**UNIVERSIDADE FEDERAL DE JUIZ DE FORA**
**FACULDADE DE ENGENHARIA**
**ENGENHARIA ELÉTRICA - ROBÓTICA E AUTOMAÇÃO INDUSTRIAL**

**Allan Braga Corrêa**

# Inteligência Artificial em Sistemas Embarcados

Juiz de Fora
2025

**Allan Braga Corrêa**

# Inteligência Artificial em Sistemas Embarcados

Trabalho de Conclusão de Curso (TCC) apresentado à Engenharia Elétrica - Robótica e Automação Industrial da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Leandro Rodrigues Manso Silva

Juiz de Fora

2025

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Braga, Allan.
Inteligência Artificial em Sistemas Embarcados / Allan Braga Corrêa. – 2025.
36 f. : il.

Orientador: Leandro Rodrigues Manso Silva
Trabalho de Conclusão de Curso – Universidade Federal de Juiz de Fora, Faculdade de engenharia. Engenharia Elétrica - Robótica e Automação Industrial , 2025.

1. Inteligência Artificial (AI). 2. Sistemas Embarcados. 3. Aprendizado de Máquina (ML). 4. ESP32. 5. TensorFlow Lite (LiteRT). I. Rodrigues, Leandro.

**Allan Braga Corrêa**

**Inteligência Artificial em Sistemas Embarcados**

Trabalho de Conclusão de Curso (TCC) apresentado à Engenharia Elétrica - Robótica e Automação Industrial da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Engenheiro Eletricista.

Aprovada em (dia) de março de 2025

BANCA EXAMINADORA

---

Prof. Dr. Leandro Rodrigues Manso Silva - Orientador
Universidade Federal de Juiz de Fora (UFJF)

---

Prof. Dr. Cristiano Gomes Casagrande
Universidade Federal de Juiz de Fora (UFJF)

---

Prof. Dr. Kennedy Martins Pedroso
Universidade Federal de Juiz de Fora (UFJF)

## AGRADECIMENTOS

Primeiramente, quero agradecer imensamente à minha família e à minha namorada/noiva, Mariana (Bãobão), que foram alicerces fundamentais ao longo desta jornada desafiadora, com destaque para minha mãe, Sumáia, a qual foi minha maior base.

Em especial, ao meu pai, Júlio Antônio, por me ensinar a amar o Botafogo desde criança, pelo incentivo constante, apoio incondicional e por todos os esforços em me proporcionar as melhores condições possíveis para que eu pudesse me dedicar aos estudos. Essa conquista se deve completamente ao senhor, e essas singelas palavras são uma pequena maneira de retribuir tudo o que o senhor sempre fez por mim.

À minha irmã, Ludimilla, cuja paixão pelos estudos e dedicação ao aprendizado sempre foram uma inspiração para mim, sua presença, apoio incondicional e exemplo de perseverança foram determinantes para o meu crescimento pessoal e acadêmico, você sempre será o meu mais elevado exemplo e fonte de inspiração para o conhecimento.

Agradeço também ao professor orientador Leandro Manso por sua orientação e apoio ao longo do desenvolvimento deste trabalho. Sou profundamente grato por suas valiosas contribuições, tanto neste trabalho quanto nas disciplinas que tive o privilégio de cursar sob sua orientação (Eletrônica Digital e Software Embarcado).

Aos professores do curso de Engenharia, que contribuíram enormemente em minha formação com seus ensinamentos e apoio ao longo do curso. Jamais poderia me esquecer dos excelentíssimos professores do Instituo de Ciências Exatas, principalmente os do Departamento de Matemática, estendo minha gratidão a todos vocês.

Não poderia deixar de mencionar as amizades construídas durante a graduação, principalmente ao Maicon, Iury, Gabriel e Camila. Vocês foram parte essencial desta jornada, oferecendo apoio, companheirismo e motivação nas horas mais díficeis.

Por fim, dedico este parágrafo exclusivamente ao meu grande amigo Raul. Nossa amizade, de mais de 10 anos, construída na adolescência, é algo que permanecerá para o resto de nossas vidas. Considero-o um irmão, alguém que sempre me apoiou em minhas decisões e sempre esteve ao meu lado em todos os momentos. Deixo aqui o meu mais sincero obrigado.

“Tudo aquilo que o homem ignora, não existe para ele. Por isso o universo de cada um, se resume no tamanho de seu saber.”

Albert Einstein

**RESUMO**

O presente estudo possui como centralidade uma pesquisa que inclui uma análise teórica, didática e prática entre a integração de Inteligência Artificial (AI) e Sistemas Embarcados, com foco na aplicação do framework TensorFlow Lite (LiteRT) no microcontrolador ESP32. A motivação surgiu devido à crescente demanda por soluções inteligentes e eficientes em um mundo cada vez mais tecnológico, onde a necessidade de atingir melhores resultados em dispositivos com recursos limitados se torna maior e mais relevante, especialmente em Internet das Coisas (IoT) e automação. O principal objetivo é fornecer um material didático de fácil compreensão, que possa contribuir com estudantes, pesquisadores, profissionais da área e entusiastas do assunto, abordando conceitos primordiais e apresentando um exemplo prático da previsão de uma onda senoidal utilizando redes neurais artificiais. A metodologia constitui-se na adaptação de um projeto já desenvolvido, contendo a criação, treinamento e otimização de um modelo de Aprendizado de Máquina (ML) no TensorFlow, seguido da conversão desse modelo para o formato TensorFlow Lite e a implementação no microcontrolador ESP32. Os resultados foram promissores, indicando uma alta precisão do modelo desenvolvido e demonstrando a eficácia da solução proposta. Conclui-se que o TensorFlow Lite no ESP32, além de permitir a execução de modelos de Aprendizado de Máquina mais complexos, o mesmo apresenta uma excelente eficácia e mostra-se promissor para tais desenvolvimentos de soluções em dispositivos embarcados.

**Palavras-chave:** Inteligência Artificial (AI); Sistemas Embarcados; Aprendizado de Máquina (ML); ESP32; TensorFlow Lite (LiteRT).

**ABSTRACT**

The present study focuses on research that includes a theoretical, didactic, and practical analysis of the integration between Artificial Intelligence (AI) and Embedded Systems, with an emphasis on applying the TensorFlow Lite (LiteRT) framework to the ESP32 microcontroller. The motivation arose from the growing demand for intelligent and efficient solutions in an increasingly technological world, where the need to achieve better results in devices with limited resources becomes more significant and relevant, especially in Internet of Things (IoT) and automation applications. The main objective is to provide an easy-to-understand didactic material that can contribute to students, researchers, professionals in the field, and enthusiasts of the subject, addressing fundamental concepts and presenting a practical example of sine wave prediction using artificial neural networks. The methodology consists of adapting an already developed project, including the creation, training, and optimization of a Machine Learning (ML) model in TensorFlow, followed by the conversion of this model to the TensorFlow Lite format and its implementation on the ESP32 microcontroller. The results were promising, indicating high accuracy of the developed model and demonstrating the effectiveness of the proposed solution. It is concluded that TensorFlow Lite on ESP32 not only allows the execution of more complex Machine Learning models but also demonstrates excellent efficiency and proves to be a promising approach for developing solutions in embedded devices.

**Keywords:** Artificial Intelligence (AI); Embedded Systems; Machine Learning (ML); ESP32; TensorFlow Lite (LiteRT).

## LISTA DE ILUSTRAÇÕES

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| AI | Inteligência Artificial, do Inglês *Artificial Intelligence* |
| AIoT | Inteligência Artifical das Coisas |
| CPU | Unidade Central de Processamento, do Inglês *Central Processing Unit* |
| FPGA | Arranjo de Portas Programáveis em Campo, do Inglês *Field Programmable Gate Array* |
| GPU | Unidade de Processamento Gráfico, do Inglês *Graphics Processing Unit* |
| IoT | Internet das Coisas, do Inglês *Internet of Things* |
| KB | Quilobytes, do Inglês *kilobytes* |
| LiteRT | Tempo de Execução Leve, do Inglês *Lite Runtime* |
| MAE | Erro Absoluto Médio, do Inglês *Mean Absolute Error* |
| ML | Aprendizado de Máquina, do Inglês *Machine Learning* |
| NPU | Unidade de Processamento Neural, do Inglês *Neural Processing Unit* |
| RAM | Memória de Acesso Aleatório, do Inglês *Random Acess Memory* |
| RMSE | Erro Quadrático Médio, do Inglês *Root Mean Squared Error* |
| TinyML | Aprendizado de Máquina Minúsculo, do Inglês *Tiny Machine Learning* |
| TPU | Unidade de Processamento de Tensor, do Inglês *Tensor Processing Unit* |

## SUMÁRIO

# 1 INTRODUÇÃO

## 1.1 Contextualização

Diante de tamanho avanço tecnológico global, no que diz respeito à Inteligência Artificial (do inglês, *Artificial Intelligence* (AI)) tem-se uma revolução em termos de inteligência e eficiência, permitindo que máquinas atuem de forma autônoma, com qualidade e produtividade. Dentre as áreas de influência da AI ressalta-se a de sistemas embarcados, com a junção de hardware, software e interfaceamento de dispositivos integrados na execução de determinadas tarefas. Este sistema se encontra cada vez mais presente no dia a dia das pessoas, fornecendo serviços essenciais e permitindo que agora dispositivos com menores demandas de consumo de energia e com recursos limitados executem tarefas que anteriormente eram realizadas apenas por máquinas com maior poder computacional.

No passado, modelos de AI, especialmente Aprendizado de Máquina (do inglês, *Machine Learning* (ML)), eram utilizados apenas em ambientes computacionais altamente robustos, como por exemplo: data centers e supercomputadores. Contudo, com o crescente poder de processamento, sendo cada vez maior com o passar do tempo, o mundo em que vivemos a cada dia se torna mais micro conectado, com isso, têm-se o avanço de hardwares menores, otimização de algoritmos e consequentemente criação de novos frameworks, que é o caso do TensorFlow Lite (TF Lite) e Aprendizado de Máquina Minúsculo (do inglês, *Tiny Machine Learning* (TinyML)), que fazem a interconexão entre AI e sistemas embarcados. Com o crescimento acelerado da quantidade de dados coletados por dispositivos móveis e de Internet das Coisas (do inglês, *Internet of Things* (IoT)), faz com que haja um destaque em computação de borda, pois essa necessita distribuição em que parte da computação ocorra perto de onde os dados são criados, tendo um processamento local e diminuindo os servidores remotos, mostrando ser fundamental ML em sistemas embarcados (WARDEN, P.; SITUNAYAKE, D., 2020).

As aplicações de AI vem otimizando vários serviços e abrangendo diversas áreas, como é o caso da automação residencial, que utiliza dispositivos Internet das Coisas (do inglês, Internet of Things (IoT)) para controlarem e automatizarem vários dispositivos da casa, como iluminação, janelas, câmeras e muito mais (SILVA et al., 2022); saúde, com monitoramento de saúde, como pulseiras inteligentes e relógios, permitindo aos pacientes acompanharem diversos parâmetros da saúde, como frequência cardíaca e pressão arterial, tudo de maneira contínua e autônoma (OLIVEIRA & SANTOS, 2021); agrícultura, como a transformação de aparelhos tradicionais em “pulverizadores inteligentes”, essa foi uma proposta criada por uma empresa brasileira chamada SAVEFARM., a qual ao deixar esses aparelhos inteligentes faz com que os mesmos sejam capazes de otimizar a aplicação de agroquímicos (SAVEFARM., 2023). Além disso, a indústria automotiva se beneficia também dessas tecnologias, um exemplo é a direção autonôma, que usa ML para processar

dados de sensores e câmeras, os quais identificam possíveis riscos na pista que o veículo está percorrendo e evita que ocorra um acidente, dessa maneira aumenta a segurança do(s) passageiro(s) (GOMES & PEREIRA, 2020).

Conforme apontado por Vijay Madisetti e Arshdeep Bahga em *MADISETTI, V.; BAHGA, A. Embedded Systems: Fundamentals and Applications, 2017*, "os sistemas embarcados inteligentes representam uma convergência de tecnologia que exige um equilíbrio delicado entre funcionalidade avançada e as restrições inerentes ao hardware", isso nos mostra que mesmo com grandes avanços ainda existem barreiras a serem quebradas, desafios que a cada dia vem sendo superados. (MADISSETI & BAHGA, 2017)

Um framework/biblioteca criado de suma importância para desenvolvimento dessa tecnologia é o TensorFlow Lite, o qual faz essa união de ML com dispositivos de pequeno porte, ou seja, aplicação de ML em microcontroladores e outros dispositivos usando alguns kilobytes de memória. Assim como TensorFlow Lite, o TinyML tem a mesma função, que é executar modelos de ML em microcontroladores.

Por mais que o TensorFlow Lite e o TinyML pareçam ser a mesma coisa, na realidade são duas abordagens distintas. O TinyML tem um foco maior na execução de modelos de ML em dispositivos minúsculos, menores que os executados pelo TensorFlow Lite. O TinyML representa o campo de técnicas de otimização, compactação de modelos e algoritmos para permitir AI em hardwares muito restritos (TINYML FOUNDATION., 2023). O TensorFlow Lite pode-se dizer que é uma biblioteca mais geral e aplicável a gama de dispositivos com recursos limitados muito maior, enquanto o TinyML é voltado para dispositivos onde consumo e tamanho são críticos, como sensores inteligentes e wearables (WARDEN, P.; SITUNAYAKE, D., 2020).

## 1.2 Motivação

A AI ganha cada vez mais holofotes na sociedade, impactando cada vez mais a economia e os avanços tecnológicos nas mais variadas áreas. A junção de AI juntamente com sistemas embarcados não é diferente, ela irá permitir novas soluções, sendo mais baratas e de consumo energético baixo, o que no caso de AI baseada em nuvem não é possível. Diante disso, novas perspectivas tecnológicas são alcançaveis. O presente estudo tem como motivação a inovação e revolução que a junção das tecnologias poderão causar no futuro, proporcionando novos empregos, pesquisas e desenvolvimentos.

A justificativa para a escolha do TensorFlow Lite se deve à sua capacidade de conversão de modelos pré-treinados do Tensor Flow em formatos especiais otimizados para velocidade e armazenamento. Modelos com o Lite são muito leves de maneira a obter uma baixa latência em dispositivos de ponta e dispositivos incorporados, como por exemplo ambiente Mobile e microcontroladores, respectivamente.

## 1.3 Objetivos

O presente trabalho objetiva proporcionar uma abordagem mais teórica e didática, fornecendo um arcabouço relevante para estudantes, profissionais da área e entusiastas no assunto de AI em sistemas embarcados. tendo um foco maior em ML, utilizando a abordagem TensorFlow Lite, mas citando outra relevante nesse contexto, TinyML. O exemplo prático de implementação que será abordado mais adiante usa um microcontrolador, o ESP32, destacando sua importância nessa tecnologia.

## 1.4 Divisão do Trabalho

Este trabalho foi dividido em cinco capítulos, os quais se interagem entre si, o intuito deles é fornecer uma estrutura clara, coesa e que deixe o fluxo de ideias da proposta do trabalho de fácil compreensão, tornando o aprendizado mais acessível e valioso, oferecendo abordagens teóricas e práticas sobre o uso de AI em sistemas embarcados, como o uso do framework TensorFlow Lite. Diante dessa ideia, será apresentado a estrutura de forma minuciosa.

O primeiro capítulo, Introdução, foi dividido em quatro subcapítulos. O primeiro é a Contextualização, que apresenta uma introdução ao tema do trabalho, oferecendo uma visão geral da AI em sistemas embarcados, com foco em ML, destacando seus avanços, relevância atual e futura, além dos desafios.

Na parte que diz respeito a Motivação, há a justificativa da escolha do tema, importância das tecnologias e o impacto delas na atualidade.

Nos Objetivos, são apresentados o intuito geral do trabalho, especificando o público alvo e as tecnoligias utilizadas, exemplificadas por meio de um problema prático, com a finalidade de fornecer um material didático acadêmico.

O capítulo 2 aborda uma revisão sobre o TensorFlow Lite visa fornecer uma fundamentação teórica sobre essa framework, abordando a estrutura do software e também a história e funcionamento interno da biblioteca.

O capítulo 3 apresenta um tutorial de utilização do ESP32. Há um passo a passo para a implementação de ML nesse microcontrolador, descrevendo de forma eficaz e detalhada de como se utiliza o TensorFlow Lite no ESP32. Antes desse passo a passo, há uma breve teoria sobre o ESP32 para posteriormente, entrar nas abordagens supracitadas.

O capítulo 4 traz um exemplo de utilização, será apresentado uma aplicação prática, tendo o desenvolvimento e implementação de um modelo de ML utilizando o microcontrolador ESP32, onde terá todo o problema de forma detalhada, objetivos, mostrando cada etapa da construção do projeto e seus resultados. O intuito desse capítulo é ilustrar os conceitos discutidos no trabalho.

Por fim, o capítulo V traz a conclusão do trabalho, que é a parte final, onde é discutido sobre tudo que foi abordado anteriormente, refletindo os resultados das teorias e as contribuições. São mostradas as limitações da AI no microcontrolador ESP32 com o uso do TensorFlow Lite e sugestões para diminuir essas limitações, contribuindo para trabalhos futuros.

# 2 REVISÃO SOBRE O TENSORFLOW LITE

Este capítulo abordará uma revisão sobre o TensorFlow Lite, desde o seu surgimento até suas vantagens, detalhando os conceitos e estruturas desse software, apresentando seus principais aspectos e funções.

Antes de abordar diretamente o TensorFlow Lite, é imprescindível falar sobre o TensorFlow, o qual foi desenvolvido pelo Google e foi disponibilizado ao público em 2015. O TensorFlow é uma biblioteca de ML utilizada globalmente em grande escala, e uma característica notável é que sua plataforma é de código aberto (do Inglês, *open source*). Seus desenvolvedores aderiram um lema muito famoso que a grande maioria dos apaixonados por tecnologia, em especial ML, o conhecem: "Uma estrutura de Aprendizado de Máquina (ML) de código aberto para todos", tendo como principal linguagem de interface o Python (WARDEN, P.; SITUNAYAKE, D., 2020).

Criada com o intuito de ser utilizada em desktops e servidores Linux, Windows e macOS, o TensorFlow fornece muitas ferramentas e exemplos para que modelos de ML sejam executados, otimizados, treinados e implantados em modelos na nuvem, além de vários tutoriais disponíveis auxiliando quem deseja explorá-lo. Suas ações são executadas em processadores que são as Unidades Centrais de Processamento (do inglês, *Central Processing Unit* (CPU)), Unidades de Processamento Gráfico (do inglês, *Graphics Processing Unit* (GPU)) e também por Unidades de Processamento de Tensor (do inglês, *Tensor Processing Unit* (TPU)). As TPUs são circuitos integrados de aplicação específica projetados pelo Google para redes neurais. São otimizados para treinar modelos de Aprendizado Profundo (do inglês, *Deep Learning*) maiores e mais complexos. Entretanto, há limitações quando se trata de dispositivos móveis, como Android e iOS, os quais possuem menos Memória de Acesso Aleatório (do inglês, *Random-Acess Memory* (RAM)) e menor espaço de armazenamento (WARDEN, P.; SITUNAYAKE, D., 2020).

Diante dessa limitação que o TensorFlow não era capaz de atender, foi desenvolvido também pelo próprio Google, em 2017, o TensorFlow Lite, que atualmente sofreu uma alteração em seu nome, passando a ser chamado de Tempo de Execução Leve (do inglês, *Lite Runtime* (LiteRT)) (GOOGLE., 2024). A partir desse momento, no decorrer de todo o restante do trabalho, ele será referenciado somente como LiteRT.

O LiteRT é uma biblioteca que tem como propósito permitir a execução de modelos de redes neurais em dispositivos móveis, sendo eficiente e de fácil implementação. Os desenvolvedores do LiteRT para atender essa demanda foram modificando os recursos, tipos de dados e reduziram as operações do TensorFlow, para que fosse possível tal execução nos dispositivos com rescursos bem menores. Reduziram tamanho e a complexidade da estrutura, dispensando recursos que não haveria necessidade ou seriam poucos utilizados nessas plataformas, como por exemplo, o suporte ao treinamento de modelos. Também

não há suporte a tipos de dados os quais há no TensorFlow em LiteRT (WARDEN, P.; SITUNAYAKE, D., 2020).

Com essas retiradas, o LiteRT tornou-se mais compacto, facilitando seu encaixe para melhor adequação em aplicativos com restrições de tamanho, tornando-se dessa maneira ideal para esses casos. Além disso, ele tem bibliotecas otimizadas para determinadas CPUs e também um suporte de aceleração por meio de APIs - Interface de Programação de Aplicação (do Inglês, *Application Programming Interface)* de redes neurais do Android. As APIs tornam mais fáceis a integração entre os modelos de ML em sistemas embarcados, como por exemplo a conversão de modelo TensorFlow em LiteRT. Uma vantagem expressiva é a melhora no desempenho da inferência, alcançada através de caminhos para utilização de uma execução otimizada para dados menores e assim uma redução significativa em tamanhos de modelos (WARDEN, P.; SITUNAYAKE, D., 2020).

A equipe do Google, responsável pelo desenvolvimento do LiteRT viu o sucesso que foi o framework para dispositivos móveis, porém não atendia outros produtos dentro do próprio Google assim como produtos exteriores no mercado global, os quais também poderiam se beneficiar do ML para otimizar seus serviços. Vale ressaltar que o maior empecilho ainda era o tamanho do arquivo binário, mesmo sendo relativamente pequeno centenas de quilobytes (do inglês, *kilobytes* (KB)), o mesmo ainda precisava ser menor, ou seja, mais compacto (WARDEN, P.; SITUNAYAKE, D., 2020).

Como consequência, em 2018, os desenvolvedores do Google começaram a adaptar e experimentar o LiteRT exclusivamente para sistemas embarcados. Nesse processo o objetivo era reutilizar o máximo possível das tecnologias já bem estruturadas dos dispositivos móveis, de maneira a atender os requisitos dos ambientes embarcados. Primeiramente, a equipe teve em mente algo prático, de uso real, que foi o reconhecimento de uma palavra falada para ativar e acessar os serviços (WARDEN, P.; SITUNAYAKE, D., 2020).

Diante dessa ideia, de uma assistente virtual que faz reconhecimento de palavras, surgiu a Siri, em 2011, que foi introduzida no iPhone 4S, uma aquisição da Apple. Ela foi pioneira nessa tecnologia para smartphones e expandiu ainda mais sua capacidade, indo além do simples reconhecimento de palavras, podendo realizar inúmeras tarefas, como reproduzir músicas de vários aplicativos de mídia, programar eventos de calendário, lembretes e alarmes, traduzir idiomas, entre outras funções (MCDONOUGH, 2025). Com isso, outras empresas aderiram a essa ideia e, um pouco mais tarde, a Amazon desenvolveu sua própria assistente, a Alexa, lançada em 2014 junto com a Echo, que basicamente possui as mesmas funções da Siri (KITAMURA, 2023).

Outras aplicações práticas que podem ser citadas em que o LiteRT está sendo aplicado são: sistemas de segurança e automação industrial. Uma aplicação importante do uso dessa tecnologia é em dispositivos médicos portáteis, como monitores de glicose e oxímetros, no qual o modelo de ML permite diagnosticar com rapidez e precisão sem a

necessidade de conexão em nuvem (CHEN, J et al., 2020).

Um outro exemplo do uso do LiteRT é em veículos autônomos e sistemas de assistência ao motorista. Nesse caso, a alta eficiência energética e a baixa latência são críticas, diante desse cenário, o LiteRT entra com a função de permitir a execução de modelos de visão computacional direto e exclusivo nos sistemas embarcados desses veículos (GRIGORESCU, S. et al., 2020).

Essa tecnologia também é amplamente usada em IoT, onde ela processa informações em tempo real, fazendo com que a resposta do sistema seja melhorada devido à diminuição da dependência com servidores que são remotos (LANE, N. D., 2015).

O LiteRT para microcontroladores tem como uma de suas principais propostas tornar o uso de ML de fácil aplicação e entendimento para os mais diversos dispositivos, impactando diretamente no desenvolvimento de software embarcado. Além disso, um de seus principais recursos e vantagens é a capacidade de otimização, que torna os modelos de ML mais eficientes e compactos. A otimização preserva a precisão dos modelos, enquanto diminui o tamanho dos recursos computacionais assim como o próprio modelo. Uma técnica largamente utilizada com esse intuito é a quantização, a qual faz os ajustes necessários e remove aquilo que é desnecessário no modelo (EITCA ACADEMY., s.d.).

A quantização faz um ajuste essencial que faz com que o modelo fique menor e que acelere a inferência, impactando minimamente a precisão, a qual faz uma redução nos números utilizados no modelo, passando os valores de 32 bits, que são floats, para 8 bits, que são inteiros, ou seja, descartando as casas decimais (JACOB et al., 2018). Dessa maneira ela também permite que hardwares que não possuam unidades de ponto flutuante possam utilizar o LiteRT, aumentando ainda mais sua compatibilidade com dispositivos mais limitados.

Para dispositivos com baterias limitidas, o LiteRT dispõe de uma técnica chamada poda, a qual remove conexões desnecessárias das redes neurais, sejam elas redundantes e/ou pouco significativas para essas redes, tornando o modelo de tamanho menor e consumo mais reduzido de energia (HAN et al., 2015).

Dentre os vários recursos do LiteRT, pode-se destacar os mais relevantes. Com relação às restrições, pode-se citar a latência, privacidade, conectividade e consumo de energia. Quanto aos suportes, eles podem ser divididos em dois: suporte as plataformas, tendo compatibilidade com dispositivs Android e iOS, Linux incorporado e, o principal nesse estudo, microcontroladores; o outro suporte são as várias linguagens de programação que o mesmo comporta, o tradicional C e C++ e o Python, que vem sendo amplamente utilizado, além de outras linguagens (EITCA ACADEMY., s.d.).

O LiteRT também têm opções de modelos com vários frameworks e um recurso fundamental: a aceleração de hardware, que vem aumentando em grande escala o desem-

penho. Todas essas vantagens vem ampliando a viabilidade da execução de AI em vários aplicativos e sistemas embarcados, permitindo soluções únicas (EITCA ACADEMY., s.d.).

Assim como toda tecnologia existente em que há vantagens e desvantagens, para o LiteRT não é diferente, ela enfrenta desafios e limitações significativos. Por mais que o LiteRT tenha compatibilidade com diversos dispositivos ela ainda enfrenta problemas de portabilidade entre diferentes arquiteturas de hardware, sendo assim, a otimização para cada plataforma específica pode exigir ajustes manuais, tornando o desenvolvimento dessa aplicação mais dificultosa e complexa (LIN et al., 2021).

Com relação ao futuro do LiteRT ele está intrinsecamente ligado à computação de borda. A computação de borda é o processamento, análise e armazenamento de dados o mais próximo possível de onde eles foram gerados, com o objetivo de tornar as respostas mais rápidas assim como a análise, sendo praticamente tudo em tempo real. Seu uso pode ser efetuado em diversas áreas, como o varejo, que teria o objetivo de deixar o fornecimento e o desenvolvimento de produtos mais eficiente, um exemplo é a utilização de sensores e câmeras para se obter os objetivos ditos anteriormente (INTEL, 2025).

Os dispositivos cada vez mais vêm incorporando modelos de ML, isso faz com que aumente a demanda por bibliotecas mais leves e eficientes, que é o caso do LiteRT, tornando sua aplicação cada vez maior. Um destaque que tende a ser cada vez mais promissor é a união entre LiteRT e frameworks de aprendizado federado, que é uma configuração de ML, o qual faz os modelos treinarem de forma colaborativa, mantendo a conservação dos dados (KAIROUZ et al., 2019).

Além disso, pode-se esperar melhorias no suporte de hardwares especializados, como Arranjos de Portas Programáveis em Campo (do inglês, *Field Programmable Gate Array* (FPGA)) e Unidades de Processamento Neural (do inglês, *Neural Processing Unit* (NPU)). Esses hardwares são especializados em executar modelos de AI, o qual oferecem maiores desempenhos, com a união com o LiteRT, faz com que potencialize ainda mais esses hardwares e amplie suas capacidades (JOUPPI et al., 2017).

# 3 TUTORIAL DE UTILIZAÇÃO NO ESP32

Antes de abordar como se utiliza o ESP32 com ML, será apresentada uma breve introdução sobre sua história e conceitos.

O ESP32 é um microcontrolador criado e desenvolvido pela empresa chinesa Espressif, o qual alia alta performance e um baixo consumo de energia. Destaca-se pelo baixo custo de aquisição e por seus recursos de conectividade, tendo como principais o wifi e o bluetooth, ambos integrados, o que a torna uma excelente escolha para projetos de robótica, automação e IoT (SMARTKITS, 2023).

A parte física do ESP32 é dotada de um processador dual-core de 32 bits e possui 520 kB de memória flash. Com relação a pinagem, o mesmo possui 34 pinos GPIO, que são pinos digitais e pinos analógicos, dentre esses 34, 22 são digitais e 12 são analógicos, permitindo a interação do ESP32 com outros dispositivos eletrônicos, sensores e também atuadores (VICTOR VISION, 2023).

Os pinos digitais e analógicos têm suas funcionalidades no qual o digital são pinos configuráveis que podem atuar tanto na entrada quanto na saída, já os pinos analógicos são utilizados para leitura de sinais analógicos, como variações de temperatura e tensão (VICTOR VISION, 2023).

Com relação a programação, o mircrocontrolador ESP32 é bem versátil, pois é compátivel com várias linguagens de programação, dentre as quais se destacam C, C++ e Python. A escolha da linguagem vai depender de qual o principal objetivo do projeto: o Python é amplamente utilizado para projetos de AI, IoT e projetos de prototipagem, já o C e C++ se configuram mais em projetos que exigem alto desempenho e controle de hardware, sendo mais utilizados em projetos de sistemas embarcados (VICTOR VISION, 2023).

No que tange ao desenvolvimento com o ESP32, há inúmeras ferramentas que têm compatibilidade com o microcontrolador e podem ser utilizadas sem grandes dificuldades. O uso dessas ferramentas vai depender exclusivimente das necessidades do projeto, assim como a familiaridade do desenvolvedor com os diversos tipos de ferramentas. Será apresentado neste momento alguma das ferramentas mais comumentemente utilizadas com o ESP32:

O ESP-IDF é o framework desenvolvido pela própria criadora do ESP32, sendo assim o ambiente de desenvolvimento oficial do microcontrolador, o qual oferece um amplo conjunto de ferramentas, bibliotecas e exemplos práticos, facilitando o entendimento e proporcionando maior capacidade de desenvolvivemento e aplicações, sendo eficiente e robusto. O grande destaque quanto às vantagens é o completo suporte ao ESP32 (EMBARCADOS, 2023).

Eclipse IDE é uma plataforma de desenvolvimento extensível e integrada, sendo uma

das plataformas mais populares nesse meio. Ela em conjunto com o plugin do ESP-IDF torna o ambiente para com o ESP32 mais robusto. Por possuir uma vasta flexibilidade isso a torna extremamente compatível e ideal com projetos mais complexos, proporcionando excelentes gerenciamentos, avançadas ferramentas de depuração e uma integração direta com o ESP-IDF (STARTBIT, 2023).

MicroPython é uma implementação reduzida e eficiente que tem algumas bibliotecas padrões da linguagem de programação Python tendo como função a otimização para rodar em microcontroladores e também em ambientes com recursos limitados. Seu ambiente é interativo onde você pode utilizar os comandos do Python no próprio microcontrolador, por ser muito compacto ele cabe e roda em no máximo 256k de espaço para código e 16k de memória RAM, também possui diversos suportes a módulos específicos para determinado tipo de hardware, como GPIO e PWM. No ESP32 as principais utilidades do MicroPython estão na prototipagem rápida e acesso ao hardware (MICROPYTHON, 2023).

PlatformIO é uma plataforma de desenvolvimento que suporta o microcontrolador ESP32, assim como outros hardwares além desse. Além disso, possui suporte ao MicroPython oferendo divesas funcionalidades que facilitam a implementação e desenvolvimento, as principais são gerenciamento de projetos, depuração e instalação de bibliotecas (PLATFORMIO, 2023).

## 3.1 Passo a passo para implementação

Como já se sabe, o LiteRT possibilita os desenvolvedores a criarem modelos de ML em dispositivos limitados, que é o caso do microcontrolador ESP32. Dessa forma, isso se torna extremamente importante, visto o aumento acelerado da utilização de redes neurais com relação a AI, pois a mesma é capaz de reconhecer padrões complexos e realizar previsões com alta precisão (ELETROGATE., 2023).

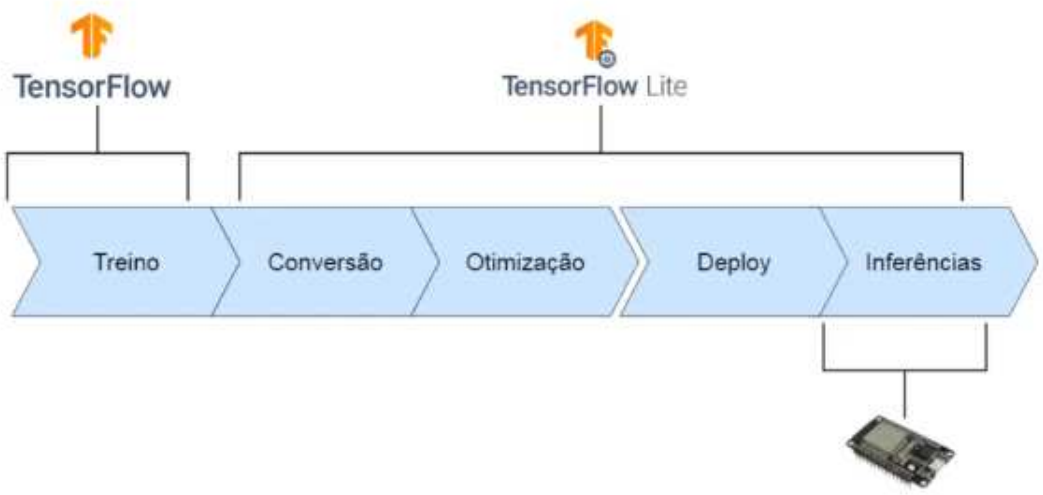
Diante desse cenário, destaca-se a Inteligência Artificial das Coisas (AIoT), que é a união da AI com a IoT. Sabendo que a IoT consiste em dispositivos inteligentes interconectados, equipados com sensores e atuadores, além da conectividade de rede, o que possibilita a coleta e transmissão de dados, combina-se a AI com a IoT. Isso permite que a AI analise os dados, processe-os em tempo real e tome decisões de forma autônoma, automatizando a tarefa desejada. Como resultado, o sistema se torna mais otimizado, preciso e extremamente adaptável (ELETROGATE, 2023).

Nesse contexto, as redes neurais artificais, foram baseadas/inspiradas na estrutura neural do cérebro humano. Sua disposição se dá através dos neurônios, que são interligados entre si e tendo sua organização em camadas. Suas camadas são divídidas basicamente em três níveis, sendo a camada de entrada, as camadas intermediárias, podendo ser uma ou mais, e por fim a camada de saída. Inicialmente a rede recebe um sinal, no qual a seguir

serão aplicados pesos as entradas, tendo uma soma e uma função de ativação. O papel dessa função é introduzir a rede a não-linearidade, fazendo com que a rede seja capaz de aprender padrões ainda mais difíceis (ELETROGATE., 2023).

O que foi dito anteriormente recebe o nome de propagação feedforward, ou retro-propagação. Onde o mesmo alimenta as entradas e saídas da rede. Para que ocorra o treinamento dessa rede é essencial expor a mesma a um vasto conjunto de dados com entradas e saídas conhecidas, diante dessa exposição ela será capaz de ajustar pesos e vieses se baseando nos erros que foram obtidos na comparação entre a saída prevista e a saída real. Desse modo, a rede se torna capaz de aprender com seus erros e com o passar do treinamento consegue aprimorar sua capacidade de precisão nos dados. No término dessa etapa, a rede é capaz de avaliar dados desconhecidos (ELETROGATE., 2023).

A Figura 1 resume bem o passo a passo de como proceder para implementar um modelo de ML no ESP32 será mostrada e explicada a seguir:




– Figura 1 - Diagrama das etapas do modelo e seus respectivos frameworks

Fonte: blog.eletrogate.com, 2023

Primeiramente, é necessário treinar o modelo utilizando o TensorFlow, no qual há etapas bem definidas a serem seguidas:

1. coletar os dados para alimentar o modelo;
2. processamento inicial dos dados, no qual há limpeza, pois podem haver valores faltantes, erros e valores fora do padrão, depois tem que normalizar os dados e por fim dividi-los em conjunto de treino e teste;
3. construir o modelo, ou seja, definir a rede neural;

4. treinar o modelo com os dados que foram alimentados na entrada fazendo os ajustes dos pesos e vies;

5. avaliar o modelo para verificar se está desempenhando bem sua função.

O segundo passo, como bem sugere a imagem, é fazer a conversão do TensorFlow para o TensorFlow Lite (LiteRT). É utilizado uma ferramenta em Python para fazer essa conversão, chamada "TFLiteConverter", que transforma o modelo de formato .pb ou SavedModel do TensorFlow para o formato .tflite do TensorFlow Lite (LiteRT). Dessa maneira, otimizando os dados para os dispositivos limitados, além da redução do tamanho do modelo, para o nosso caso o ESP32.

O terceiro passo é a otimização do modelo, para alcançar modelos mais eficiente e precisos. Como a capacidade de processamento do ESP32 é baixa, o modelo precisa diminuir o consumo de memória e tempo de inferência, para isso, faz-se necessário quantizar o modelo, ou seja, converter os pesos de float32 para int8, dessa maneira se reduz o tamanho do modelo e melhorando seu desempenho. Mas a depender do modelo pode-se usar outras ferramentas para otimizar, como: poda ou distilação de conhecimento.

O quarto passo é o Deploy no ESP32, que é carregar todos os passos anteriores no microcontrolador, que vai depender da ferramenta utilizada para programar, podendo ser um Arduíno IDE, ESP-IDF e etc.

Para finalizar, o quinto e último passo é a inferência, que é a alimentação do modelo com dados novos. No ESP32 sua inferência é basicamente coletar os dados através de sensores, processar os dados de maneira a ficar no formato que o modelo foi programado, executar o modelo com o TensorFlow Lite para ter a previsão e para finalizar ele irá interpretar os resultados obtido e tomará uma decisão de acordo com o que se deseja.

# 4 EXEMPLO DE UTILIZAÇÃO

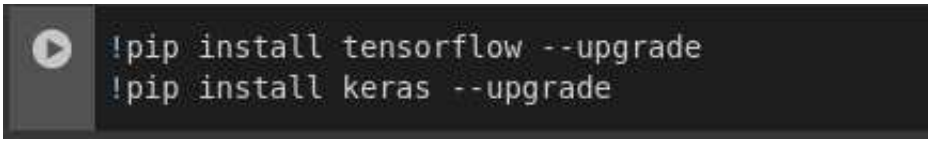
Nesta seção será implementado um projeto sobre toda a teoria vista nos capítulos anteriores, a fim de comprovar através de um exemplo prático a combinação do ESP32 com ML, no qual será usado como base todo o projeto desenvolvido pelo professor Abhishek Singh. A abordagem em questão tem como tema a previsão de saída de uma onda senoidal e visualização da rede de aprendizado profundo.

A motivação para esse projeto é devida a toda importância das ondas senoidais, que são a forma de representação ondulatória natural de muitos fenômenos físicos, além de serem as mais presentes no mundo real e, consequentemente, uma das mais estudadas e trabalhadas. Isso ocorre pelo fato de que muitos fenômenos podem ser representados matematicamente por essa forma de onda. Sua ocorrência é observada em sinais de corrente alternada e em ondas de rádio, que são elementos comuns no cotidiano da humanidade. No entanto, há diversas outras ocorrências dessas ondas além das mencionadas, como no movimento pendular simples (SINGH, A. 2022).

O objetivo desse projeto é prever a saída da onda senoidal, e para isso será utilizado um modelo de ML, o qual fará essa previsão. Posteriormente, para visualização será utilizado o aplicativo de navegador Netron. (SINGH, A. 2022)

Será usado para implementação desse projeto a linguagem de programação Python com o pacote Anaconda e o Google Colab para implementar a sintaxe da linguagem e plotar os gráficos pertinentes. (SINGH, A. 2022)

Pelo fato de estarmos usando o Google Colab faz-se necessário primeiramente fazer as seguintes instalções:




– Figura 2 - Instalações necessárias

Fonte: Elaborado pelo autor através do Google Colab

Os comandos “pip install” fornecem a instalação, atualização e o gerenciamento de ferramentas que não vem pré instaladas no Google Colab, e nesse caso foi instalado o TensorFLow para usar ML e a Keras para redes neurais, “–upgrade” faz com que seja instalada a versão mais atual de cada biblioteca.

Tendo feitas as devidas instalações, o próximo passo será importar um conjunto de bibliotecas para prosseguir com a proposta.

```
import tensorflow as tf
import keras as ks
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import math
import sys
from tensorflow.keras import layers
```

– Figura 3 - Bibliotecas importadas

Fonte: Elaborado pelo autor através do Google Colab

Neste momento, será criado um conjunto de dados. Para esse exemplo em questão, serão utilizadas 7000 amostras, nas quais a distribuição será da seguinte maneira: 20% para validação, 20% para testes e 60% para treinamento. Dessa forma, será garantido que o modelo seja capaz de aprender e ser avaliado da maneira necessária para dados não vistos antes. A Figura 4 mostra o código utilizado para executar o que foi descrito:

```
[ ]  nsamples = 7000
     val_sam_per = 0.2
     test_sam_per = 0.2
     tflite_model_name = 'sine_model'
```

– Figura 4 - Amostras, Validação, Teste e Treinamento

Fonte: Elaborado pelo autor através do Google Colab

Após executar o passo anterior, para prosseguir com o modelo, faz-se necessário gerar um sinal aleatório. É isso que a Figura 5, a seguir, está mostrando: o código responsável por gerar esse sinal aleatório.

```
np.random.seed(2)
x_values = np.random.uniform(low=0, high=2 * math.pi, size=nsamples)
plt.plot(x_values)
```

– Figura 5 - Código para gerar um sinal aleatório

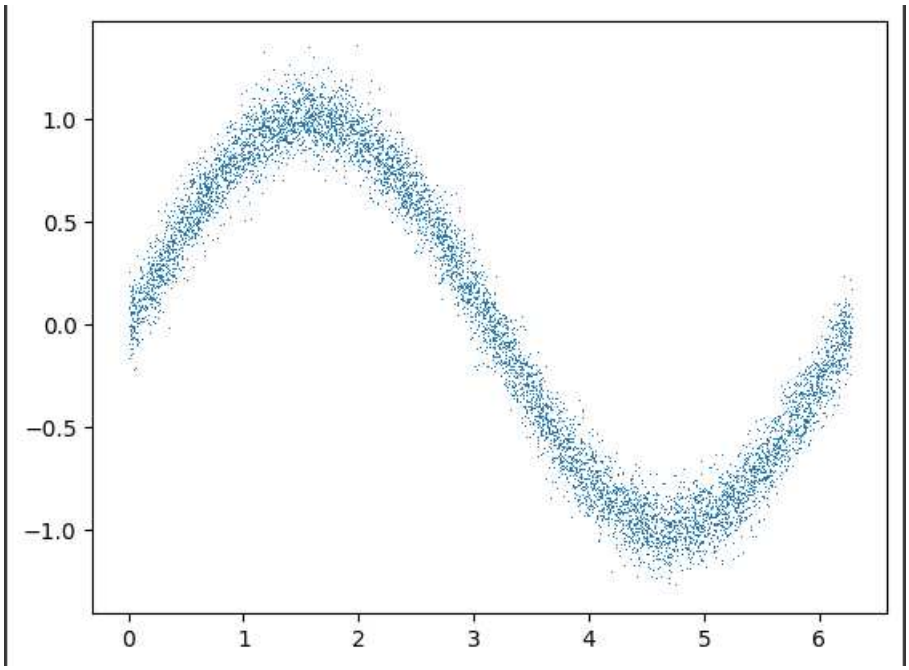
Fonte: Elaborado pelo autor através do Google Colab

Após compilar, com a utilização do Numpy (“np” no código), foi gerado um sinal com as amostras, que não é uma onda senoidal pura, possuindo uma parcela de aleatoriedade. Esse sinal, que combina a forma de onda senoidal com ruído, é descrito na Figura 6. Em seguida, será criada a forma de onda senoidal com ruído, apresentado na Figura 7. (SINGH, A. 2022)

```
y_values = np.sin(x_values) + (0.1 * np.random.randn(x_values.shape[0]))
plt.plot(x_values, y_values, ',')
```

– Figura 6 - Sinal aleatório das amostras

Fonte: Elaborado pelo autor através do Google Colab




– Figura 7 - Sinal de onda senoidal ruidoso

Fonte: Elaborado pelo autor através do Google Colab

O conjunto de dados para treinar e testar serão gerados através de x e y, onde x representa os dados de entrada do modelo, enquanto y contém os rótulos ou valores esperados, que o modelo irá aprender a prever com base em x. Para evitar que o modelo decore padrões duvidosos e consequentemente cometa erros, vamos embaralhar de forma aleatória, tornando o modelo mais eficiente e com treinamento mais estável. Para isso será importada uma nova biblioteca para executar essa função de embaralhamento. (SINGH, A. 2022)

```
from sklearn.utils import shuffle
x_values, y_values = shuffle(x_values, y_values)
```

– Figura 8 - Embaralhamento

Fonte: Elaborado pelo autor através do Google Colab

Fundamentalmente o processo de aprendizagem de modelos de ML são divididos em três etapas: treinamento, validação e teste.

- Treinamento: treina o modelo

- Validação: faz a comparação entre diferentes modelos
- Teste: faz testes para comprovar se o modelo funciona

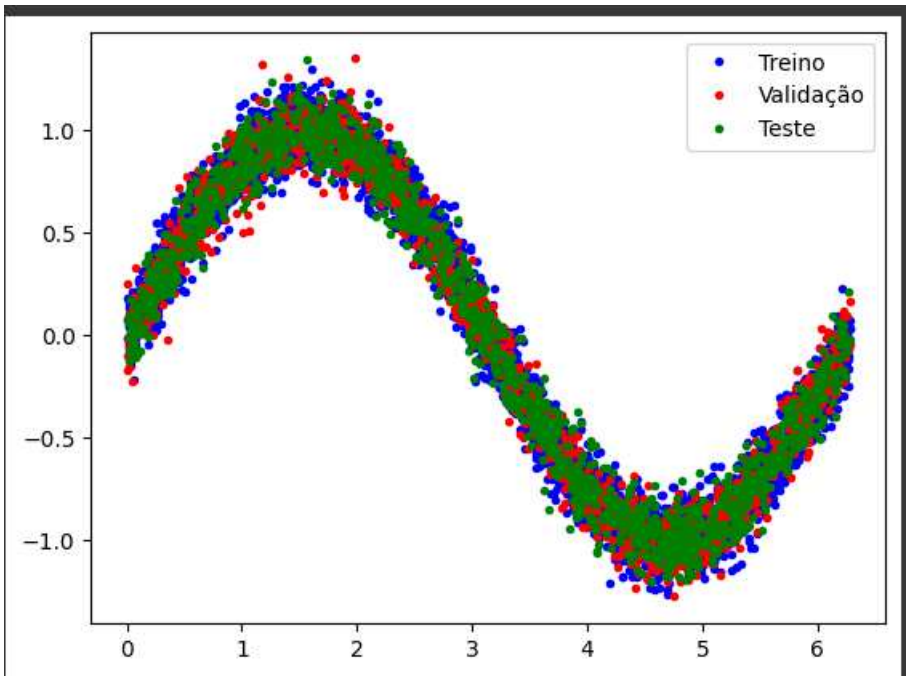
Nesse modelo foi feita essa divisão, na qual o Numpy é responsável pelo processo, além disso, com o matplotilib plotamos a curva com essas divisões. (SINGH, A. 2022)

```
#Divisão dos dados
val_split = int(val_sam_per * nsamples)
test_split = int(val_split + (test_sam_per * nsamples))
x_val, x_test, x_train = np.split(x_values, [val_split, test_split])
y_val, y_test, y_train = np.split(y_values, [val_split, test_split])

#Plot dos dados
plt.plot(x_train, y_train, 'b.', label='Treino')
plt.plot(x_val, y_val, 'r.', label='Validação')
plt.plot(x_test, y_test, 'g.', label='Teste')
plt.legend()
plt.show()
```

– Figura 9 - Código da divisão dos dados e plotagem

Fonte: Elaborado pelo autor através do Google Colab




– Figura 10 - Divisão dos Dados em Treinamento, Validação e Teste

Fonte: Elaborado pelo autor através do Google Colab

Com o plot do gráfico podemos concluir que tivemos uma boa divisão de dados, no qual o modelo aprenderá de forma eficiente como prever uma senoide ruidosa. Tendo em vista as amostras sobrepostas e bem distribuídas, dá para presumir que esse modelo será consistente em seus dados para aprender e avaliá-los.

No próximo passo, será criado o modelo de rede neural, conforme ilustrado na Figura 11. Esse modelo terá duas camadas densas plenamente conectadas. Será utilizado

a função RELU para ativar e projetar as redes neurais, e será utilizado o Adam para otimizar o modelo. (SINGH, A. 2022). Código utilizado:

```
model = tf.keras.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(1,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1))

model.compile(optimizer='adam', loss='mae', metrics=['mae'])

history = model.fit(x_train, y_train, epochs=500, batch_size=100, validation_data=(x_val, y_val))
```

– Figura 11 - Código Rede Neural

Fonte: Elaborado pelo autor através do Google Colab

Foi utilizado um tamanho de lote 100 e 500 iterações. Para exibir o resumo de uma maneira mais fácil de ser visualizada e entendida foi usado o model.summary o qual nos apresenta os seguintes dados: (SINGH, A. 2022).

```
model.summary()
```

```
Model: "sequential"
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense (Dense) | (None, 16) | 32 |
| dense_1 (Dense) | (None, 16) | 272 |
| dense_2 (Dense) | (None, 1) | 17 |

```
Total params: 965 (3.77 KB)
Trainable params: 321 (1.25 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 644 (2.52 KB)
```

– Figura 12 - Resultado Rede Neural

Fonte: Elaborado pelo autor através do Google Colab

Com os resultados expressos da Figura 12, é indicado que o presente modelo de rede neural treinado têm três camadas densas interconectadas (dense, dense_1 e dense_2). A primeira e segunda camada possuem 16 neurônios cada, e a camada de saída possui apenas um. O número de amostras necessárias para prever uma nova amostra vai depender do tamanho da janela de entrada, nesse modelo foi definido um vetor de dimensão fixa com N pontos anteriores.

Foram plotados valores específicos dos parâmetros, no qual se evidencia que o modelo treinou todos os seus parâmetros, sem a ocorrência de elementos que não foram treinados pela rede neural. Com um total de 965 parâmetros, dos quais 644 foram otimizados, dessa forma apresentando a eficácia desse processo.

Podemos agora calcular $R^2$ (r2), que é o coeficiente de determinação, além disso, o cálculo do Erro Quadrático Médio (do inglês, *Root Mean Squared Error* (RSME)) e o Erro

Absoluto Médio (do inglês, *Mean Absolute Error* (MAE)), o qual irão verificar a precisão do modelo. (SINGH, A. 2022).

Matematicamente as métricas são definidas da seguinte maneira:

- para o $R^2$:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{4.1}$$

  onde $y_i$ são os valores observados, $\hat{y}_i$ são os valores previstos, $\bar{y}$ é a média dos valores observados e $n$ o número total de amostras.

- para o RMSE:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.2}$$

  onde $y_i$ são os valores observados, $\hat{y}_i$ são os valores previstos e $n$ o número total de amostras.

- para o MAE:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.3}$$

  onde $y_i$ são os valores observados, $\hat{y}_i$ são os valores previstos e $n$ o número total de amostras.

$R^2$ tem por objetivo avaliar se o modelo se ajusta corretamente ao dados, indicando a qualidade do modelo. $RMSE$ tem por objetivo fornecer uma medida de precisão do modelo, mostrando o quanto as previsões estão diferentes dos valores reais. $MAE$ tem como intuito avaliar a precisão do modelo, mostrando o erro médio das previsões.

A figura 13 apresenta o código e os resultados dos cálculos de $R^2$, RMSE e MAE, os quais são utilizados para avaliar o modelo treinado, permitindo uma análise mais precisa, pois possibilitam a quantificação do erro de previsão em relação ao valores reais.

Diante dos resultados, nitidamente podemos perceber que a pontuação de r2 é próxima de um, 97,82%. Esse valor para r2 e os valores baixos de RMSE e MAE, indica que o modelo está fazendo boas previsões, onde os 97,82% de r2 é a capacidade de explicação da variância dos dados, fornencendo uma linha de menor ajuste (SINGH, A. 2022).

A Figura 14 mostra as formas de onda reais e previstas para fazer uma análise da aproximação entre elas. Como r2 está bem próximo de um, isso sugere que ambas devem seguir uma a outra com pouca variação (SINGH, A. 2022).

```
# Previsão do modelo treinado
y_pred = model.predict(x_test)

#Métricas
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
print('R2: ', r2_score(y_pred, y_test))
print('MAE: ', mean_absolute_error(y_pred, y_test))
print('MSE: ', mean_squared_error(y_pred, y_test))
```

```
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
R2:  0.9782311903844815
MAE:  0.08149942690082296
MSE:  0.010562300075766397
```

– Figura 13 - Cálculo das métricas

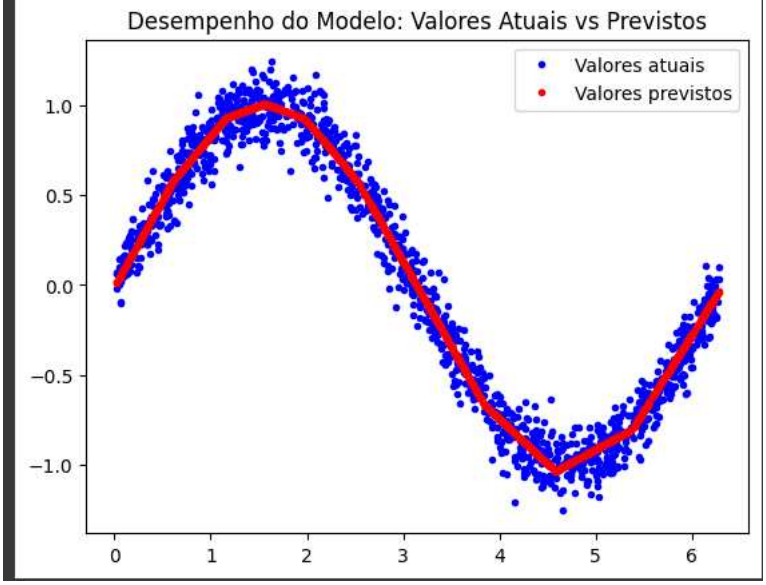
Fonte: Elaborado pelo autor através do Google Colab

Após a etapa de treinamento do modelo de rede neural no Google Colab utilizando o TensorFlow, o modelo foi convertido para TensorFlow Lite e, em seguida, implementado no ESP32 para a realização da inferência. A Figura 14 apresenta o desempenho do modelo, comparando os valores atuais (reais) com os valores previstos pelo modelo após ser executado no ESP32.

```
# Previsão do modelo treinado
y_pred = model.predict(x_test)

plt.clf()
plt.title("Desempenho do Modelo: Valores Atuais vs Previstos")
plt.plot(x_test, y_test, 'b.', label='Valores atuais')
plt.plot(x_test, y_pred, 'r.', label='Valores previstos')
plt.legend()
plt.show()
```

```
44/44 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```




– Figura 14 - Desempenho do Modelo: Valores Atuais vs Previstos

Fonte: Elaborado pelo autor através do Google Colab

Como pode ser visto no plot do gráfico, os valores previstos acompanham os valores

atuais, o que significa que o modelo seguiu satisfatóriamente a forma de onda senoidal. Nas regiões de curvatura o modelo dos valores atuais teve uma pequena dificuldade para prever os resultados, isso se dá devido ao ruído, mas nada que afete o modelo, porém tem como melhorá-lo, assim como o mesmo possui leve dispersão dos pontos azuis mais na região das curvaturas.

É importante salientar que o modelo foi realizado de maneira offline, utilizando a linguagem de programação Python, tudo dentro do ambiente do Google Colab. Em contrapartida, a inferência foi executada rigorosamente no ESP32, visando a execução em tempo real. O ESP32 foi programado para coletar os dados de entrada, em seguida processá-los e, finalmente executar o modelo TensorFlow Lite para a realização das previsões. Sendo assim, essa abordagem e processo permitem que modelos mais complexos possam ser treinados em ambientes com maior capacidade computacional e, em seguida, possam ser convertidos para serem implantados em dispositivos embarcados que possuem recursos mais limitados, que é o caso do ESP32 que foi utilizado.

O software utilizado para visualização desse modelo será o Netron. O Netron é um programa capaz de fazer a visualização de sistemas neurais, o qual pode ser baixado ou utilizado de forma online pelo próprio navegador (EDIVALDO, B. 2024).

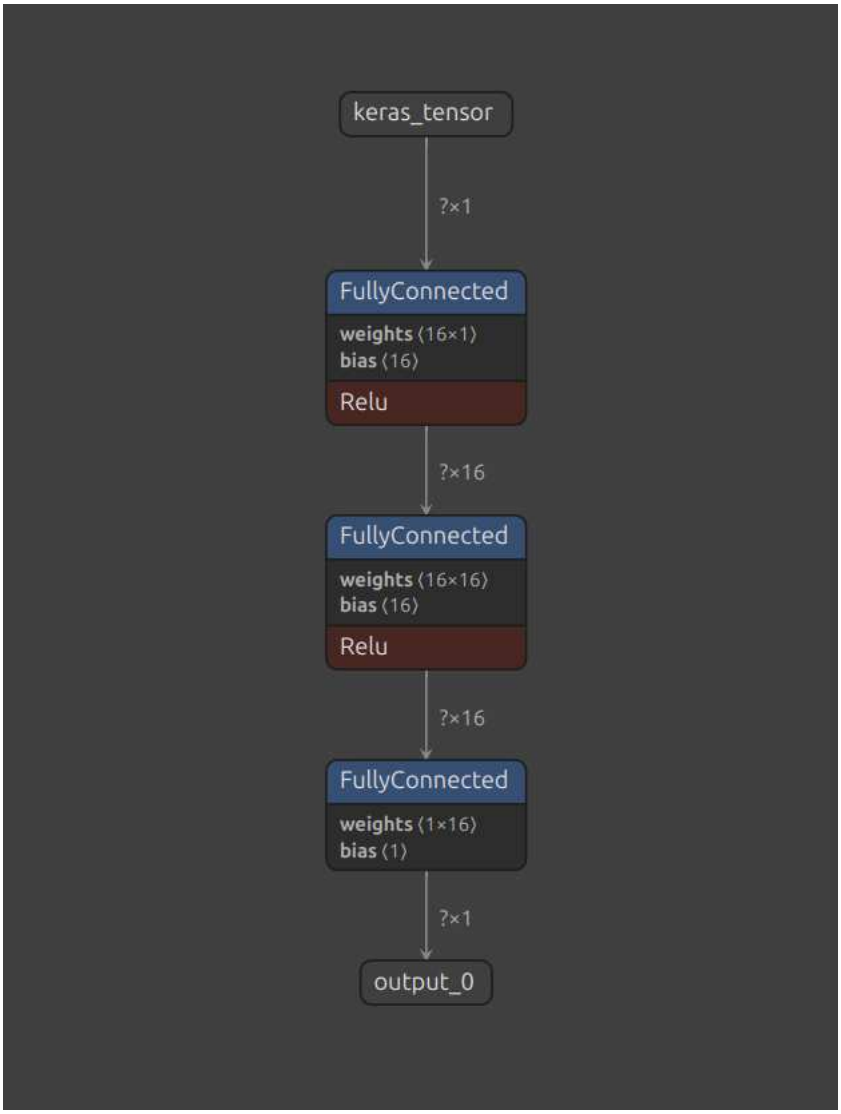
Antes de utilizar o Netron para visualizar nosso modelo é preciso converter o modelo de TensorFlow para TensorFlow Lite (LiteRT), o qual irá criar um arquivo model.tflite no diretório do trabalho.

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
with tf.io.gfile.GFile('model.tflite', 'wb') as f: f.write(tflite_model)
```

– Figura 15 - Conversão do modelo para TensorFlow Lite (LiteRT)

Fonte: Elaborado pelo autor através do Google Colab

Após esse procedimento, podemos acessar o site do Netron e carregar o arquivo convertido, o que gerou a imagem mostrada na Figura 16.




– Figura 16 - Modelo Rede Neural no Netron

Fonte: Elaborado pelo autor através do Netron

Podemos observar o diagrama de fluxo da rede neural convertida. No qual cada parte dessa rede tem suas propriedades e funções que serão melhores detalhadas nesse instante:

- keras_tensor é a entrada, no qual essa entrada é um vetor unidimensional e sua dimensão não é definida, o símbolo de interrogação nos indica essa indefinição, ou seja, seu tamanho é dinâmico e pode variar de acordo com a inferência. Isso se aplica as outras camadas que tem o símbolo de interrogação, isso se dá pelo fato de na hora da conversão o modelo não ter sido especificado a sua dimensão, deixando assim o mesmo com maior flexibilidade e eficiência.
- FullConnected primeira camada: o qual tem os Pesos (16 x 1) que significa a conexão de 1 neurônio de entrada a 16 neurônios, Bias (16) indica que cada neurônio dos 16

possui um viés epecífico. O RELU é ativação no qual introduz a não linearidade do sistema, a Saída (? x 16) indica que a mesma produz 16 valores para cada entrada.

- FullConnected segunda camada: os Pesos (16 x 16) tem a função de conectar os 16 neurônios da primeira camada a outros 16, seu Bias e RELU tem o mesmo sentido e significado da primeira camada e sua Saída (? x 16) mantém 16 neurônios.
- FullConnected terceira camada: seu Peso (1 x 16) faz uma redução de neurônios, passando de 16 para somente 1 na saída, Bias (1) indica um viés para saída e sua Saída faz o retorno de um valor por amostra
- output_0 é a saída final do sistema

# 5 CONCLUSÃO

Este trabalho buscou elucidar os conceitos e técnicas da integração entre AI e Sistemas Embarcados, o qual foi embasado no framework TensorFlow Lite (LiteRT) em um dispositivo com recurso limitado, no caso o microcontrolador ESP32. Com foco teórico e um exemplo prático, foi possível demonstrar um pouco da tecnologia, sua relevância, vantagens e desvantagens.

Apesar da Inteligência Artificial ser uma ferramenta extremamente poderosa no que diz respeito a otimização de processos e automatização de tarefas, para grandes dados ela ainda apresenta certos desafios no campo de Sistemas Embarcados, devido à baixa memória desses dispositivos, assim como a energia e capacidade de processamento. Devido a esses desafios surgiu o TensorFlow Lite (LiteRT) para atender essa demanda de dispositivos de recursos limitados, como o ESP32.

O presente estudo nos mostrou que o TensorFlow Lite (LiteRT) é extremamente fundamental para implementar Aprendizado de Máquina no microcontrolador ESP32, fornecendo técnicas para otimização dos dados, diminuição do tamanho do modelo e menor consumo de energia, deixando o modelo eficiente.

O exemplo prático elucidou todos os conceitos e técnicas fornecidos nos escopos anteriores, mostrando uma alta eficácia do modelo, que propunha prever uma onda senoidal utilizando uma rede neural. O coeficiente de determinação atingiu satisfatórios 97,82%, demonstrando sua excelente capacidade de previsão em relação aos dados reais.

Por mais que o estudo e seus resultados tenham sido satisfatórios, vale ressaltar que o campo de Inteligência Artificial em Sistemas Embarcados ainda não é muito maduro, apresentando alguns desafios a serem superados, como a necessidade de ajustes manuais em plataformas específicas.

Com um mundo cada vez mais conectado e a busca incessante por tecnologias de ponta, a intenção deste estudo foi contribuir para a disseminação do conhecimento sobre a integração entre Aprendizado de Máquina e Sistemas Embarcados, fornecendo uma base teórica, aplicações, exemplo prático e uma perspectiva futura para essa tecnologia, na qual a tendência é crescer de forma exponencial, visto as necessidades da sociedade.

Para futuras pesquisas, são sugeridos os seguintes temas de investigação:

- Implementação do modelo proposto em um outros microcontroladores, como, por exemplo, o Raspberry Pi, realizando o mesmo estudo e comparando os desempenhos entre os microcontroladores.
- Aprimoraramento do modelo de ML, utilizando outras redes neurais, explorando técnicas de aprendizado profundo mais complexas e empregando técnicas de quantização

mais avançadas.

- Aplicação do modelo desenvolvido em um sistema real, fazendo a integraçãodo do ESP32 com sensores físicos, como sensores de temperatura, com o intuito de validar a aplicação em situações reais e práticas.
- Investigação de outras técnicas de redes neurais para reduzir ainda mais o uso da memória e o consumo de energia, ou seja, visando uma maior otimização dos recursos
- Desenvolvimento do modelo em áreas mais usuais e práticas, como automação industrial e monitoramento da saúde, tornando a tomada de decisões locais.

# REFERÊNCIAS


1 CHEN, J. *et al.* **Edge Computing for AI-enabled Medical Devices**. IEEE Transactions on Biomedical Engineering, 2020.

2 EDIVALDO, B. **Blog do Edivaldo**. 2024. Disponível em: https://www.edivaldobrito.com.br/netron-um-programa-para-visualizar-modelos-de-redes-neurais/.

3 EITCA ACADEMY. **O que é o TensorFlow Lite e qual é a sua finalidade?** Disponível em: https://pt.eitca.org/artificial-intelligence/eitc-ai-tff-tensorflow-fundamentals/programming-tensorflow/tensorflow-lite-for-android/examination-review-tensorflow-lite-for-android/what-is-tensorflow-lite-and-what-is-its-purpose/.

4 ELETROGATE. **TensorFlow Lite no ESP32**. 2023. Disponível em: https://blog.eletrogate.com/tensorflow-lite-no-esp32/.

5 EMBARCADOS. **Desenvolvimento para ESP32: ESP-IDF, Zephyr, NuttX ou Arduino?** 2023. Disponível em: https://submazine.com/quais-as-plataformas-de-programacao-utilizadas-para-programar-o-esp32/.

6 GOOGLE. **Edge for Microcontrollers Overview**. 2024. Disponível em: https://ai.google.dev/edge/litert/microcontrollers/overview?hl=pt-br.

7 GOMES, L.; PEREIRA, M. **Machine learning applications in autonomous driving systems**. Journal of Automotive Technology, v. 15, n. 3, p. 120-135, 2020.

8 GRIGORESCU, S. *et al.* **A Survey of Deep Learning Techniques for Autonomous Driving**. Journal of Field Robotics, 2020.

9 HAN, S. *et al.* **Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding**. International Conference on Learning Representations (ICLR), 2016.

10 INTEL. **O que é computação de borda?** 2025. Disponível em: https://www.intel.com.br/content/www/br/pt/edge-computing/what-is-edge-computing.html.

11 JACOB, B. *et al.* **Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference**. In: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR), 2018.

12 JOUPPI, N. P. *et al.* **In-Datacenter Performance Analysis of a Tensor Processing Unit**. In: INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE (ISCA), 2017.

13 KAIROUZ, P. *et al.* **Advances and Open Problems in Federated Learning**. arXiv preprint arXiv:1912.04977, 2019.

14 KITAMURA, Celso. **A história da Alexa**. 2023. Disponível em: https://celsokitamura.com.br/a-historia-da-alexa/.

15 LANE, N. D. *et al.* **Squeezing deep learning into mobile and embedded devices**. IEEE Pervasive Computing, v. 16, n. 3, p. 82-88, 2015.

16 LIN, J. *et al.* **Challenges in Deploying TinyML on Heterogeneous Hardware**. TinyML Research Symposium, 2021.

17 MADISETTI, V.; BAHGA, A. **Embedded Systems: Fundamentals and Applications**. 1. ed. 2017.

18 MCDONOUGH, Michael. **Siri**. 2025. Disponível em: https://www.britannica.com/technology/Siri.

19 MICROPYTHON. **Site oficial do Micropython**. 2023. Disponível em: https://micropython.org/.

20 OLIVEIRA, R.; SANTOS, P. **Wearable devices for continuous health monitoring: A review**. HealthTech Journal, v. 8, n. 2, p. 45-59, 2021.

21 PLATFORMIO. **Site oficial do PlatformIO**. 2023. Disponível em: https://platformio.org/.

22 SAVEFARM. **Tecnologia de pulverização inteligente: Inovando a agricultura brasileira**. 2023. Disponível em: https://www.savefarm.com.br/.

23 SILVA, J.; ALMEIDA, C.; COSTA, L. **IoT na automação residencial: Benefícios e desafios**. Revista Brasileira de Tecnologia, v. 10, n. 1, p. 60-75, 2022.

24 SINGH, A. **Previsão da saída da onda senoide e visualização da rede de aprendizado profundo**. 2022. Disponível em: https://medium.com/@abhi16.2007/predicting-sine-wave-output-and-visualizing-the-deep-learning-network-979a11f2af31.

25 SMARTKITS. **ESP32: Modelos mais populares**. 2023. Disponível em: https://blog.smartkits.com.br/esp32-modelos-mais-populares/.

26 STARTBIT. **Ambiente de desenvolvimento com Eclipse IDE para ESP32**. 2023. Disponível em: https://startbit.com.br/md/ambiente-de-desenvolvimento-com-eclipse-ide-para-esp32-L2G2A8.

27 TINYML FOUNDATION. **What is TinyML?** Disponível em: https://www.tinyml.org/.

28 VICTOR VISION. **Placa ESP32: o que é, para que serve e como programar?** 2023. Disponível em: https://victorvision.com.br/blog/placa-esp32/.

29 WARDEN, P.; SITUNAYAKE, D. **TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers**. 2020. O'Reilly Media.