Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

Trabalho de Conclusão de Curso

Sistema de Diagnóstico Preditivo para Veículos a Combustão Usando Machine Learning e Deep Learning

Alexandre Pedro Yure Cariri Gomes

Campina Grande - PB
Outubro de 2024

Alexandre Pedro Yure Cariri Gomes

Sistema de Diagnóstico Preditivo para Veículos a Combustão Usando Machine Learning e Deep Learning

Trabalho de Conclusão de Curso submetido à Coordenação de Curso de Graduação de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Elétrica.

Kyller Costa Gorgônio, UFCG
Professor Orientador

Gutemberg Gonçalves dos Santos Júnior, UFCG
Professor Avaliador

Campina Grande - PB
Outubro de 2024

Dedico este trabalho aos meus pais, por todo apoio e suporte dado ao longo desses anos.

Agradecimentos

Gostaria de expressar minha profunda gratidão a todas as pessoas e instituições que me apoiaram durante a realização deste estágio.

Primeiramente, agradeço a Deus pelo dom da vida e por me dar forças para continuar, apesar das dificuldades e desafios enfrentados.

Agradeço ao meu pai, João Batista, e à minha mãe, Maria de Lourdes, por me acolherem, por seu suporte e motivação ao longo dessa jornada.

Agradeço à Universidade Federal de Campina Grande (UFCG) e ao Departamento de Engenharia Elétrica pela oportunidade de realizar este estágio e por todo o suporte educacional fornecido durante minha formação. Em especial, agradeço ao professor Kyller por sua orientação, paciência e por compartilhar seus conhecimentos e experiências, o que foi fundamental para o meu desenvolvimento profissional.

"A vitória pertence ao mais perseverante ."

Napoleão Bonaparte

Resumo

Este trabalho apresenta o desenvolvimento de um sistema de diagnóstico preditivo para veículos a combustão, utilizando técnicas de Machine Learning e Deep Learning. O objetivo é criar um modelo capaz de prever falhas e verificar a saúde do motor. Para isso, foram utilizados dados de sensores automotivos, como temperatura, pressão, e rotação, e aplicadas técnicas de pré-processamento e feature engineering para otimização dos dados.

Modelos de ML, como Random Forests e Support Vector Machines, foram comparados com Redes Neurais Convolucionais e Redes Neurais Recorrentes, utilizando séries temporais para a detecção de padrões de falha. A validação dos modelos foi realizada em um ambiente de simulação de Hardware-in-the-Loop, utilizando o Model Based Design, permitindo a integração do sistema com hardware real para testes robustos.

Este estudo reforça a viabilidade da aplicação de inteligência artificial na manutenção automotiva, destacando o potencial da tecnologia para prever e evitar falhas antes que ocorram.

Palavras-chave: Diagnóstico preditivo, Machine Learning, Deep Learning, Veículos a combustão, Manutenção preditiva, Hardware-in-the-Loop, Model Based Design.

Abstract

This work presents the development of a predictive diagnostic system for combustion vehicles, using Machine Learning and Deep Learning techniques. The objective is to create a model capable of predicting failures and assessing engine health. For this, data from automotive sensors, such as temperature, pressure, and rotation, were used, and data preprocessing and feature engineering techniques were applied for data optimization.

ML models, such as Random Forests and Support Vector Machines, were compared with Convolutional Neural Networks and Recurrent Neural Networks, using time series to detect failure patterns. The validation of the models was carried out in a Hardware-in-the-Loop simulation environment, using Model Based Design, allowing the system to be integrated with real hardware for robust testing.

This study reinforces the feasibility of applying artificial intelligence in automotive maintenance, highlighting the potential of technology to predict and prevent failures before they occur.

Key-words: Predictive diagnostics, Machine Learning, Deep Learning, Combustion vehicles, Predictive maintenance, Hardware-in-the-Loop, Model Based Design.

Lista de Ilustrações

Lista de abreviaturas e siglas

IA	Inteligência Artificial
LED	Diodo Emissor de Luz
LCD	Display de Cristal Líquido
TFLite	TensorFlow Lite
HIL	Duty Cycle
MBD	Model Based Design
NLP	Natural Language Processing
ML	Machine Learning
LLM	Large Language Model
SVM	Support Vector Machines
LSTM	Long Short-Term Memory
XAI	Explainable artificial intelligence
IoT	Internet of Things
BLE	Bluetooth Low Energy
GPIO	General Purpose Input/Output
DAC	Conversores Digital-Analógic
ADC	Analog to Digital Converter
ReLU	Rectified Linear Unit
MSE	Mean Squared Error
RNN	Recurrent Neural Network
CNN	Convolutional neural network

Lista de tabelas

Sumário

1 Introdução

A manutenção de veículos evoluiu significativamente nas últimas décadas com o avanço das tecnologias automotivas e a crescente complexidade dos sistemas embarcados. No entanto, apesar dessas inovações, os métodos tradicionais de manutenção, como a manutenção corretiva e preventiva, ainda enfrentam desafios significativos, incluindo a identificação precoce de falhas e a otimização dos processos de reparo (JARDINE; LIN; BANJEVIC, 2006). O diagnóstico preditivo surge como uma solução promissora para superar essas limitações, oferecendo uma abordagem proativa e baseada em dados (LEE; LAPIRA, 2013).

O diagnóstico preditivo se baseia na análise de dados históricos e em tempo real para prever falhas futuras e permitir intervenções antes que problemas críticos ocorram. Essa abordagem é particularmente relevante no contexto automotivo, onde a detecção antecipada de falhas pode reduzir significativamente os custos de manutenção, melhorar a segurança e aumentar a vida útil dos componentes (TSUI et al., 2015). Com o advento de técnicas avançadas de Machine Learning(ML) e Deep Learning(DL), tornou-se possível analisar grandes volumes de dados de sensores e históricos de manutenção de forma mais eficiente e precisa, permitindo a criação de sistemas que aprendem continuamente com o desempenho do veículo (ZHANG; GANESAN; SHI, 2017).

Este trabalho propõe o desenvolvimento de um sistema de diagnóstico preditivo para veículos a combustão, empregando técnicas de ML e DL para prever falhas e problemas de desempenho, além de monitorar a saúde geral do motor. O modelo utiliza dados de sensores automotivos, como temperatura, pressão, rotação e outros parâmetros críticos, a fim de detectar padrões e sinais de possíveis falhas antes que elas se manifestem (MOHAMMADI; MOMENI; VAFERI, 2021).

visamos nesse trabalho contribuir para a evolução das práticas de manutenção automotiva, aproveitando tecnologias avançadas para oferecer uma abordagem mais precisa e eficaz. Ao empregar diagnósticos preditivos, espera-se que o sistema desenvolvido possa otimizar os processos de manutenção, reduzir tempos de inatividade e melhorar a confiabilidade dos veículos a combustão, proporcionando benefícios tanto para as montadoras quanto para os consumidores (BOUSDEKIS et al., 2015).

1.1 Objetivo Geral

Desenvolver um sistema inteligente que utilize técnicas de ML e DL para estimar a saúde do motor e prever possíveis falhas. O sistema será validado através de simulações

e testes utilizando um ambiente Hardware-in-the-Loop (HIL), com Model Based Design (MBD) para a modelagem e simulação das condições de operação.

1.2 Objetivos Específicos

Para a realização do objetivo geral, são definidos os seguintes objetivos específicos:

- Coletar ou buscar, dados relevantes de sensores e históricos de manutenção dos veículos e realizar o pré-processamento necessário para garantir a qualidade e a integridade dos dados;
- Desenvolver e treinar modelos de Machine Learning e Deep Learning para identificar padrões e prever falhas futuras com base nos dados;
- Avaliar a precisão e a eficácia dos modelos preditivos desenvolvidos, utilizando métricas de desempenho como acurácia, precisão, recall e F1-score;
- Testar o sistema em um ambiente real ou simulado para validar sua eficácia na previsão de falhas e avaliar sua capacidade de reduzir custos e melhorar a manutenção;
- Analisar os impactos e benefícios do sistema de diagnóstico preditivo, incluindo a redução de custos de manutenção, aumento da segurança e eficiência operacional.

1.3 Organização do documento

Este trabalho está organizado em cinco capítulos, além de anexos que complementam as informações apresentadas. O **Capítulo 1** introduz o contexto do problema, os objetivos gerais e específicos da pesquisa, além da motivação para o desenvolvimento de um sistema de diagnóstico preditivo para veículos a combustão utilizando técnicas de Machine Learning e Deep Learning. No **Capítulo 2**, é apresentada a fundamentação teórica, que aborda os principais conceitos relacionados ao trabalho, como manutenção preditiva, inteligência artificial, aprendizado de máquina, aprendizado profundo, sistemas embarcados e Hardware-in-the-Loop (HIL). O **Capítulo 3** descreve a metodologia adotada para o desenvolvimento do projeto, detalhando o processo de aquisição e pré-processamento dos dados, a criação e avaliação dos modelos de IA, e a sua implementação em microcontroladores de baixo custo. O **Capítulo 4** apresenta os resultados obtidos, incluindo a simulação do desgaste dos motores e a aplicação prática dos modelos desenvolvidos. Por fim, o **Capítulo 5** traz as conclusões da pesquisa, discutindo as contribuições do trabalho, as limitações encontradas e as sugestões para trabalhos futuros. Os anexos contêm os códigos utilizados para a implementação dos algoritmos, simulações e integração com o ESP32, além de outras informações complementares.

2 Fundamentação Teórica

Este capítulo tem como objetivo abordar os conceitos e técnicas utilizados no desenvolvimento deste trabalho, fornecendo uma base teórica sólida para a aplicação da IA em sistemas eletrônicos. Esses conceitos serão apresentados de forma clara e organizada, visando facilitar a compreensão dos aspectos técnicos abordados ao longo do estudo, assegurando uma visão completa e integrada do tema.

2.1 Manutenção Preditiva em Veículos de Combustão

A manutenção preditiva tem ganhado relevância em diversas indústrias, especialmente na automotiva, por sua capacidade de antecipar falhas com base em dados operacionais e históricos dos sistemas. Essa abordagem contrasta com as técnicas tradicionais de manutenção, como a manutenção corretiva e a manutenção preventiva, que possuem limitações inerentes em termos de eficiência e custo. A seguir, exploraremos as diferenças entre essas abordagens e como a manutenção preditiva, auxiliada por técnicas de IA, tem o potencial de superar os desafios enfrentados pelas práticas convencionais.

2.1.1 Manutenção Corretiva

A manutenção corretiva é caracterizada pela reparação de componentes ou sistemas após a ocorrência de uma falha. Embora seja a forma mais simples e direta de manutenção, ela acarreta custos elevados e períodos de inatividade inesperada, o que pode comprometer a operação de veículos. Segundo (SMITH, 2020), essa abordagem, por ser reativa, não permite qualquer tipo de planejamento antecipado e, por isso, pode resultar em falhas catastróficas que afetam não só os componentes diretamente envolvidos, mas todo o sistema do automóvel.

2.1.2 Manutenção Preventiva

A manutenção preventiva é baseada em intervalos fixos de tempo ou uso (quilometragem, horas de operação, etc.), independentemente do real estado do componente. O objetivo é realizar a troca ou reparo de peças antes que uma falha ocorra, minimizando os riscos associados à manutenção corretiva. No entanto, essa técnica apresenta limitações, pois nem sempre leva em consideração as condições reais de operação. A variabilidade das condições de condução, como o tipo de estrada, a carga transportada e o estilo de direção, faz com que alguns componentes se desgastem mais rapidamente do que o pre-

visto, enquanto outros podem ser substituídos de forma desnecessária (JARDINE; LIN; BANJEVIC, 2006).

2.1.3 O Conceito de Manutenção Preditiva

A manutenção preditiva surge como uma evolução das abordagens tradicionais, permitindo que a manutenção seja realizada apenas quando realmente necessário, com base no monitoramento contínuo de variáveis críticas. Por meio da coleta de dados em tempo real (como temperatura, vibração, pressão e velocidade de rotação), combinada com algoritmos avançados de IA e ML, é possível prever com precisão quando um componente está prestes a falhar (KIM; AN; CHOI, 2017).

Diferentemente da manutenção preventiva, a preditiva não depende de intervalos fixos. Em vez disso, utiliza uma abordagem baseada em condição, onde sensores capturam dados operacionais e algoritmos analisam esses dados para identificar padrões de desgaste ou anomalias. Isso permite uma estimativa mais precisa da vida útil de componentes críticos, como motores de combustão, transmissões e sistemas de freio, adaptando a manutenção ao ciclo de vida real de cada peça (DALPIAZ; RIVOLA; RUBINI, 2000).

2.1.4 Benefícios e Limitações da Manutenção Preditiva

Os benefícios da manutenção preditiva incluem:

- Redução de Custos: a manutenção é realizada apenas quando necessária, evitando trocas prematuras de peças e economizando recursos.
- Aumento da Confiabilidade: a previsão de falhas reduz o risco de paradas inesperadas, melhorando a confiabilidade do sistema.
- Otimização do Tempo de Manutenção: permite que a manutenção seja programada em momentos mais convenientes, minimizando o impacto na operação dos veículos.

No entanto, apesar de suas vantagens, a implementação da manutenção preditiva enfrenta alguns desafios. Entre eles, a necessidade de infraestrutura para a coleta e processamento de dados em tempo real, e a complexidade dos modelos de IA, que podem exigir grandes volumes de dados históricos para treinamento. Além disso, a precisão das previsões depende da qualidade dos dados obtidos pelos sensores, o que pode ser afetado por interferências externas e falhas no próprio hardware de monitoramento (DEMPSEY; AFJEH, 2002).

Tabela 1 – Comparativo entre as abordagens de manutenção

Característica	**Manutenção Corretiva**	**Manutenção Preventiva**	**Manutenção Preditiva**
Base de Ação	Após a falha	Intervalos fixos	Dados em tempo real
Custo Operacional	Alto (imprevisível)	Moderado (custo fixo)	Baixo (otimizado por necessidade)
Confiabilidade	Baixa	Moderada	Alta
Necessidade de Planejamento	Não	Sim	Sim, baseado em previsões
Tecnologia Necessária	Baixa	Moderada	Alta (sensores, IA, ML)

2.2 Inteligencia artificial

A Inteligência Artificial é uma área da ciência da computação que visa criar sistemas ou máquinas capazes de realizar tarefas que normalmente exigiriam inteligência humana, como tomada de decisão, reconhecimento de padrões e resolução de problemas. Segundo (RUSSELL; NORVIG, 2020), a IA pode ser dividida em duas categorias principais: IA fraca, que é projetada para realizar tarefas específicas (como assistentes virtuais e sistemas de recomendação), e IA forte, que tenta replicar a inteligência humana em sentido amplo, ainda um desafio significativo na área.

No contexto da manutenção preditiva de veículos, a IA é aplicada para prever falhas em componentes a partir de grandes volumes de dados coletados por sensores instalados nos veículos. Esses dados podem incluir informações sobre vibrações, pressão, temperatura, velocidade do motor, entre outros. A capacidade da IA de identificar padrões e realizar previsões com alta precisão torna-a uma ferramenta essencial para aumentar a confiabilidade e reduzir os custos operacionais.

A aplicação de IA tem sido o diferencial da manutenção preditiva, possibilitando a análise de grandes volumes de dados complexos que seriam impossíveis de processar manualmente. As técnicas de Machine Learning e Deep Learning podem identificar padrões ocultos nos dados operacionais, permitindo prever falhas antes que se tornem críticas. Por exemplo, algoritmos como Redes Neurais Artificiais (RNA) e Random Forest podem ser treinados com dados históricos de falhas e condições normais de operação para prever com precisão o momento em que uma falha ocorrerá.

Essas técnicas vão além da simples detecção de anomalias; elas aprendem continuamente com novos dados, ajustando seus modelos para melhorar as previsões ao longo do tempo (BREIMAN, 2001a) . Isso proporciona uma visão em tempo real do estado de saúde dos componentes, facilitando a tomada de decisões proativas e otimizando os custos

de manutenção. Segundo (JARDINE; LIN; BANJEVIC, 2006) , a manutenção preditiva baseada em IA tem o potencial de reduzir o tempo de inatividade dos veículos em até 50%, além de diminuir os custos operacionais relacionados à manutenção não planejada.

2.2.1 Machine Learning

Machine Learning envolve o uso de algoritmos para permitir que os computadores aprendam a partir de dados e realizem previsões ou decisões sem programação explícita. Dentro da manutenção preditiva, ML é utilizado para prever falhas em componentes críticos de veículos a partir de dados de sensores e variáveis operacionais, como pressão, temperatura, e vibração.

Existem vários algoritmos de aprendizado supervisionado que são amplamente utilizados para tarefas de classificação e regressão, essenciais para prever o comportamento dos sistemas mecânicos.

2.2.1.1 Regressão Logística

A Regressão Logística é um dos algoritmos mais simples e amplamente utilizados para tarefas de classificação binária, como prever se um componente do motor falhará (1) ou não falhará (0) em um determinado período de tempo. A função de regressão logística estima a probabilidade de um evento binário com base em um conjunto de variáveis preditoras. Por ser um modelo fácil de interpretar, ele é uma boa escolha inicial para a previsão de falhas (HOSMER; LEMESHOW; STURDIVANT, 2013).

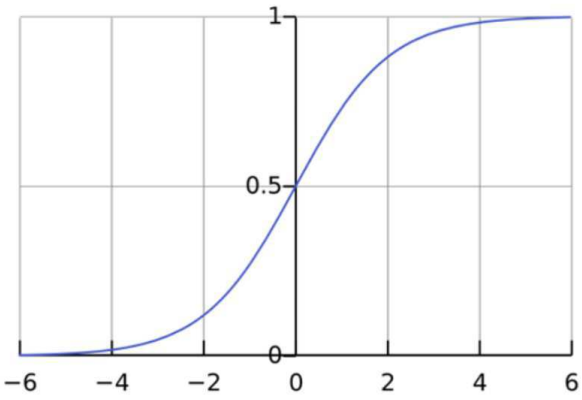
A regressão logística é um modelo estatístico que utiliza a função logística, também conhecida como função logit, para estabelecer a relação entre as variáveis x e y. A função logit transforma y em uma curva sigmoide, que varia suavemente em função de x, permitindo modelar a probabilidade de diferentes resultados.

A função sigmoide, que é frequentemente utilizada em regressão logística, é dada por:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

O gráfico desta função pode ser visto na Figura 1

Figura 1 – Curva sigmoide



Fonte: (Amazon Web Services, 2023)

No contexto de manutenção preditiva, a regressão logística pode ser aplicada para prever se um motor ou componente está em risco de falha iminente, com base em variáveis como temperatura, vibração e desgaste ao longo do tempo.

2.2.1.2 Árvore de Decisão

As Árvores de Decisão são algoritmos de aprendizado supervisionado que dividem iterativamente os dados em subconjuntos com base nas variáveis mais relevantes para prever o valor da variável de destino (Figura: 2). Elas são amplamente utilizadas em manutenção preditiva devido à sua capacidade de lidar com dados complexos e interpretar facilmente os resultados.

Cada nó de uma árvore de decisão representa uma variável preditora, e cada folha representa um resultado. No caso da manutenção preditiva, a árvore de decisão pode classificar se um motor precisa ou não de manutenção com base em medições contínuas dos sensores (QUINLAN, 1986). Além disso, árvores de decisão lidam bem com variáveis categóricas e numéricas, tornando-as úteis para sistemas com diferentes tipos de sensores.

Figura 2 – Árvore de Decisões

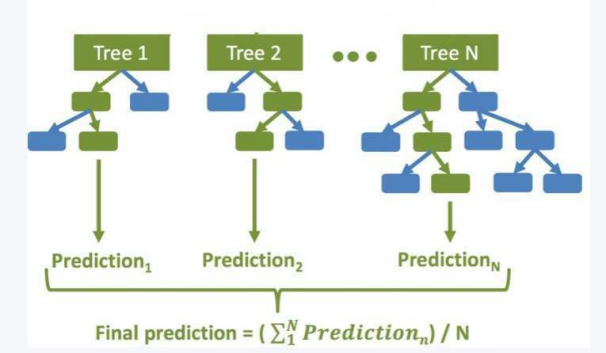
Fonte: (Colaborae Blog, 2023)

2.2.1.3 Random Forest

O Random Forest é uma extensão das árvores de decisão e consiste em um conjunto de várias árvores treinadas com diferentes subconjuntos de dados (Figura 3). Cada árvore em uma floresta decide sobre o resultado final, e a previsão geral é feita com base na votação da maioria das árvores (BREIMAN, 2001b). Essa abordagem é conhecida por sua robustez e alta precisão, sendo menos suscetível ao sobreajuste em comparação com uma única árvore de decisão.

No caso da manutenção preditiva, o Random Forest é útil para prever falhas de componentes de veículos a partir de grandes volumes de dados sensoriais, permitindo que o modelo combine sinais de várias fontes para tomar decisões precisas sobre o estado de saúde do motor ou de seus componentes. Por exemplo, pode prever a falha de um sistema de lubrificação ao considerar a pressão, a temperatura do óleo e o tempo de operação.

Figura 3 – Random Forest



Fonte: (PRAMOD, 2023)

2.2.1.4 Support Vector Machine (SVM)

O SVM é um algoritmo poderoso para tarefas de classificação, especialmente em casos em que os dados não são linearmente separáveis. O SVM tenta encontrar o hiperplano que melhor separa as diferentes classes de dados. Ele é amplamente utilizado para manutenção preditiva quando há necessidade de distinguir entre diferentes estados de operação do sistema, como funcionamento normal e falha iminente (CORTES; VAPNIK, 1995).

O SVM pode ser aplicado na previsão de falhas de motores de combustão interna, onde os dados de sensores complexos, como a análise de vibrações ou sinais acústicos, podem ser difíceis de classificar com algoritmos tradicionais. O SVM lida bem com esses problemas de classificação não linear, tornando-o uma escolha popular em muitas aplicações industriais.

2.2.2 Deep Learning

Deep Learning é uma subárea de ML que utiliza redes neurais profundas para modelar e resolver problemas complexos, especialmente aqueles com grandes quantidades de dados. O DL revolucionou diversos campos, incluindo visão computacional, processamento de linguagem natural, e manutenção preditiva, devido à sua capacidade de lidar com dados não estruturados e detectar padrões complexos que seriam difíceis para algoritmos tradicionais.

2.2.2.1 Estrutura de Redes Neurais Artificiais

Uma RNA é composta por unidades chamadas de neurônios, organizadas em camadas. Uma rede neural típica possui uma camada de entrada, uma ou mais camadas ocultas, e uma camada de saída.

- **Camada de Entrada**: recebe os dados de entrada, que podem ser números (dados estruturados) ou imagens, textos, sinais de sensores, entre outros (dados não estruturados).
- **Camadas Ocultas**: realizam a maior parte do processamento. Cada neurônio de uma camada oculta aplica uma função matemática aos dados que recebe da camada anterior, com a função mais comum sendo a função de ativação ReLU (Rectified Linear Unit), que resolve problemas de não linearidade.
- **Camada de Saída**: retorna a previsão final da rede. Em problemas de classificação, a saída pode ser uma probabilidade para cada classe.

Cada conexão entre os neurônios é associada a um peso, e os neurônios têm um valor de bias. O objetivo do treinamento de redes neurais é ajustar esses pesos e *biases* para minimizar o erro entre a saída prevista e o valor real.

2.2.2.2 Funcionamento do Algoritmo de Treinamento

O processo de aprendizado em uma rede neural é realizado através de algoritmos de retropropagação (*backpropagation*), que funcionam em conjunto com o algoritmo de gradiente descendente.

1. **Propagação Direta (*Forward Propagation*)**: os dados de entrada são alimentados na camada de entrada da rede e passam por todas as camadas ocultas até atingir a camada de saída. Cada neurônio realiza cálculos lineares baseados nos pesos das conexões e depois aplica uma função de ativação.

2. **Cálculo da Função de Custo (*Loss Function*)**: a função de custo mede a diferença entre a previsão da rede e o valor real. Um exemplo comum de função de custo para tarefas de classificação é a **cross-entropy**, enquanto para regressão pode-se usar o **erro quadrático médio (MSE)**.

3. **Retropropagação (*Backpropagation*)**: durante a retropropagação, o erro calculado pela função de custo é propagado de volta pela rede, ajustando os pesos e biases para reduzir esse erro. O **algoritmo de gradiente descendente** atualiza os pesos de cada neurônio, movendo-os na direção que minimiza a função de custo.

4. **Atualização dos Pesos (*Gradient Descent*)**: o gradiente descendente ajusta os pesos usando uma taxa de aprendizado (learning rate), que determina o tamanho do passo que deve ser dado para atualizar os pesos a cada iteração. Em redes neurais profundas, uma variante chamada **Adam Optimizer** é amplamente utilizada, pois ajusta a taxa de aprendizado de forma adaptativa ao longo do treinamento.

Esse processo se repete por várias iterações (chamadas de épocas) até que o modelo atinja um ponto de convergência, minimizando o erro e maximizando a precisão da previsão.

2.2.2.3 Tipos de Redes Neurais Profundas

Dentro do Deep Learning, há vários tipos de redes neurais, cada uma adequada para diferentes tipos de dados e problemas.

As **Redes Neurais Convolucionais (CNNs)** são amplamente utilizadas para processar dados que têm uma estrutura espacial, como imagens. Ao invés de conectar todos os neurônios entre si, as CNNs aplicam filtros convolucionais que deslizam sobre a entrada e detectam características locais, como bordas ou texturas.

Cada camada convolucional gera um mapa de características que é então processado por camadas seguintes. Isso permite que a rede extraia gradualmente características de baixo nível (como contornos) até chegar a características de alto nível (como formas e objetos). CNNs são usadas em tarefas como a detecção de falhas visuais em peças de máquinas, reconhecimento de padrões em dados de sensores, e inspeção de componentes mecânicos em manutenção preditiva.

As **Redes Neurais Recorrentes (RNNs)** são adequadas para dados sequenciais, como séries temporais ou dados dependentes do tempo, como as leituras contínuas de sensores de um motor. As RNNs possuem conexões internas que permitem que a rede retenha informações sobre entradas anteriores, essencial para tarefas como previsão de falhas em manutenção preditiva, onde a relação temporal entre os eventos é crucial.

Um tipo avançado de RNN, chamado de Long Short-Term Memory (LSTM)**,** é especialmente eficaz para aprender dependências de longo prazo em dados sequenciais.

As LSTMs têm sido aplicadas na previsão da degradação de componentes em sistemas automotivos, analisando como os padrões de vibração ou temperatura mudam ao longo do tempo.

2.2.2.4 Aplicação de Deep Learning na Manutenção Preditiva

A **manutenção preditiva** é uma área em que o DL tem se mostrado promissor, especialmente para sistemas complexos como motores de combustão interna, onde há uma enorme quantidade de dados sensoriais para análise. A capacidade do DL de lidar com dados em larga escala e identificar padrões ocultos é uma vantagem significativa(SMITH; WANG, 2020) .

- **Análise de Sinais de Vibração**: usando CNNs, padrões complexos de vibração em motores podem ser detectados e classificados, ajudando a identificar componentes desgastados antes que ocorra uma falha catastrófica.
- **Previsão da Vida Útil de Componentes**: LSTMs podem ser treinadas para analisar dados sequenciais, como temperatura, pressão e desgaste, para prever a vida útil restante de um componente crítico do motor, permitindo que a manutenção seja planejada de maneira eficiente.
- **Detecção de Anomalias**: usando Autoencoders, redes neurais não supervisionadas podem ser treinadas para detectar anomalias nos dados de sensores. Quando a rede é incapaz de reconstruir um dado de entrada com precisão, isso pode indicar uma falha ou anomalia no sistema, disparando um alerta para manutenção.

2.2.2.5 Desafios e Limitações

Apesar de seu sucesso, o DL enfrenta alguns desafios na manutenção preditiva. Modelos de DL podem ser computacionalmente caros e demandar grandes quantidades de dados rotulados para o treinamento. Além disso, a natureza de "caixa-preta"dos modelos de DL pode dificultar a explicação dos resultados, o que é uma preocupação em sistemas críticos como motores automotivos. No entanto, técnicas como explainable AI (XAI) estão sendo desenvolvidas para melhorar a interpretabilidade desses modelos.

2.3 *Sistemas embarcados*

Sistemas embarcados são dispositivos eletrônicos projetados para realizar funções específicas, integrando hardware e software. Diferentemente dos computadores de propósito geral, que podem executar uma ampla variedade de tarefas, os sistemas embarcados são otimizados para realizar uma função particular de maneira eficiente e confiável. Eles

estão presentes em diversos setores, como automotivo, eletrodomésticos, telecomunicações e controle industrial (BURNS et al., 2001).

Os sistemas embarcados são compostos por:

- **Microcontrolador ou microprocessador**: o núcleo computacional responsável por processar instruções.
- **Periféricos**: componentes de entrada/saída, sensores, atuadores e interfaces de comunicação (HEATH, 2002).
- **Memória**: armazenamento de programas e dados.
- **Software embarcado**: código que controla o hardware e implementa as funcionalidades específicas do sistema (BURNS et al., 2001).

A principal vantagem dos sistemas embarcados é sua capacidade de operar com restrições de energia, processamento e tamanho, o que os torna ideais para aplicações em tempo real e controle direto de dispositivos (WOLF, 2001).

2.3.1 O ESP32 como Plataforma de Sistemas Embarcados

O ESP32 é uma plataforma de microcontrolador muito popular no desenvolvimento de sistemas embarcados, especialmente por suas capacidades de conectividade e processamento. Desenvolvido pela Espressif Systems, o ESP32 se destaca por ser um microcontrolador de baixo custo e de alto desempenho, utilizado em uma ampla gama de projetos de IoT (Internet das Coisas), automação, controle de sistemas e monitoramento remoto (ESPRESSIF, 2016).

Características principais do ESP32:

- **CPU Dual-core**: o ESP32 possui dois núcleos de processamento de 32 bits, baseados na arquitetura Xtensa LX6, operando a uma frequência de até 240 MHz. Isso o torna capaz de lidar com várias tarefas simultaneamente, o que é útil para aplicações que requerem multitarefa (ESPRESSIF, 2016).
- **Conectividade Wi-Fi e Bluetooth**: uma das grandes vantagens do ESP32 é sua capacidade de conexão sem fio. Ele suporta Wi-Fi 802.11 b/g/n e Bluetooth 4.2/Bluetooth Low Energy (BLE), permitindo sua utilização em redes sem fio e comunicações diretas entre dispositivos (ESPRESSIF, 2016).
- **Baixo consumo de energia**: o ESP32 é projetado para operar em modos de baixo consumo, o que é essencial para aplicações em que a eficiência energética é crucial, como dispositivos móveis e sensores remotos (ESPRESSIF, 2016).

- **Entradas e saídas analógicas e digitais**: o microcontrolador possui uma ampla gama de GPIOs (General Purpose Input/Output) que podem ser configuradas como entradas ou saídas. Ele também suporta ADC (Conversores Analógico-Digital) e DAC (Conversores Digital-Analógico), permitindo a leitura de sinais analógicos e a geração de sinais de controle (ESPRESSIF, 2016).
- **Interfaces de comunicação**: além de Wi-Fi e Bluetooth, o ESP32 suporta UART, I2C, SPI, I2S, entre outros protocolos de comunicação, facilitando a integração com sensores, atuadores e outros dispositivos embarcados (ESPRESSIF, 2016).
- **Capacidades de processamento multimídia**: o ESP32 pode ser utilizado em projetos mais complexos, como processamento de áudio e vídeo em tempo real, graças ao suporte para o protocolo I2S e recursos de hardware acelerado para criptografia e segurança (ESPRESSIF, 2016).

2.3.2 Aplicações do ESP32

O ESP32 tem sido amplamente utilizado em aplicações de IoT e controle de sistemas devido à sua versatilidade e conjunto de recursos integrados. Algumas das áreas em que o ESP32 se destaca incluem:

- **Automação residencial**: controle de iluminação, segurança e eletrodomésticos via Wi-Fi (JANKOWSKI et al., 2016).
- **Monitoramento remoto**: sensores de temperatura, umidade e presença que se conectam à internet para fornecer dados em tempo real (MISHRA et al., 2018).
- **Automotivo**: monitoramento e controle de sistemas embarcados em veículos, como diagnósticos em tempo real de componentes (ROSARIO et al., 2020).
- **Sistemas de controle industrial**: coleta e processamento de dados de sensores em linhas de produção, transmitindo informações em tempo real para sistemas de gerenciamento (LIN et al., 2017).

2.3.3 ESP32 no Contexto de Manutenção Preditiva

No contexto de manutenção preditiva, o ESP32 pode ser utilizado para coletar dados de sensores (como vibração, temperatura e pressão) e transmitir essas informações para sistemas centrais de análise, onde técnicas de machine learning e deep learning podem ser aplicadas para prever falhas em equipamentos (ZHOU et al., 2019). A conectividade sem fio e os modos de baixo consumo de energia tornam o ESP32 ideal para monitoramento remoto e em tempo real de componentes críticos em veículos e sistemas industriais (ROSARIO et al., 2020).

O uso do ESP32 em manutenção preditiva envolve sua integração com sensores de diagnóstico, transmissão de dados para a nuvem ou servidores locais e processamento local de dados, o que permite a detecção antecipada de falhas e a otimização do ciclo de vida dos componentes monitorados (ZHOU et al., 2019).

O ESP32 oferece uma plataforma poderosa e flexível para o desenvolvimento de sistemas embarcados, com aplicações que vão desde a automação e controle de dispositivos até o monitoramento remoto e manutenção preditiva. Seu equilíbrio entre conectividade, desempenho e baixo consumo de energia o torna uma escolha popular entre desenvolvedores e engenheiros para projetos de IoT e sistemas embarcados (ESPRESSIF, 2016).

2.4 Hardware In the Loop

A técnica Hardware-in-the-Loop (HIL) é amplamente utilizada na validação e teste de sistemas de controle e automação, especialmente em aplicações onde o desenvolvimento de software precisa ser testado em um ambiente o mais próximo possível da realidade, mas sem os riscos ou custos associados ao uso direto do sistema físico completo. O HIL permite a integração do software de controle com modelos matemáticos de sistemas reais em um ambiente de simulação, enquanto partes específicas do hardware estão diretamente conectadas ao sistema em teste (LU et al., 2005).

Em um teste HIL, o sistema sob controle é modelado em um ambiente de simulação, e os sinais de controle são enviados a partir do controlador físico (como uma unidade de controle eletrônica de um motor). Isso garante que o controlador funcione conforme o esperado ao interagir com o hardware real, sem a necessidade de uma planta física em cada estágio de desenvolvimento (NORTON, 2017).

2.4.1 MATLAB para Simulação HIL

O MATLAB é uma das plataformas mais utilizadas para o desenvolvimento de testes HIL devido à sua flexibilidade, integração com sistemas físicos e capacidade de modelar sistemas complexos em tempo real. A interface gráfica do Simulink permite a modelagem de sistemas dinâmicos usando blocos que representam equações diferenciais, entradas e saídas de controle, o que facilita a simulação e o teste de controle em sistemas embarcados (MATHWORKS, 2021).

2.4.2 Componentes de um Sistema HIL

Um sistema HIL típico consiste em três componentes principais:

- **Modelo da planta**: Um modelo matemático da planta (o sistema físico a ser controlado), que simula o comportamento dinâmico real do sistema em tempo real.

- **Controlador físico**: O hardware de controle, como uma ECU (Electronic Control Unit) em veículos, que interage com o sistema simulado da planta.
- **Interfaces de comunicação**: Conexões entre o controlador físico e o ambiente de simulação. Isso inclui conversores A/D e D/A para leitura e escrita de sinais analógicos e digitais (SCHNEIDER et al., 2016).

2.4.3 Vantagens do Uso de HIL

O uso de HIL oferece várias vantagens, especialmente em projetos complexos, como no desenvolvimento de sistemas de controle automotivo e aeroespacial:

- **Testes seguros**: componentes perigosos ou de alto custo, como motores, podem ser simulados, reduzindo o risco de danos ou falhas durante o desenvolvimento.
- **Redução de custos**: a necessidade de protótipos físicos é minimizada, economizando recursos em termos de tempo e dinheiro (LU et al., 2005).
- **Testes repetíveis**: o HIL permite a realização de testes sob condições controladas e repetíveis, facilitando a identificação de falhas no software ou hardware.
- **Desenvolvimento acelerado**: permite o desenvolvimento e a validação de controladores antes mesmo que o sistema físico esteja disponível (SCHNEIDER et al., 2016).

2.4.4 Aplicações de HIL em Sistemas Automotivos

No setor automotivo, o HIL é amplamente utilizado para o desenvolvimento e teste de sistemas de controle de motor, transmissão e freios, além de sistemas de assistência ao motorista, como controle de estabilidade (ESC) e frenagem automática. Utilizando MATLAB/Simulink, engenheiros podem modelar o comportamento do veículo em tempo real e verificar como a ECU interage com a planta modelada (NORTON, 2017).

Exemplo: Testes HIL em Veículos de Combustão Interna Nos veículos de combustão interna, o HIL pode ser usado para testar sistemas de injeção de combustível, controle de válvulas e sistemas de ignição. A planta que simula o motor recebe os sinais do controlador físico (ECU) e gera respostas realistas de saída, como temperatura, pressão e velocidade do motor, que são lidas pelo controlador, permitindo que os engenheiros validem as estratégias de controle em tempo real (LU et al., 2005).

3 Metodologia

As etapas de desenvolvimento deste projeto foram divididas em três fases principais: Aquisição de Dados, Pré-processamento de Dados e Geração de Modelos de IA, e Implantação em Microcontrolador. Neste capítulo, serão detalhados os métodos empregados em cada uma dessas fases, com uma descrição minuciosa das ferramentas, técnicas e modelos utilizados. O objetivo é oferecer uma visão clara e organizada de todo o processo.

3.1 **Ambiente de Desenvolvimento**

O Google Colab foi escolhido por sua conveniência e pela possibilidade de acessar recursos computacionais avançados, como GPUs, sem a necessidade de configurar ambientes locais complexos. Essa plataforma baseada em nuvem permite a execução de notebooks Jupyter, facilitando a escrita de código Python e a visualização de resultados de forma interativa.

3.2 Aquisição de Dados

A aquisição de dados representa um dos desafios mais complexos neste projeto, especialmente quando se trata de dados reais de manutenção de veículos. A coleta de dados de forma experimental, ou seja, diretamente dos veículos em funcionamento, implica em diversas dificuldades práticas. Primeiramente, essa abordagem requer acesso a uma frota de veículos, equipamentos especializados para monitoramento em tempo real, e um longo período de observação para se coletar informações sobre falhas e padrões de desgaste. Além disso, o custo envolvido na instalação de sensores e a necessidade de monitoramento contínuo tornam o processo oneroso e demorado.

Dada a complexidade e o custo elevado da aquisição de dados experimentais, optou-se por realizar uma pesquisa em bases de dados disponíveis publicamente, focando em conjuntos de dados de manutenção preditiva e monitoramento de veículos. Durante a pesquisa, uma fonte útil de dados foi encontrada no site Kaggle, uma plataforma que disponibiliza uma grande variedade de datasets relacionados a diferentes áreas, incluindo a manutenção de veículos. O dataset selecionado forneceu dados históricos de manutenção com informações relevantes, como rotações por minuto (RPM), temperatura do óleo, temperatura do líquido de arrefecimento, pressão do óleo, pressão do líquido de arrefecimento e pressão do combustível. Esses dados foram utilizados como variáveis independentes, enquanto a condição do motor foi considerada a variável dependente. Essas informações

permitiram avançar no desenvolvimento do modelo de previsão de falhas (KAGGLE, 2024).

A partir dos dados obtidos no Kaggle, foi possível realizar a limpeza, pré-processamento e engenharia de características, etapas fundamentais para a construção dos modelos de machine learning aplicados no projeto.

Os dados obtidos no site de origem não continham informações detalhadas sobre como foram coletados, tampouco apresentavam especificações detalhadas dos veículos utilizados. Esta falta de clareza sobre a origem e a natureza dos dados apresentou um desafio significativo durante o processo de tratamento e inferência. Sem essas informações, foi necessário realizar diversas suposições sobre os tipos de veículos e condições de operação representadas, o que pode introduzir vieses no modelo. Além disso, a ausência de metadados detalhados complicou a verificação da consistência e qualidade dos dados, exigindo maior esforço no pré-processamento, como a identificação de *outliers*.

3.3 Pre-Processamento de Dados e Geração De Modelos de IA

3.3.1 Pré-Processamento de Dados

O pré-processamento de dados é uma etapa crítica para garantir a qualidade dos modelos de inteligência artificial (IA). Os dados brutos frequentemente apresentam valores faltantes, inconsistências ou *outliers* que podem comprometer o desempenho dos modelos preditivos. Neste projeto, após a coleta dos dados de manutenção de veículos, foram aplicadas várias técnicas de pré-processamento para garantir que os dados estivessem adequadamente estruturados para a geração dos modelos de IA.

Para o pré-processamento, utilizamos o ambiente Google Colab, que permite o desenvolvimento de notebooks com suporte a GPU e bibliotecas populares para ciência de dados. As seguintes bibliotecas foram utilizadas:

- **NumPy**: para operações numéricas e manipulação de arrays;
- **Pandas**: para manipulação e análise de dados estruturados, principalmente em DataFrames;
- **Random**: para amostragem de dados e criação de subconjuntos aleatórios;
- **Scikit-learn**: para realizar tarefas de pré-processamento, como normalização e divisão de conjuntos de dados;
- **Seaborn** e **Matplotlib**: para visualização gráfica dos dados, o que facilitou a análise exploratória e a identificação de padrões relevantes.

O processo de pré-processamento incluiu as seguintes etapas:

- **Tratamento de valores ausentes**: remoção ou imputação de valores faltantes em colunas críticas.
- **Normalização e padronização**: utilizamos a função `StandardScaler` do Scikit-learn para normalizar os dados, garantindo que todas as variáveis estivessem em uma mesma escala.
- **Engenharia de características**: novas variáveis foram criadas a partir dos dados originais, como a eficiência do óleo e a eficiência do líquido de arrefecimento, para melhorar o desempenho dos modelos de machine learning.

3.3.2 Geração de Modelos de IA

Com os dados pré-processados, foi possível avançar para a fase de modelagem. Diversos algoritmos de aprendizado de máquina e redes neurais profundas foram implementados e testados. O Google Colab foi utilizado também nessa etapa, permitindo o uso eficiente das GPUs para o treinamento dos modelos. As principais bibliotecas e frameworks empregados foram:

- **TensorFlow**: para a construção e treinamento de modelos de deep learning;
- **PyTorch**: Para o desenvolvimento de redes neurais profundas, com flexibilidade para pesquisa e experimentação;
- **XGBoost**: para a implementação de algoritmos de gradient boosting, amplamente utilizados em competições de machine learning devido à sua eficiência e desempenho;
- **LightGBM**: outra ferramenta baseada em gradient boosting, utilizada para lidar com grandes volumes de dados de forma eficiente.

Foram utilizados os seguintes algoritmos de aprendizado de máquina:

- **Regressão Logística**: para tarefas de classificação binária, como a previsão de falhas ou não falhas de componentes do veículo.
- **Árvore de Decisão**: um modelo interpretável que foi testado para identificar as variáveis mais importantes na previsão de falhas.
- **Random Forest**: um algoritmo de ensemble que combina várias árvores de decisão para melhorar a precisão.

- **Support Vector Machines (SVM)**: utilizado em conjunto com o kernel radial para separação não linear dos dados.
- **Deep Learning**: modelos de redes neurais profundas foram treinados com TensorFlow e PyTorch para capturar padrões mais complexos nos dados, utilizando múltiplas camadas densamente conectadas e técnicas como dropout para evitar overfitting.

3.3.3 Avaliação dos Modelos

Para avaliar o desempenho dos modelos, foram utilizadas algumas métricas de desempenho que permitem uma análise abrangente da eficácia de cada algoritmo.

- **Acurácia**: esta métrica indica a proporção de previsões corretas em relação ao total de previsões realizadas. A acurácia é uma boa indicação do desempenho geral do modelo, especialmente em conjuntos de dados balanceados;

$$\text{Acurácia} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

- **Precisão**: a precisão mede a proporção de verdadeiros positivos em relação ao total de previsões positivas. Essa métrica é particularmente útil quando o custo de falsos positivos é alto, pois indica a confiabilidade das previsões positivas do modelo;

$$\text{Precisão} = \frac{TP}{TP + FP} \tag{3.2}$$

- **Recall (Sensibilidade)**: o recall avalia a proporção de verdadeiros positivos em relação ao total de casos reais positivos. Essa métrica é crucial quando o foco é minimizar falsos negativos, como em aplicações de diagnóstico, onde é importante identificar corretamente todas as instâncias positivas;

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.3}$$

- **F1-Score**: o F1-Score fornece uma média harmônica entre precisão e recall, oferecendo uma única métrica que considera tanto falsos positivos quanto falsos negativos. Essa métrica é valiosa em cenários onde é necessário equilibrar precisão e recall .

$$\text{F1-Score} = 2 \times \frac{\text{Precisão} \times \text{Recall}}{\text{Precisão} + \text{Recall}} \tag{3.4}$$

- **Matriz de Confusão**: a matriz de confusão é uma ferramenta fundamental para avaliar o desempenho de modelos de classificação, fornecendo uma visão clara das previsões do modelo em relação aos rótulos verdadeiros. Permitindo saber a quantidade de falsos positivos e falso negativos (Tabela 2).

Tabela 2 – Matriz de Confusão

	Predito Positivo	**Predito Negativo**
Real Positivo	TP	FN
Real Negativo	FP	TN

3.3.4 Interpretação dos Termos

- **Verdadeiros Positivos (TP):** casos em que o modelo previu corretamente a classe positiva.
- **Verdadeiros Negativos (TN):** casos em que o modelo previu corretamente a classe negativa.
- **Falsos Positivos (FP):** casos em que o modelo previu a classe positiva, mas a classe real era negativa. Isso também é conhecido como "erro do tipo I".
- **Falsos Negativos (FN):** casos em que o modelo previu a classe negativa, mas a classe real era positiva. Isso é chamado de "erro do tipo II".

3.4 Implantação em Microcontrolador

A fase de implantação envolveu a integração dos modelos de IA em um microcontrolador ESP32, utilizando uma abordagem baseada em modelos de machine learning otimizados para dispositivos com recursos limitados. Abaixo, detalhamos as principais ferramentas e processos utilizados para a implantação.

3.4.1 Conversão do Modelo para TensorFlow Lite (TFLite)

Após o desenvolvimento e a validação dos modelos de IA em ambientes de desenvolvimento como Google Colab e Jupyter Notebooks, foi necessário converter o melhor modelo para um formato otimizado para execução em dispositivos embarcados. Utilizou-se o TensorFlow Lite (TFLite), que permite a execução de modelos de machine learning em microcontroladores e dispositivos com recursos limitados, como o ESP32.

A conversão do modelo foi realizada utilizando o TensorFlow Lite Converter. Este processo reduziu o tamanho do modelo e otimizou o desempenho em termos de velocidade e uso de memória. Foram aplicadas técnicas de quantização durante a conversão, reduzindo a precisão dos parâmetros para inteiros de 8 bits, o que é mais eficiente para o ESP32.

3.4.2 Desenvolvimento em C++ usando VSCode e PlatformIO

A etapa seguinte envolveu a implementação do código em C++ para integrar o modelo TFLite ao ESP32. O ambiente de desenvolvimento escolhido foi o Visual Studio

Code (VSCode), em conjunto com a extensão PlatformIO, que fornece suporte para microcontroladores e facilita o processo de compilação e upload do código para o dispositivo.

No código C++, o modelo TFLite foi carregado diretamente no ESP32, e a inferência foi realizada a partir dos dados de entrada fornecidos pelos sensores do sistema. Utilizamos a biblioteca *TFLite for Microcontrollers* para que o ESP32 fosse capaz de realizar previsões utilizando o modelo convertido, garantindo o funcionamento do sistema de IA em tempo real.

3.4.3 MATLAB e Comunicação Serial

O MATLAB foi utilizado para a geração de dados de entrada e comunicação com o ESP32. Os dados simulados de um veículo, como pressão do óleo, temperatura do motor e quilometragem, foram enviados ao ESP32 por meio de comunicação serial. O script MATLAB foi configurado para selecionar aleatoriamente amostras de dados, normalizá-las e enviá-las via Bluetooth para o ESP32, a cada intervalo de 5 segundos, utilizando o protocolo serial.

3.4.4 Interação com o Display LCD 16x2 com I2C

Para a visualização dos resultados das previsões feitas pelo modelo de IA no ESP32, foi utilizado um display LCD 16x2 com interface I2C. O display permitiu apresentar informações críticas, como o status do sistema de manutenção preditiva ou a probabilidade de falhas em componentes do veículo.

A interface I2C foi escolhida pela simplicidade de conexão (apenas dois pinos são necessários, SDA e SCL), e a biblioteca LiquidCrystal_I2C foi utilizada para gerenciar a comunicação entre o ESP32 e o display. O display mostrava informações como:

- **Previsão de falha**: se o componente estava em boas condições ou próximo de uma falha;
- **Dados do sensor**: leituras em tempo real dos sensores simulados de manutenção do veículo.

3.4.5 Validação e Testes

Após a integração do modelo de IA no ESP32, foram realizados testes para validar o sistema completo. Os dados foram enviados em intervalos regulares via Bluetooth, processados pelo modelo TFLite no ESP32, e os resultados foram exibidos no display LCD. O sistema foi monitorado para avaliar a precisão das previsões e o desempenho em tempo real, garantindo que o ESP32, mesmo com recursos limitados, fosse capaz de

executar o modelo eficientemente e fornecer feedback instantâneo sobre a manutenção dos componentes.

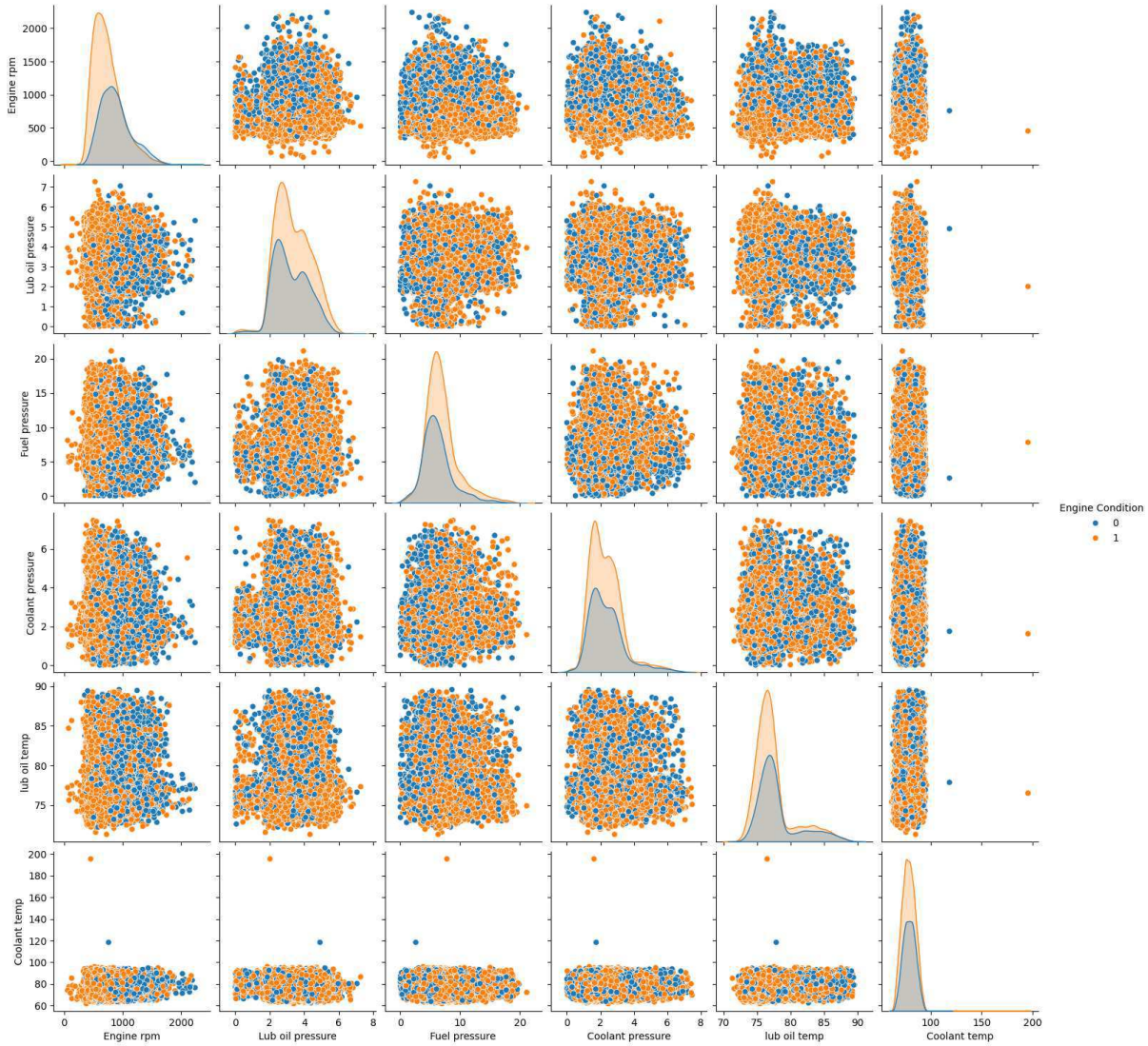
Essa implementação mostrou-se eficiente para rodar modelos de machine learning em hardware embarcado, facilitando o desenvolvimento de sistemas de manutenção preditiva em veículos, utilizando inteligência artificial em microcontroladores.

4 Resultados

4.1 Processamento dos Dados

Para dar início ao projeto, os dados foram importados para o ambiente do Google Colab, onde o pré-processamento foi realizado. Os dados consistem nas condições de 19.535 veículos e incluem informações como: Rotações por Minuto (RPM), temperatura do óleo, temperatura do líquido de arrefecimento, pressão do líquido de arrefecimento, pressão do óleo, pressão do combustível e condição do motor (0 para motor ruim e 1 para motor bom). Após a importação dos dados, foi plotado um gráfico para visualizar a distribuição dos atributos, conforme mostrado na Figura 4

Figura 4 – Visualizando a distribuição de todos os atributos



Fonte: Autoria Própria.

A parir da visualização dos dados foi possível inferir que a maioria dos atributos fornecidos pelo conjunto de dados são bem distribuídos em relação a todas as colunas, e que a temperatura do líquido de arrefecimento precisa ser dimensionada para tornar sua distribuição mais uniforme.

Em seguida, foi calculada a correlação (Tabela 3) de cada atributo em relação à condição dos motores. A análise de correlação é uma etapa crucial para entender como as variáveis independentes se relacionam com a variável dependente, que, neste caso, representa a condição dos motores.

Para realizar essa análise, foi utilizado o coeficiente de correlação de Pearson, que mede a força e a direção da relação linear entre duas variáveis. Os valores da correlação variam de -1 a 1, onde:

- **1** indica uma correlação positiva perfeita, significando que, à medida que uma variável aumenta, a outra também aumenta.
- **-1** indica uma correlação negativa perfeita, indicando que, à medida que uma variável aumenta, a outra diminui.
- **0** indica que não há correlação linear entre as variáveis.

Os passos seguintes foram realizados:

Essa abordagem sistemática para calcular e analisar a correlação entre os atributos fornece *insights* valiosos que embasam a modelagem preditiva, facilitando a identificação de padrões que podem ser utilizados para prever falhas e otimizar a manutenção de veículos.

Tabela 3 – Tabela de Correlação entre Atributos e Condição do Motor

Atributos	**Correlação com a Condição do Motor**
Fuel Pressure	0.116259
Lub Oil Pressure	0.060904
Coolant Pressure	-0.024054
Coolant Temp	-0.046326
Lub Oil Temp	-0.093635
Engine RPM	-0.268201

Em posse da Correlação, foi possível observar que, a condição do motor tem a maior correlação positiva com a pressão do combustível, e maior correlação negativa com a rotação do motor.

4.1.1 *Feature Engineering*

Com o intuito de tornar a temperatura do líquido de arrefecimento e do óleo mais expressiva em relação à variável dependente, foram criados novos atributos denominados "Eficiência do Óleo"e "Eficiência do Líquido de Arrefecimento"(Listing 4.1). Esses atributos foram relacionados à variável de RPM, permitindo uma análise mais profunda do desempenho do motor em diferentes condições operacionais.

```
# Coolant Efficiency
engine_df["Coolant Efficiency"] = (1 / engine_df["Engine rpm"]) *
    engine_df["Coolant temp"]

# Oil Efficiency
engine_df["Oil Efficiency"] = 1 / (engine_df["Engine rpm"] * engine_df["
    lub oil temp"])
```

Listing 4.1 – Cálculo da Eficiência do Líquido de Arrefecimento e do Óleo.

Em seguida foi calculada a correlação das novas variáveis como se ver na Tabela 4.

Tabela 4 – Correlação entre a Condição do Motor e Novas Variáveis

Variável	**Correlação**
Oil Efficiency	0.274301
Coolant Efficiency	0.252389

4.1.2 Remoção de *Outliers*

Dentre o conjunto de dados também foi percebido motores que apresentavam RPMs muito baixo do usual, que seria de aproximadamente 600 RPM (HEYWOOD, 1988), adotou-se a técnica do intervalo interquartil (IQR) para identificar e eliminar valores extremos que poderiam distorcer os resultados e a performance dos modelos de inteligência artificial, com essa filtragem foram removidos 126 veículos.

Primeiramente, foram calculados os quartis do conjunto de dados, sendo o primeiro quartil (Q1) o valor que separa os 25% menores dos dados, enquanto o terceiro quartil (Q3) separa os 25% maiores. O intervalo interquartil (IQR) é, então, determinado pela diferença entre Q3 e Q1 (IQR = Q3 - Q1). Essa métrica é utilizada para entender a dispersão e a variação dos dados, focando na parte central da distribuição.

```
# Calcular o primeiro quartil (Q1) e o terceiro quartil (Q3)
Q1 = engine_df.quantile(0.25)
Q3 = engine_df.quantile(0.75)
```

```
IQR = Q3 - Q1

# Definir os limites
lower_bound = Q1 - 1.5 * IQR

# Filtrar os dados para remover os outliers
engine_df_clean = engine_df[~((engine_df < lower_bound)).any(axis=1)]
```

Listing 4.2 – Código para remoção de outliers utilizando o método do intervalo interquartil (IQR).

4.1.3 Normalização

A normalização de dados é um passo crucial no pré-processamento de dados para modelos de aprendizado de máquina. O objetivo principal da normalização é ajustar a escala das variáveis de entrada, assegurando que cada uma delas contribua igualmente para o modelo, evitando que variáveis com magnitudes maiores dominem o processo de aprendizado.

A normalização padrão transforma os dados de modo que tenham uma média igual a 0 e um desvio padrão igual a 1. Essa transformação é essencial, especialmente em algoritmos que utilizam medidas de distância, como regressão logística, SVM e redes neurais, pois a escala das variáveis pode impactar a eficiência do algoritmo e a qualidade das previsões.

$$z = \frac{x - \mu}{\sigma}$$

onde:

- z é o valor transformado,
- x é o valor original,
- μ é a média da característica,
- σ é o desvio padrão da característica.

Essa transformação garante que, após o processo, a distribuição dos dados tenha uma média de 0 e um desvio padrão de 1, permitindo que todas as características sejam tratadas de forma equivalente durante o treinamento do modelo. Os valores da media e desvio padrão de cada requisito foi salvo, para normalização de novas amostras futuras.

4.2 Criação de Modelos de IAs

Neste trabalho, foram selecionados diversos algoritmo de inteligência artificial para comparar seus desempenhos na tarefa de previsão da condição de motores a combustão. A escolha dos algoritmos se baseou em suas características e eficácia em problemas de classificação.

A divisão dos dados foi realizada utilizando uma proporção de 70% para o conjunto de treino e 30% para o conjunto de teste. Esta estratégia permite que o modelo seja treinado em uma quantidade suficiente de dados, enquanto ainda reserva uma parte significativa para avaliar o desempenho e a generalização do modelo em dados não vistos. Essa abordagem é essencial para garantir que os resultados obtidos não sejam fruto de *overfitting* (superajuste), proporcionando uma avaliação mais realista do modelo aplicado.

4.2.1 Resultados

Submetendo os dados os Algoritmos de ML e DL já citados, foi possível obter os resultado presentes na Tabela 5

Tabela 5 – Comparação entre algoritmos de aprendizado de máquina e deep learning

	LR	**SVM**	**AD**	**RF**	**DL**
Acurácia	0.6733	0.6589	0.5905	0.6592	0.6698
F1-Score motores ruins	0.45	0.29	0.44	0.47	0.46
F1-Score motores bons	0.77	0.78	0.68	0.75	0.76
FP (Falsos Positivos)	1302	1689	1137	1203	1263
FN (Falsos Negativos)	600	297	1247	781	695

A Tabela 5 evidencia que o desempenho dos algoritmos varia significativamente em função da métrica avaliada. **Random Forest** e **Deep Learning** são as opções mais promissoras para detecção de motores ruins, uma vez que apresentam um bom equilíbrio entre acurácia, F1-Score e taxas de falsos positivos e negativos. **SVM** e **Árvore de Decisão**, embora tenham mostrado bons resultados para certos aspectos, apresentaram falhas que podem comprometer a confiabilidade do sistema de manutenção preditiva, especialmente na detecção de motores com defeitos.

Portanto, para um sistema de manutenção preditiva eficiente e personalizado, a escolha do algoritmo deve considerar não apenas a acurácia geral, mas também as taxas de falsos positivos e negativos, uma vez que erros na detecção de motores defeituosos podem levar a falhas críticas ou a custos desnecessários de manutenção. **Random Forest** e **Deep Learning** mostraram-se os mais adequados para essa tarefa, combinando robustez e precisão.

4.3 Otimização

Após a criação inicial dos modelos de IA e a avaliação de seu desempenho, foi identificada a necessidade de aprimorar os resultados ajustando os hiperparâmetros dos modelos. Hiperparâmetros são valores que controlam o processo de aprendizado e afetam diretamente a performance final de um algoritmo, mas não são aprendidos pelos dados. Alguns exemplos de hiperparâmetros incluem a taxa de aprendizado, o número de camadas em uma rede neural, e os parâmetros de regularização em algoritmos como SVM, Random Forest e Redes Neurais.

Para realizar essa otimização, utilizamos a técnica **GridSearchCV** do Scikit-Learn, que nos permite realizar uma busca exaustiva pelos melhores hiperparâmetros, testando todas as combinações possíveis dentro de um conjunto predefinido. A busca é feita utilizando validação cruzada, garantindo que os resultados sejam robustos e não estejam sujeitos ao *overfitting* no conjunto de treino.

O modelo de machine learning escolhido para otimização foi o **Random Forest**, pois ele apresentou a melhor acurácia entre os modelos disponíveis que permitem ajustes de hiperparâmetros. Esse algoritmo é altamente versátil e robusto, pois combina múltiplas árvores de decisão para melhorar a precisão e reduzir o risco de overfitting. A capacidade de ajustar parâmetros como o número de estimadores e a profundidade máxima da árvore torna o Random Forest uma excelente escolha para otimização de desempenho em tarefas de classificação.

No caso de redes neurais profundas (Deep Learning), a otimização de hiperparâmetros foi feita utilizando o **Keras Tuner**, uma biblioteca específica para o ajuste de hiperparâmetros em modelos desenvolvidos com o **TensorFlow**. O Keras Tuner facilita a busca automática por hiperparâmetros ideais, ajustando elementos como:

- O número de neurônios em cada camada densa,
- O número de camadas ocultas,
- A função de ativação (ReLU, Sigmoid, etc.),
- A taxa de aprendizado do otimizador (Adam, SGD),
- O uso de dropout para regularização.

Utilizando o **Keras Tuner** com a abordagem de **Random Search** ou **Hyperband**, foi possível testar diversas configurações de redes neurais para encontrar a arquitetura mais adequada ao problema. A busca foi realizada com validação cruzada e com a métrica de F1-Score e acurácia como critério principal de seleção, uma vez que o foco era equilibrar a performance na detecção de motores bons e ruins.

Ao final do processo, foi observado um aumento sutil na precisão dos modelos após a otimização dos hiperparâmetros. Conforme apresentado na Tabela 6, o modelo Random Forest e o modelo de Deep Learning mostraram melhorias consistentes na acurácia, evidenciando a importância de ajustes finos para melhorar o desempenho. Embora os ganhos tenham sido modestos, a otimização contribuiu para maximizar a performance em tarefas de classificação, comprovando a eficácia desse processo em ambos os casos analisados.

Como o modelo de Deep Learning se destacou como o mais eficaz, os tópicos a seguir se concentrarão exclusivamente nesse método. Essa escolha reflete sua capacidade de capturar padrões complexos e melhorar a precisão das previsões em comparação com outros modelos. Portanto, todos os estudos subsequentes e análises serão realizados utilizando este modelo, garantindo uma abordagem mais robusta e confiável em nossas investigações.

Tabela 6 – Comparação de acurácia dos modelos Random Forest e Deep Learning com e sem otimização.

	Random Forest	**Deep Learning**
Sem Otimização	0.6592	0.6698
Com Otimização	0.6670	0.6766

4.4 Simulando de Desgaste Continuo

A ideia central do experimento foi simular o desgaste progressivo de um carro desde o momento em que ele está novo até o ponto de falha. Para isso, o modelo inicialmente identificou veículos com mais de 90% de probabilidade de estarem em boas condições, considerando suas variáveis médias como um ponto de partida para um carro "zero". A partir daí, o desgaste foi acompanhado ao longo de 200 mil km, com incrementos simulados a cada 10 km, ajustados por variações aleatórias para replicar as condições reais de uso até o momento da falha.

```
max_km = 200000
km_interasion = 10
carro = media_motores_excelentes
km_accumulated = 0

wear_parameters = {
    'Engine_rpm': (0.90, 1.1),
    'lub_oil_pressure': (0.90, 1.1),
    'Fuel_pressure': (0.90, 1.1),
    'coolant_pressure': (0.90, 1.1),
    'lub_oil_temp': (0.90, 1.1),
    'coolant_temp': (0.90, 1.1),
    'coolant_efficiency': (0.90, 1.1),
```

```
    'oil_efficiency': (0.90, 1.1),
}

while km_accumulated <= max_km:
    carro['Engine rpm'] += -desgaste_10km['Engine rpm'] * np.random.
        uniform(*wear_parameters['Engine_rpm'])
    carro['Lub oil pressure'] += desgaste_10km['Lub oil pressure'] * np.
        random.uniform(*wear_parameters['lub_oil_pressure'])
    carro['Fuel pressure'] += desgaste_10km['Fuel pressure'] * np.random
        .uniform(*wear_parameters['Fuel_pressure'])
    carro['Coolant pressure'] += -desgaste_10km['Coolant pressure'] * np
        .random.uniform(*wear_parameters['coolant_pressure'])
    carro['lub oil temp'] += -desgaste_10km['lub oil temp'] * np.random.
        uniform(*wear_parameters['lub_oil_temp'])
    carro['Coolant temp'] += -desgaste_10km['Coolant temp'] * np.random.
        uniform(*wear_parameters['coolant_temp'])
    carro['Coolant Efficiency'] = (1 / carro['Engine rpm']) * carro['
        Coolant temp']
    carro['Oil Efficiency'] = 1 / (carro['Engine rpm'] * carro['lub oil
        temp'])

    carro_array = carro.to_numpy()
    carro_array = carro_array.reshape(1, -1)
    carro_array_normaliz = (carro_array - means) / scales

    if km_accumulated % 100 == 0:
        print(f"Quilometragem acumulada: {km_accumulated} km")
        print(carro)
        probabilities = melhor_modelo.predict(carro_array_normaliz)
        print(f"Amostra {{km_accumulated}}: Probabilidade de ser 'Motor
            Bom': {probabilities * 100}%")
    km_accumulated += km_interasion

print(f"Fim da Simula  o!")
```

Listing 4.3 – Simulação de desgaste contínuo

A Tabela 7 apresentada resume as probabilidades de um motor ser classificado como "Bom"em diferentes etapas de quilometragem acumulada, variando de 92,07% a 55,58% ao longo de 200.000 km. Observa-se uma tendência de redução na probabilidade à medida que a quilometragem aumenta, refletindo o desgaste progressivo do motor. Essa informação é crucial para entender como a saúde do motor pode ser avaliada ao longo do tempo e destaca a importância do monitoramento contínuo para manutenção preventiva.

Tabela 7 – Probabilidade de ser "Motor Bom"ao longo da quilometragem acumulada

Quilometragem Acumulada (km)	Probabilidade de ser 'Motor Bom' (%)
0	92.07
25000	83.54
50000	75.52
75000	68.32
100000	64.26
125000	62.34
150000	60.20
175000	57.81
200000	55.58

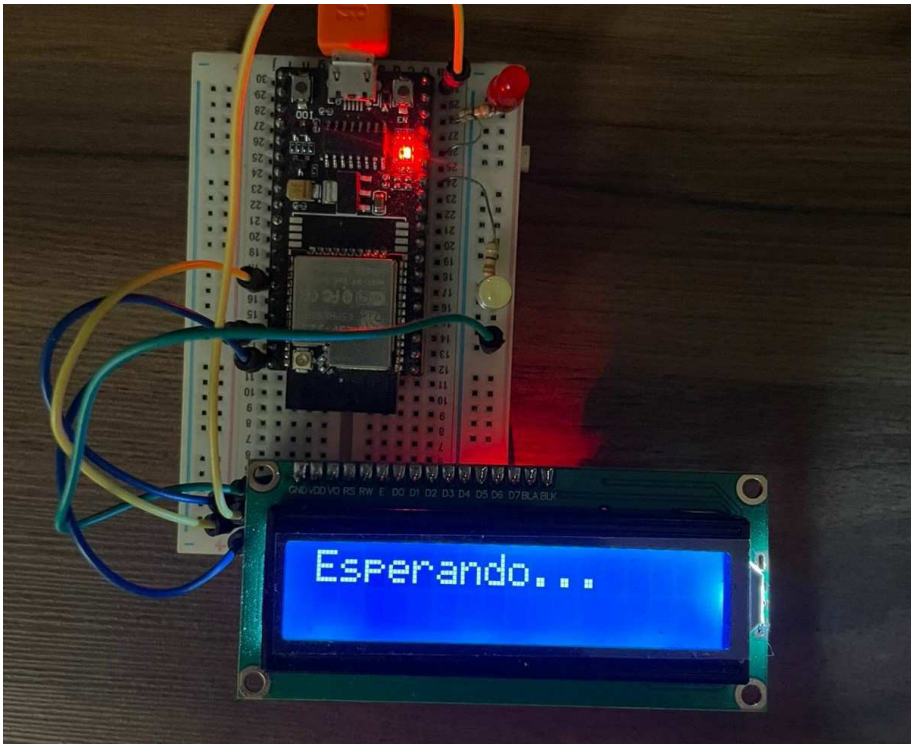
4.5 Aplicação em Microcontrolador

Após a otimização e avaliação dos modelos de IA, o próximo passo foi implementar a solução em um ambiente de hardware embarcado, no caso o microcontrolador ESP32. Esse processo envolveu a conversão do modelo de IA treinado para um formato que pudesse ser executado eficientemente no ESP32, um dispositivo com recursos limitados de processamento e memória. Para alcançar isso, foi utilizado o TensorFlow Lite, uma versão leve do TensorFlow desenvolvida especificamente para dispositivos com baixa capacidade computacional.

O processo de exportação começou com a conversão do modelo treinado em um arquivo .tflite. Essa etapa foi realizada utilizando a API do TensorFlow, que permite a conversão direta de modelos desenvolvidos e otimizados para um formato compatível com o TensorFlow Lite. Durante a conversão, são aplicadas diversas otimizações, como a quantização dos pesos, que podem em alguns casos, reduz a precisão de alguns parâmetros para economizar memória e melhorar a eficiência sem comprometer drasticamente a precisão do modelo.

A acurácia do novo modelo simplificado foi medida em 67,66%, mantendo-se inalterada em relação ao modelo original. Mesmo com a redução na quantidade de bits representativos e a utilização de menos recursos de memória, o desempenho não foi comprometido. O modelo também foi otimizado para funcionar eficientemente no ESP32, atendendo aos requisitos de um sistema de baixo custo e com hardware limitado. Essa manutenção de acurácia com uso reduzido de recursos indica que o modelo pode ser uma solução viável para aplicações em microcontroladores sem perda significativa de desempenho preditivo.

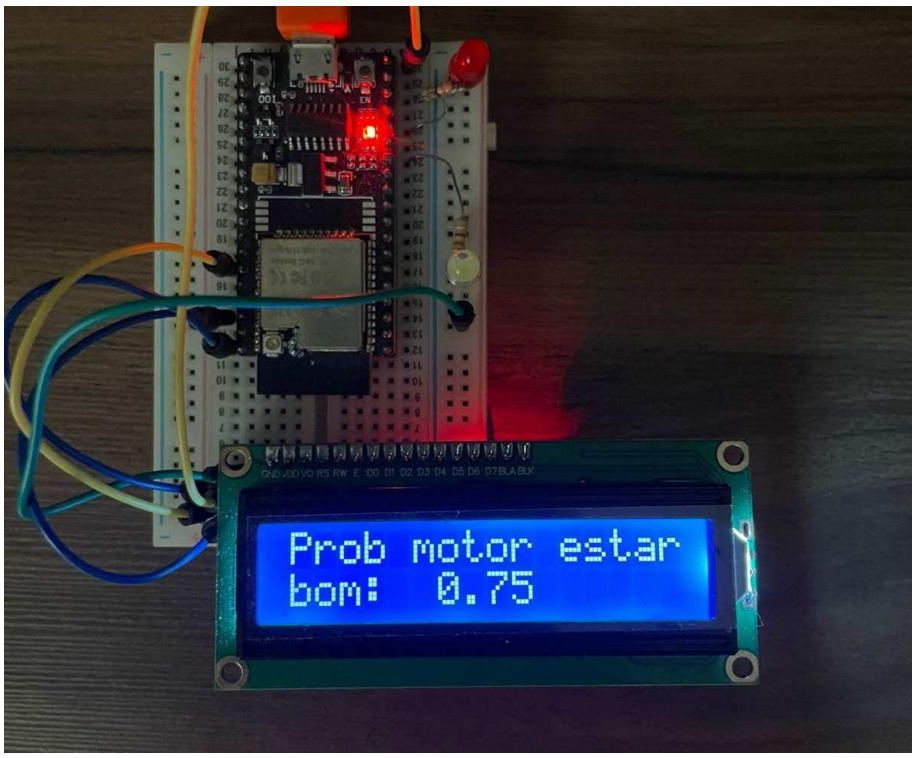
Figura 5 – ESP32 aguardando comunicação Bluetooth



Fonte: Autoria Própria.

Uma vez que o modelo foi convertido para o formato .tflite, ele foi integrado ao código C++ desenvolvido no VSCode com o auxílio do PlatformIO. Para a comunicação entre o ESP32 e o ambiente externo, foi utilizada uma interface de Bluetooth, que recebe os dados do motor em tempo real. O modelo embarcado faz as inferências necessárias para diagnosticar a condição do motor e exibe os resultados em um display LCD 16x2 com I2C conectado ao ESP32.

Figura 6 – ESP32 exibindo resultado de amosta 1



Fonte: Autoria Própria.

Esse processo de exportação e implantação foi essencial para transformar o modelo de IA em uma ferramenta prática de diagnóstico preditivo, aproximando-o ainda mais de uma aplicação real em veículos de combustão. Ao prever falhas e otimizar a manutenção, a solução desenvolvida não apenas aumenta a eficiência operacional, mas também reduz custos associados à manutenção corretiva. A integração da IA com o hardware embarcado, como o ESP32, demonstra ser uma solução de baixo custo, eficiente e escalável para o setor automotivo, trazendo inovações significativas no monitoramento e na gestão preventiva de veículos.

Devido à limitação de recursos e à falta de acesso direto a um veículo de teste, o MATLAB foi utilizado para processar e enviar dados (os mesmos dados usados para testar os modelos na Seção 4.3). Após o processamento, as amostras de dados são enviadas para o ESP32 via Bluetooth. O ESP32, por sua vez, recebe essas amostras e as submete ao modelo embarcado para prever a probabilidade de cada motor estar em boas condições de funcionamento.

Para apresentar os resultados, um display LCD foi integrado ao sistema, além de retornar os dados ao MATLAB para análise adicional. O display exibe em tempo real as probabilidades de classificação, permitindo que os operadores visualizem rapidamente o estado de cada motor. As informações mostradas incluem a probabilidade de o motor ser classificado como "Bom". Essa interface é essencial, pois oferece uma forma simples e intuitiva de monitorar o estado dos motores, facilitando a tomada de decisões relacionadas

à manutenção, e garantindo que ações corretivas possam ser tomadas com base em dados preditivos.

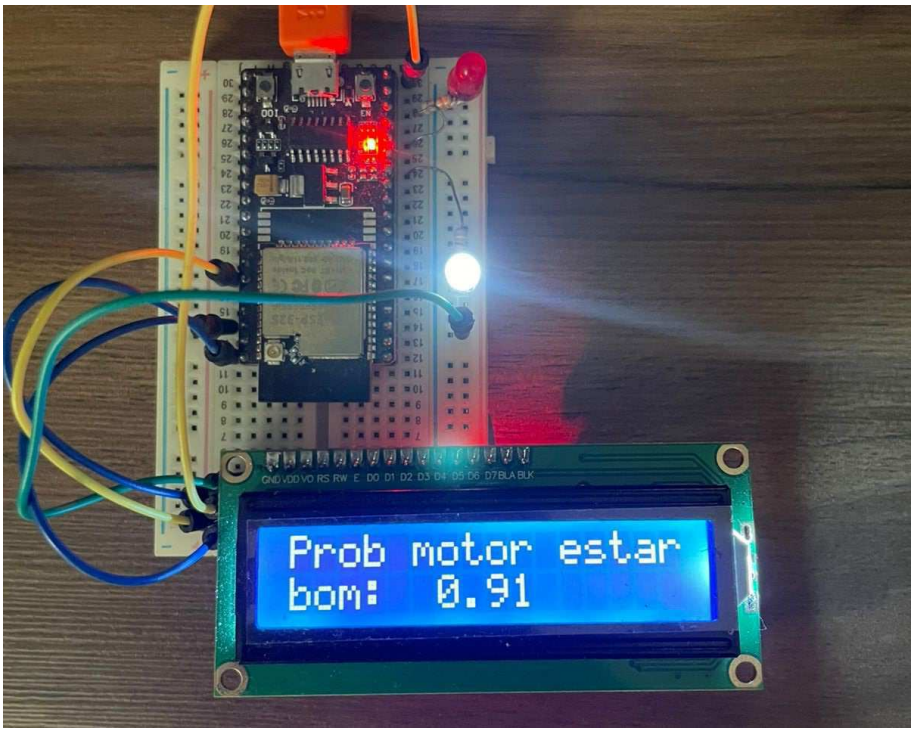
Figura 7 – Resultado de saída do MATLAB

Fonte: Autoria Própria.

Para melhorar a visualização dos resultados e fornecer um feedback rápido, também foram incluídos dois LEDs: um vermelho e um branco. O LED vermelho acende quando a probabilidade de o motor estar em boas condições é inferior a 60%, indicando um alerta de possível falha. Quando a probabilidade de bom funcionamento do motor é superior a 80%, o LED branco acende, sinalizando que o motor está em condições ótimas. Para probabilidades entre 60% e 80%, ambos os LEDs permanecem apagados, sugerindo uma condição intermediária que não requer ação imediata. Essa solução oferece

uma interface visual clara e intuitiva, permitindo que os operadores avaliem rapidamente o estado dos motores sem a necessidade de observar constantemente o display ou os dados no MATLAB.

Figura 8 – ESP32 exibindo resultado de amosta 2



Fonte: Autoria Própria.

5 Conclusão

A busca por diminuir os custos de manutenção é constante, e, nesse contexto, a manutenção preditiva se mostra uma excelente opção quando comparada às técnicas tradicionais de manutenção. Isso ocorre porque é possível acompanhar o desgaste de componentes de forma individualizada, permitindo a personalização dos intervalos de manutenção e, consequentemente, a redução de custos e a otimização do desempenho do sistema.

Nesse sentido, este trabalho cumpriu com seu objetivo, desenvolvendo uma ferramenta eficaz para calcular o desgaste do conjunto do motor. Dessa forma, foi possível calcular tempos de manutenção personalizados para cada caso. Além disso, a utilização de um microcontrolador de baixo custo, como o ESP32, viabilizou a implementação da solução em veículos, tornando-a acessível para diversas aplicações.

Contudo, durante o desenvolvimento, foi observado que alguns aspectos poderiam ser otimizados para resultados ainda melhores. Um exemplo seria o uso de um conjunto de dados mais confiável e com informações específicas de cada veículo. Isso traria maior confiança nos resultados e, possivelmente, melhoraria a acurácia do modelo de predição.

Outro ponto que seria interessante aprimorar é a inclusão, no banco de dados, de informações sobre os defeitos que o veículo apresentava no momento da coleta dos parâmetros. Com esses dados adicionais, seria possível estimar com maior precisão qual peça do conjunto estaria com defeito, proporcionando uma análise mais detalhada e assertiva, melhorando a capacidade preditiva do sistema de manutenção.

Referências

Amazon Web Services. *O que é regressão logística?* 2023. Acessado em: 26 de setembro de 2024. Disponível em: <https://aws.amazon.com/pt/what-is/logistic-regression/>. Citado na página 18.

BOUSDEKIS, A. et al. Review, analysis and synthesis of prognostic-based decision support methods for condition-based maintenance. *Journal of Intelligent Manufacturing*, Springer, p. 1–20, 2015. Citado na página 12.

BREIMAN, L. Random forests. *Machine Learning*, Springer, v. 45, n. 1, p. 5–32, 2001. Citado na página 16.

BREIMAN, L. Random forests. *Machine Learning*, Springer, v. 45, n. 1, p. 5–32, 2001. Citado na página 19.

Colaborae Blog. *Árvore de Decisão: O que é e como funciona?* 2023. Acessado em: 26 de setembro de 2024. Disponível em: <https://colaborae.com.br/blog/2023/07/19/arvore-de-decisao/>. Citado na página 18.

CORTES, C.; VAPNIK, V. Support-vector networks. *Machine Learning*, Springer, v. 20, n. 3, p. 273–297, 1995. Citado na página 19.

DALPIAZ, G.; RIVOLA, A.; RUBINI, R. Effectiveness and sensitivity of vibration processing techniques for local fault detection in gears. *Mechanical Systems and Signal Processing*, Elsevier, v. 14, n. 3, p. 387–412, 2000. Citado na página 15.

DEMPSEY, P. J.; AFJEH, A. A. Integrating vibration-based diagnostics, prognostics, and condition-based maintenance. *Mechanical Systems and Signal Processing*, Elsevier, v. 16, n. 6, p. 745–762, 2002. Citado na página 15.

HEYWOOD, J. B. *Internal Combustion Engine Fundamentals*. New York: McGraw-Hill, 1988. Citado na página 36.

HOSMER, D. W.; LEMESHOW, S.; STURDIVANT, R. X. *Applied Logistic Regression*. 3rd. ed. Hoboken, NJ: Wiley, 2013. ISBN 978-0470582473. Citado na página 17.

JARDINE, A. K.; LIN, D.; BANJEVIC, D. A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical systems and signal processing*, Elsevier, v. 20, n. 7, p. 1483–1510, 2006. Citado 3 vezes nas páginas 12, 15 e 17.

KIM, B. H.; AN, D. H.; CHOI, Y. R. Machine learning for failure diagnosis of vehicle components based on can data. *Journal of Mechanical Science and Technology*, Springer, v. 31, n. 7, p. 3351–3358, 2017. Citado na página 15.

LEE, J.; LAPIRA, E. *Predictive Maintenance for Automotive Systems*. [S.l.]: Springer, 2013. 57–81 p. Citado na página 12.

MOHAMMADI, S.; MOMENI, S.; VAFERI, B. Machine learning-based models in predictive maintenance: a review of the state-of-the-art. *Journal of Intelligent Manufacturing*, Springer, 2021. Citado na página 12.

PRAMOD, O. *Random Forests*. 2023. Acessado em: 26 de setembro de 2024. Disponível em: <https://medium.com/@ompramod9921/random-forests-32be04c8bf76>. Citado na página 19.

QUINLAN, J. R. Induction of decision trees. *Machine Learning*, Springer, v. 1, n. 1, p. 81–106, 1986. Citado na página 18.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 4th. ed. London: Pearson, 2020. ISBN 978-0134610993. Citado na página 16.

SMITH, J. *Maintenance Engineering Handbook*. 8th. ed. New York: McGraw-Hill Education, 2020. Citado na página 14.

SMITH, R.; WANG, X. Predictive maintenance using deep learning techniques: Applications and trends. *IEEE Transactions on Industrial Informatics*, IEEE, v. 16, n. 5, p. 3601–3610, 2020. Citado na página 22.

TSUI, K. L. et al. Prognostics and health management: A review on data driven approaches. *Mathematical Problems in Engineering*, Hindawi, v. 2015, p. 1–17, 2015. Citado na página 12.

ZHANG, Y.; GANESAN, R.; SHI, J. Machine learning in automotive predictive maintenance: A comprehensive review. *IEEE Transactions on Automation Science and Engineering*, v. 14, n. 4, p. 2096–2108, 2017. Citado na página 12.

Anexos

Este capítulo apresenta os códigos e materiais complementares utilizados durante o desenvolvimento deste trabalho.

Anexo A - Código Python para Criação do Modelo

O código abaixo foi utilizado para criar e treinar o modelo de IA no Python, aplicando técnicas de machine learning para a análise preditiva.

```
# -*- coding: utf-8 -*-
"""TCC_Criar_modelo.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/10e3Iio1sz934q43GILtw-kbpJOz1iF6C
"""

!pip install keras-tuner

"""# Adicionando Bibliotecas




"""

# Manipula  o e an lise de dados
import pandas as pd
import numpy as np
import random

# Algoritmos de machine learning
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.base import TransformerMixin, BaseEstimator
```

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC
#Deeplearn
import tensorflow as tf

# Deep Learning
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization,
    Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import torch
import keras_tuner as kt
# Gradient Boosting
import xgboost as xgb
import lightgbm as lgb

# Visualiza  o de dados
import seaborn as sns
import matplotlib.pyplot as plt

from google.colab import drive
drive.mount('/content/drive')

"""# Importando dados

"""

caminho_engine_df = '/content/drive/MyDrive/TCC/engine_data.csv'
engine_df = pd.read_csv(caminho_engine_df)
print(engine_df.head())

"""Exibir um resumo conciso de um DataFrame. Esse m todo     til  para
    obter uma vis o geral da estrutura dos dados, como o n mero de
    entradas, o n mero de colunas, os nomes das colunas, o tipo de dado
    de cada coluna, e o n mero de valores n o nulos em cada coluna."""

engine_df.info()

"""Gerar estat sticas descritivas que resumem a tend ncia central, a
    dispers o e a forma da distribui  o de um conjunto de dados,
    excluindo valores NaN. Esse m todo    particularmente  til  para ter
     uma vis o geral r pida das propriedades num ricas de um DataFrame
    ."""

engine_df.describe()
```

```
72
73 """Contar as condi  es quantos motores do banco de dados est o bons,
       e quantos est o ruins. **(0 - Ruim ; 1 - Bom)**"""
74
75 engine_df["Engine Condition"].value_counts()
76
77 """Separar a coluna 'engine condiction' das demais"""
78
79 engine_features = engine_df.drop("Engine Condition", axis=1)
80 engine_labels = engine_df["Engine Condition"]
81 print(engine_features.head())
82 print("\n")
83 print(engine_labels.head())
84
85 """Visualizando a distribui  o de todos os atributos"""
86
87 sns.pairplot(engine_df, hue="Engine Condition")
88
89 """Infer ncia
90
91 * A maioria dos atributos fornecidos pelo conjunto de dados s  o bem
       distribu dos em rela  o a todas as colunas
92 * A temperatura do l quido de arrefecimento precisa ser dimensionada
       para tornar sua distribui  o mais uniforme
93
94 Calcular correla  o
95 """
96
97 corr_matrix = engine_df.corr()
98 corr_matrix["Engine Condition"].sort_values(ascending=False)
99
100 """Infer ncia
101 *  A condi  o do motor tem a maior correla  o positiva com a
       press o do combust vel
102 *  A condi  o do motor tem a maior correla  o negativa com a
       rota  o do motor
103
104 # Tentando novos atributos (Engenharia de recursos)
105 """
106
107 # Oiling system
108 #engine_df["Oil System"] = engine_df["Lub oil pressure"] * engine_df["
       lub oil temp"]
109
110 # Coolant System
111 #engine_df["Coolant System"] = engine_df["Coolant pressure"] * engine_df
       ["Coolant temp"]
```

```
112
113 # Coolant Efficiency
114 engine_df["Coolant Efficiency"] = (1 / engine_df["Engine rpm"]) *
       engine_df["Coolant temp"]
115
116 # Oil Efficiency
117 engine_df["Oil Efficiency"] = 1 / (engine_df["Engine rpm"] * engine_df["
       lub oil temp"])
118
119 engine_df.head()
120
121 """
122
123 Calculando as Correla  es com os Novos Atributos"""
124
125 corr_matrix = engine_df.corr()
126 corr_matrix["Engine Condition"].sort_values(ascending=False)
127
128 """# Pr -processamento de dados
129 Construindo um transformador personalizado para adicionar os novos
       atributos
130 """
131
132 rpm_idx, oil_pressure_idx, coolant_pressure_idx, oil_temp_idx,
       coolant_temp_idx = 0, 1, 3, 4, 5
133
134 class AttributesAdder(BaseEstimator, TransformerMixin): #Cria uma Classe
        para reallizar adi   o de colunas na tabela de atributos.
135
136     # Constructor of the Class
137     def __init__(self, add_oil_system=True, add_coolant_system=True):
138         self.add_oil_system = add_oil_system
139         self.add_coolant_system = add_coolant_system
140
141     def fit(self, X, y=None):
142         return self
143
144     def transform(self, X):
145         if self.add_oil_system:
146             # oil_system = X[:, oil_pressure_idx] * X[:, oil_temp_idx]
147             oil_efficiency = 1 / (X[:, rpm_idx] * X[:, oil_temp_idx])
148             # X = np.c_[X, oil_system, oil_efficiency]
149             X = np.c_[X, oil_efficiency]
150
151         if self.add_coolant_system:
152             # cool_system = X[:, coolant_pressure_idx] * X[:,
                coolant_temp_idx]
```

```
153             cool_efficiency = (1 / X[:, rpm_idx]) * X[:,
                    coolant_temp_idx]
154             # X = np.c_[X, cool_system, cool_efficiency]
155             X = np.c_[X, cool_efficiency]
156
157         return X
158
159 engine_features.head()
160
161 attr_addr = AttributesAdder()
162 engine_prep = attr_addr.transform(engine_features.values) #.values
    converte de panda para um array numpay; .transform aplica a regra da
    classe para adicionar as colunas
163 print(f"Transformed Data: {engine_prep[0, :]}")
164
165 """Construindo o pipeline para automatizar o pr -processamento de dados
    """
166
167 engine_prep_pipe = Pipeline([
168     ("attr_adder", AttributesAdder()), # aplica a transforma  o
        definida pela classe AttributesAdder, que adiciona as colunas de
        efici ncia do  leo  e do sistema de refrigera  o aos dados
169     ("std_scaler", StandardScaler())   # aplica a normaliza  o usando
        StandardScaler. Ele padroniza os dados para que cada
        caracter stica tenha m dia 0 e desvio padr o 1, o que     til
         para muitos algoritmos de machine learning que s o sens veis
           escala dos dados.
170 ])
171
172 engine_data_prepared = engine_prep_pipe.fit_transform(engine_features.
    values) #Esta fun  o aplica todas as transforma  es do pipeline
    aos dados engine_features.values. Primeiro, o AttributesAdder
    adiciona as colunas de efici ncia, e em seguida, o StandardScaler
    normaliza todas as colunas resultante
173 engine_data_prepared[0, :] # apresenta a primeira linha do conjunto de
    dados
174
175 engine_data_prepared.shape
176
177 """
178 Dividindo o conjunto de dados preparado"""
179
180 X_train, X_test, y_train, y_test = train_test_split(engine_data_prepared
    , engine_labels, test_size=0.3, random_state=42) #test_size=0.3 =>
    tamanho do meu conjunt de dados em %; random_state => semente para a
    sele  o de dados, caso seja None, toda vez que o codigo
    executado teremos resultados diferentes.
```

```
181 print(f"Shape X Train: {X_train.shape}")
182 print(f"Shape y Train: {y_train.shape}\n")
183 print(f"Shape X Test: {X_test.shape}")
184 print(f"Shape y Test: {y_test.shape}\n")
185
186 """# Retirando Outliers"""
187
188 # Calcular o primeiro quartil (Q1) e o terceiro quartil (Q3)
189 Q1 = engine_df.quantile(0.25)
190 Q3 = engine_df.quantile(0.75)
191 IQR = Q3 - Q1
192
193 # Definir os limites
194 lower_bound = Q1 - 1.5 * IQR
195 upper_bound = Q3 + 1.5 * IQR
196
197 # Filtrar os dados para remover os outliers
198 engine_df_clean = engine_df[~((engine_df < lower_bound)).any(axis=1)]
199
200 # Exibir o novo DataFrame sem outliers
201 print(engine_df_clean.head())
202
203 engine_df_clean["Engine Condition"].value_counts()
204
205 engine_features = engine_df_clean.drop("Engine Condition", axis=1)
206 engine_labels = engine_df_clean["Engine Condition"]
207 print(engine_features.head())
208 print("\n")
209 print(engine_labels.head())
210
211 engine_prep_pipe = Pipeline([
212     ("std_scaler", StandardScaler())   # aplica a normaliza  o usando
        StandardScaler. Ele padroniza os dados para que cada
        caracter stica tenha m dia 0 e desvio padr o 1, o que     til
         para muitos algoritmos de machine learning que s  o sens veis
           escala dos dados.
213 ])
214
215 engine_data_prepared = engine_prep_pipe.fit_transform(engine_features.
    values) #Esta fun  o aplica todas as transforma  es do pipeline
    aos dados engine_features.values. Primeiro, o AttributesAdder
    adiciona as colunas de efici ncia, e em seguida, o StandardScaler
    normaliza todas as colunas resultante
216 engine_data_prepared[0, :] # apresenta a primeira linha do conjunto de
    dados
217
218 engine_data_prepared.shape
```

```
219
220 X_train, X_test, y_train, y_test = train_test_split(engine_data_prepared
    , engine_labels, test_size=0.3, random_state=42) #test_size=0.3 =>
    tamanho do meu conjunt de dados em %; random_state => semente para a
    sele  o de dados, caso seja None, toda vez que o codigo
    executado teremos resultados diferentes.
221 print(f"Shape X Train: {X_train.shape}")
222 print(f"Shape y Train: {y_train.shape}\n")
223 print(f"Shape X Test: {X_test.shape}")
224 print(f"Shape y Test: {y_test.shape}\n")
225
226 # Acessar o StandardScaler do pipeline
227 scaler = engine_prep_pipe.named_steps['std_scaler']
228
229 # Obter os par metros de normaliza  o
230 means = scaler.mean_
231 scales = scaler.scale_
232
233 # Exibir os par metros
234 print("Means:")
235 print(means)
236
237 print("\nScales:")
238 print(scales)
239
240 # Initilisation
241 log_reg = LogisticRegression() #Cria uma inst ncia da classe
    LogisticRegression
242
243 # Training
244 log_reg.fit(X_train, y_train) #Este m todo treina o modelo usando os
    dados de treino (X_train) e os r tulos correspondentes (y_train). O
    modelo aprende a rela  o entre as vari veis independentes (
    caracter sticas) e a vari vel dependente (r tulos) durante este
    processo.
245
246 validation = log_reg.predict(X_test) #O m todo predict retorna as
    previs es do modelo com base nos dados de entrada.
247 score = sum(validation == y_test)
248 #validation == y_test: Esta express o cria um array booleano onde True
    indica uma predi  o correta (a predi  o do modelo corresponde ao
    r tulo real) e False indica uma predi  o incorreta.
249 #sum(validation == y_test): A fun  o sum conta quantas predi  es
    foram corretas.
250 #Acur cia: A acur cia  calculada dividindo o n mero de predi  es
    corretas pelo n mero total de amostras no conjunto de teste (len(
    y_test)). A acur cia  uma m trica comum para avaliar a
```

```
    performance de modelos de classifica  o.
251
252 # Calculando a matriz de confus o
253 cm = confusion_matrix(y_test, validation)
254
255 # Separando os quadrantes
256 tn, fp, fn, tp = cm.ravel() #A fun  o ravel() do NumPy "achata" a
    matriz de confus o de uma matriz 2x2 para um array unidimensional
    com quatro elementos
257
258 # Imprimindo os valores dos quadrantes
259 print(f"Verdadeiros Positivos (TP): {tp}")
260 print(f"Verdadeiros Negativos (TN): {tn}")
261 print(f"Falsos Positivos (FP): {fp}")
262 print(f"Falsos Negativos (FN): {fn}")
263
264
265 # Relat rio de Classifica  o
266 print("\nRelat rio de Classifica  o:")
267 print(classification_report(y_test, validation, target_names=["Motor
    Ruim", "Motor Bom"]))
268
269
270 print(f"Score: {score / len(y_test)}")
271
272 # Inicializa  o
273 svm_cls = SVC(kernel='linear', random_state=42)  # Voc  pode ajustar o
    kernel se necess rio
274
275 # Treinamento
276 svm_cls.fit(X_train, y_train)
277
278 # Previs es
279 validation = svm_cls.predict(X_test)
280
281 # Calcular o score de acur cia
282 score = accuracy_score(y_test, validation)
283 print(f"Score: {score}")
284
285 # Calculando a matriz de confus o
286 cm = confusion_matrix(y_test, validation)
287
288 # Separando os quadrantes
289 tn, fp, fn, tp = cm.ravel()  # "Achata" a matriz de confus o de 2x2
    para um array unidimensional com quatro elementos
290
291 # Imprimindo os valores dos quadrantes
```

```
292 print(f"Verdadeiros Positivos (TP): {tp}")
293 print(f"Verdadeiros Negativos (TN): {tn}")
294 print(f"Falsos Positivos (FP): {fp}")
295 print(f"Falsos Negativos (FN): {fn}")
296
297 # Relat rio de Classifica  o
298 print("\nRelat rio de Classifica  o:")
299 print(classification_report(y_test, validation, target_names=["Motor
    Ruim", "Motor Bom"]))
300
301 # Initilisation
302 tree_cls = DecisionTreeClassifier() #
303
304 # Training
305 tree_cls.fit(X_train, y_train) #
306
307 validation = tree_cls.predict(X_test)
308 score = sum(validation == y_test)
309 print(f"Score: {score / len(y_test)}")
310
311 # Calculando a matriz de confus o
312 cm = confusion_matrix(y_test, validation)
313
314 # Separando os quadrantes
315 tn, fp, fn, tp = cm.ravel() #A fun  o ravel() do NumPy "achata" a
    matriz de confus o de uma matriz 2x2 para um array unidimensional
    com quatro elementos
316
317 # Imprimindo os valores dos quadrantes
318 print(f"Verdadeiros Positivos (TP): {tp}")
319 print(f"Verdadeiros Negativos (TN): {tn}")
320 print(f"Falsos Positivos (FP): {fp}")
321 print(f"Falsos Negativos (FN): {fn}")
322
323
324 # Relat rio de Classifica  o
325 print("\nRelat rio de Classifica  o:")
326 print(classification_report(y_test, validation, target_names=["Motor
    Ruim", "Motor Bom"]))
327
328 # Initilisation
329 forest_cls = RandomForestClassifier(max_features=4, n_estimators=100)
330
331 # Training
332 forest_cls.fit(X_train, y_train)
333
334 validation = forest_cls.predict(X_test)
```

```
335 score = sum(validation == y_test)
336 print(f"Score: {score / len(y_test)}")
337
338 # Calculando a matriz de confus o
339 cm = confusion_matrix(y_test, validation)
340
341 # Imprimindo a matriz de confus o
342 print("Matriz de Confus o:")
343 print(cm)
344
345 # Separando os quadrantes
346 tn, fp, fn, tp = cm.ravel() #A fun   o ravel() do NumPy "achata" a
    matriz de confus o de uma matriz 2x2 para um array unidimensional
    com quatro elementos
347
348 # Imprimindo os valores dos quadrantes
349 print(f"Verdadeiros Positivos (TP): {tp}")
350 print(f"Verdadeiros Negativos (TN): {tn}")
351 print(f"Falsos Positivos (FP): {fp}")
352 print(f"Falsos Negativos (FN): {fn}")
353
354
355 # Relat rio de Classifica  o
356 print("\nRelat rio de Classifica  o:")
357 print(classification_report(y_test, validation, target_names=["Motor
    Ruim", "Motor Bom"]))
358
359 probabilities = forest_cls.predict_proba(X_test)
360 # percentages = probabilities[:, 1] * 100 # Converter a probabilidade da
     classe 1 em porcentagem
361
362 for i in range(10):
363     prob_class_1 = probabilities[i, 1]  # Probabilidade de classe 1 (
        Motor Bom)
364     print(f"Amostra {i}: Probabilidade de ser 'Motor Bom': {prob_class_1
         * 100:.2f}%")
365
366 # Acur cia: D  uma vis o geral, mas pode ser enganosa se as classes
    forem desbalanceadas.
367 # Precis o: Indica quantas das previs es positivas estavam corretas.
       importante quando falsos positivos s o custosos.
368 # Recall: Indica a capacidade do modelo de capturar todos os positivos
    verdadeiros.    importante quando falsos negativos s o custosos.
369 # F1-Score: Equilibra precis o e recall e     til  em cen rios com
    classes desbalanceadas.
370
371 # Definir o par metro para o GridSearchCV
```

```
372 param_grid = [
373     {'n_estimators': [10, 50, 100], 'max_features': [2, 4, 6, 8]},
374     {'bootstrap': [False], 'n_estimators': [10, 50], 'max_features': [2,
            3, 4]},
375 ]
376
377 # Executar GridSearchCV para encontrar os melhores par metros
378 grid_search = GridSearchCV(forest_cls, param_grid, scoring="accuracy",
        cv=5)
379
380 # Ajustar o modelo com os dados de treino
381 grid_search.fit(X_train, y_train)
382
383 # Exibir os melhores par metros encontrados
384 print("Melhores par metros:", grid_search.best_params_)
385
386 # Avaliar a performance no conjunto de teste
387 grid_validation = grid_search.predict(X_test)
388 score = accuracy_score(y_test, grid_validation)
389 print(f"Score: {score}")
390
391 # Relat rio de Classifica  o
392 print("\nRelat rio de Classifica  o:")
393 print(classification_report(y_test, grid_validation, target_names=["
        Motor Ruim", "Motor Bom"]))
394
395 # Definindo par metros
396 param_dist = {'n_estimators': [10, 50, 100], 'max_features': [2, 4, 6,
        8]}
397
398 # Executando RandomizedSearchCV
399 randomized_search = RandomizedSearchCV(forest_cls, param_distributions=
        param_dist, n_iter=5, cv=3, random_state=42)
400 randomized_search.fit(X_train, y_train)
401 #
402 # Exibir os melhores par metros encontrados
403 print("Melhores par metros:", randomized_search.best_params_)
404
405 """#Deep Learning"""
406
407 # 1. Cria  o do Modelo
408 model = tf.keras.Sequential([
409     tf.keras.layers.Dense(128, activation='relu', input_shape=(X_train.
            shape[1],)),
410     tf.keras.layers.Dense(64, activation='relu'),
411     tf.keras.layers.Dense(32, activation='relu'),
412     tf.keras.layers.Dense(1, activation='sigmoid')  # Sa da bin ria
```

```
413 ])
414
415 # 2. Compila  o do Modelo
416 model.compile(optimizer='adam',
417               loss='binary_crossentropy',  # Como estamos trabalhando
                      com uma tarefa bin ria
418               metrics=['accuracy'])
419
420 # 3. Treinamento do Modelo
421 model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split
      =0.3)
422
423 # 4. Avalia  o e Previs o
424 # Previs o dos r tulos (classe bin ria)
425 validation = (model.predict(X_test) > 0.5).astype("int32")
426
427 # C lculo da acur cia
428 score = np.sum(validation.flatten() == y_test) / len(y_test)
429 print(f"Score: {score:.4f}")
430
431 # Calculando a matriz de confus o
432 cm = confusion_matrix(y_test, validation)
433
434 # Imprimindo a matriz de confus o
435 print("Matriz de Confus o:")
436 print(cm)
437
438 # Separando os quadrantes
439 tn, fp, fn, tp = cm.ravel()
440
441 # Imprimindo os valores dos quadrantes
442 print(f"Verdadeiros Positivos (TP): {tp}")
443 print(f"Verdadeiros Negativos (TN): {tn}")
444 print(f"Falsos Positivos (FP): {fp}")
445 print(f"Falsos Negativos (FN): {fn}")
446
447 # Relat rio de Classifica  o
448 print("\nRelat rio de Classifica  o:")
449 print(classification_report(y_test, validation, target_names=["Motor
      Ruim", "Motor Bom"]))
450
451 # Previs o das probabilidades
452 probabilities = model.predict(X_test)
453
454 # Exibindo as probabilidades das 10 primeiras amostras
455 for i in range(10):
```

```
456     prob_class_1 = probabilities[i, 0]  # Probabilidade da classe 1 (
            Motor Bom)
457     print(f"Amostra {i}: Probabilidade de ser 'Motor Bom': {prob_class_1
            * 100:.2f}%")
458
459 def build_model(hp):
460     model = tf.keras.Sequential()
461     model.add(Input(shape=(X_train.shape[1],)))
462
463     # Configura  o otimizada das camadas densas
464     for i in range(hp.Int('num_layers', 2, 6)):
465         model.add(Dense(units=hp.Int(f'units_{i}', min_value=32,
                max_value=512, step=32),
466                         activation='relu'))
467         model.add(Dropout(rate=hp.Float(f'dropout_{i}', min_value=0.2,
                max_value=0.5, step=0.1)))
468         model.add(BatchNormalization())
469
470     model.add(Dense(1, activation='sigmoid'))
471
472     # Otimiza  o da taxa de aprendizado
473     model.compile(optimizer=Adam(learning_rate=hp.Float('lr', 1e-5, 1e
            -2, sampling='log')),
474                   loss='binary_crossentropy',
475                   metrics=['accuracy'])
476     return model
477
478 # Configura  o do Hyperparameter Tuner
479 tuner = kt.RandomSearch(
480     build_model,
481     objective='val_accuracy',
482     max_trials=20,
483     executions_per_trial=2,
484     directory='my_dir',
485     project_name='hyperparam_opt')
486
487 early_stopping = EarlyStopping(monitor='val_loss', patience=10,
        restore_best_weights=True)
488 reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience
        =5, min_lr=1e-6)
489
490 # Otimiza  o com early stopping e redu  o da taxa de aprendizado
491 tuner.search(X_train, y_train, epochs=100, validation_split=0.2,
492              callbacks=[early_stopping, reduce_lr])
493
494 best_model = tuner.get_best_models(num_models=1)[0]
495
```

```
496 # Treinamento do melhor modelo encontrado
497 history = best_model.fit(X_train, y_train, epochs=100, validation_split
       =0.2,
498                           callbacks=[early_stopping, reduce_lr])
499
500 """#Salvando Modelo de IA"""
501
502 from keras_tuner import RandomSearch
503 best_model = tuner.get_best_models(num_models=1)[0]
504 best_model.save('melhor_modelo.h5')
505
506 best_model.save('melhor_modelo.keras')
507
508 !ls
509
510 !mv melhor_modelo.h5 /content/drive/MyDrive/TCC/
511
512 import joblib
513
514 # Exportar o modelo para um arquivo
515 joblib.dump(log_reg, 'logistic_model.pkl')
516
517 !mv logistic_model.pkl /content/drive/MyDrive/TCC/
518
519 """#Criando modelo do veiculo"""
520
521 melhor_modelo = tf.keras.models.load_model('/content/drive/MyDrive/TCC/
       melhor_modelo.h5')
522
523 validation = (melhor_modelo.predict(X_test) > 0.5).astype("int32")
524
525 # C lculo da acur cia
526 score = np.sum(validation.flatten() == y_test) / len(y_test)
527 print(f"Score: {score:.4f}")
528
529 # Calculando a matriz de confus o
530 cm = confusion_matrix(y_test, validation)
531
532 # Imprimindo a matriz de confus o
533 print("Matriz de Confus o:")
534 print(cm)
535
536 # Separando os quadrantes
537 tn, fp, fn, tp = cm.ravel()
538
539 # Imprimindo os valores dos quadrantes
540 print(f"Verdadeiros Positivos (TP): {tp}")
```

```
541 print(f"Verdadeiros Negativos (TN): {tn}")
542 print(f"Falsos Positivos (FP): {fp}")
543 print(f"Falsos Negativos (FN): {fn}")
544
545 # Relat rio de Classifica  o
546 print("\nRelat rio de Classifica  o:")
547 print(classification_report(y_test, validation, target_names=["Motor
    Ruim", "Motor Bom"]))
548
549 # Previs o das probabilidades
550 probabilities = melhor_modelo.predict(X_test)
551
552 # Exibindo as probabilidades das 10 primeiras amostras
553 for i in range(10):
554     prob_class_1 = probabilities[i, 0]  # Probabilidade da classe 1 (
        Motor Bom)
555     print(f"Amostra {i}: Probabilidade de ser 'Motor Bom': {prob_class_1
        * 100:.2f}%")
556
557 # Reverter a normaliza  o
558 engine_data_original = (X_test * scales) + means
559 #criando rotulos das colunas
560 colunas = ['Engine rpm', 'Lub oil pressure', 'Fuel pressure', 'Coolant
    pressure',
561            'lub oil temp', 'Coolant temp', 'Coolant Efficiency', 'Oil
               Efficiency']
562 #Criando df
563 df_test = pd.DataFrame(engine_data_original, columns=colunas)
564
565 # Adicionando a coluna y_test
566 df_test['Engine Condition'] = y_test.values  # ou diretamente y_test se
    j  for uma Series
567
568 print(df_test.head)
569
570 probabilidades = melhor_modelo.predict(X_test)
571
572 # Exibir as probabilidades das primeiras amostras
573 for i in range(10):  # Exibir para as primeiras 10 amostras
574     prob_classe_1 = probabilidades[i, 0]  # Probabilidade da classe 1 (
        Motor Bom)
575     print(f"Amostra {i}: Probabilidade de ser 'Motor Bom': {
        prob_classe_1 * 100:.2f}%")
576
577 # Criar um DataFrame com as probabilidades
578 resultados = pd.DataFrame({
```

```
579     'Probabilidade': probabilidades.flatten()  # Flatten para garantir
            que seja uma s rie unidimensional
580 })
581
582 # Adicionar os dados originais ao DataFrame de resultados
583 resultados = pd.concat([df_test, resultados], axis=1)
584 print(resultados.head())
585
586 new_order = ['Engine rpm', 'Lub oil pressure', 'Fuel pressure', 'Coolant
        pressure', 'lub oil temp', 'Coolant temp', 'Coolant Efficiency', '
    Oil Efficiency','Engine Condition', 'Probabilidade']
587 resultados = resultados[new_order]
588 print(resultados.head())
589
590 """## Motores Excelentes"""
591
592 motores_excelentes = resultados[(resultados['Probabilidade'] > 0.90) & (
    resultados['Engine Condition'] == 1)]
593 print(motores_excelentes)
594
595 media_motores_excelentes = motores_excelentes[['Engine rpm', 'Lub oil
    pressure', 'Fuel pressure',
596                                                 'Coolant pressure', 'lub
                                                        oil temp', 'Coolant
                                                        temp',
597                                                 'Coolant Efficiency', '
                                                        Oil Efficiency']].mean
                                                        ()
598
599 # Exibir a m dia de cada vari vel
600 print("M dia das vari veis dos motores excelentes:")
601 print(media_motores_excelentes)
602
603 """## Motores com indicios de problemas"""
604
605 motores_ruins = resultados[(resultados['Probabilidade'] < 0.40) & (
    resultados['Engine Condition'] == 0)]
606 print(motores_ruins)
607
608 media_motores_ruins = motores_ruins[['Engine rpm', 'Lub oil pressure', '
    Fuel pressure',
609                                      'Coolant pressure', 'lub oil temp',
                                             'Coolant temp',
610                                      'Coolant Efficiency', 'Oil
                                         Efficiency']].mean()
611
612 # Exibir a m dia de cada vari vel
```

```
613 print("M dia das vari veis dos motores ruins:")
614 print(media_motores_ruins)
615
616 """## Desgaste"""
617
618 desgaste = media_motores_excelentes - media_motores_ruins
619 print(desgaste)
620
621 desgaste_10km = desgaste/20000
622 print(desgaste_10km)
623
624 """# MODELO DO VEICULO"""
625
626 max_km = 00000
627 km_interasion = 10
628 carro = media_motores_excelentes
629 km_accumulated = 0
630
631 wear_parameters = {
632     'Engine_rpm': (0.90, 1.1),  # Exemplo: Varia  o de  10 %
633     'lub_oil_pressure': (0.90, 1.1),
634     'Fuel_pressure': (0.90, 1.1),
635     'coolant_pressure': (0.90, 1.1),
636     'lub_oil_temp': (0.90, 1.1),
637     'coolant_temp': (0.90, 1.1),
638     'coolant_efficiency': (0.90, 1.1),
639     'oil_efficiency': (0.90, 1.1),
640 }
641
642 while km_accumulated <= max_km:
643     carro['Engine rpm'] += -desgaste_10km['Engine rpm'] * np.random.
        uniform(*wear_parameters['Engine_rpm'])
644     carro['Lub oil pressure'] += desgaste_10km['Lub oil pressure'] * np.
        random.uniform(*wear_parameters['lub_oil_pressure'])
645     carro['Fuel pressure'] += desgaste_10km['Fuel pressure'] * np.random
        .uniform(*wear_parameters['Fuel_pressure'])  # Corrigido: use '
        Fuel_pressure'
646     carro['Coolant pressure'] += -desgaste_10km['Coolant pressure'] * np
        .random.uniform(*wear_parameters['coolant_pressure'])
647     carro['lub oil temp'] += -desgaste_10km['lub oil temp'] * np.random.
        uniform(*wear_parameters['lub_oil_temp'])
648     carro['Coolant temp'] += -desgaste_10km['Coolant temp'] * np.random.
        uniform(*wear_parameters['coolant_temp'])
649     carro['Coolant Efficiency'] = (1 / carro['Engine rpm']) * carro['
        Coolant temp']
650     carro['Oil Efficiency'] = 1 / (carro['Engine rpm'] * carro['lub oil
        temp'])
```

```
651
652     carro_array = carro.to_numpy()
653     carro_array = carro_array.reshape(1, -1)
654     carro_array_normaliz = (carro_array - means) / scales
655
656     if km_accumulated % 100 == 0:
657         print(f"Quilometragem acumulada: {km_accumulated} km")
658         #print(carro)
659         probabilities = melhor_modelo.predict(carro_array_normaliz)
660         print(f"Amostra {{km_accumulated}}: Probabilidade de ser 'Motor
               Bom': {probabilities * 100}%")
661     km_accumulated += km_interasion
662
663 print(f"Fim da Simula  o!")
```

Listing 5.1 – Código Python para Criação do Modelo

Anexo B - Código MATLAB para Simular Veículos

O código abaixo foi utilizado para enviar amostas de veículos do MATLAB para o microcontrolador.

```
1  clear all
2  clc
3
4  % Carregar um arquivo CSV
5  df_test = readtable('df_test.csv');
6  % Exibir as primeiras linhas da tabela
7  head(df_test)
8
9  % Carregar o arquivo .mat
10 load('param_norm.mat')
11 % Agora, 'means' e 'scales' estar o dispon veis no workspace
12 disp(means)
13 disp(scales)
14
15 % Conecte    porta serial associada ao ESP32 (substitua 'COMX' pela
      porta correta)
16 device = serialport("COM6", 115200); % Ajuste para a porta correta
17
18 % Loop para enviar dados a cada 5 segundos
19 while true
20     try
21         % Selecionar uma linha aleat ria
22         rowIdx = randi(height(df_test)); % Gera um  ndice  aleat rio
23         selectedRow = df_test(rowIdx, :); % Seleciona a linha
24
```

```
25          % Remover colunas 'Var1' e 'engine_condition'
26          selectedRow(:, {'Var1', 'EngineCondition'}) = [];
27
28          % Normalizar os dados
29          normalizedData = (selectedRow{1, :} - means) ./ scales;
30
31          % Enviar os dados normalizados via porta serial
32          write(device, join(string(normalizedData), ' '), "char");
33
34          % Aguarde 5 segundos
35          pause(5);
36          % Verifica se h  bytes dispon veis para leitura
37          if device.NumBytesAvailable > 0
38              % Leia a resposta do ESP32
39              data = read(device, device.NumBytesAvailable, "char");
40
41              % Exibe a resposta no console MATLAB
42              disp("Resposta do ESP32:");
43              disp(data);
44          else
45              disp("Nenhum dado foi recebido.");
46          end
47
48      catch ME
49      % Captura e exibe erros
50          disp("Ocorreu um erro:");
51          disp(ME.message);
52      end
53  end
```

Listing 5.2 – Código MATLAB para Modelo de Carro

Anexo C - Código C++ para Implementar IA em ESP32

O código abaixo foi utilizado para realizar a integração do modelo de IA com o ESP32, utilizando LEDs e display LCD para indicar o status do motor e enviar informações via Bluetooth.

```
1  #include <Arduino.h>
2  #include <math.h>
3  #include "tensorflow/lite/experimental/micro/kernels/all_ops_resolver.h"
4  #include "tensorflow/lite/experimental/micro/micro_error_reporter.h"
5  #include "tensorflow/lite/experimental/micro/micro_interpreter.h"
6  #include "melhor_modelo.h"
7
8  //bibliotecas que eu adicionei
9  #include <Wire.h>
```

```
10 #include <LiquidCrystal_I2C.h>
11 #include "BluetoothSerial.h"
12
13 #define LED_VERMELHO 13  // Pino do LED vermelho
14 #define LED_BRANCO 12    // Pino do LED branco
15
16 // Create a memory pool for the nodes in the network
17 constexpr int tensor_pool_size = 10 * 1024;
18 uint8_t tensor_pool[tensor_pool_size];
19
20 // Define the model to be used
21 const tflite::Model* melhor_modelo;
22
23 // Define the interpreter
24 tflite::MicroInterpreter* interpreter;
25
26 // Input/Output nodes for the network
27 TfLiteTensor* input;
28 TfLiteTensor* output;
29
30 // Inicializa o Bluetooth Serial
31 BluetoothSerial SerialBT;
32
33 // Configura  o do display LCD (ajuste o endere o se necess rio, 0
      x27 ou 0x3F)
34 LiquidCrystal_I2C lcd(0x27, 16, 2);
35
36
37 // Set up the ESP32's environment.
38 void setup() {
39   // Start serial at 115200 baud
40   Serial.begin(115200);
41   Serial.println("Iniciando configura  o...");
42
43     // Inicializa o Bluetooth
44     SerialBT.begin("ESP32_BT"); // Nome do dispositivo Bluetooth
45     Serial.println("Bluetooth iniciado, esperando conex o...");
46
47     // Configura os pinos dos LEDs como sa da
48     pinMode(LED_VERMELHO, OUTPUT);
49     pinMode(LED_BRANCO, OUTPUT);
50
51     // Configura  o inicial dos LEDs
52     digitalWrite(LED_VERMELHO, LOW);
53     digitalWrite(LED_BRANCO, LOW);
54
55   // Load the sample sine model
```

```
56    Serial.println("Carregando Tensorflow model....");
57    melhor_modelo = tflite::GetModel(model);
58    Serial.println("Modelo de IA Carregado!");
59
60    // Define ops resolver and error reporting
61    static tflite::ops::micro::AllOpsResolver resolver;
62
63    static tflite::ErrorReporter* error_reporter;
64    static tflite::MicroErrorReporter micro_error;
65    error_reporter = &micro_error;
66
67    // Instantiate the interpreter
68    static tflite::MicroInterpreter static_interpreter(
69      melhor_modelo, resolver, tensor_pool, tensor_pool_size,
          error_reporter
70    );
71
72    interpreter = &static_interpreter;
73
74    // Allocate the the model's tensors in the memory pool that was
         created.
75    Serial.println("Modelo carregado e tensores alocado");
76    if(interpreter->AllocateTensors() != kTfLiteOk) {
77      Serial.println("ocoreu um erro ao alocar os tensore na memoria...
          ooof");
78      return;
79    }
80
81    // Define input and output nodes
82    input = interpreter->input(0);
83    output = interpreter->output(0);
84    Serial.println("Starting inferences... Input a number! ");
85
86    // Inicializa o display LCD
87    lcd.begin();
88      lcd.backlight();
89      lcd.setCursor(0, 0);
90      lcd.print("Esperando...");
91
92  }
93
94  // Logic loop for taking user input and outputting the sine
95  void loop() {
96      // Verifica se h  dados dispon veis via Bluetooth
97      if (SerialBT.available()) {
98        lcd.clear(); // Limpa o display antes de mostrar nova mensagem
```

```
99        String receivedMessage = ""; // Vari vel para armazenar a
              mensagem recebida
100
101       // L   os dados dispon veis do Bluetooth
102       while (SerialBT.available()) {
103           char incomingChar = SerialBT.read();
104           receivedMessage += incomingChar; // Concatena os caracteres na
                  string
105       }
106
107       // Imprime os dados recebidos para verifica  o
108       Serial.println("Dados recebidos: " + receivedMessage);
109
110       // Converter a string recebida em um array de floats
111       float input_data[8];
112       int index = 0;
113       char* token = strtok((char*)receivedMessage.c_str(), " ");
114       while (token != nullptr && index < 8) {
115           input_data[index] = atof(token);  // Converte string para
                  float
116           token = strtok(nullptr, " ");
117           index++;
118     }
119
120       // Exibir o array de floats para depura  o
121       Serial.println("Dados convertidos para float:");
122       for (int i = 0; i < 8; i++) {
123           Serial.println(input_data[i]);
124       }
125
126       // Passar os dados para o modelo
127       float* input = interpreter->input(0)->data.f;
128       for (int i = 0; i < 8; i++) {
129           input[i] = input_data[i];  // Define os dados de entrada do
                  modelo
130       }
131
132       // Run inference on the input data
133       if(interpreter->Invoke() != kTfLiteOk) {
134         Serial.println("Erro ao executar o modelo!");
135         return;
136       }
137
138       // Ler a sa da do modelo
139       float* output = interpreter->output(0)->data.f;
140       Serial.print("Resultado: ");
141       Serial.println(output[0]);
```

```
142
143       // Enviar a resposta ao MATLAB
144       SerialBT.println("A inferencia foi concluida. Resultado: " +
              String(output[0]));
145
146     // Exibir o resultado no display LCD
147       lcd.clear(); // Limpa o display antes de mostrar o resultado
148       lcd.setCursor(0, 0); // Posiciona o cursor no in cio da primeira
              linha
149       lcd.print("Prob motor estar"); // Escreve "Resultado:"
150     lcd.setCursor(0, 1); // Posiciona o cursor no in cio da primeira
           linha
151       lcd.print("bom: "); // Escreve "Resultado:"
152       lcd.setCursor(6, 1); // Posiciona o cursor no in cio da segunda
              linha
153       lcd.print(output[0]); // Exibe o valor do resultado da infer ncia
154
155       if (output[0] < 0.6) {
156           // Resultado abaixo de 60%, acende o LED vermelho
157           digitalWrite(LED_VERMELHO, HIGH);
158           digitalWrite(LED_BRANCO, LOW); // Desliga o LED branco
159       } else if (output[0] > 0.8) {
160           // Resultado acima de 80%, acende o LED branco
161           digitalWrite(LED_BRANCO, HIGH);
162           digitalWrite(LED_VERMELHO, LOW); // Desliga o LED vermelho
163       } else {
164           // Resultado entre 60% e 80%, desliga ambos os LEDs
165           digitalWrite(LED_VERMELHO, LOW);
166           digitalWrite(LED_BRANCO, LOW);
167       }
168   }
169 }
```

Listing 5.3 – Código para implementar IA no ESP32