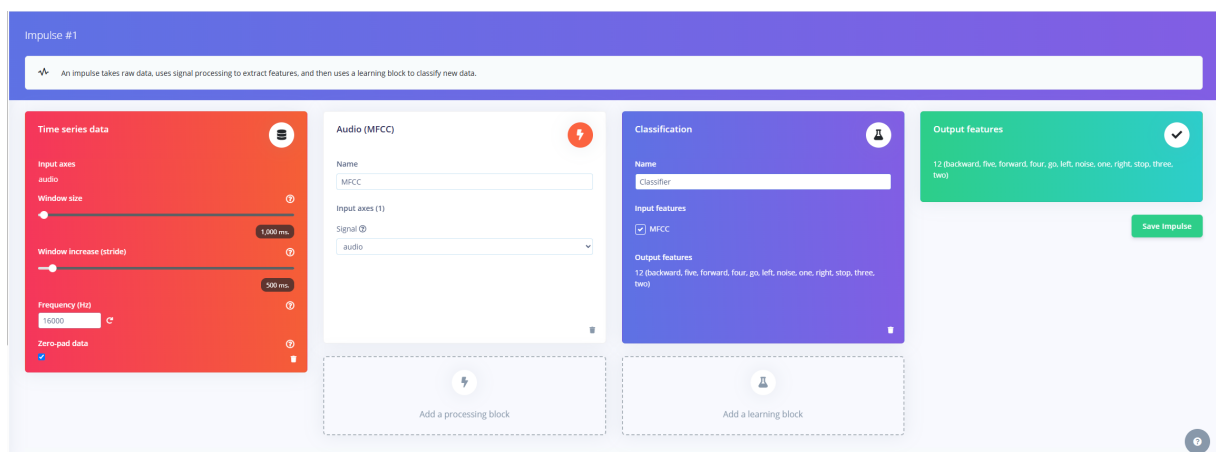


Figura 11: Molde do modelo de IA classificando onze palavras e ruído

Fonte: Edge Impulse

As Figuras 10 e 11 apresentam os moldes dos modelos de IA na etapa do *Impulse Design*. A única diferença entre os dois modelos é a quantidade de *features* de saída.

A segunda etapa é a configuração do *MFCC*. Foram definidos os coeficientes padrões recomendados pela plataforma, apresentados na tabela 3.2.

<i>Number of cepstral coefficients</i>	13
<i>Frame length</i>	0.02 s
<i>Frame stride</i>	0.02 s
<i>Filter number</i>	32
<i>FFT length</i>	256
<i>Normalization window size</i>	101
<i>Low frequency</i>	0 Hz
<i>High frequency</i>	0 Hz
<i>Pre-emphasis coefficient</i>	0.98

Tabela 3.2: Parâmetros do MFCC

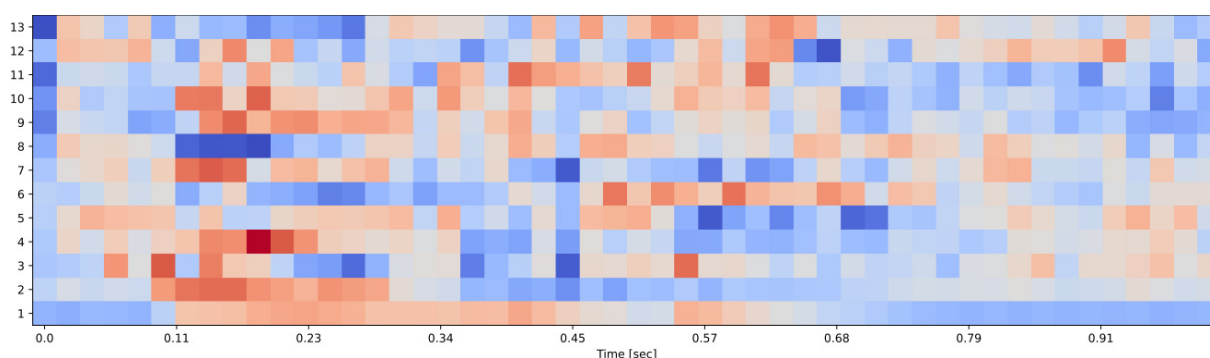


Figura 12: Coeficientes cepstrais de um dado gerado pelo Edge Impulse

Fonte: Edge Impulse.

A Figura 12 mostra um exemplo dos coeficientes cepstrais de um dado qualquer dos dados de treinamento do modelo classificando onze palavras após processado pelo MFCC. São esses coeficientes que serão utilizados como entrada da rede neural para a classificação do sinal de áudio, que são mostrados no primeiro bloco (*Input layer*) nas Figuras 13 e 14. São um total de 650 *features*, pois é o tamanho total do histograma (13 x 50).

A terceira etapa é a configuração da rede neural. A tabela 3.3 apresenta os parâmetros definidos para ambos os modelos.

Número de épocas	100
Taxa de aprendizagem	0.005
Tamanho do <i>batch</i>	32

Tabela 3.3: Parâmetros da Rede Neural

Para a arquitetura da rede neural, foram definidas as camadas recomendadas pela plataforma:

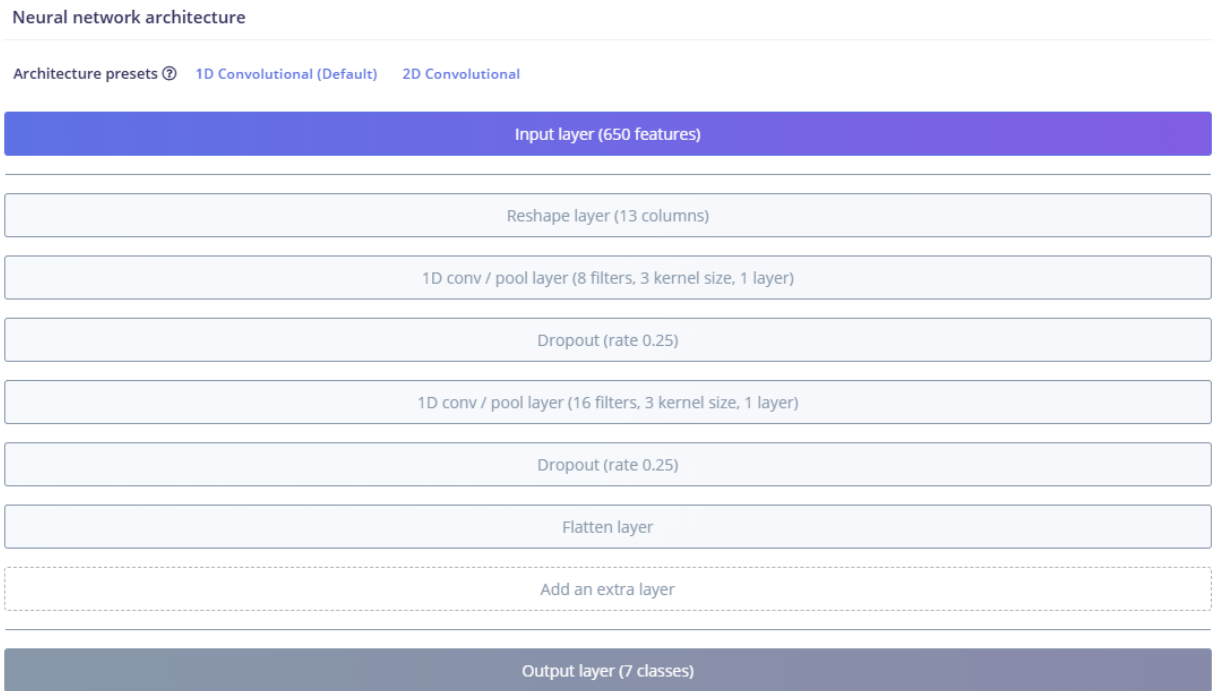


Figura 13: Arquitetura do modelo da rede neural classificando seis palavras
Fonte: Edge Impulse.

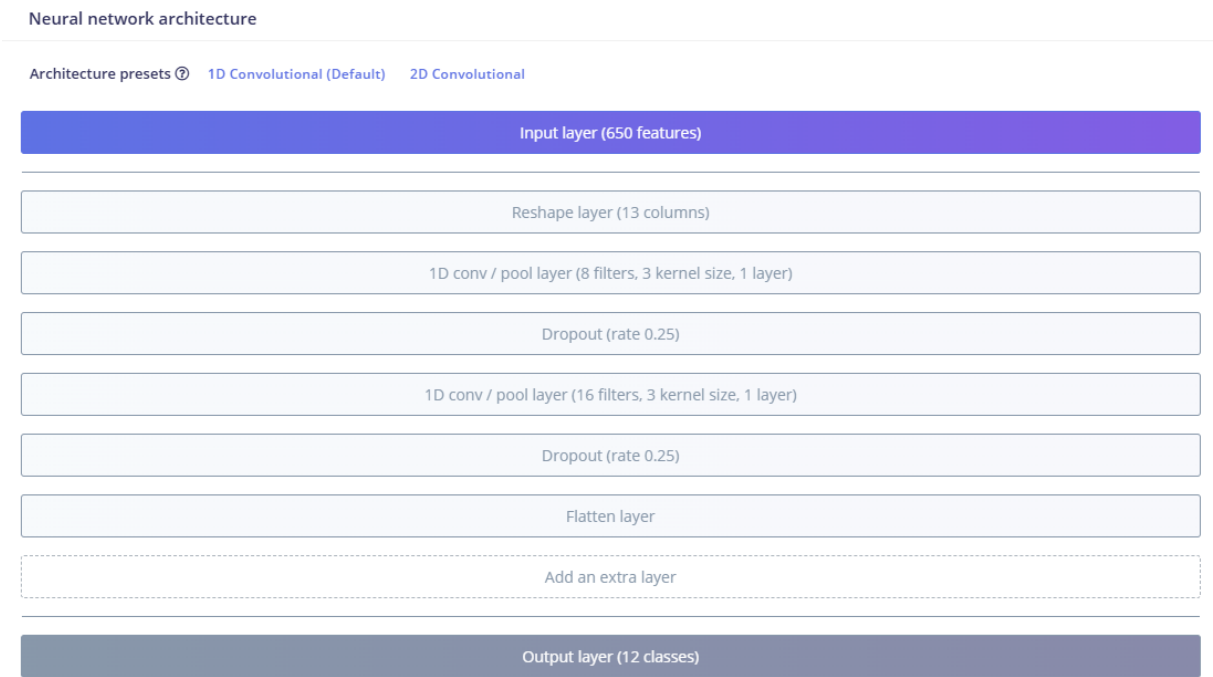


Figura 14: Arquitetura do modelo da rede neural classificando onze palavras
Fonte: Edge Impulse.

As Figuras 13 e 14 apresentam as camadas padrões recomendadas pela plataforma *Edge Impulse*. A única diferença entre os dois modelos é o último bloco (*Output layer*), que é a saída da rede neural, sendo respectivamente os sete e os doze comandos possíveis para os modelos classificando seis e onze palavras e ruído. A tabela 3.4 apresenta a função de cada camada da rede neural.

<i>Camada (Layer)</i>	<i>Função</i>
<i>Input</i>	Representa as 650 <i>features</i> de entrada (espectrograma gerado pelo MFCC)
<i>Reshape</i>	Redimensiona as <i>features</i> de entrada de 650 para 13 (quantidade de coeficientes cepstrais do MFCC)
<i>1D conv (8 filtros)</i>	Aprende as características mais aparentes
<i>Dropout (0.25)</i>	Desliga 25% dos neurônios para evitar <i>overfitting</i>
<i>1D conv (16 filtros)</i>	Aprende as características mais complexas
<i>Flatten</i>	Reduz a dimensionalidade para 1 dimensão
<i>Output</i>	Representa as classes de saída

Tabela 3.4: Camadas da Rede Neural

Depois de configurada, a rede neural de cada modelo foi treinada com o subconjunto de treinamento. Na quarta etapa, os modelos foram testados com seu subconjunto de teste respectivo. Após testados, os modelos passaram pela quinta etapa, onde foram treinados novamente com base nos resultados que obtiveram na etapa anterior. Por fim, após serem retreinados, os modelos foram exportados para poderem ser utilizados no microcontrolador do Arduino Nano 33 BLE Sense.

Resultados

Neste capítulo, serão apresentados os resultados obtidos durante o desenvolvimento e testes dos modelos de inteligência artificial e do sistema embarcado.

4.1 Modelos de IA

Inicialmente, o banco de dados foi dividido em 70% para o subconjunto de treinamento e 30% para teste. Porém, ao treinar o modelo classificando seis palavras pela primeira vez, a plataforma sugeriu dividir em 80% para treinamento e 20% para teste, então foi realizada essa mudança.

As Figuras 15 e 16 representam as matrizes de confusão durante o treinamento e o teste do modelo detectando seis palavras, obtendo respectivamente **89,9%** e **81,84%** de acurácia.

	BACKWARD	FORWARD	GO	LEFT	NOISE	RIGHT	STOP
BACKWARD	81.5%	5.0%	2.1%	5.5%	1.7%	2.9%	1.3%
FORWARD	0%	96.2%	0.8%	0%	1.3%	0.8%	0.8%
GO	0%	2.0%	93.1%	2.0%	1.6%	0.4%	0.8%
LEFT	0.9%	0%	2.7%	86.2%	2.2%	5.8%	2.2%
NOISE	1.9%	0.6%	0.6%	0%	93.5%	1.3%	1.9%
RIGHT	0%	0.9%	2.7%	4.4%	1.3%	90.3%	0.4%
STOP	0%	1.3%	5.3%	1.3%	2.2%	0.4%	89.4%
F1 SCORE	0.89	0.93	0.90	0.86	0.90	0.89	0.91

Figura 15: Matriz de Confusão da rede neural do modelo detectando seis palavras após treinamento gerado pelo Edge Impulse

Fonte: Edge Impulse.

	BACKWARD	FORWARD	GO	LEFT	NOISE	RIGHT	STOP	UNCERTAIN
BACKWARD	75.9%	1.0%	0.3%	1.0%	0.7%	0.7%	0.3%	20%
FORWARD	0%	86.6%	0.7%	0%	0.7%	0.3%	0.3%	11.4%
GO	1%	0.7%	79.7%	0.3%	0%	0%	1%	17.3%
LEFT	0%	0%	0.5%	82.3%	0%	1.6%	0.5%	15.1%
NOISE	0%	0%	0.3%	1.6%	75.6%	0.3%	2.3%	19.9%
RIGHT	0%	0.3%	0%	3.7%	0.7%	85%	0%	10.3%
STOP	0%	0.3%	1.0%	0.3%	0.3%	0%	87.6%	10.4%
F1 SCORE	0.86	0.92	0.87	0.89	0.85	0.90	0.91	

Figura 16: Resultados da rede neural do modelo detectando seis palavra após teste gerado pelo Edge Impulse

Fonte: Edge Impulse.

Após obter bons resultados com o modelo detectando seis palavra, foi decidido ir além e criar um modelo com mais *features* de saída, detectando onze palavras. No primeiro treinamento da rede neural do modelo detectando onze palavras foi selecionado a opção de adicionar grande quantidade de ruído no subconjunto de treinamento, mas isso não foi bom para a rede neural pois aumentava o erro de classificação de ruído.

	BACKWARD	FIVE	FORWARD	FOUR	GO	LEFT	NOISE	ONE	RIGHT	STOP	THREE	TWO
BACKWARD	73.8%	1.7%	1.3%	2.1%	4.2%	7.2%	2.5%	3.4%	0.8%	0.4%	0%	2.5%
FIVE	0.8%	87.3%	1.2%	0.4%	0.4%	2.8%	1.6%	0.4%	0.8%	1.6%	2.0%	0.8%
FORWARD	1.6%	3.3%	71.6%	13.2%	6.6%	0%	0.8%	0.4%	0%	1.6%	0%	0.8%
FOUR	0.4%	2.3%	23.6%	54.0%	13.7%	0.8%	0.4%	1.9%	0%	1.5%	0%	1.5%
GO	1.7%	0.8%	2.1%	1.3%	85.6%	0.8%	1.3%	0%	0%	1.7%	0.4%	4.2%
LEFT	2.1%	1.2%	0.4%	0.4%	1.2%	90.9%	1.2%	0.4%	0.4%	0%	0.4%	1.2%
NOISE	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%
ONE	1.3%	7.8%	0.9%	1.7%	1.7%	2.2%	0.4%	82.7%	0.9%	0%	0.4%	0%
RIGHT	2.1%	7.6%	0.4%	0%	0.8%	7.6%	0%	1.7%	76.9%	0%	2.1%	0.8%
STOP	0.4%	2.9%	0.4%	0.8%	4.2%	1.7%	1.7%	0%	0%	87.4%	0%	0.4%
THREE	0.4%	0.9%	0.4%	0.4%	0.4%	2.2%	2.2%	0%	1.3%	0.4%	82.0%	9.2%
TWO	0%	0%	0.4%	0%	9.2%	0.4%	2.6%	0%	0%	0.9%	0.9%	85.6%
F1 SCORE	0.80	0.81	0.70	0.63	0.75	0.84	0.84	0.86	0.85	0.89	0.87	0.82

Figura 17: Matriz de Confusão da rede neural do modelo detectando onze palavras com alto ruído nos dados após treinamento gerado pelo Edge Impulse

Fonte: Edge Impulse.

	BACKWARD	FIVE	FORWARD	FOUR	GO	LEFT	NOISE	ONE	RIGHT	STOP	THREE	TWO	UNCERTAIN
BACKWARD	54.9%	0.7%	0.7%	0%	2.0%	0.3%	0.3%	0.3%	0%	0%	0%	0.7%	40%
FIVE	0%	75.8%	0%	0%	0%	0%	0.3%	0.7%	0%	0.3%	0.3%	0%	22.6%
FORWARD	0.3%	1.7%	31.5%	1.7%	1.0%	0%	0.3%	0%	0%	0%	0%	0%	63.4%
FOUR	0%	2.4%	7.2%	22.9%	2.0%	0.7%	1.0%	0.3%	0%	0.3%	0%	0.7%	62.5%
GO	0.3%	0%	0%	0%	66.3%	0%	0%	0%	0%	0.3%	0%	1.7%	31.3%
LEFT	0.3%	0.2%	0%	0%	0%	79.5%	0.7%	0.2%	0%	0.2%	0%	0%	18.9%
NOISE	0%	0%	0%	0%	0.3%	1.0%	70.4%	0%	0%	2.9%	0%	0%	25.4%
ONE	0.3%	0%	0.3%	0%	0.3%	1.3%	0%	67.3%	0%	0%	0%	0%	30.3%
RIGHT	0%	1.7%	0%	0%	0%	3.3%	0%	1%	62%	0%	1%	0%	31%
STOP	0%	1.0%	0%	0%	0.3%	0%	0%	0%	0%	84.9%	0%	0%	13.7%
THREE	0%	0.3%	0%	0%	0%	0%	0.7%	0%	0.3%	0%	83.4%	1.0%	14.1%
TWO	0%	0%	0%	0%	5.7%	0%	0.3%	0%	0%	1.0%	0.3%	71.2%	21.4%
F1 SCORE	0.70	0.82	0.45	0.37	0.75	0.87	0.81	0.79	0.76	0.89	0.90	0.81	

Figura 18: Resultados da rede neural do modelo detectando onze palavras com alto ruído nos dados após teste gerado pelo Edge Impulse

Fonte: Edge Impulse.

As Figuras 17 e 18 representam as matrizes de confusão do modelo detectando onze palavras durante o treinamento e o teste, com adição de auto ruído nos dados durante treinamento, obtendo respectivamente **80,3%** e **65,34%** de acurácia. Embora pouco aparente ao analisar a matriz de confusão, na prática o erro de classificação era mais perceptível. Foi removida essa opção, e a taxa de erro diminuiu.

	BACKWARD	FIVE	FORWARD	FOUR	GO	LEFT	NOISE	ONE	RIGHT	STOP	THREE	TWO
BACKWARD	72.2%	5.1%	2.1%	0.4%	3.8%	10.5%	2.1%	1.3%	0.4%	0%	0%	2.1%
FIVE	0.8%	88.5%	1.2%	0.8%	0%	2.4%	0.8%	0.8%	0.4%	1.2%	2.8%	0.4%
FORWARD	0.4%	5.3%	69.5%	18.1%	3.3%	0.4%	0%	2.1%	0%	0.4%	0.4%	0%
FOUR	0.4%	2.3%	18.3%	68.4%	6.5%	0%	0.4%	0.4%	0%	1.1%	0%	2.3%
GO	0.4%	1.3%	2.1%	2.5%	84.3%	1.3%	0.4%	0.4%	0.4%	1.7%	0%	5.1%
LEFT	1.7%	2.9%	1.2%	0.8%	0.8%	89.7%	0.4%	0%	1.7%	0%	0%	0.8%
NOISE	0%	0%	0%	0%	1.1%	0%	98.9%	0%	0%	0%	0%	0%
ONE	0.9%	6.9%	2.2%	3.5%	2.6%	1.7%	0%	80.5%	0.9%	0%	0.4%	0.4%
RIGHT	2.5%	6.7%	0.8%	0.4%	0.4%	4.2%	0%	2.5%	78.6%	0%	3.4%	0.4%
STOP	0.4%	3.3%	1.3%	1.7%	0.8%	1.3%	0.4%	0%	0%	90.0%	0%	0.8%
THREE	0%	2.2%	0.9%	0.4%	0%	1.3%	1.3%	0.9%	0.4%	0.9%	82.5%	9.2%
TWO	0%	0.9%	2.2%	0.4%	6.6%	1.7%	0%	0%	0%	0.4%	0.9%	86.9%
F1 SCORE	0.80	0.79	0.69	0.70	0.80	0.84	0.92	0.85	0.86	0.92	0.86	0.83

Figura 19: Matriz de Confusão da rede neural do modelo detectando onze palavras sem ruído nos dados após treinamento gerado pelo Edge Impulse

Fonte: Edge Impulse.

	BACKWARD	FIVE	FORWARD	FOUR	GO	LEFT	NOISE	ONE	RIGHT	STOP	THREE	TWO	UNCERTAIN
BACKWARD	59.0%	1.4%	0.3%	0%	0.7%	0.3%	0%	0.3%	0%	0%	0%	0.3%	37.6%
FIVE	0%	78.5%	0.3%	0.3%	0%	0.3%	0%	0.3%	0%	0.3%	0%	0%	19.9%
FORWARD	0%	1.7%	37.6%	3.4%	0%	0%	0.3%	0%	0%	0%	0%	0.3%	56.7%
FOUR	0%	2.7%	6.5%	44.0%	0%	0%	0.3%	0%	0%	0.3%	0%	0.7%	45.4%
GO	0%	0%	0%	0%	60%	0%	0%	0%	0%	1%	0%	2%	37%
LEFT	0%	0.2%	0%	0%	0%	80.4%	0%	0.3%	0.3%	0.2%	0%	0%	18.6%
NOISE	0%	0%	0%	0%	0%	7.4%	50.8%	0%	0%	1.9%	0.3%	0.6%	38.9%
ONE	0%	0.7%	0.7%	0.3%	0%	1.3%	0%	66%	0.7%	0%	0%	0%	30.3%
RIGHT	0%	1%	0%	0%	0%	2.7%	0%	1%	65.3%	0%	1.3%	0%	28.7%
STOP	0%	0.7%	0%	0%	0.7%	0%	0%	0%	0%	83.3%	0%	0%	15.4%
THREE	0%	0.3%	0%	0%	0%	0%	0.7%	0%	0%	0%	76.6%	1.7%	20.7%
TWO	0%	0%	0%	0%	3.0%	0.3%	0.7%	0%	0%	0.3%	0%	75.6%	20.1%
F1 SCORE	0.74	0.84	0.52	0.59	0.73	0.86	0.67	0.78	0.78	0.89	0.86	0.83	

Figura 20: Resultados da rede neural do modelo detectando onze palavras sem ruído nos dados após teste gerado pelo Edge Impulse

Fonte: Edge Impulse.

As Figuras 19 e 20 representam as matrizes de confusão do modelo detectando onze palavras durante o treinamento e o teste, sem nenhuma adição de ruído nos dados durante treinamento, obtendo assim respectivamente **81,5%** e **65,84%** de acurácia. Embora a acurácia treinando o modelo detectando onze palavras retirando o ruído no subconjunto de treinamento tenha sido maior que a acurácia treinando-o com baixo ruído (próximo teste), ao testá-lo tanto com o subconjunto de teste como também ao vivo com o microcontrolador, o modelo detectando onze palavras treinado com baixo ruído na base de treinamento obteve o melhor resultado geral.

Por fim, no treinamento final do modelo detectando onze palavras, foi adicionado baixa quantidade de ruído no subconjunto, obtendo assim um resultado melhor que o anterior. Após o treinamento final da rede neural, obteve-se a seguinte matriz de confusão.

	BACKWARD	FIVE	FORWARD	FOUR	GO	LEFT	NOISE	ONE	RIGHT	STOP	THREE	TWO
BACKWARD	72.2%	1.7%	3.8%	0.4%	9.7%	6.3%	1.3%	3.8%	0.4%	0.4%	0%	0%
FIVE	0.4%	88.5%	2.0%	0.8%	1.2%	1.6%	0.4%	1.6%	0.8%	0.8%	0.8%	1.2%
FORWARD	0.8%	4.1%	79.4%	9.9%	3.7%	0%	0%	1.2%	0%	0.4%	0%	0.4%
FOUR	0%	3.4%	31.6%	55.5%	4.2%	0.4%	0%	2.7%	0%	1.5%	0.4%	0.4%
GO	1.7%	1.3%	1.7%	3.4%	85.6%	1.3%	0%	0.4%	0%	2.5%	0%	2.1%
LEFT	0.8%	2.1%	0.8%	0.4%	2.9%	87.6%	0.8%	2.9%	1.7%	0%	0%	0%
NOISE	0%	0%	0%	0%	2.2%	0%	97.8%	0%	0%	0%	0%	0%
ONE	2.2%	6.5%	1.3%	2.2%	0.4%	1.7%	0%	84.8%	0.9%	0%	0%	0%
RIGHT	1.3%	6.3%	1.7%	0.4%	1.7%	6.3%	0%	2.1%	78.2%	0%	2.1%	0%
STOP	0.4%	3.8%	1.3%	1.7%	1.3%	1.7%	0%	0%	0%	89.5%	0%	0.4%
THREE	0%	0.9%	0.9%	0%	0.9%	2.2%	0.4%	1.3%	0%	0.4%	87.7%	5.3%
TWO	0.4%	1.3%	2.2%	0.9%	11.8%	0%	0%	0%	0%	3.1%	1.7%	78.6%
F1 SCORE	0.80	0.81	0.69	0.64	0.76	0.84	0.95	0.84	0.86	0.90	0.91	0.83

Figura 21: Matriz de Confusão da rede neural final do modelo detectando onze palavras após treinamento gerado pelo Edge Impulse

Fonte: Edge Impulse.

	BACKWARD	FIVE	FORWARD	FOUR	GO	LEFT	NOISE	ONE	RIGHT	STOP	THREE	TWO	UNCERTAIN
BACKWARD	60.7%	0.7%	0.7%	0%	3.4%	1.0%	0%	0.7%	0%	0%	0.3%	0%	32.5%
FIVE	0%	76.8%	1.0%	0%	0%	0%	0%	1.0%	0%	0%	0.7%	0%	20.5%
FORWARD	0%	1.3%	53.7%	3.4%	0.7%	0%	0.3%	0.3%	0%	0%	0%	0%	40.3%
FOUR	0%	1.7%	13.0%	27.6%	0.7%	0%	0%	0.3%	0%	0%	0%	0.7%	56.0%
GO	0.3%	0%	0%	0%	68.3%	0%	0%	0.3%	0%	0.7%	0%	1.3%	29%
LEFT	0%	0.2%	0%	0%	0.2%	76.0%	0.2%	1.0%	0%	0.5%	0%	0%	21.9%
NOISE	0%	0%	0%	0%	1.6%	5.1%	49.5%	0%	0%	1.9%	0.6%	0.3%	40.8%
ONE	0%	0.7%	0.3%	0.3%	0.3%	1%	0%	76%	0%	0%	0%	0%	21.3%
RIGHT	0.3%	2.3%	0%	0%	0%	4.7%	0%	1%	65.7%	0%	2.3%	0%	23.7%
STOP	0%	0.7%	0%	0%	0%	0%	0%	0%	0%	84.9%	0%	0%	14.4%
THREE	0%	0.3%	0.3%	0%	0%	0%	0.3%	0%	0.3%	0%	81.7%	0.3%	16.6%
TWO	0%	0%	0%	0%	6.4%	0%	0.3%	0%	0%	1.7%	1.3%	63.9%	26.4%
F1 SCORE	0.75	0.83	0.64	0.42	0.75	0.83	0.66	0.84	0.79	0.89	0.87	0.77	

Figura 22: Resultado da rede neural final do modelo detectando onze palavras após teste gerado pelo Edge Impulse

Fonte: Edge Impulse.

As Figuras 21 e 22 representam as matrizes de confusão do modelo detectando onze palavras durante o treinamento e o teste, com baixa adição de ruído nos dados durante treinamento, obtendo respectivamente **81,1%** e **66,15%** de acurácia. A tabela 4.1 compara os valores de acurácia dos modelos testados, ressaltando que todos os modelos, além das palavras, classificam também ruído.

<i>Modelo</i>	<i>Acurácia</i>
<i>Detectando 6 palavras, sem adições</i>	81,84%
<i>Detectando 11 palavras, com alto ruído</i>	65,34%
<i>Detectando 11 palavras, sem ruído</i>	65,84%
<i>Detectando 11 palavras, com baixo ruído (sua versão final)</i>	66,15%

Tabela 4.1: Acurácia dos modelos

Pode-se perceber que as redes neurais do modelo detectando onze palavras confundem entre as palavras *four* e *forward*, o que era esperado devido ao pronunciamento similar entre elas. Os modelos também têm uma menor acurácia na detecção de ruído, o que também é esperado, pois o ruído é muito variável, então uma amostra com volume mais baixo, ou gravada com um microfone com chiado pode ser confundida com ruído.

4.2 Sistema Embarcado

O *firmware* e o *hardware* demonstraram ser robustos e eficientes para o sistema. O microcontrolador do Arduino foi capaz de executar a rede neural, levando apenas cerca de 4 ms para classificar um dado, como também enviar os comandos via USART. O microcontrolador STM32F103C8T6 também conseguiu suportar o *firmware*, utilizando apenas 45.6% da memória flash e 77.8% da memória RAM.

O desenvolvimento do *firmware* e do *hardware* de maneira modular deixou o sistema funcional, onde não é necessária uma profunda observação para entender seu funcionamento. Esse método também facilita para a manutenção futura do código, pois como cada módulo funciona individualmente, uma alteração interna de um módulo específico não afetará a sua utilização externamente, funcionando parecido como uma arquitetura baseada em *microserviços*.

Inicialmente, havia a preocupação de que o protocolo USART não fosse capaz de suprir a demanda de informações. No entanto, essa preocupação foi rapidamente descartada, pois o protocolo funcionou de forma eficiente. Isso se deve ao *delay* existente no *firmware* do Arduino entre as aquisições de *samples*, que proporciona tempo suficiente para o envio das informações pelo protocolo enquanto aguarda a próxima classificação.

Por ter utilizado módulos para os testes iniciais não houveram problemas de *hardware*, que geralmente costuma acontecer quando se monta um protótipo. Havia um certo receio se a tensão de 3.7 V iria suprir a necessidade de tensão dos motores, mas funcionou bem.

4.3 Desafios, soluções e lições aprendidas

Esta seção descreve os principais desafios enfrentados durante o desenvolvimento do trabalho, as estratégias adotadas para superá-los e as lições extraídas a partir de cada situação.

4.3.1 Banco de Dados

Os áudios iniciais de ruído utilizados foram apenas retirados do banco de dados do *Speech Commands: A dataset for Limited-Vocabulary Speech Recognition*, porém após os primeiros testes o modelo não se apresentou muito eficiente, pois ao detectar voz de plano de fundo o modelo retornava alguma das palavras-chave ao invés de detectar como ruído. Adicionando demais áudios retirados do site *pixbay*, houve uma considerável redução desse problema.

Um problema que não foi detectado inicialmente analisando a matriz de confusão do modelo foi durante a classificação da palavra *Left*. O modelo as vezes classificava como *Left* quando detectava algum ruído. Realizando uma busca mais profunda no banco de dados, percebeu-se que nos dados desta palavra tinha uma demora maior para aparecer os primeiros sinais de áudio. Enquanto nas outras palavras a média de aparecer os primeiros sinais de áudio era de aproximadamente 200 ms, nos dados da palavra *Left* a média era cerca de 500 ms, o que acarretou na

classificação errada. Como as palavras foram escolhidas aleatoriamente, o algoritmo de seleção de palavras foi executado novamente, apenas para os dados desta palavra. Após treinar o modelo novamente e testar, obteve uma melhora, mas não mitigou 100% do problema.

4.3.2 Firmware

Nos testes iniciais, como haviam menos de dez palavras-chave os comandos eram enviados entre os microcontroladores como um valor de 0 a 9. Depois da inserção de novas palavras, a quantidade de comandos aumentou para doze, sendo necessário alterar o modo de envio. Por ter tamanho igual ao tamanho máximo de envio do protocolo USART (1 *byte*), os comandos passaram a ser enviados com o tipo *char*, resolvendo assim a questão e aumentando a quantidade possível de comandos, caso necessário.

Por padrão, o arquivo de *overlay* do microcontrolador STM32F103C8T6 utiliza os canais 1 e 2 do protocolo USART para comunicação com o monitor serial, o que estava causando confusão no canal 1, que foi o canal utilizado para comunicar com o Arduino. As portas dos canais 2 e 3 estavam ocupadas pois estavam sendo utilizadas pelos *drivers* do *display* e do *driver* de motor. Após realizar pesquisas, o problema foi sanado, apenas sendo necessário sobrescrever os valores padrões do canal 1 para o canal 3 e desabilitando o canal 2.

Também houve uma dificuldade durante o desenvolvimento do *driver* do *display*, pois na documentação do *driver solomon,ssd1306fb* (Solomon Systech Limited, 2025) no Zephyr não mostra um componente que é obrigatório para o *driver* funcionar, mas após pesquisar profundamente foi descoberto o componente *cs-gpios*, que corresponde ao pino CS da comunicação SPI, solucionando assim o problema.

4.3.3 Hardware

A PCI começou a ser montada inicialmente com apenas duas camadas, mas a medida que se foram adicionando os componentes a complexidade do roteamento aumentou, sendo necessário colocar mais camadas, o que atrasou um pouco a montagem, pois foi necessário refazer as trilhas. Porém, ao adicionar mais camadas, o roteamento se tornou mais simples, pois bastava conectar os componentes ou criar máscaras diretamente nas camadas de *ground* ou 3.3 V, sem a necessidade de criar várias trilhas. Adicionar uma camada inteira de *ground* também ajuda a reduzir possíveis interferências entre os componentes.

4.4 Síntese dos Resultados

Os resultados obtidos foram satisfatórios para a proposta do projeto. Embora o modelo às vezes classifique uma palavra errada, como *Left* ao invés de *Noise*, ele consegue classificar todas as

palavras, o que era o objetivo principal. A adição de pouca quantidade de ruído no subconjunto de treinamento foi de grande importância para que o modelo conseguisse classificar melhor.

Sem as técnicas de compressão do Edge Impulse, os modelos detectando seis e onze palavras teriam respectivamente tamanhos de 168.4 e 172.5 KB de flash. Embora ainda coubessem na memória do Arduino Nano 33 BLE Sense, que possui 1 MB de flash, após as compressões os modelos possuem apenas cerca de 34 KB.

A acurácia de 66% do modelo detectando onze palavras foi um resultado relevante, pois devido a limitação de *hardware* é difícil que um modelo de *TinyML* atinja altos valores, ainda mais precisando classificar doze *features*, quantidade bastante superior ao usual. O MFCC demora cerca de 258 ms para processar o áudio e o modelo demora 4 ms para classificar, e, juntamente com o *delay* de 100 ms do envio de informação via USART, o sistema leva aproximadamente 362 ms para identificar um comando, ocupando cerca de 15.5KB de memória RAM e 33.8KB de memória flash do microcontrolador Arduino Nano 33 BLE Sense, valores bons considerando o hardware utilizado. O microcontrolador STM32F103C8T6 utilizou cerca de 15.56 KB de memória RAM e 29.2KB de memória flash para executar o RTOS. Esse é outro fato relevante e importante para uma aplicação em tempo real.

Somando o consumo de memória dos dois microcontroladores, foram utilizados no total 31.06KB de memória flash e 63.0KB de memória RAM. O microcontrolador Arduino Nano 33 BLE Sense supriria sozinho o quesito do consumo de memória, pois possui 1MB de flash e 256KB de RAM, mas seria necessário analisar outros quesitos como poder de processamento para definir se esse microcontrolador executaria sozinho todo o *firmware* do sistema.

O *firmware* e o *hardware* conseguiram operar de maneira concisa, fazendo com que o sistema embarcado funcionasse de forma concreta, sem travamentos ou gargalos. Às vezes, é comum que uma das partes tenha algum *delay* ou falha, acarretando instabilidade no sistema, podendo até ocasionar um travamento. Esse fato que não ocorreu no projeto, onde o sistema em um todo conseguiu operar de forma fluida e contínua.

O fato do *hardware* operar no máximo com 3.7 V também é um valor relevante para um sistema embarcado, pois um dos objetivos é o baixo consumo. Outro objetivo cumprido foi a utilização de uma fonte de energia móvel, sem a necessidade de fios, no caso uma bateria. Utilizando duas baterias 18650 em paralelo permitiu que o veículo funcionasse sem a necessidade de recarga recorrente.

Conclusão

O desenvolvimento deste trabalho mostrou como a inteligência artificial pode ser utilizada eficientemente em dispositivos que possuam limitações de memória, processamento, gasto energético, entre outras. Foram implementados um banco de dados contendo as palavras-chave dos comandos e o modelo de inteligência artificial, que foram refinados e convertidos para o modelo compatível com os microcontroladores na plataforma *Edge Impulse*. Foi desenvolvido o *firmware* e o *hardware* do veículo, que se comunicam com o *devkit* do Arduino, onde está presente modelo de inteligência artificial, para assim controlar o veículo. Os resultados obtidos em todas as seções do projeto foram satisfatórios, reforçando assim o conceito de *TinyML*. O vídeo com a demonstração do funcionamento do veículo está disponível no YouTube.⁶

A seção 5.1 cita as contribuições deste projeto, e a seção 5.2 apresenta possíveis pontos que poderão aprimorar este projeto em trabalhos futuros.

5.1 Contribuições do Projeto

Este trabalho contribui de forma significativa para o desenvolvimento de aplicações de inteligência artificial, especificamente na detecção de palavras-chave, em dispositivos de borda (*edge devices*).

A base de dados implementada pode ser utilizada para o desenvolvimentos de outras aplicações, já contendo com os dados de treino e de teste separados, sendo apenas necessário importar os arquivos no Edge Impulse. Com os resultados obtidos, é possível escolher os melhores parâmetros, a depender da aplicação. Embora seja recomendado voz humana, o MFCC também pode ser utilizado para processar outros tipos de áudio, como sons da natureza, etc.

⁶Elaboração própria, *Vídeo de funcionamento do veículo*. Acessado em: 14 mai. 2025. Disponível em: <<https://youtu.be/iiWRqvPNBO4>>

O *firmware* e o *hardware* não se limitam apenas ao projeto. Por terem sido desenvolvidos de maneira modular, os seus componentes podem ser facilmente utilizados individualmente em outras aplicações.

5.2 Trabalhos Futuros

Com base nos resultados que foram apresentados nas seções anteriores, é possível destacar alguns pontos que poderão estender este trabalho, como:

- **Refazer a base de dados:** Refazer a base de dados gravando os áudios através do próprio microfone do *devkit* do microcontrolador Arduino Nano 33 BLE Sense, o que poderá melhorar a acurácia do modelo.
- **Testar outros blocos de processamento:** Além do MFCC, existem outros blocos para processamento de áudio. Embora o MFCC seja o recomendado para voz humana, poderá haver algum outro bloco que venha a obter melhores resultados, como menor tempo de processamento e menor uso de memória.
- **Aprimorar o modelo:** Testar outros parâmetros do modelo, a fim de encontrar uma acurácia melhor que a atual.
- **Aprimorar a comunicação:** Testar se a implementação de outros protocolos de comunicação (como o SPI e I²C) diminuirá o tempo total de classificação de um comando.
- **Verificar a execução total do *firmware* no Arduino Nano 33 BLE Sense:** Verificar se o microcontrolador Arduino Nano 33 BLE Sense executaria sozinho o *firmware* total do sistema, ou seja, tanto o modelo de IA como o RTOS, visto que este microcontrolador possui memória RAM e flash suficientes e é suportado tanto pelo Edge Impulse como pelo Zephyr. Talvez seja necessário analisar como exportar o modelo de IA para o Zephyr.
- **Expandir o gerenciador do veículo:** Implementar novas funcionalidades do gerenciador do veículo, como por exemplo armazenar os comandos recebidos em uma fila e depois executá-los sequencialmente.
- **Elaborar um modelo 3D:** Elaborar um modelo 3D específico para o projeto, integrando o chassi com *case* para proteção do circuito.
- **Melhorar a estética do veículo:** Por se tratar de um veículo, é possível adicionar novos componentes para melhorar a sua estética, como *leds* para simular faróis, servomotores, entre demais componentes.

Referências Bibliográficas

Edgeimpulse, Inc. *Edge Impulse - The Leading Edge AI Platform*. 2025. Acessado em: 28 mar. 2025. Disponível em: <<https://edgeimpulse.com/>>.

Google Cloud. *O que é inteligência artificial (IA)?* 2025. Acessado em: 18 mar. 2025. Disponível em: <<https://cloud.google.com/learn/what-is-artificial-intelligence?hl=pt-BR>>.

Harvard Professional and Lifelong Learning. *Fundamentals of TinyML*. 2024. Acessado em: 27 fev. 2025. Disponível em: <<https://pll.harvard.edu/course/fundamentals-tinyml>>.

IBM. *O que é um microcontrolador?* 2024. Acessado em: 27 fev. 2025. Disponível em: <<https://www.ibm.com/br-pt/think/topics/microcontroller>>.

IBM. *O que é uma rede neural?* 2024. Acessado em: 18 mar. 2025. Disponível em: <<https://www.ibm.com/br-pt/topics/neural-networks>>.

Nama. *O que é TinyML? Conheça o conceito, os benefícios e as fontes para se aprofundar no tema*. 2020. Acessado em: 28 abr. 2025. Disponível em: <<https://simple.nama.ai/post/tinyml-conheca-uma-nova-perspectiva-de-ia>>.

pixabay. *2,422 royalty-free people-talking sound effects*. 2025. Acessado em: 19 mar. 2025. Disponível em: <<https://pixabay.com/sound-effects/search/people-talking/>>.

Solomon Systech Limited. *solomon,ssd1306fb (on spi bus)*. 2025. Acessado em: 19 mar. 2025. Disponível em: <<https://docs.zephyrproject.org/latest/build/dts/api/bindings/display/solomon%2Csd1306fb-spi.html>>.

ST. *STM32F103x8 Datasheet*. 2023. Acessado em: 10 mar. 2025. Disponível em: <<https://www.st.com/resource/en/datasheet/cd00161566.pdf>>.

TensorFlow. *TensorFlow Lite para microcontroladores*. 2021. Acessado em: 28 abr. 2025. Disponível em: <<https://www.tensorflow.org/lite/microcontrollers?hl=pt-br>>.

uday. *MFCC Technique for Speech Recognition*. 2023. Acessado em: 04 abr. 2025. Disponível em: <<https://www.analyticsvidhya.com/blog/2021/06/mfcc-technique-for-speech-recognition/>>.

uTensor. *uTensor GitHub*. 2025. Acessado em: 28 abr. 2025. Disponível em: <<https://github.com/uTensor/uTensor>>.

WARDEN, P. Speech commands: A dataset for limited-vocabulary speech recognition. *Google Brain*, 2018. Disponível em: <<https://arxiv.org/pdf/1804.03209>>.

Zephyr Project. *About the Zephyr Project*. 2023. Acessado em: 10 mar. 2025. Disponível em: <<https://zephyrproject.org/learn-about/>>.

Apêndice A

PCI

A primeira etapa da montagem da PCI foi criar o diagrama de blocos. O diagrama de blocos, apresentado na figura 23, representa os principais componentes do *hardware* e a conexão entre eles. No diagrama do projeto, temos como os principais componentes:

- Bateria (duas baterias do modelo 18650).
- LDO 3.3V, que é o regulador da tensão de entrada para 3.3V.
- MCUs, que são os microcontroladores STM32F103C8T6 e o Arduino Nano 33 BLE Sense.
- Display, que é o display 0.96"OLED SPI.
- Driver de motor, que é o MX1616H.
- Motores (quatro motores genéricos com rodas).
- Cristal, que é o cristal de 16MHz do microcontrolador STM32F103C8T6.
- LED, que é um led genérico para indicar se o sistema está ligado ou desligado.
- Chave de Boot, que é para indicar qual será o boot do microcontrolador STM.

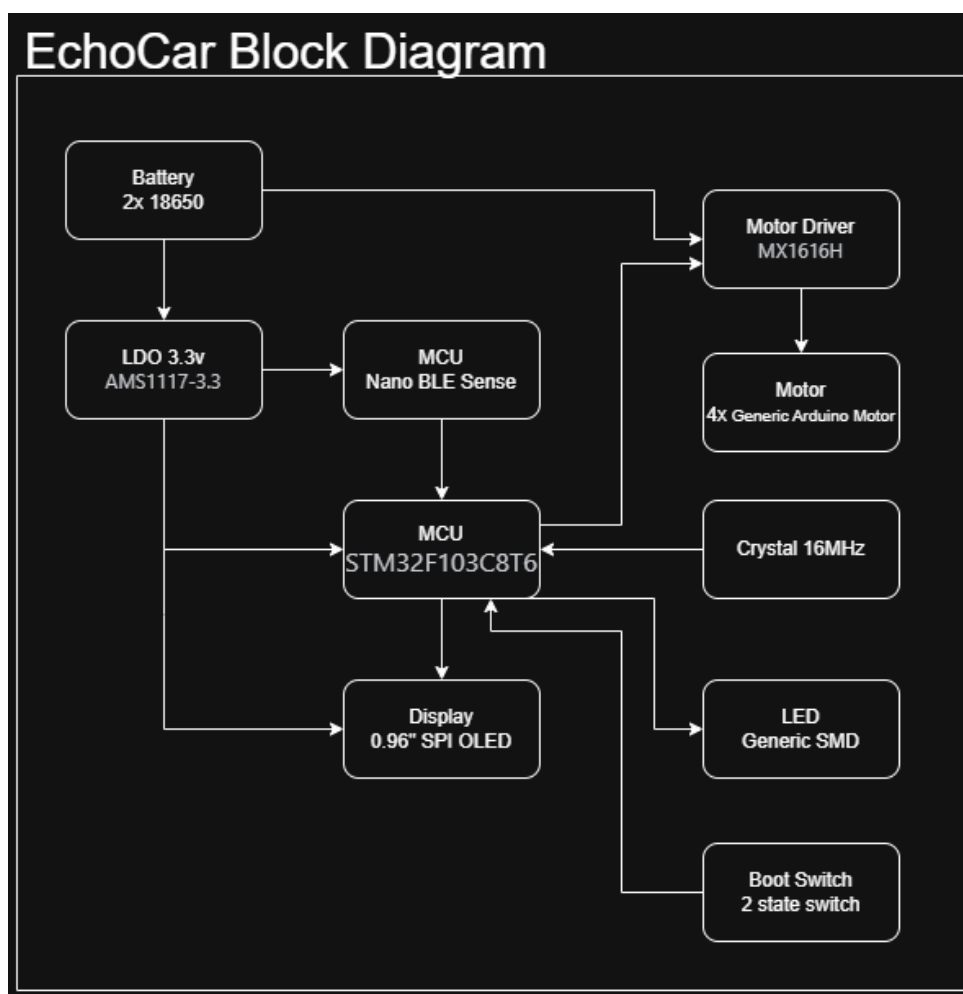


Figura 23: Diagrama de Blocos do hardware

Fonte: Autor.

Com o diagrama montado, a segunda etapa é criar a BOM (*Bill of Materials*). A BOM é uma lista que contém todos os componentes da PCI e suas informações, como referência no esquemático, valor (resistência, capacitância, etc), *link* de venda do produto, *footprint*, quantidade, entre outras. Enquanto o diagrama de blocos possui apenas os principais componentes, a BOM inclui os demais componentes elétricos, como resistores, capacitores, diodos, etc.

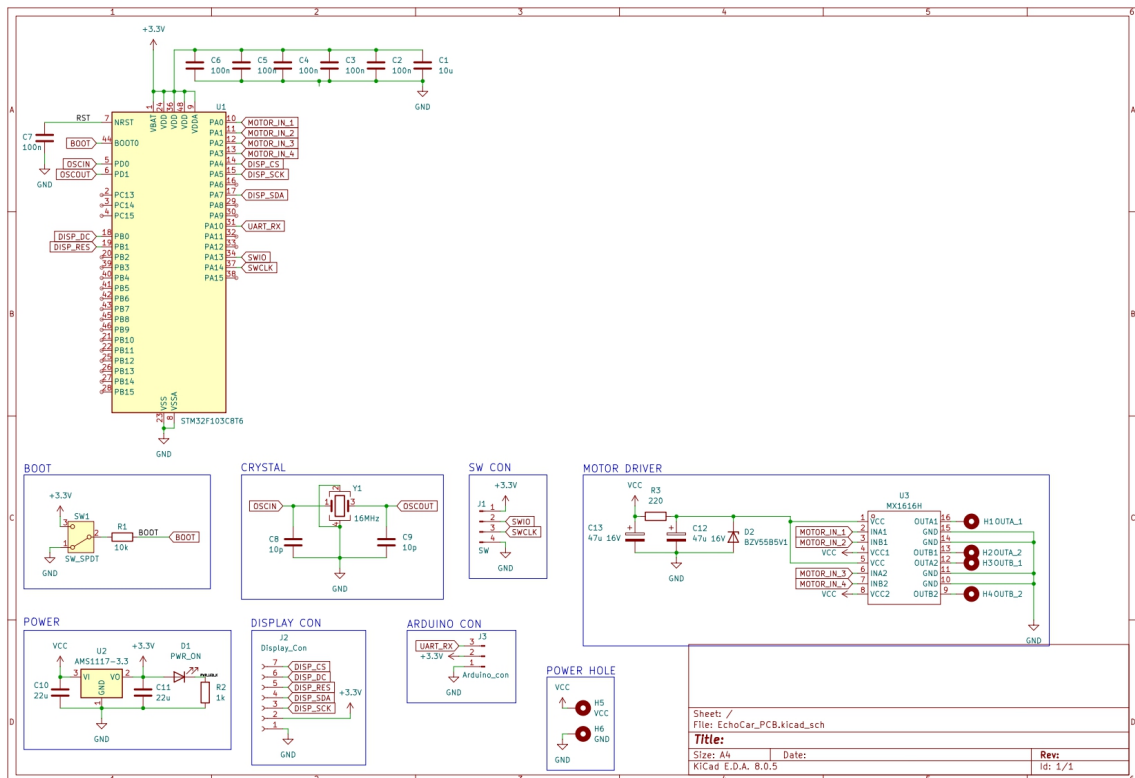
Reference	Value	Link	Footprint	Qty
C1	10u	c-Capacitors-MLCC-SMD-SMT_Murata-Electronics-GRM21BR61H106KE4	Capacitor_SMD:C_0805_2012Metric	1
C2,C3,C4,C5,C6,C7	100n	eramic-Capacitors-MLCC-SMD-SMT_TDK-C1005X7R1H104KT000F_C36	Capacitor_SMD:C_0402_1005Metric	6
C8,C9	10p	eramic-Capacitors-MLCC-SMD-SMT_TDK-CGA2B2NP01H100DT0Y0F_C	Capacitor_SMD:C_0402_1005Metric	2
C10,C11	22u	Capacitors-MLCC-SMD-SMT_Samsung-Electro-Mechanics-CL21A226MAQ	Capacitor_SMD:C_0805_2012Metric	2
C12,C13	47u 16V	ic-Capacitors-SMD_AISHI-Aihua-Group-EMK1CM470E61D00R_C528972	Capacitor_SMD:CP_Elec_5x5.3	2
D1	PWR_ON	LED-Indication-Discrete_Foshan-NationStar-Optoelectronics-NCD0603W3	LED_SMD:LED_0603_1608Metric	1
D2	BZV55B5V1	n/product-detail/Zener-Diodes_EIC-Semicon-BZV55B5V1_C2978889.html?	Diode_SMD:D_MiniMELF	1
J1	SW	x	Connector_PinHeader_2.54mm:PinHeader_1x04_P2.54mm_Vertical	1
J2	Display_Con	x	Connector_PinSocket_1.00mm:PinSocket_1x07_P2.54mm_Vertical	1
J3	Arduino_con	x	Connector_PinHeader_2.54mm:PinHeader_1x03_P2.54mm_Vertical	1
R1	10k	>-Resistor-Surface-Mount_Vishay-Intertech-RCS040210K0FKED_C302023	Resistor_SMD:R_0402_1005Metric	1
R2	1k	sistor-Surface-Mount_UNI-ROYAL-Uniroyal-Elec-CQ02WGJ0102TCE_C27	Resistor_SMD:R_0402_1005Metric	1
R3	220	tail/Chip-Resistor-Surface-Mount_RALEC-RTT022200FTH_C102889.html?	Resistor_SMD:R_0402_1005Metric	1
SW1	SW_SPDT	roduct-detail/Slide-Switches_XKB-Connection-SK12D07L3B_C480339.htm	Button_Switch_THT:SW_E-Switch_EG1224_SPDT_Angled	1
U1	STM32F103C8T6	Microcontrollers-MCU-MPU-SOC_STMicroelectronics-STM32F103C8T6_C	Package_QFP:LQFP-48_7x7mm_P0.5mm	1
U2	AMS1117-3.3	s-Linear-Low-Drop-Out-LDO-Regulators_UMW-Youtai-Semiconductor-Co-L	Package_TO_SOT_SMD:SOT-223-3_TabPin2	1
U3	MX1616H	n/product-detail/Brushed-DC-Motor-Drivers_Mixic-MX1616H_C5119044.htr	MX1616H_SMD:MX1616H	1
Y1	16MHz	il/Crystals_Shenzhen-SCTF-Elec-SX3B16-000F0910F30_C2901667.html?	Crystal:Crystal_SMD_3225-4Pin_3.2x2.5mm	1

Figura 24: BOM

Fonte: Autor.

A figura 24 representa a BOM do projeto.

A terceira etapa é criar o esquemático do projeto, representado na figura 25. É no esquemático do projeto que se posiciona os símbolos de todos os componentes da BOM e as conexões entre eles. Os componentes foram dispostos de acordo com a parte elétrica a qual pertence, como por exemplo a parte de suprimento de energia, de controle dos motores, de seleção de *boot* do microcontrolador, entre outras.



A quarta etapa é criar a PCI em si, ou seja, posicionar o *footprint* dos componentes na placa e criar as conexões físicas entre eles (trilhas), apresentada na figura 26. É a etapa mais complexa do desenvolvimento, pois não existe uma única maneira correta de posicionar os componentes, então é necessário ficar posicionando, testando e reposicionando, até que se fique da melhor maneira possível.

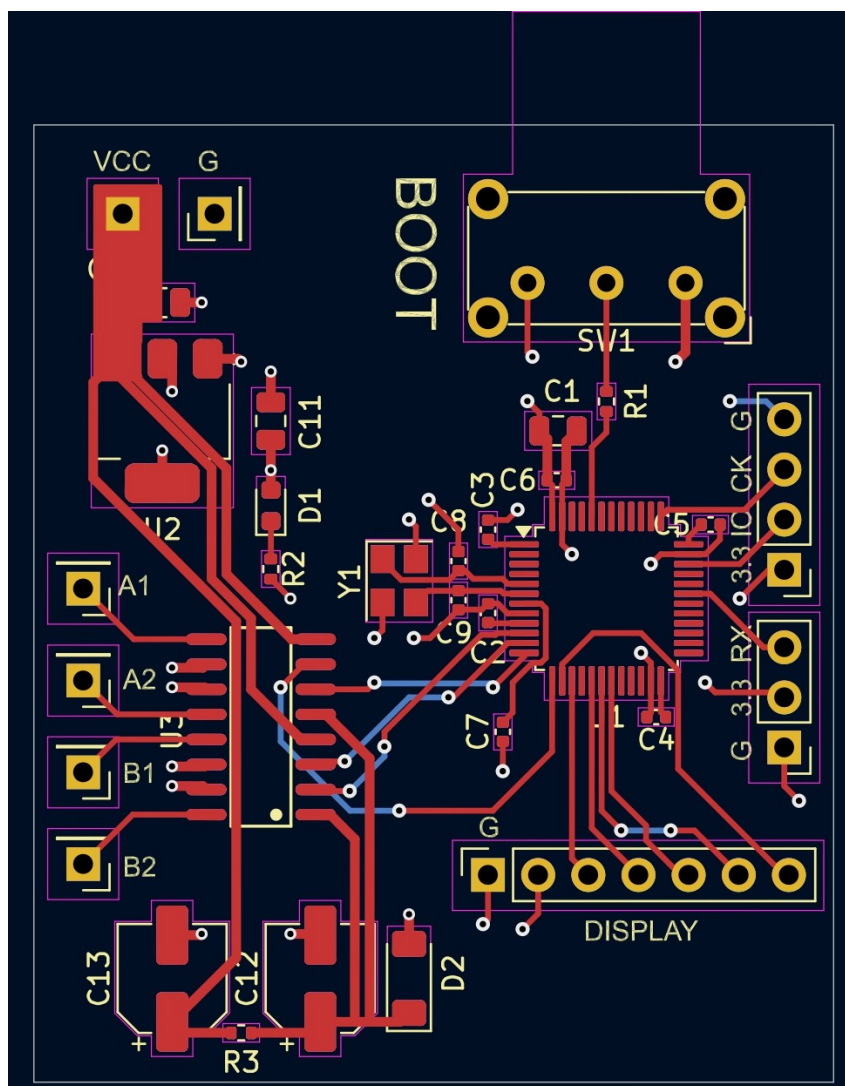


Figura 26: PCI

Fonte: Autor.

Após posicionar e rotear todos os componentes, a quinta e última etapa é gerar os arquivos necessários para produzir a PCI. Esses arquivos são utilizados para encomendar a produção das placas nos sites fornecedores.

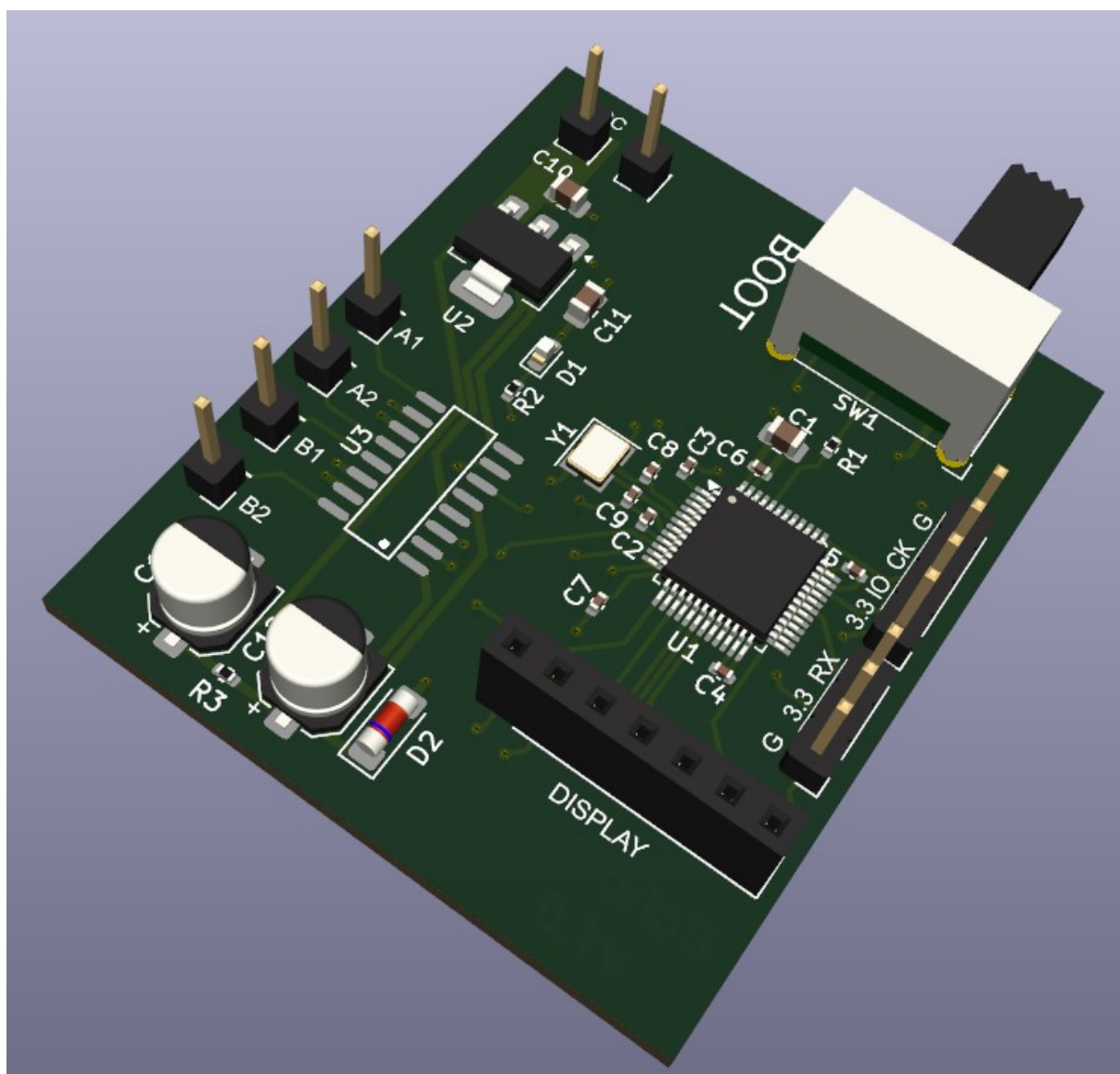


Figura 27: PCI em 3D

Fonte: Autor.

A figura 27 apresenta a PCI finalizada em 3D.