UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Gabriel Victor Alves Santana

Detecção de anomalias na qualidade do ar utilizando aprendizado de máquina em microcontroladores

São Carlos
2025

Gabriel Victor Alves Santana

Detecção de anomalias na qualidade do ar utilizando aprendizado de máquina em microcontroladores

Monografia apresentada ao Curso de Engenharia Elétrica com ênfase em Eletrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Pedro de Oliveira Conceição Junior

São Carlos

2025

Ficha catalográfica elaborada pela Biblioteca Prof. Sérgio Rodrigues Fontes
e pelo Serviço de Comunicação e Marketing da EESC-USP,
com dados inseridos pelo(a) autor(a).

A231d

Alves Santana, Gabriel Victor

Detecção de anomalias na qualidade do ar utilizando aprendizado de máquina em microcontroladores / Gabriel Victor Alves Santana ; orientador Pedro de Oliveira Conceição Junior. -- São Carlos, 2025.

70 p.

Monografia - Graduação em Engenharia Elétrica com ênfase em Eletrônica -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2025.

1. Microcontroladores. 2. Aprendizado de máquina. 3. tinyML. 4. Internet das Coisas (IoT). 5. Detecção de Anomalias. I. de Oliveira Conceição Junior, Pedro, orient. II. Título.

Responsáveis pela estrutura de catalogação da publicação segundo a AACR2: Bibliotecários da EESC/USP.

FOLHA DE APROVAÇÃO

Nome: Gabriel Victor Alves Santana

Título: "Detecção de anomalias na qualidade do ar utilizando aprendizado de máquina em microcontroladores"

Trabalho de Conclusão de Curso defendido e aprovado em 04/12/2025**,**

com NOTA 9,5 (nove, cinco)**, pela Comissão Julgadora:**

Prof. Dr. Pedro de Oliveira Conceição Júnior - Orientador - SEL/EESC/USP

Prof. Dr. Ivan Nunes da Silva - Professor Titular SEL/EESC/USP

Prof. Dr. Marcos Rogério Fernandes - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Professor Associado José Carlos de Melo Vieira Júnior

AGRADECIMENTOS

Agradeço, antes de tudo, a Deus, por me guiar e permitir a conclusão deste trabalho. Agradeço também aos meus pais e amigos, cujo apoio, incentivo e companhia foram indispensáveis para superar os desafios desta trajetória acadêmica. Finalmente, registro meus agradecimentos à Universidade de São Paulo e à Escola de Engenharia de São Carlos, bem como a todos os professores e funcionários, pelo inestimável auxílio, pelo suporte e ambiente de excelência e por todo o aprendizado que me foi oferecido.

RESUMO

SANTANA, G. V. A. **Detecção de anomalias na qualidade do ar utilizando aprendizado de máquina em microcontroladores**. 2025. 70p. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2025.

Modelos de aprendizado de máquina não supervisionados são úteis para identificar padrões complexos em dados não classificados, mas seu alto custo computacional limita sua portabilidade. Para superar essas limitações, este trabalho propõe a aplicação de um modelo de aprendizado de máquina em um sistema embarcado para detecção de anomalias na qualidade do ar por meio da utilização de técnicas de tinyML. O objetivo é unir a capacidade de detecção de padrões do aprendizado de máquina ao baixo custo e portabilidade dos sistemas embarcados. Para isso, foi desenvolvido um modelo autoencoder não supervisionado que, otimizado com quantização INT8, teve seu tamanho reduzido em 47,02% sem perdas significativas de performance. Na validação experimental, feita em comparação a uma abordagem tradicional, o sistema embarcado (TinyML) foi 1,3 vezes mais rápido na detecção de anomalias (tempo médio de 4,5s contra 5,8s). Conclui-se que o sistema TinyML é uma solução robusta e mais rápida, reforçando a viabilidade do processamento na borda para aplicações de IoT que demandam respostas imediatas.

Palavras-chave: microcontroladores. aprendizado de máquina. tinyML. internet das Coisas (IoT). detecção de anomalias.

ABSTRACT

SANTANA, G. V. A. **Air quality anomaly detection using machine learning on microcontrollers**. 2025. 70p. Monograph (Conclusion Course Paper) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2025.

Unsupervised machine learning models are useful for identifying complex patterns in unlabeled data, but their high computational cost limits their portability. To overcome these limitations, this work proposes the application of a machine learning model on an embedded system for air quality anomaly detection through the use of tinyML techniques. The objective is to unite the pattern detection capabilities of machine learning with the low cost and portability of embedded systems. To this end, an unsupervised autoencoder model was developed which, optimized with INT8 quantization, had its size reduced by 47.02% without significant performance losses. In experimental validation, compared to a traditional approach, the embedded system (TinyML) was 1.3 times faster at detecting anomalies (average time of 4.5s versus 5.8s). It is concluded that the TinyML system is a robust and faster solution, reinforcing the feasibility of edge processing for IoT applications that demand immediate responses.

Keywords: microcontrollers. machine learning. tinyML. internet of things (IoT). anomaly detection.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A Internet das coisas (IoT) é uma área de pesquisa que possui uma ampla gama de aplicações que vão desde a indústria e automação [1] à agricultura [2] e até à aplicações na medicina [3]. Segundo [4], o número de dispositivos conectados em 2023 foi de 16 bilhões e a estimativa é que este número chegue a 40 bilhões até 2030. O baixo custo, a grande variedade de sensores e a alta capacidade de processamento são alguns dos motivos desse crescimento contínuo.

Os dispositivos IoT geralmente são a primeira camada de processamento em uma rede de comunicação que começa com a aquisição de dados por meio de de sensores chegando até o armazenamento em nuvem ou à integração com sistemas analíticos. Como esses dispositivos estão na borda de uma rede de comunicação eles são conhecidos como *edge devices*, e o processamento realizado por eles é chamado de *edge computing*. [5, p. 18]

Segundo [5], as principais vantagens na realização de *edge computing* estão relacionadas à conectividade dos dispositivos com a nuvem. O envio de dados tem um custo elevado e um alto consumo de energia. Dessa maneira, processando os dados na borda pode-se limitar o número de conexões e a quantidade de dados enviados. Além disso, conforme aponta [6, p. 4], o envio e recebimento de dados para a nuvem não é instantâneo e pode afetar aplicações que precisam de uma resposta em curto período de tempo.

A partir desse cenário, surge o conceito de tinyML. De acordo com [7], tinyML é definido como o paradigma que facilita a execução de algoritmos de *machine learning* em dispositivos com requisitos mínimos de processador e memória, e com consumo de potência inferior a alguns miliwatts. Apesar de suas limitações, esses dispositivos são baratos, econômicos e portáteis, o que os tornam atraentes para diversas aplicações.

1.1 Motivação

Segundo [8], “qualidade do ar é uma medida do nível de poluentes atmosféricos à qual a população está exposta”. No estado de São Paulo, o monitoramento da qualidade do ar e a definição de padrões a serem seguidos são realizados pela Companhia Ambiental do Estado de São Paulo (CETESB). Os poluentes considerados como parâmetros nas medidas são: partículas inaláveis (MP10 e MP2,5), fumaça, ozônio (O_3), monóxido de carbono (CO), dióxido de nitrogênio (NO_2) e dióxido de enxofre (SO_2) [9]. Segundo a CETESB:

> Considera-se poluente qualquer substância presente no ar e que, pela sua concentração, possa torná-lo impróprio, nocivo ou ofensivo à saúde, causando inconveniente ao bem estar público, danos aos materiais, à fauna e à flora ou prejudicial à segurança, ao uso e gozo da propriedade e às atividades normais da comunidade [10].

A tabela abaixo apresenta a classificação da qualidade do ar com base na concentração do poluente e o tempo de amostragem.

Tabela 1 – Estrutura do índice de qualidade do ar

Qualidade	**Índice**	**MP10 (µg/m3) 24h**	**MP2,5 (µg/m3) 24h**	**O_3 (µg/m3) 8h**	**CO (ppm) 8h**	**NO_2 (µg/m3) 1h**	**SO_2 (µg/m3) 24h**
Boa	0 – 40	0 – 50	0 – 25	0 – 100	0 – 9	0 – 200	0 – 20
Moderada	41 – 80	>50 – 100	>25 – 50	>100 – 130	>9 – 11	>200 – 240	>20 – 40
Ruim	81 – 120	>100 – 150	>50 – 75	>130 – 160	>11 – 13	>240 – 320	>40 – 365
Muito Ruim	121 – 200	>150 – 250	>75 – 125	>160 – 200	>13 – 15	>320 – 1130	>365 – 800
Péssima	>200	>250	>125	>200	>15	>1130	>800

Fonte: [9].

Para cada poluente medido é atribuído um índice com base em sua concentração. Para efeito de divulgação, a qualidade ar é considerada igual ao nível mais alto entre todos os poluentes. Essa classificação está associada aos efeitos à saúde humana [9]. A tabela abaixo apresenta o significado de cada nível.

Tabela 2 – Qualidade do ar e efeitos à saúde

Qualidade	**Índice**	**Significado**
Boa	0 – 40	Pessoas de grupos sensíveis (crianças, idosos e pessoas com doenças respiratórias e cardíacas) podem apresentar sintomas como tosse seca e cansaço. A população, em geral, não é afetada.
Moderada	41 – 80	Toda a população pode apresentar sintomas como tosse seca, cansaço, ardor nos olhos, nariz e garganta. Pessoas de grupos sensíveis (crianças, idosos e pessoas com doenças respiratórias e cardíacas) podem apresentar efeitos mais sérios na saúde.
Ruim	81 – 120	Toda a população pode apresentar sintomas como tosse seca, cansaço, ardor nos olhos, nariz e garganta. Pessoas de grupos sensíveis (crianças, idosos e pessoas com doenças respiratórias e cardíacas) podem apresentar efeitos mais sérios na saúde.
Muito Ruim	121 – 200	Toda a população pode apresentar agravamento dos sintomas como tosse seca, cansaço, ardor nos olhos, nariz e garganta e ainda falta de ar e respiração ofegante. Efeitos ainda mais graves à saúde de grupos sensíveis (crianças, idosos e pessoas com doenças respiratórias e cardíacas).

Péssima	>200	Toda a população pode apresentar sérios riscos de manifestações de doenças respiratórias e cardiovasculares. Aumento de mortes prematuras em pessoas de grupos sensíveis.

Fonte: [9].

O alto índice de qualidade do ar está associado a diversos problemas de saúde e ambientais. De acordo com [8], os impactos da poluição do ar estão relacionados às mortes prematuras, doenças pulmonares e cardiovasculares, acidentes vasculares cerebrais, disposição ao câncer e ao diabetes, e problemas cognitivos de crianças e idosos. Além disso, [11] aponta que a poluição do ar pode ser causador de risco para outros problemas como o baixo peso ao nascer, anemia falciforme e partos prematuros. Da mesma forma a qualidade do ar tem um impacto direto no meio ambiente, são exemplos disso, segundo [8], "a ação do ozônio na redução da produtividade agrícola, a ocorrência de chuva ácida e a acidificação de lagos e rios pela deposição de sulfato e nitrato".

Dados os riscos associados à qualidade do ar ruim, fica evidente a importância do monitoramento contínuo e abrangente. Segundo relatório do Instituto de Energia e Meio Ambiente (IEMA) apenas 13 dos 26 estados brasileiros possuem redes de monitoramento [12]. Um dos efeitos da falta de monitoramento é a incapacidade de fiscalizar o cumprimento de normas governamentais [8]. Enfatiza que a falta de dados é uma importante fragilidade que inviabiliza a avaliação da implementação e os impactos das políticas públicas relativas à qualidade do ar.

Nesse contexto, a aplicação da Internet das Coisas (IoT) oferece uma alternativa viável e eficiente para expandir a capacidade de monitoramento. Por meio do uso de dispositivos embarcados juntamente com sensores específicos para monitorar a condição do ar é possível implementar uma rede de monitoramento de baixo custo, com grande cobertura e conectividade. No entanto, apenas a coleta de dados brutos não é suficiente para um monitoramento eficiente e em larga escala. Em muitas regiões, a conectividade com a nuvem é limitada ou intermitente. Assim, para superar essa limitação as técnicas de *edge computing* e tinyML são de grande valor. A aplicação de técnicas de tinyML permite que os próprios dispositivos sejam capazes de interpretar os dados coletados, identificando padrões e detectando anomalias de maneira local, diminuindo a necessidade de comunicação e envio de dados.

1.2 Objetivos

A proposta deste trabalho foi desenvolver um dispositivo portátil capaz de detectar anomalias na qualidade do ar em tempo real. Além disso, o algoritmo de aprendizado de máquina implementado deveria ser do tipo não supervisionado onde não existe a necessidade de dados rotulados [5, p. 225]. Este trabalho delimita sua análise às séries temporais de sensores de qualidade do ar, focando especificamente nas medições de Monóxido de Carbono (CO) e Ozônio (O_3). Tal seleção justifica-se pela disponibilidade comercial e pelo baixo custo dos sensores. A análise de outros poluentes, como partículas inaláveis (PM2.5), Dióxidos de Nitrogênio (NO_2) ou Dióxido de Enxofre (SO_2), foge ao escopo desta pesquisa.

1.3 Organização do Trabalho

O presente trabalho é constituído por 5 capítulos conforme a descrição que segue:

Capítulo 1: Contextualiza o desenvolvimento do trabalho e apresenta as motivações para a sua realização.

Capítulo 2: Apresenta a fundamentação teórica usada como base no desenvolvimento do projeto.

Capítulo 3: Descreve os materiais e métodos utilizados na implementação do dispositivo e na avaliação da implementação.

Capítulo 4: Discussão do resultados obtidos

Capítulo 5: Apresenta as conclusões do trabalho.

2 REFERENCIAL TEÓRICO

2.1 Detecção de anomalias

Anomalia é um conceito intuitivo, mas de difícil definição formal. Segundo Hawkins [13], uma definição intuitiva de anomalia seria uma observação que difere tanto das outras que gera uma suspeita se ela foi gerada pelo mesmo processo. Dessa definição pode-se perceber que uma anomalia não pode ser identificada longe de um contexto, ou seja, um dado não pode ser considerado anômalo isoladamente. Com isso, pode-se dizer que uma anomalia é um dado não usual, incomum em relação a um conjunto de dados que o contém.

Dá-se o nome de detecção de anomalias ao processo ou algoritmo capaz de identificar dados que fogem do esperado. Adari [14] define a detecção de anomalias como o processo de identificar dados ou padrões de dados incomuns para um certo conjunto de dados. Além disso, Kennedy [15] acrescenta que o que se entende por comuns ou normais pode mudar, entretanto, a detecção de anomalias pressupõe que a maioria dos dados em um conjunto pode ser considerado normal. Esse desvio do padrão é o que torna a identificação de anomalias tão valiosa, já que a maioria dos sistemas é projetado para operar sob condições normais e a detecção prematura da anormalidade tende a minimizar os riscos. Entretanto, nem todos os dados que não seguem o comportamento esperado são relevantes para a detecção.

Erros na aquisição de dados ou ruídos podem ser considerados anomalias dependendo do processo de detecção, contudo, apesar de se enquadrarem como anomalias, não representam dados de interesse ou valor para a análise. Boniol [16] destaca que dependendo da aplicação as anomalias podem ser constituídas de ruídos e dados errôneos ou então de dados de interesse real. No primeiro caso, os dados devem ser corrigidos ou removidos para não comprometer nenhuma análise posterior, e no último caso, as anomalias podem identificar eventos significativos como falhas ou mudanças de comportamento e são a base para análises subsequentes. Cabe ao processo de detecção de anomalias diferenciar entre esses dois casos. A falha em distinguir cada tipo de anomalia pode acarretar em um grande número de falsos positivos e falsos negativos, o que compromete a acurácia e a confiabilidade do processo de detecção. A detecção precisa de anomalias permite que suas aplicações sejam confiáveis e eficazes.

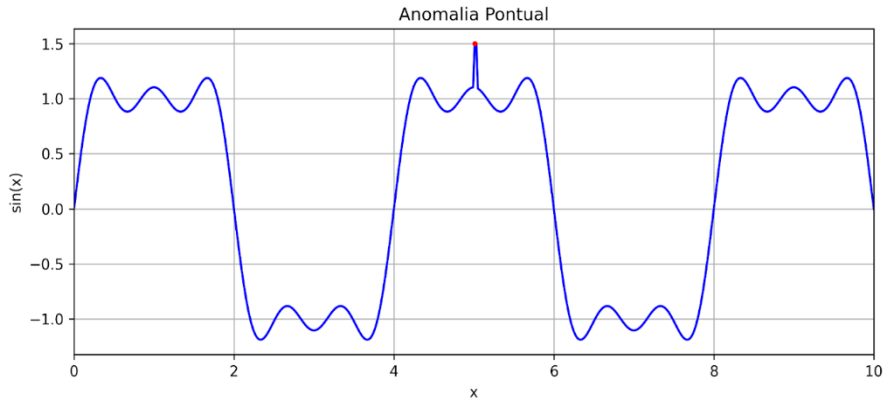
A detecção de anomalias possui aplicações em diversas áreas conforme discutido por diferentes autores citados em [17]. Entre elas, destacam-se a remoção de ruídos em dados, a

prevenção de ataques de envenenamento de dados e, na área médica, a detecção de condições anormais a partir de sensores IoT e o monitoramento de idosos. Em ambientes domésticos inteligentes, pode indicar aumentos atípicos de temperatura possivelmente relacionados a invasões, enquanto, no setor industrial, pode auxiliar no gerenciamento de recursos e na identificação de variações ambientais como temperatura, umidade e fumaça. Apesar da diversidade de contextos de aplicação, as anomalias podem ser classificadas em tipos comuns à todos os contextos.

2.1.1 Tipos de anomalias

As anomalias podem ser categorizadas em relação a um conjunto global de dados, à um subconjunto de dados ou à uma sequência de dados. Boniol [16] define três tipos de anomalias: pontual, contextual e coletiva. As anomalias pontuais são aquelas que divergem significativamente de todo um conjunto de dados. A figura 1 apresenta um exemplo desse tipo de anomalia. Observar-se que o ponto vermelho difere de todos os outros pontos e está fora da distribuição dos dados

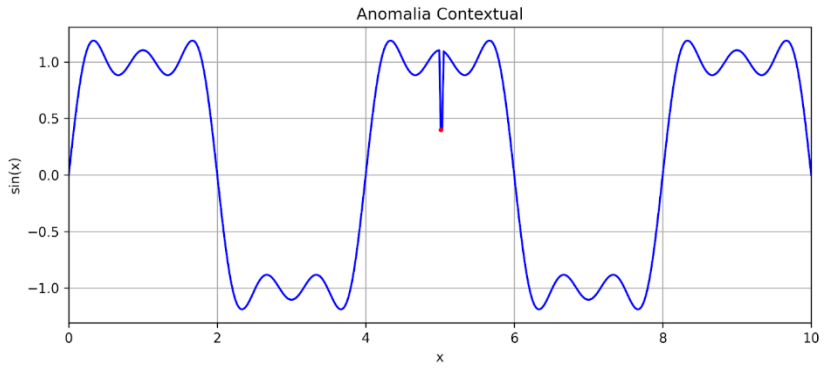
Figura 1 – Representação gráfica de uma anomalia pontual



Fonte: Elaborado pelo autor.

Segundo Boniol [16], as anomalias contextuais são aquelas que parecem compatíveis com a distribuição global, mas se tornam anômalas quando analisadas em um contexto específico, como em relação aos dados vizinhos. A figura 2 apresenta um exemplo de anomalia contextual. Diferentemente do ponto vermelho da figura 1, que está fora da distribuição, nesse caso, o valor encontra-se dentro da distribuição. O que caracteriza esse ponto como anômalo é a discrepância em relação aos pontos próximos.

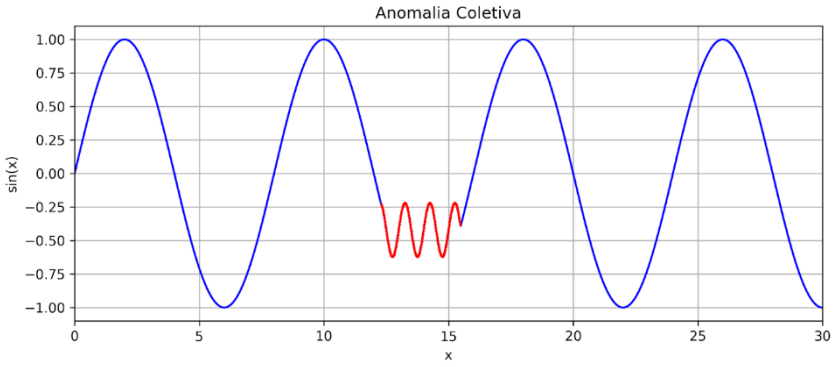
Figura 2 – Representação gráfica de uma anomalia contextual



Fonte: Elaborado pelo autor.

As anomalias coletivas referem-se a sequências de pontos que não seguem o padrão típico observado em uma distribuição. A figura 3 mostra um exemplo de uma anomalia sequencial. Diferentemente dos exemplos anteriores, aqui não é um único ponto que se destaca, mas sim um subconjunto de dados que, considerado em conjunto, forma um padrão anômalo.

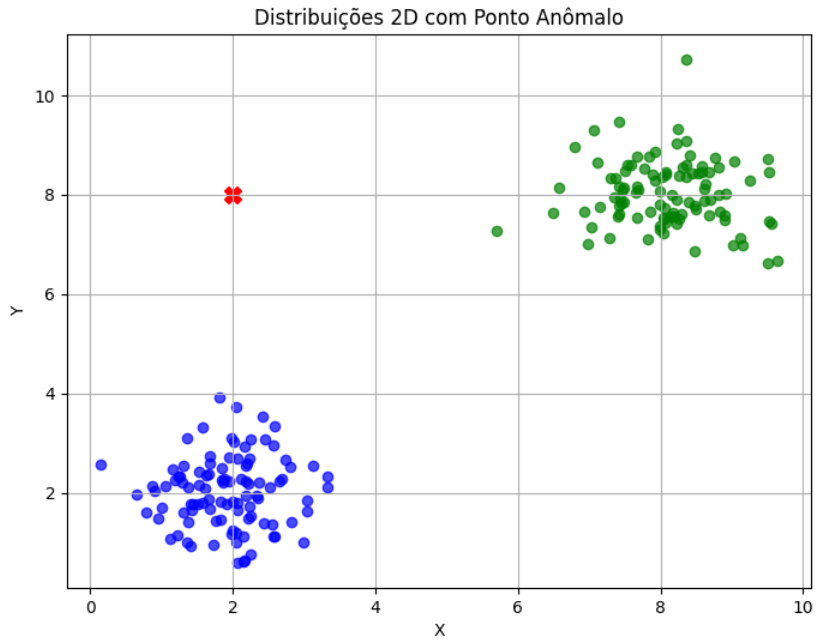
Figura 3 – Representação gráfica de uma anomalia coletiva



Fonte: Elaborado pelo autor.

Além desses tipos básicos, também existem as anomalias multidimensionais. Conforme Kennedy [15] aponta, existem duas maneiras de identificar uma anomalia multidimensional: Esse ponto é incomum em uma única dimensão ou então a combinação de valores em múltiplas dimensões é atípica. A figura 4 apresenta um exemplo de uma combinação incomum de valores. Observa-se que, ao analisar apenas o eixo x o valor da anomalia está próximo ao centro da distribuição em azul e, ao considerar apenas o eixo y, encontra-se próximo ao centro da distribuição verde. Entretanto quando se analisa a combinação dos dois eixos fica evidente que o ponto vermelho é uma anomalia.

Figura 4 – Representação gráfica de uma anomalia multidimensional



Fonte: Elaborado pelo autor.

2.1.2 Etapas do Processo de Detecção de Anomalias

Apesar da aplicação em diferentes domínios, os processos de detecção de anomalias apresentam uma estrutura semelhante na maior parte das aplicações, conforme aponta Boniol [16]. Segundo os autores, pode-se dividir o processo de detecção em quatro etapas: pré-processamento de dados, aplicação do método de detecção, *scoring* e pós-processamento. O pré-processamento é a etapa inicial e inclui todas as transformações necessárias para preparar ou facilitar a aplicação dos métodos subsequentes. Em seguida, diferentes métodos de detecção podem ser empregados cada um com uma abordagem específica. Esses métodos de detecção e suas respectivas abordagens serão discutidos com maior detalhe na próxima seção.

Na etapa de *scoring*, os resultados desses métodos são convertidos em valores numéricos que representam o grau de anormalidade de um ponto ou subsequência. Por fim, na última etapa, os pontos ou intervalos anômalos são extraídos, geralmente por meio da definição de um limiar que separa valores normais de valores anômalos usando os valores do grau de anormalidade de cada ponto dados pela etapa anterior.

2.1.3 Classificação dos métodos de detecção

A classificação das diversas abordagens de detecção, depende em grande parte, do contexto de aplicação. Cada contexto lida com tipos específicos de variáveis, como dados tabulares, séries temporais, dados categóricos, textos e imagens, entre outros. Por sua vez, cada tipo de variável apresenta características e desafios próprios, que demandam métodos específicos de detecção. Considerando que o tema desta monografia é a detecção de anomalias na qualidade do ar, representada por séries temporais obtidas por sensores conectados à dispositivos IoT, adotou-se a classificação proposta por Samara et al [18]. Este trabalho foi selecionado por sua abrangente revisão de técnicas de detecção de anomalias especificamente no contexto da Internet das Coisas. De acordo com os autores, os métodos de detecção podem ser classificados em 7 categorias: baseados em estatística, baseados em agrupamentos, baseados nos vizinhos mais próximos, baseados em classificação, baseados em inteligência artificial, baseados em decomposição espectral e métodos híbridos.

Nesta monografia, a categoria de métodos baseados em vizinhos mais próximos será abordada dentro da seção de métodos de distância. Essa escolha se justifica pois a análise por distância é o princípio fundamental desta abordagem, sendo a análise de vizinhança sua forma mais comum de implementação. Além disso, também foi alterada a categoria de inteligência artificial para métodos baseados em *deep learning*, já que o nome antigo é mais abrangente e inclui algoritmos de outras categorias, enquanto o último é mais específico e contempla apenas os algoritmos da própria categoria.

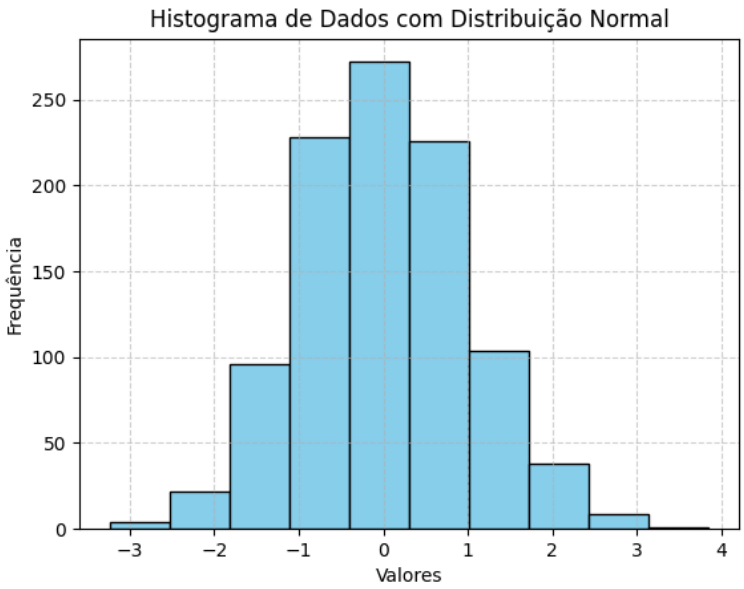
2.1.3.1 Métodos baseados em estatística

Os métodos baseados em estatística são aqueles que assumem ou estimam um modelo da distribuição de dados. Os dados que têm baixa probabilidade, de acordo com o modelo, são considerados anomalias [18]. Esses métodos podem ser divididos em duas subcategorias: os métodos paramétricos, no qual se assume que os dados são gerados a partir de uma distribuição conhecida, por exemplo, a distribuição normal; e os métodos não paramétricos em que a distribuição não é conhecida [18].

Uma abordagem não paramétrica comum citada por Samara et al. [18] é o histograma, ilustrado na Figura 5. Nesse método não paramétrico divide-se um conjunto de dados em intervalos regulares e então conta-se a quantidade de dados presentes em cada intervalo para gerar uma taxa de ocorrência. Conforme explicam os autores [18], intervalos com altas taxas

de ocorrência são considerados normais, enquanto intervalos com taxas nulas ou pequenas são considerados anomalias.

Figura 5 – Exemplo de um histograma



Fonte: Elaborado pelo autor.

Uma outra abordagem muito usada e simples de se aplicar é o método *z-score* , descrito por Kennedy [15]. O autor explica que nesse método paramétrico utiliza-se a média e o desvio padrão de uma distribuição para determinar as anomalias. Para encontrar o *z-score* de um valor (x), subtrai-se a média (μ) e divide-se o resultado da subtração pelo desvio padrão (σ). Segundo Kennedy [15], em seguida é necessário definir um valor limite. Qualquer valor maior que esse limite, ou então menor que o negativo desse valor é considerado uma anomalia.

$$z = \frac{x-\mu}{\sigma} \tag{2.1}$$

Samara et al. [18] apontam que, os métodos estatísticos funcionam bem quando o modelo é corretamente definido, nesse caso não é necessário manter os dados usados para criar o modelo. No entanto, os autores [18] alertam que para garantir que os métodos funcionem corretamente é necessário ter um conhecimento prévio sobre a distribuição dos dados, o que nem sempre é possível, ou então obter esse conhecimento pela aquisição de dados, o que nem sempre é fácil de se realizar. Eles concluem [18] que os métodos paramétricos são muitas vezes inadequados para o uso em IoT e os métodos não paramétricos são computacionalmente

custosos para dados multivariados, o que pode potencialmente inviabilizar a sua aplicação na detecção de anomalias na qualidade do ar.

2.1.3.2 Métodos baseados em agrupamentos

Os métodos de agrupamentos dividem os dados, como o próprio nome indica, em agrupamentos utilizando a similaridade ou proximidade desses dados para tal. Pontos que não pertencem a nenhum agrupamento ou agrupamentos com poucos dados são considerados anomalias, enquanto agrupamentos muitos dados são considerados normais [18].

Um algoritmo de agrupamento muito popular é o DBSCAN, que Géron [19] define como um método que define *clusters* como regiões contínuas de alta densidade. Além disso, conforme Kennedy [15] aponta, esse algoritmo possui a propriedade de permitir que alguns pontos não pertençam a nenhum agrupamento, o que o torna especialmente útil na detecção de anomalias. O DBSCAN, conforme descrito por Géron [19], funciona da seguinte maneira: para cada ponto ele conta quantos outros pontos estão próximos até uma distância ε (epsilon). Se esse ponto tiver pelo menos N pontos dentro dessa distância, ele é considerado uma instância core, isto é, ele pertence a uma região densa. Todos os vizinhos e todos as outras instâncias core, que estão a uma distância menor do que ε, pertencem ao mesmo *cluster*. Finalmente, Géron [19] explica que qualquer ponto que não é uma instância core ou que não é parte de algum cluster é considerado uma anomalia.

Kennedy [15] indica que uma grande vantagem dos métodos de agrupamento é que independente do modo que o agrupamento é feito e da forma que as anomalias são identificadas, eles são aplicáveis para diversos formatos: dados tabulares, séries temporais, texto, imagens, etc; desde de que uma métrica de distância ou similaridade pode ser definida. Além disso, estes métodos não exigem nenhum conhecimento prévio do conjunto de dados para a sua aplicação. No entanto, Samara et al. [18] alertam que, para dados multivariados o cálculo das distâncias pode se tornar computacionalmente custosos.

2.1.3.3 Métodos baseados em distâncias

Os métodos de distância, conforme explica Kennedy [15], tentam determinar pontos que possuem poucos dados similares utilizando métricas de distância. A ideia geral é identificar pontos que não possuem muitos pontos próximos ou então cuja distância até os vizinhos é grande. Para isso é necessário escolher uma métrica de distância adequada para o contexto da

aplicação. O autor [15] aponta que a principal métrica é a distância euclidiana. Entretanto outros estudos [16, 19] também citam métricas como Mahalanobis, Hamming e Minkowski.

Kennedy [15] aponta o *K-nearest neighbour* (KNN) como um dos principais algoritmos dessa categoria. O autor descreve que, nesse método, para cada ponto, é medido sua distância até o seu k vizinho mais próximo. Essa distância é considerada o "grau de anormalidade" do ponto e é usada para categorizar as anomalias [15].

Uma vantagem destacada por Kennedy [15] é que os métodos baseados em distância não exigem nenhum conhecimento da distribuição de dados, além disso, podem ser aplicados para diferentes tipos de dados. Entretanto, como aponta Samara et al. [18], os cálculos das distâncias podem exigir muita capacidade de processamento o que pode inviabilizar a sua aplicação em microcontroladores de menor capacidade.

2.1.3.4 Métodos baseados em classificação

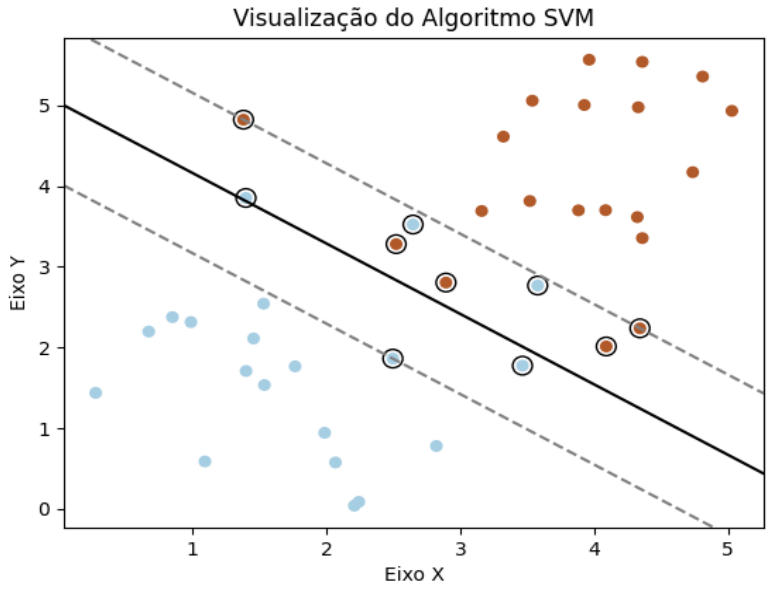
Os métodos baseados em classificação treinam um modelo, utilizando uma amostra dos dados, para classificar os pontos entre dados normais e anomalias. A criação desses modelos é dividida em duas fases: treinamento, na qual o modelo é criado usando um conjunto dos dados; e uma fase de validação onde um outro conjunto de dados é usado para avaliar a acurácia do modelo [18].

Kennedy [15] apresenta o Algoritmo OCSVM (*one class SVM*) como um exemplo de método baseado em classificação. O autor explica que este algoritmo utiliza o conceito de *one-class* para treinar o modelo. Na prática isso significa que o algoritmo vai assumir que existe apenas uma única classe na amostra e vai criar um modelo para representá-la tão fielmente quanto possível. Depois que o modelo é criado ele pode ser usado para testar novos dados. Qualquer dado que esteja em conformidade com o modelo é considerado normal e se não estiver é considerado anomalia. Kennedy [15] nota, que por ser *one-class*, esse algoritmo é sensível a anomalias presentes no conjunto de dados de treinamento, uma vez que o modelo vai considerá-las como pertencentes à mesma classe de dados normais. Entretanto, o autor aponta que esse algoritmo é especialmente eficaz na detecção de novidades, que é um subcategoria da detecção de anomalias onde não se tem nenhuma informação sobre as características das anomalias.

Conforme descrito por Géron [19], o OCSVM é uma adaptação do algoritmo SVM (*support vector machine*) para o caso de uma classe utilizado para permitir uma modelagem não supervisionada. Já o SVM, ilustrado na figura 6, é um classificador supervisionado usado para distinguir dados em duas ou mais classes. O autor explica [19] que esse algoritmo tenta

criar um hiperplano entre duas classes de dados para maximizar a margem entre a instância mais próxima das duas classes, essa fronteira é chamada de limite de decisão (*decision boundary*). As duas instâncias usadas como base para o cálculo da distância são chamadas de vetores de suporte e dão nome ao algoritmo. Além disso, Géron [19] também aponta que o SVM faz o uso de funções de kernel, que permitem mapear os dados para um espaço de dimensão superior, possibilitando a criação de um limite de decisão não-linear para separar classes complexas.

Figura 6 – Representação visual do algoritmos SVM em duas dimensões



Fonte: Elaborado pelo autor.

Segundo Samara et al. [18], os métodos de classificação possuem excelentes resultados para a detecção de anomalias. Eles podem ser supervisionados ou não-supervisionados, sendo o último caso muito utilizado na detecção de novidades. Entretanto, os autores [18] alertam que esses métodos consomem mais recursos computacionais do que os métodos baseados em estatísticas e agrupamentos, o que pode representar um desafio para a sua aplicação em microcontroladores.

2.1.3.5 Métodos baseados em *Deep Learning*

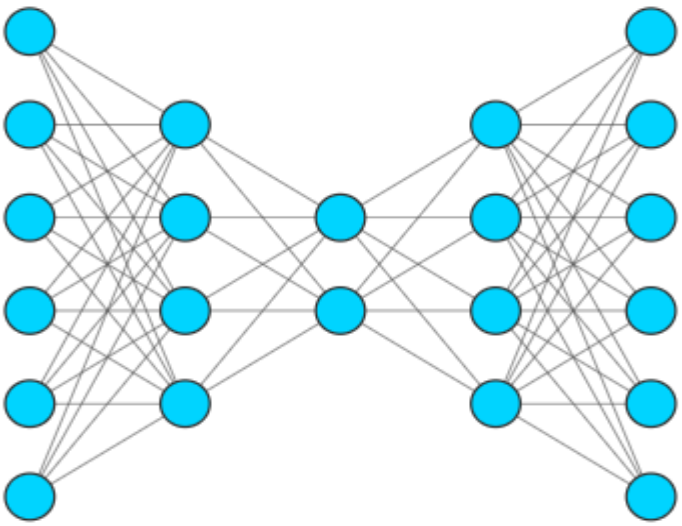
Segundo Kennedy [15], os métodos baseados em *deep learning* utilizam alguma forma de rede neural detectar de anomalias. Conforme explica Iodice [20], uma rede neural é formada por várias camadas utilizadas para aprender padrões de um conjunto de dados. Cada camada

por sua vez é formada por diversos neurônios. Um neurônio recebe várias entradas e produz uma saída. Esse processo envolve primeiro uma transformação linear (a soma ponderada das entradas mais um viés) e, em seguida, a aplicação de uma função de ativação não-linear a este resultado, o que permite à rede aprender padrões complexos.

Kennedy [15] aponta o algoritmo *autoencoder* como um tipo de rede neural utilizado para detecção de anomalias. O autor descreve sua arquitetura como sendo composta por duas partes. A primeira é chamada de *encoder*, onde as camadas da rede neural diminuem progressivamente a quantidade de neurônios até chegar na metade da rede. A partir desse ponto começa a segunda parte, chamada de *decoder*. Essa parte geralmente é simétrica a primeira e tem como objetivo reconstruir a entrada inicial tão fielmente quanto possível. A figura 7 mostra um exemplo de um *autoencoder*. O autor explica [15] que os erros de reconstrução podem ser usados para detectar anomalias. Um ponto com um alto erro indica que ele não segue o mesmo padrão que permitiu a reconstrução dos dados após a compressão.

Kennedy [15] observa que os métodos baseados em *deep learning* exigem maior quantidade de dados e tempo para serem treinados e um maior *fine-tuning* dos seus parâmetros. O autor [15] também aponta que, esses métodos apresentam melhores resultados para dados não estruturados como imagens, vídeo e textos e dados de séries temporais.

Figura 7 – Representação visual das camadas de uma rede neural do algoritmo autoencoder



Fonte: Elaborado pelo autor.

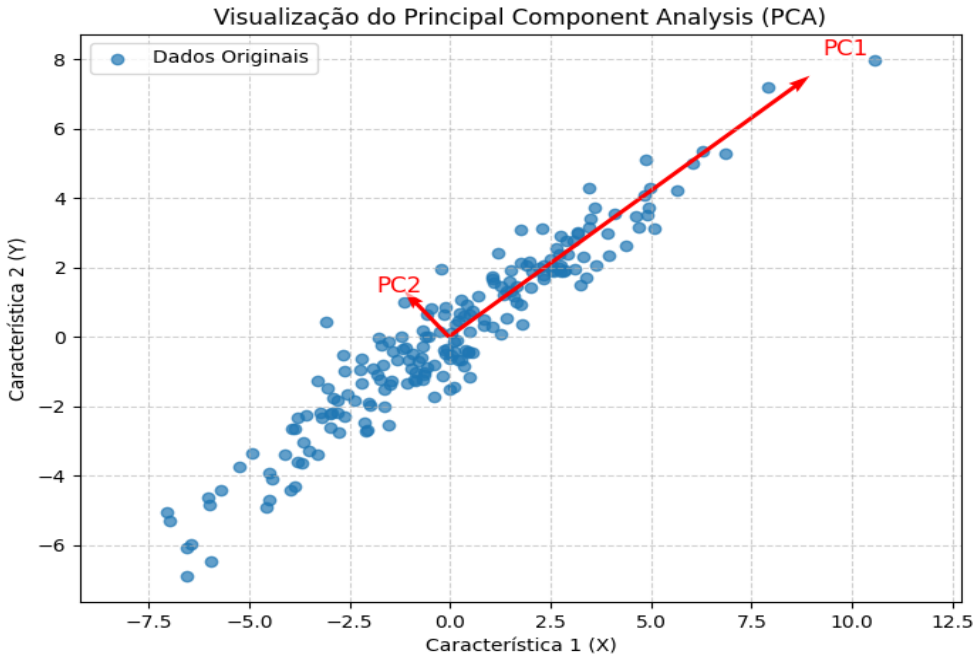
2.1.3.6 Métodos baseados em decomposição espectral

Segundo Samara et al. [18], os métodos baseados em decomposição espectral se baseiam em algoritmos de redução de dimensionalidade para simplificar um conjunto de dados e preservar apenas os componentes com maior variância. O principal algoritmo dessa categoria é o PCA (*Principal Component Analysis*) ilustrado na figura 8.

Géron [19] explica que esse algoritmo identifica o hiperplano mais próximo aos dados e então projeta os dados nele. Primeiro ele identifica o eixo que possui a maior variância no conjunto de dados, então ele identifica um segundo eixo, ortogonal ao primeiro, de maior variância. O algoritmo continua identificando eixos de maior variância até que o número de eixos seja igual ao número de dimensões do conjunto de dados [19]. No contexto da detecção de anomalias, Samara et al. [18] apontam que pontos com alto erro de reconstrução ou que divergem muito da variabilidade apresentada pelos primeiros componentes são considerados anômalos.

Samara et al. [18] concluem que esses métodos podem ser usados em uma grande variedade de dados e dados com muitas dimensões. Entretanto, o grande consumo de recursos computacionais inviabiliza a sua aplicação em microcontroladores.

Figura 8 – Representação visual de um algoritmos de decomposição espectral



Fonte: Elaborado pelo autor

2.1.3.7 Métodos híbridos

Os métodos híbridos são aqueles que combinam dois ou mais algoritmos de detecção a fim de obter um resultado mais acurado. Kennedy [15] explica que esses métodos têm dois objetivos: melhorar a confiabilidade do sistema diminuindo a taxa de falso positivos, isto é, pontos normais que foram identificados como anomalias; e melhorar a identificação de anomalias diminuindo a taxa de falsos negativos, isto é, anomalias que foram consideradas normais. O autor [15] justifica essa abordagem pelo fato de que os detectores de anomalias, no geral, identificam apenas algum tipo específico de anomalias e deixam passar outros, por isso combinar mais de um detector pode melhorar significativamente os resultados. Contudo, Samara et al. [18] apontam que a combinação de dois ou mais algoritmos ou modelos pode aumentar em muito o uso de recursos computacionais que dificulta sua aplicação em dispositivos IoT.

2.1.4 Detecção de anomalias em IoT

Cook et al. [21] definem a Internet das Coisas (IoT) como um paradigma de programação onde dispositivos, compostos por sensores, atuadores e computadores, interagem com o ambiente e se comunicam entre si sem a necessidade de intervenção humana. Dentro desse contexto, ss autores [21] apontam a existência de uma ampla gama de aplicações de detecção de anomalias abrangendo desde o monitoramento de sistemas industriais, até a identificação de falhas em redes de distribuição de energia. Entretanto, existem diversos desafios que dificultam a aplicação de técnicas de detecção de anomalias.

Segundo Cook et al. [21] séries temporais são o principal formato de dados que os dispositivos IoT capturam, e por isso demandam atenção especial durante o processamento. Eles explicam [21] que a natureza desses dados é frequentemente não estacionária, podendo apresentar fenômenos como o *concept drift* (mudança da distribuição estatística dos dados no decorrer do tempo) e sazonalidade. Nesses casos é necessário que o modelo seja capaz de aprender os novos padrões a fim de detectar corretamente as anomalias. Outra dificuldade apontada pelos autores [21] está relacionada a falta de conhecimento prévio de um contexto de aplicação ou então a falta de dados históricos o que torna inviável a aplicação de algoritmos supervisionados.

Todavia, Cook et al. [21] afirmam que, o maior desafio na detecção de anomalias em IoT é o custo computacional. Eles descrevem [21] que em um modelo tradicional de detecção

os dispositivos IoT seriam usados apenas para registrar os dados. Esses dados então seriam enviados para um servidor para serem processados. Esse modelo tradicional permite o uso de um grande poder computacional, entretanto para aplicações que exigem um ação em tempo real a latência da resposta pode tornar a aplicação inviável. Nesses casos, concluem os autores [21], não existe outra alternativa, senão processar os dados na borda. Dessa forma, a implementação de detecção de anomalias em IoT requer o uso de técnicas de otimização de recursos computacionais [21]. Essas técnicas devem reduzir o consumo de memória, CPU e potência; enquanto se busca minimizar a perda de acurácia do modelo de detecção [21]

2.2 TinyML: Otimização de Modelos para Borda

Warden e Situnayake [22] definem TinyML como um paradigma que possibilita a aplicação de algoritmos de Aprendizado de Máquina (ML) em dispositivos de borda com recursos computacionais limitados e baixo consumo de energia, na ordem de alguns miliwatts. Já Iodice [20] entende o conceito como um conjunto de tecnologias de ML e sistemas embarcados que permite a criação de sistemas inteligentes capazes de perceber o ambiente por meio de sensores, processar os dados aplicando modelos de ML e, então, agir com base nos resultados desses modelos [20].

A utilização de técnicas de TinyML apresenta vantagens em comparação às abordagens tradicionais, em que os dados são enviados para processamento em nuvem. A primeira vantagem, apontada por Iodice [20], é a latência. A transmissão de dados para um servidor não é instantânea e pode comprometer o desempenho de aplicações que requerem uma resposta rápida. Além disso, o autor [20] nota que a transmissão e recepção de dados é uma operação de grande consumo energético e mesmo com protocolos de baixa potência, como o *Bluetooth*, consome mais energia que as computações realizadas pela CPU. Dessa forma, como conclui Iodice [20], para maximizar o tempo de autonomia de dispositivos alimentados por baterias é necessário reduzir a quantidade de dados transmitidos para a nuvem. Por fim, Tsoukas et al. [23] destacam que o processamento local evita a exposição de dados sensíveis a possíveis ameaças de segurança, como ataques *man-in-the-middle* e *eavesdropping*. Logo, garante-se maior segurança e privacidade ao usuário ao processar os dados na borda.

Outros fatores importantes para a viabilização da aplicação de TinyML, apontados por Iodice [20], estão relacionados às características dos microcontroladores. Esses dispositivos são baratos, de fácil programação, e são facilmente integrados a uma ampla gama de sensores. Apesar de suas limitações são potentes suficientes para executar algoritmos complexos de *deep*

learning. Segundo Iodice [20], essas características justificam a sua popularidade e sua presença em vários dispositivos eletrônicos do dia a dia e em outros setores como a indústria e a área da saúde.

Entretanto, a aplicação de aprendizado de máquina na borda enfrenta diversos desafios, principalmente relacionados a limitação de recursos computacionais. No geral os dispositivos possuem apenas alguns kilobytes de memória RAM e, em alguns casos, os processadores não possuem aceleração de hardware para aritmética de ponto flutuante [20]. Na revisão de literatura feita por Capogrosso [24], os autores constataram que os requisitos de hardware para memória são menores do que 1MB, normalmente estão entre 64 KB e 256 KB, e as unidades de processamento operam na faixa de 40 a 400 MHz. Conforme ressaltado por Capogrosso et al. [24], essas características ressaltam a importância de técnicas de otimização específicas para viabilizar a aplicação do algoritmos de ML.

Outros desafios estão relacionados à variedade de dispositivos e aos diferentes ambientes de aplicação. Tsoukas et al. [23] explicam que a grande heterogeneidade de microcontroladores inviabiliza a criação de um *framework* universal para treinamento, otimização e implantação de modelos. Um modelo construído para funcionar em uma arquitetura específica de MCU pode não funcionar em outra, mesmo que as especificações de hardware sejam semelhantes. Tsoukas et al. [23] concluem, portanto, que um modelo é aplicável somente para o contexto e o hardware em que ele foi desenvolvido. Ademais, Iodice [20] nota que o ambiente de implantação pode ser fonte de adversidades. Condições ambientais como clima, calor e poeira podem interferir na execução da aplicação.

2.2.1 Otimização de Modelos para Sistemas Embarcados

Os métodos de otimização são utilizados para viabilizar a implementação de modelos complexos de aprendizado de máquinas em dispositivos IoT. A aplicação dessas técnicas geralmente produz uma menor utilização de memória e um menor consumo de energia, além da diminuição do tempo de resposta [24]. As principais abordagens presentes na literatura, que serão detalhadas a seguir, incluem a Quantização, a Poda, a Destilação de Conhecimento, a Otimização de hiperparâmetros, a Busca de arquitetura neural e as otimizações baseadas em hardware.

- **Quantização:** Conforme descrito por Capogrosso et al. [24], a quantização é o processo de reduzir a quantidade de bits utilizados para representar pesos e as ativações de uma rede neural. Ao invés de utilizar uma representação numérica

de ponto flutuante com 32 bits, utiliza-se representações com menos bits. Por exemplo, uma representação do tipo float com 16 bits ou do tipo inteiro com 8 bits. Essa técnica produz modelos mais compactos, além de reduzir significativamente os custos de computação, sem reduzir significativamente a acurácia do modelo.

- **Poda:** A poda, como definida por Capogrosso et al. [24], é o processo de remover conexões com pesos próximos de zero entre as camadas de um rede neural. Ela serve para reduzir o tamanho total do modelo e melhorar o tempo de inferência. A aplicação da poda pode ocorrer tanto durante o processo de treinamento quanto em um modelo já treinado. Durante o treinamento ela serve para evitar o sobreajuste e após ela ajuda a evitar redundância, melhorando a eficiência do modelo.
- **Destilação do conhecimento:** Segundo os autores [24], nessa técnica um modelo maior e mais complexo é utilizado para treinar um modelo menor e mais simples. Esse processo reduz a demanda computacional de um modelo mantendo uma precisão aceitável.
- **Otimização de hiperparâmetros:** Capogrosso et al. [24] indica que esse conjunto de técnicas automatiza a busca de hiperparâmetros de um modelo para maximizar a performance em um contexto específico. Para esse fim, algoritmos de buscas como o *Grid Search* e *Random Search* são utilizados para investigar e identificar a melhor combinação de hiperparâmetros.
- **Busca de arquitetura neural:** A busca de arquitetura neural, ou *neural architecture search* (NAS), segundo os autores [24], é um método utilizado para automatizar o processo de busca por arquiteturas ideais de redes neurais para um fim específico. Além disso, Tsoukas et al. [23] apontam que esse processo seleciona o modelo de maior acurácia possível dentro de um espaço pré definido de NNs.
- **Otimizações Baseadas em Hardware:** Tsoukas et al. [23] explicam que, além das otimizações de software, a performance de modelos de aprendizado de máquina pode ser maximizada por meio de otimizações de hardware. Segundo os autores [23], a aceleração por hardware serve para aumentar a velocidade de processamento ou permitir o processamento paralelo. Os autores apontam que [23] essas otimizações focam principalmente em melhorar a performance de

> operações matriciais, uma vez que essas operações formam a base para muitos algoritmos de ML. A literatura, conforme descrito no estudo [23], detalha diversas arquiteturas especializadas para este fim. Tsoukas et al. [23] citam como exemplo o *framework* PRIME, que acelera aplicações de redes neurais ao utilizar a memória RAM para realizar multiplicações de matriz-vetor, alcançando melhorias de performance de até 2.360 vezes e reduzindo o consumo de energia em 895 vezes. Outra abordagem mencionada pelos autores [23] é o acelerador *SmartShuttle*, que foca em otimizar o acesso à memória externa (*off-chip*), utilizando esquemas dinâmicos de reuso de dados para se adaptar a diferentes camadas de modelos de *deep learning*.

As abordagens apresentadas não são mutuamente exclusivas e podem se beneficiar de uma aplicação em conjunto. Além disso vale ressaltar que as técnicas quantização, poda e destilação de conhecimento são aplicáveis durante ou após o treinamento do modelo, enquanto as técnicas de otimização de hiperparâmetros e busca e arquitetura são aplicadas antes do treinamento.

2.2.2 Fluxo de Trabalho

Capogrosso et al. [24] apontam que o fluxo de trabalho para o desenvolvimento de soluções em tinyML possui duas abordagens principais: a orientada a aprendizado de máquina e a orientada ao desenvolvimento do hardware. Segundo os autores [24], na abordagem orientada a aprendizado de máquina o foco é no design e na otimização de um modelo a fim de garantir a sua aplicação em um dispositivo específico. O hardware é tratado como um componente imutável ou com pouca possibilidade de alteração. As etapas do fluxo de trabalho nesse caso consistem no design do modelo, otimização, implementação no dispositivo e avaliação da performance do modelo [24]. Já na abordagem orientada a hardware, o estudo [24] aponta que o foco é o desenvolvimento de arquiteturas de hardware otimizadas para a aplicação de algoritmos de aprendizado de máquina. Para essa abordagem as etapas do fluxo de trabalho são: desenvolvimento do hardware, implementação do modelo e avaliação da performance do modelo [24].

Os autores [24] mencionam também uma terceira abordagem é o co-design, que integra as duas abordagens anteriores desde o início do projeto. Nesse caso especialistas de ambas as áreas colaboram desde a definição de requisitos até a prototipagem, escolhendo componentes

de hardware e software com uma visão global do sistema. Dessa forma, Capogrosso et al. [24] concluem que, a otimização de modelos e o design de hardware são processos interligados que visam alcançar performance superiores e menor consumo de recursos do que as abordagens tradicionais.

3 MATERIAIS E MÉTODOS

3.1 Materiais

A seguir são apresentados os materiais utilizados para o desenvolvimento do projeto. Esse desenvolvimento pode ser dividido em duas partes: montagem do hardware para coleta de dados e, posteriormente, detecção de anomalias e o desenvolvimento do modelo de aprendizado de máquina para realizar a detecção. As partes são mutuamente dependentes. Sem os dados, não seria possível treinar um modelo e sem o modelo não seria possível realizar a detecção de anomalias.

3.1.1 Hardware

3.1.1.1 Arduino

O microcontrolador utilizado no projeto foi o Arduino Nano 33 BLE Sense Rev2. Este microcontrolador possui um processador Arm Cortex-M4F (FPU) que roda em 64 MHz, além disso ele possui 1 MB de memória flash e 256 kB de memória RAM. Uma característica importante deste microcontrolador é que seus pinos de I/O possuem um limite de tensão de 3.3V e não podem receber sinais de 5V, caso contrário pode-se danificar o microcontrolador [25]. Optou-se pela utilização desse módulo devido ao seu baixo custo e boa capacidade computacional, além de sua popularidade na literatura [21, 22, 24].

Tabela 3 – Resumo das características do Arduino

Características	**Detalhe**
Módulo	Arduino Nano 33 BLE Sense Rev2
Processador	Arm Cortex-M4F (FPU)
Clock do processador	64 MHz
Memória Flash	1 MB
Memória RAM	256 kB
Comunicação sem fio	Bluetooth 5 multiprotocol
Tensão de I/O	3.3V

Fonte: [25].

3.1.1.2 Sensores

Foram utilizados 3 sensores de gases diferentes para a coleta de dados da qualidade do ar. O sensor MQ-7 foi usado para medir a concentração do gás Monóxido de carbono (CO), um gás altamente tóxico. Esse sensor é capaz de medir concentrações entre 100 a 10.000 ppm [26]. Outro sensor utilizado foi o MQ-131, capaz de detectar ozônio, um gás incolor e reativo, em níveis baixos de concentração de 10 a 1000 ppb [27]. De modo complementar, o sensor MQ-2 foi empregado para determinar concentrações de gases inflamáveis (como o GLP, Metano, Butano, etc) e fumaça na faixa de 300 a 10.000 ppm [28]. Esses sensores possuem pinagens iguais. São alimentados por uma tensão de 5V. Possuem 4 pinos sendo dois de alimentação (GND e Vin), uma saída digital e uma saída analógica. Para este trabalho, foi utilizada apenas as saídas analógicas de cada sensor, pois elas fornecem os valores contínuos necessários para a análise da série temporal.

Tabela 4 – resumo das características dos sensores

Características	**MQ-2**	**MQ-7**	**MQ-131**
Gases detectados	Gases inflamáveis (GLP, Metano, etc.) e fumaça	Monóxido de Carbono (CO)	Ozônio (O_3)
Faixa de operação	300 a 10.000 ppm	100 a 10.000 ppm	10 a 1000 ppb
Tensão de entrada	5V	5V	5V
Sensibilidade	Ajustável via potenciômetro	Ajustável via potenciômetro	Ajustável via potenciômetro

Fonte: [27- 29].

3.1.1.3 Fonte de alimentação

A Alimentação do sistema foi fornecida por um fonte ajustável para protoboard, um dispositivo que converte uma tensão de entrada (que pode variar de 7,5 a 12 VDC ou ser fornecida via cabo USB) em saídas reguladas de 3,3V e 5V [29]. A utilização dessa fonte foi imprescindível devido a incompatibilidade das tensões do arduino (3.3V) e dos sensores (5V).

3.1.2 Software

3.1.2.1 Ambientes de Desenvolvimento

O Arduino IDE foi o ambiente de desenvolvimento utilizado para a escrita, compilação e upload de software em C++ para o Arduino. O Arduino IDE é uma aplicação multiplataforma que simplifica o processo de desenvolvimento de software embarcado. Essa IDE possui uma interface amigável e de fácil utilização, além disso é compatível com uma ampla gama de microcontroladores, incluindo placas de outros fabricantes além da linha oficial do Arduino. [30].

O Jupyter Notebook foi a ferramenta utilizada para desenvolver códigos em Python relacionados a *machine learning*. Sua interface baseada em células permite a execução individual de blocos de código, dessa forma possibilitando uma abordagem iterativa para o desenvolvimento do pré-processamento de dados, criação do modelo de aprendizado de máquina e avaliação dos resultados do treinamento [31].

O Visual Studio Code, um editor de código leve e extensível, serviu como ambiente de desenvolvimento principal para códigos feitos em Python. Ele também foi utilizado para a execução dos Jupyter Notebooks mediante uso de extensões [32].

3.1.2.2 Linguagens de programação

O C++ foi a linguagem utilizada para desenvolver o firmware executado no microcontrolador arduino. Sua capacidade de controle de baixo nível sobre o hardware foi essencial para a leitura dos sensores e controle das saídas digitais. Já o Python foi a linguagem utilizada para o desenvolvimento do modelo de *machine learning* e para a transformação desse modelo em um formato compatível com microcontroladores. O Python possui uma sintaxe simples e de fácil aprendizado, entretanto, sua principal vantagem para este projeto reside no vasto ecossistema de bibliotecas e ferramentas voltadas para a criação de projetos de aprendizado de máquina [33].

3.1.2.3 Principais bibliotecas Python

O Para o desenvolvimento do projeto foram utilizadas várias bibliotecas em Python que formaram a base para o processamento dos dados, construção do modelo, otimização do modelo e comunicação via bluetooth com o microcontrolador:

- **TensorFlow:** Esta plataforma de código aberto foi utilizada para o desenvolvimento do modelo de aprendizado de máquina *Autoencoder* criado neste projeto. Além disso, a utilização da API de alto nível Keras especializada em d*eeep learning* permitiu que a criação do modelo fosse rápida e eficiente [34, 35].
- **TensorFlow Lite:** *Framework* do ecossistema do TensorFlow criado especificamente para a aplicação em dispositivos de apenas alguns kilobytes de memória [36]. A aplicação desse framework na otimização do modelo foi o que permitiu realizar inferências no arduino em tempo de execução.
- **Pandas:** Biblioteca utilizada para a leitura, manipulação e estruturação dos dados de séries temporais coletadas pelos sensores de qualidade do ar [37].
- **NumPy:** Biblioteca especializada na cálculo numérico computacional. Essa biblioteca é a base para as operações eficientes em array multidimensionais [38].
- **Matplotlib:** Biblioteca para a criação de visualizações, sendo fundamental para a geração dos gráficos apresentados na análise dos dados obtidos e na avaliação dos resultados do modelo [39].
- **Pyserial:** Biblioteca forneceu o acesso à porta serial durante a fase de coleta de dados para estabelecer a comunicação (via USB) entre o microcontrolador Arduino e o computador que armazenava as leituras dos sensores [40].
- **Bleak:** Biblioteca para conectar com dispositivos *Bluetooth Low Energy* (BLE), foi empregada no cenário de teste comparativo ("*on cloud*") para implementar a comunicação sem fio entre o microcontrolador e o computador que executava o modelo não otimizado [41].

3.2 Métodos

3.2.1 Montagem do Protótipo e Coleta de Dados

O protótipo foi desenvolvido conforme o esquemático da figura 9. Uma vez que as tensões de alimentação dos sensores (5V) eram incompatíveis com as tensões fornecidas pelo Arduino Nano 33 BLE Sense Rev2 (3.3V) utilizou-se uma fonte de alimentação específica para protoboards, visível no canto superior esquerdo da figura 10, para fornecer energia aos sensores. O arduino foi alimentado via cabo USB-C conectado a um notebook. Além disso, para adequar a saída analógica dos sensores às entradas analógicas do arduino criou-se três divisores de tensão de forma que a tensão máxima recebida pelo arduino fosse aproximadamente 3.22 V, um pouco abaixo do seu limite de leitura (3.3 V). A Equação (3.1) descreve a relação de saída do divisor de tensão:

$$V_{arduino} = V_{sensor} \times \frac{R3}{R1+R2+R3} \tag{3.1}$$

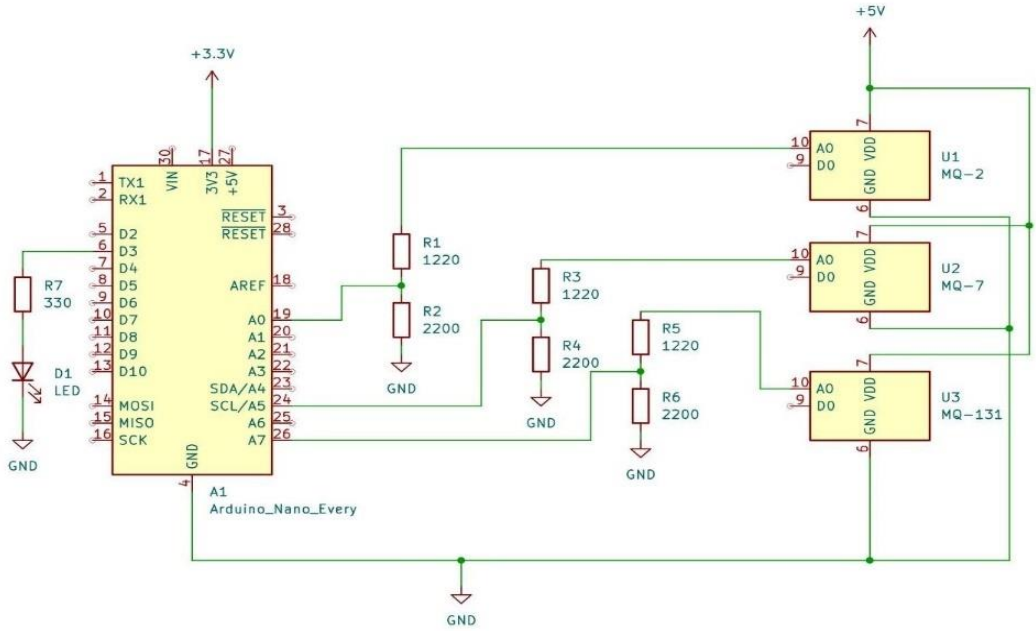
Onde:

- $Varduino$ é a tensão de saída do divisor conectada a entrada analógica do arduino;
- $Vsensor$ é a tensão de saida do sensor;
- $R1, R2\ e\ R3$ são os resistores do circuito.

Para o projeto, foram selecionados os seguintes valores de resistores comerciais: $R1 = 220\ \Omega, R2 = 1000\ \Omega\ e\ R3 = 2200\ \Omega$. Aplicando esses valores na Equação (3.1) para a tensão máxima de saida do sensor, obtém-se a tensão máxima na entrada do Arduino:

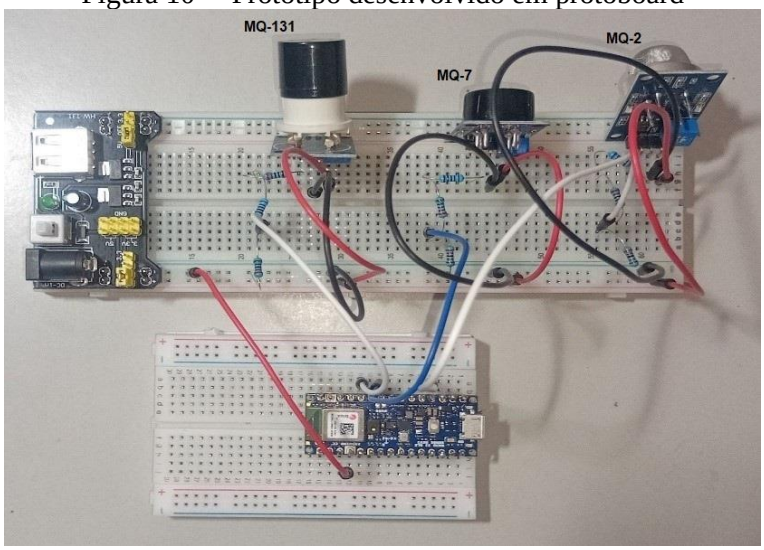
$$V_{arduino} = 5V \times \frac{2200}{220+1000+2200} \approx 3.22\ V \tag{3.2}$$

Figura 9 – Esquemático do protótipo desenvolvido



Fonte: Elaborado pelo autor.

Figura 10 – Protótipo desenvolvido em protoboard

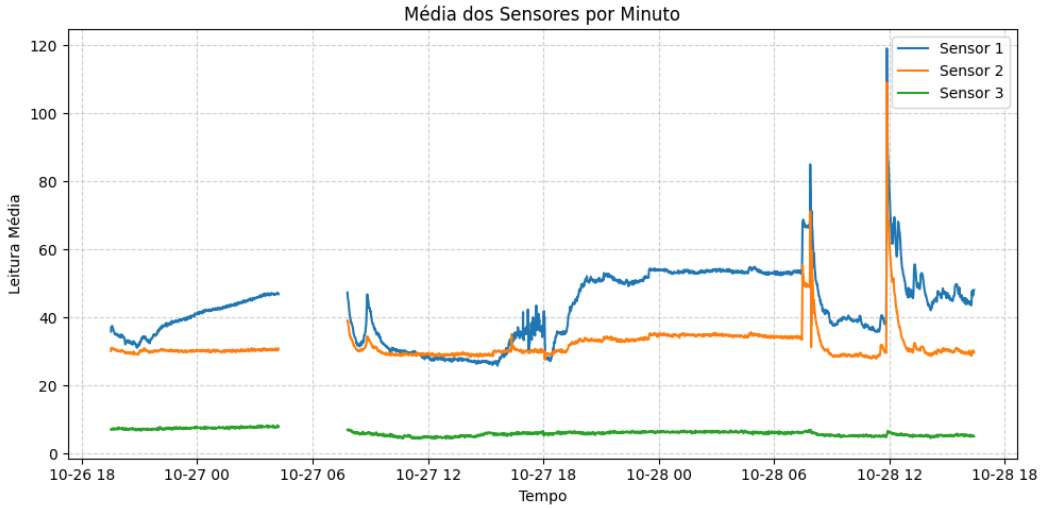


Fonte: Elaborado pelo autor.

Antes de iniciar o processo de leitura dos sensores e coleta de dados, o circuito foi alimentado e os sensores permaneceram ligados por 48 horas. Essa prática é uma recomendação do fabricante para garantir que os sensores forneçam leituras estáveis [27-29]. Após esse período de aquecimento deu-se início a coleta de dados. Para ler as entradas analógicas foi desenvolvido um programa para o arduino capaz de realizar a leitura e enviá-la via comunicação serial por um cabo USB-C. O programa utilizou uma taxa de transmissão de 9600 bits por segundo e a leitura dos sensores foi feita em intervalos de 1,5 segundos. No notebook, foi

desenvolvido um script Python para ler a porta serial e salvar os dados em um arquivo CSV. Na Figura 11 são mostrados os dados dos três sensores, com aplicação de uma média móvel de um minuto para suavizar as curvas. A coleta de dados foi realizada durante um período de aproximadamente 45 horas no interior de um residência longe de ambientes que possuem ocorrência de gases ou fumaças que poderiam comprometer o desenvolvimento do projeto. A lacuna, visível na figura 11 , foi causada por um fator externo ao protótipo, uma falha de energia elétrica no local. Para a etapa de treinamento do modelo, foram utilizados apenas os trechos de dados contínuos coletados antes e depois desta interrupção.

Figura 11 – Gráfico das leituras brutas dos sensores de CO, O_3 e Fumaça



Fonte: Elaborado pelo autor.

3.2.2 Desenvolvimento do Modelo

Iniciou-se o desenvolvimento do modelo com a leitura dos dados armazenados nos arquivos CSV. Para esse fim foi utilizado a biblioteca Pandas [33] que possui métodos nativos para a leitura desse tipo de arquivo. Em seguida, os dados foram transformados do formato de Dataframe, o formato padrão do Pandas, para o formato de Numpy array, o formato padrão do Numpy, com o objetivo de simplificar as operações com arrays multidimensionais. Após essa transformação os dados foram normalizados para o intervalo de 0 a 1. Para isso dividiram-se todos os dados por 1024 uma vez que o arduino possui uma escala de leitura analógica de 0 até 1023.

Após a normalização, os dados foram divididos em subsequências de 30 pontos. Uma subsequência é o input do modelo desenvolvido. Posteriormente, utilizou-se o método *shuffle*

do Numpy [38] para embaralhar a ordem dessas subsequências e dessa maneira evitar que o modelo se sobreajuste a tendências temporais maiores que 45 segundos. A escolha por uma janela curta se deve às limitações do hardware onde o modelo foi embarcado. Embora janelas maiores pudessem capturar tendências mais complexas, elas também exigiriam redes neurais mais complexas com mais neurônios e conexões, o que poderia comprometer a sua aplicação no microcontrolador. Por fim, os dados foram divididos em três conjuntos distintos - treino, validação e teste - nas proporções de 60%, 20% e 20%, respectivamente. Dessa forma, a distribuição final dos dados totalizou 1.979 amostras para o conjunto de treinamento, 659 para a validação e 661 para o conjunto de teste.

Antes de começar o treinamento do modelo foi necessário 'achatar' as dimensões das subsequências que possuíam dimensão de 30x3, 30 pontos para cada um dos 3 sensores, para dimensão 90x1. Isto porque, durante a fase de implantação no arduino constatou-se que *autoencoders* que possuíam camadas *flatten* e *reshape* falhavam durante a inferência no microcontrolador. A fim de resolver esse problema a fase de 'achatamento' dos dados foi transferida para o pré-processamento. Com essa abordagem foi possível executar inferências nos dispositivos com sucesso.

O modelo implementado foi um *autoencoder* empilhado com entrada e saída de 90 pontos. O *encoder* foi formado por camadas de 30, 12 e 5 neurônios, respectivamente. Já o *decoder* foi formado por camadas de 12, 30 e 90 neurônios, respectivamente. A definição desta arquitetura específica foi obtida empiricamente após a realização de diversos testes, buscando-se o equilíbrio ideal entre a capacidade de generalização e as restrições de consumo de memória do microcontrolador. A função de ativação *Rectified Linear Unit* (ReLU) foi utilizada em todas as camadas com exceção da última. Essa função foi escolhida devido a sua capacidade de modelar relações não lineares e sua alta velocidade de convergência [20, 24]. Além disso, aplicou-se uma regularização L1 de fator 0.01 em todas as camadas do *encoder* a fim de evitar o sobreajuste e reforçar a eliminação de características menos importantes [19].

Para o treinamento do modelo foi utilizada a função de perda *Mean Squared Error* (MSE) utilizando o otimizador Gradiente Descendente Estocástico com taxa de aprendizado de 0.01. Além disso, optou-se por monitorar a performance do modelo utilizando a métrica *Mean Absolute Error* (MAE). O modelo foi treinado por 200 épocas. Após o treinamento do modelo utilizou-se o conjunto de validação para determinar o limiar do erro de reconstrução a partir do qual todo valor que seja maior será considerado anômalo. Para isso optou-se por utilizar a métrica *Mean Absolute Error* para calcular os erros de reconstrução de cada subsequência. Essa métrica foi escolhida por ser robusta a presença de anomalias e por sua simplicidade de cálculo.

Em seguida, calculou-se o MAE para todo conjunto de validação e determinou-se o valor do percentil 99,5% como limiar, assim a taxa de anomalia esperada é de 0,5%. Por fim, o conjunto de teste foi utilizado para avaliar o modelo e verificar a distribuição do MAE e a taxa de anomalia resultante com base no limiar definido.

3.2.3 Otimização e Deploy

Com o modelo devidamente treinado iniciou-se o processo de otimização. Nesse processo foi realizada a quantização completa para inteiros de 8 bits (INT8) nas camadas profundas do modelo. Entretanto, optou-se por manter as camadas de entrada e saída como float, a fim de evitar a conversão de tipos de dados em tempo de execução e assim simplificar o firmware do arduino. Para realizar a quantização foi necessário passar uma função geradora à um método da biblioteca Tensor Flow Lite. Essa função retorna um valor do conjunto de validação por vez e é utilizado para calibrar a quantização do modelo [22]. Foram convertidos um modelo com quantização e um modelo sem quantização a fim de medir a redução do consumo de memória e medir a queda de performance na reconstrução. A biblioteca Tensor Flow Lite salva os modelos no formato .tflite representado como Flatbuffer, esse formato é utilizado para garantir um serialização eficiente e o permitir o carregamento direto na RAM [22].

Após a conversão para o formato .tflite o modelo foi convertido em C usando o comando *xxd* do linux. Esse comando é utilizado para converter arquivos binários nas suas respectivas representações hexadecimais, além disso utilizando a *flag* -i o comando salva a representação hexadecimal em formato de código C [42]. Dentro do código C do modelo é necessário definir a macro ‘DATA_ALIGN_ATTRIBUTE’ a fim de garantir que o modelo seja alinhado em memória e dessa forma garantir um bom desempenho [22].

Uma vez que o modelo foi convertido deu-se início ao desenvolvimento do código responsável por realizar a leitura dos sensores, invocar o modelo e fazer a detecção de anomalias. A figura 12 abaixo apresenta um fluxograma com as principais etapas do programa. A primeira etapa diz respeito a inicialização de variáveis. Nessa etapa instanciaram-se os ponteiros que apontam para o interpretador, para o modelo e para os vetores de entrada e saída, além das variáveis de controle que serviram para armazenar os valores das leituras dos sensores e controlar o fluxo. Nessa etapa também definiu-se a memória de trabalho usando a variável tensor_arena. Essa variável é de grande importância, uma vez que ela serve como memória

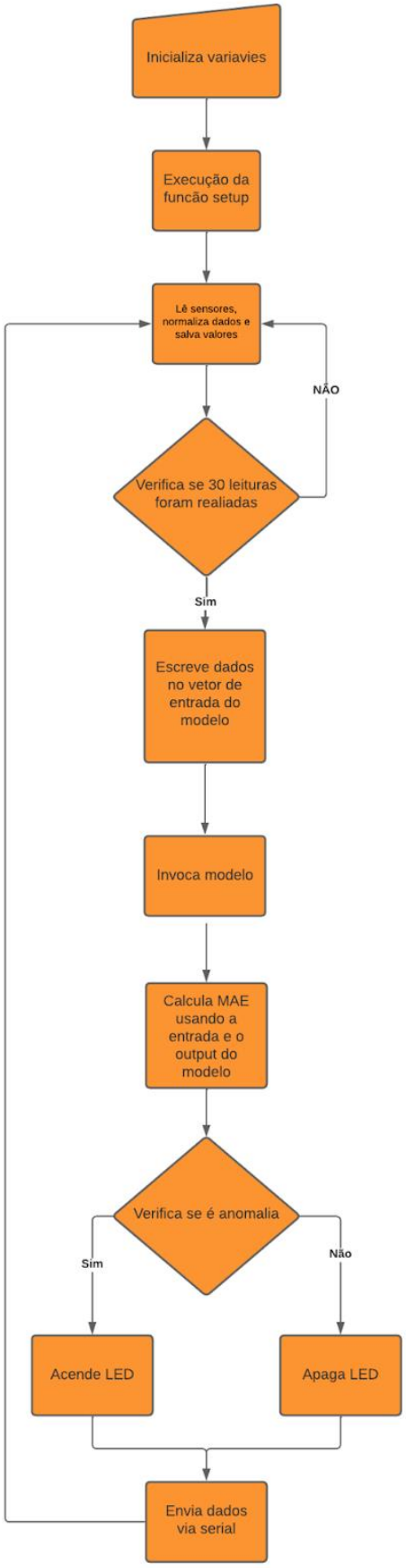
RAM de trabalho e armazena a entrada, a saída e os valores dos tensores intermediários. Sem armazenar bytes suficientes é impossível fazer inferências com o modelo [22].

A próxima etapa do fluxo é a execução da função de *setup*. Dentro dessa função instancia-se o modelo a partir do array de bytes gerados pelo comando *xxd*. Em seguida, foi isntanciada a classe responsável por fornecer as implementações das operações e a classe do intérprete responsável por realizar as inferências. Usando essa última classe executou-se o método *AllocateTensors* para alocar o espaço de memória necessário para os tensores dentro da tensor_arena definida na primeira etapa [22]. Logo após, obtiveram-se os ponteiros para a entrada e saída do modelo. Por fim o pino digital 6 foi definido como OUTPUT e deu-se início a comunicação serial com um taxa de 9600 bits por segundo.

Na etapa seguinte leram-se os valores dos sensores. Para cada sensor armazenou-se o valor das últimas 30 leituras em um array implementando uma lógica de janela deslizante, onde o dado mais recente era inserido no índice zero e o restante era 'empurrado' para direita. Dessa forma se em um ciclo um valor está no índice i, no ciclo seguinte ele estará no índice $i+1$, sendo o último valor descartado. Antes de prosseguir para a etapa de inferência o programa verificava se já haviam sido realizadas 30 leituras. Caso contrário o programa continua para a próxima leitura dos sensores depois de um delay de 1,5 segundos. Depois de terem sido realizadas 30 leituras o programa escrevia os valores dos arrays com dados de sensores no input do modelo. Diferente das APIs de alto nível como Keras, onde os dados são passados como argumento para uma função, a biblioteca Tensor Flow Lite Micro exige que os dados sejam escritos diretamente para buffers de memória [22]. Além disso, é necessário copiar os valores passados para o input do modelo em outro array para posterior comparação com o output, uma vez que o tensor flow lite pode usar a buffer de entrada para cálculos intermediários [22]. Por fim, o modelo é invocado.

Após a inferência do modelo, utilizou-se a cópia do *input* juntamente com o vetor de *output* para calcular o MAE da subsequência. Se o valor do MAE for maior do que o valor de limiar determinado durante o treinamento, a subsequência é considerada uma anomalia e a saída digital 6 recebe o valor 1 (HIGH). Caso contrário a subsequência é considerada normal e o valor 0 (LOW) é passado para a saída digital 6. Por fim, os dados dos sensores e as medidas de execução são enviados para o computador de monitoramento via conexão USB-C.

Figura 12 – Fluxograma da lógica de inferência executada no microcontrolador



Fonte: Elaborado pelo autor.

3.2.4 Procedimentos de Validação

3.2.4.1 Avaliação da Otimização

A fim de avaliar o impacto da otimização no desempenho do modelo, o modelo foi convertido duas vezes para o formato padrão do Tensor Flow Lite (.tflite). Na primeira conversão não foi aplicada nenhuma otimização e na segunda aplicou-se uma quantização completa para inteiros de 8 bits (INT8) nas camadas profundas do modelo. Dessa forma evitou-se que a diferença do formato do modelo interfira na sua performance. Após a conversão mediu-se o tamanho dos arquivos em kilobytes para verificar o efeito da quantização no consumo de memória. Em seguida, foram instanciados os dois modelos usando o interpretador do Tensor Flow Lite no ambiente Python e usou-se o conjunto de teste para realizar inferências e assim avaliar o erro de reconstrução. Utilizou-se esse erro para calcular o MAE e o MSE para os dois casos.

3.2.4.2 Validação Experimental em Tempo Real

Visando comparar a performance de um sistema que aplica o conceito de TinyML com uma abordagem tradicional, isto é, onde a inferência do modelo é realizada em um computador central, foi desenvolvido um segundo sistema que não realiza inferência dentro do microcontrolador. Esse segundo sistema compartilha o circuito e o modelo de *machine learning* com o primeiro, a única diferença está no firmware do Arduino e na utilização de um outro computador. A figura 13 apresenta o fluxo desse sistema de comparação. Nesse novo fluxo não é necessário configurar a biblioteca do Tensor Flow Lite no arduino, entretanto requer-se o recurso de *Bluetooth*.

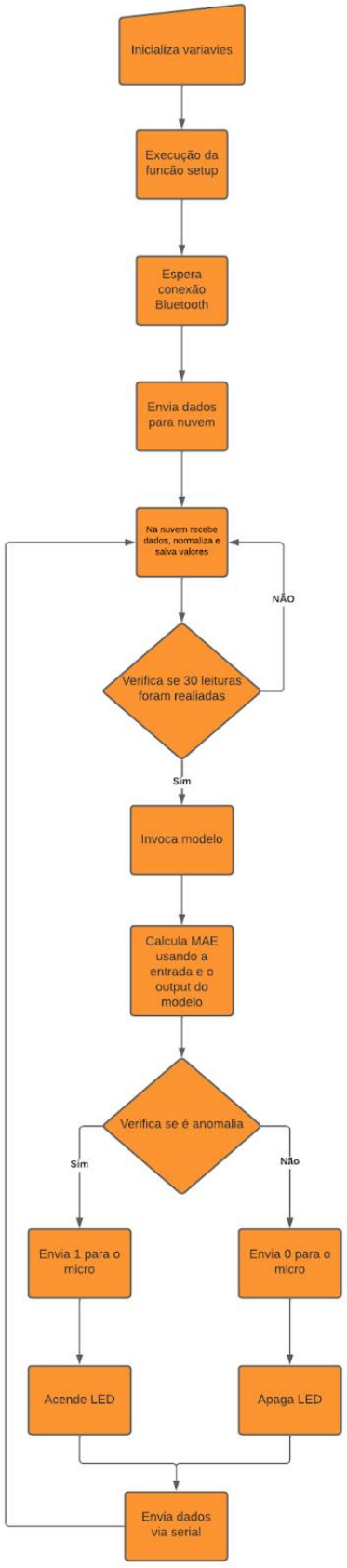
Para simular uma abordagem tradicional optou-se por utilizar uma comunicação sem fio via *Bluetooth* entre o microcontrolador e um notebook que faz o papel de um computador central. Para fins de replicabilidade, o notebook utilizado como computador central possui um processador Intel(R) Core(TM) i3-6006U CPU 2.00GHz, 8 GB de memória RAM e executava o sistema operacional Ubuntu 20.04.6 LTS. Nesse sistema de comparação, o arduino é responsável por fazer as leituras dos sensores, enviar os dados e alterar o estado do LED quando necessário. Já o computador é responsável por armazenar o valor das últimas 30 leituras, invocar o modelo, calcular o MAE e enviar para o arduino um valor indicando se a subsequência

é uma anomalia ou não. Para realizar a comunicação *Bluetooth* foi utilizado o módulo Arduino BLE presente no Arduino IDE [31] e a biblioteca Bleak [41] do Python.

Para os dois sistemas mediu-se o consumo de memória Flash e RAM no momento da compilação e upload do programa para o arduino. Os dois sistemas foram executados consecutivamente aplicando o mesmo procedimento para a introdução de anomalias. Para garantir uma comparação robusta, a coleta de dados para análise foi padronizada: em ambos os casos, os dados foram enviados ao computador via porta serial (USB) após a conclusão de cada ciclo de detecção. Esses dados correspondem aos valores dos sensores, uma *flag* de valor 0 ou 1 que indica se a subsequência é uma anomalia e o *timestamp* em que a medida foi realizada. Os sistemas possuem o mesmo delay entre cada ciclo de inferência: 1,5 segundos, que é o mesmo valor utilizado no programa de coleta de dados.

Para a validação da detecção realizou-se a introdução de anomalias reais para medir a latência do sistema em identificá-las. Para cada um dos dois sistemas o seguinte experimento foi realizado. Uma anomalia foi gerada manualmente no ambiente dos sensores. Para induzir as anomalias, utilizou-se um isqueiro para liberar gás butano (acionando os sensores MQ-2 e MQ-7) e, em seguida, acendeu-se a chama para gerar fumaça e CO. Este procedimento foi repetido 10 vezes para cada sistema. A medida do tempo de execução foi calculada posteriormente por meio da análise dos dados coletados e dos vídeos de referência. Para cada um dos eventos de anomalia induzidos foram identificados dois marcos: o tempo de início e tempo de resposta. O tempo de início é definido como o *timestamp* em que os valores dos sensores apresentaram a primeira alteração significativa, indicando o início do evento e o tempo de resposta é o tempo em que a *flag* de anomalia do sistema saiu de 0 para 1. A diferença entre esses dois marcos resulta no tempo de detecção.

Figura 13 – Fluxograma da lógica de inferência executada na nuvem.



Fonte: Elaborado pelo autor.

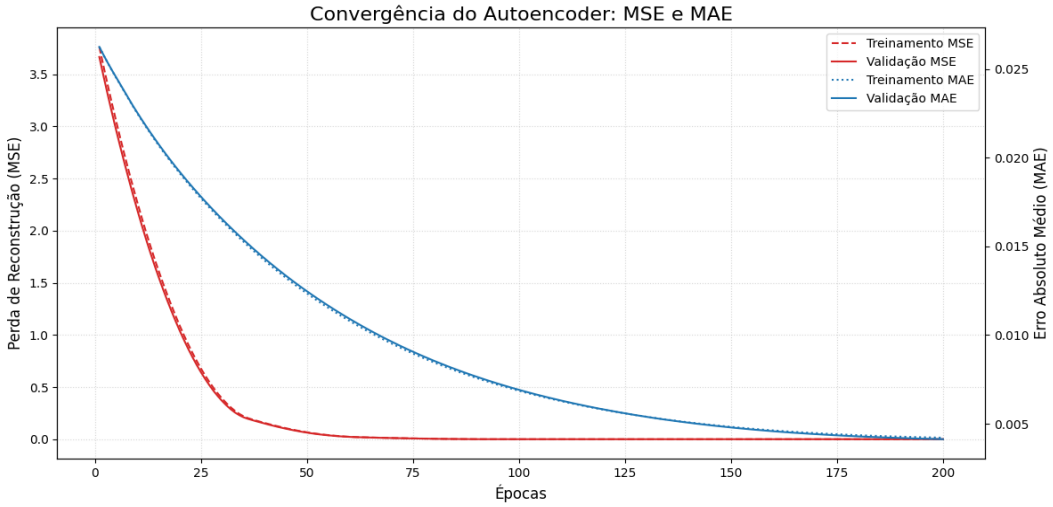
4 RESULTADOS E DISCUSSÃO

Este capítulo apresenta os resultados obtidos com o desenvolvimento do projeto. Essa exposição foi dividida em três etapas: treinamento do modelo, otimização e validação experimental. Na primeira etapa apresentam-se os resultados obtidos com o treinamento do modelo de aprendizado de máquina e sua posterior avaliação. Em seguida, analisam-se os resultados obtidos com a quantização do modelo e como essa operação impactou a sua performance. Por último, realizou-se uma análise comparativa entre o sistema tradicional e o sistema que utiliza TinyML.

4.1 Resultados do Treinamento do Modelo

A figura 14 apresenta as curvas de convergência das métricas do modelo. Observa-se que as curvas da função de perda (MSE), linhas em vermelho, convergem rapidamente em cerca de 75 épocas. Esse era um resultado esperado devido a utilização da função de ativação ReLU na maioria das camadas do *autoencoder*. Além disso, pode-se observar que as curvas de treinamento e validação estão praticamente sobrepostas, o que indica que o modelo não sofreu *overfitting*. Ou seja, o modelo desenvolvido generaliza bem para dados que não foram usados durante o treinamento. Por fim, nota-se que o modelo atingiu um platô na perda de reconstrução entre 75 e 100 épocas. O valor baixo na perda de reconstrução indica que o modelo não sofreu *underfitting* e foi capaz de aprender os padrões do conjunto de dados.

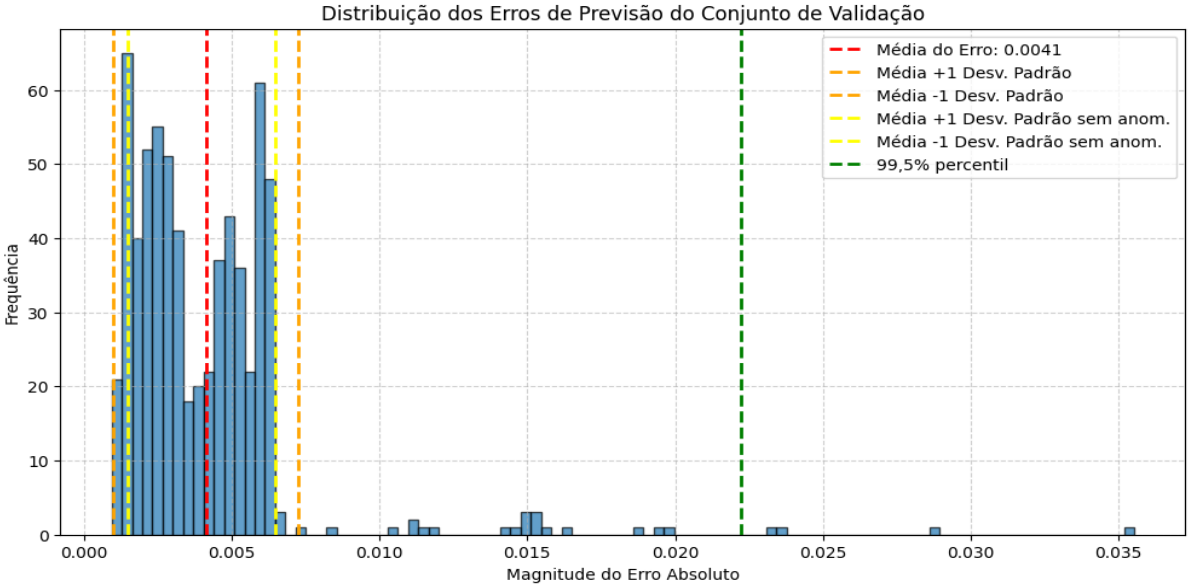
Figura 14 – Curvas de convergência das métricas de perda (Loss) e MAE durante o treinamento do autoencoder.



Fonte: Elaborado pelo autor.

Após o treinamento do modelo calculou-se o MAE para todo o conjunto de dados de validação. A figura 15 apresenta a distribuição desses erros. Pode-se observar que grande parte dos erros se encontram no intervalo entre a média menos um desvio padrão e a média mais um desvio padrão. Além disso, nota-se claramente que a distribuição dos erros não segue uma forma normal. Essa falta de normalidade inviabiliza o uso de métodos paramétricos, como por exemplo o uso da média e do desvio padrão, na escolha do limiar de detecção de anomalias. Assim conclui-se que a escolha de usar um método não paramétrico para o cálculo do limiar, neste caso o percentil 99,5% do MAE, foi a escolha mais apropriada, uma vez que permite maior robustez frente à distribuição dos dados. O limiar calculado foi de 0.022119.

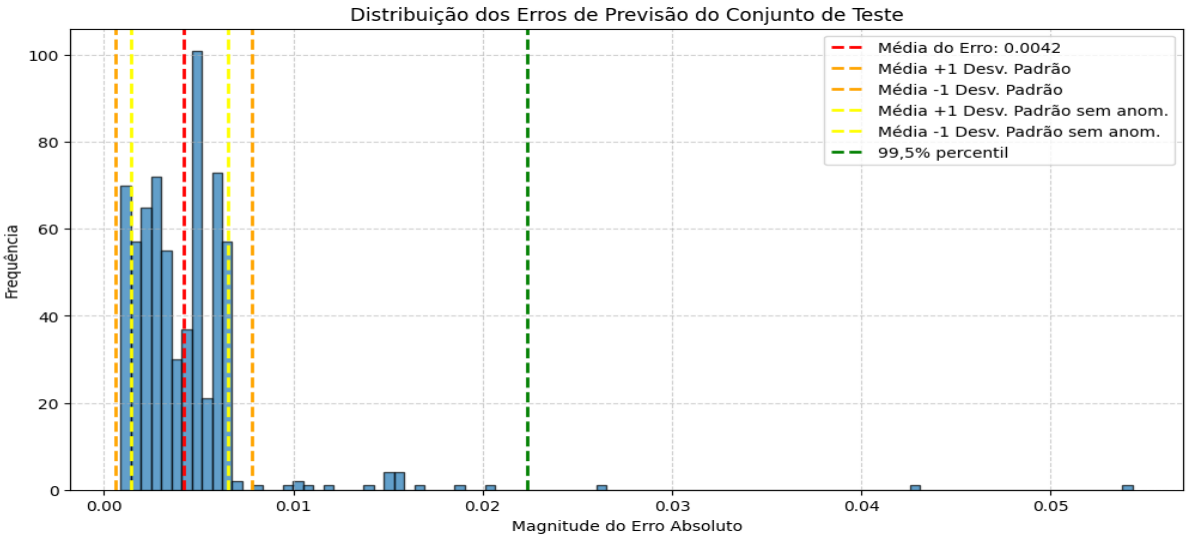
Figura 15 – Distribuição do erro absoluto médio (MAE) de reconstrução no conjunto de dados de validação..



Fonte: Elaborado pelo autor.

Posteriormente ao cálculo do limiar, calculou-se o MAE para todo o conjunto de dados de teste. A distribuição dos erros de reconstrução deste conjunto pode ser vista na figura 16. Utilizando os valores de MAE e o valor do limiar calculou-se a percentagem de anomalias no conjunto. Obteve-se um resultado de 0.454% de anomalias. Esse valor é praticamente o valor esperado de 0,5% definido pela escolha do percentil 99,5%. Esse fato reforça a indicação inicial de que o modelo desenvolvido não sofreu *overfitting*.

Figura 16 – Distribuição do erro absoluto médio (MAE) de reconstrução no conjunto de dados de teste.



Fonte: Elaborado pelo autor.

4.2 Resultados da Otimização

A tabela abaixo contém o comparativo entre o desempenho do modelo desenvolvido antes e depois da quantização. Observa-se que o modelo quantizado apresentou uma redução de 47,02% em seu tamanho. Além disso, não houve perda significativa em nenhuma das outras métricas de desempenho. Esse resultado demonstra que a quantização é uma operação ideal para reduzir o consumo de memória. De fato, esse resultado expressivo coloca a quantização como uma técnica de interesse em todo o campo de aprendizado de máquina, para além do escopo do TinyML.

Tabela 5 – Comparativo das métricas do modelo pré e pós-quantização.

Medidas	Modelo Sem Quantização	Modelo com Quantização
Tamanho (kB)	28,07	14,82
Média MAE	0.004206	0.004207
Desvio Padrão MAE	0.003594	0.003594
Média MSE	0.000052	0.000052
Desvio Padrão MSE	0.000213	0.000213

Fonte: Elaborado pelo autor.

4.3 Resultados da Validação Experimental

A tabela 6 apresenta os resultados comparativos entre o sistema embarcado e o sistema tradicional. Observa-se que a memória de programa no sistema tradicional é 62% maior do que no sistema embarcado, mesmo o programa desenvolvido no modelo embarcado sendo maior e mais complexo. Esse fato indica que a biblioteca utilizada para a comunicação via *bluetooth* consome mais memória que a biblioteca do Tensor Flow Lite. Já para a memória RAM estática o sistema embarcado consome 20% de memória a mais do que o sistema tradicional. Esse era um resultado esperado, uma vez que no sistema embarcado é preciso inicializar algumas variáveis globais como a tensor_arena, que é responsável por armazenar resultados intermediários da inferência para rodar o modelo corretamente. Dessa forma, antes mesmo da execução o sistema embarcado já utiliza mais memória RAM do que o sistema tradicional.

A tabelas 7 e 8 apresentam os resultados de cada evento de introdução de anomalias no sistema embarcado e sistema tradicional, respectivamente. Em relação ao tempo médio de detecção, verificou-se que o sistema embarcado foi 1,3 vezes mais rápido na detecção de anomalias do que o sistema tradicional (PC). Esse resultado evidencia que, apesar do maior

consumo de memória RAM, o sistema embarcado apresenta uma velocidade de detecção superior, reforçando sua adequação para aplicações em tempo real.

Tabela 6 – Comparação de recursos computacionais e tempo de inferência entre o sistema embarcado (TinyML) e o sistema local (PC).

Medidas	**Sistema Embarcado**	**Sistema Tradicional (PC)**
Memória Flash	197 Kb	321 Kb
Memória RAM Estática	82 Kb	68 kB
Tempo Médio de Detecção	4,5 s	5,8 s

Fonte: Elaborado pelo autor.

Tabela 7 – Resultados dos testes de latência de detecção do sistema embarcado (TinyML).

Evento	**Início do Evento (hh:mm:ss)**	**Momento da detecção (hh:mm:ss)**	**Latência de Detecção (s)**
1	17:37:56	17:38:02	6
2	17:39:46	17:39:49	3
3	17:41:25	17:41:30	5
4	17:42:46	17:42:48	2
5	17:44:38	17:44:41	3
6	17:46:17	17:46:19	2
7	17:47:51	17:48:06	15
8	17:49:32	17:49:35	3
9	17:52:10	17:52:13	3
10	17:53:59	17:54:02	3

Fonte: Elaborado pelo autor.

Tabela 8 – Resultados dos testes de latência de detecção do sistema de processamento tradicional (PC).

Evento	**Início do Evento (hh:mm:ss)**	**Momento da detecção (hh:mm:ss)**	**Latência de Detecção (s)**
1	18:09:55	18:10:03	8
2	18:11:17	18:11:21	4
3	18:12:24	18:12:29	5
4	18:14:20	18:14:28	8
5	18:15:53	18:15:59	6
6	18:17:57	18:18:01	4
7	18:19:33	18:19:36	3
8	18:21:08	18:21:12	4
9	18:22:38	18:22:46	8
10	18:24:17	18:24:25	8

Fonte: Elaborado pelo autor.

5 CONCLUSÃO

Este trabalho se propôs a desenvolver um sistema capaz de detectar anomalias na qualidade do ar em tempo real utilizando um microcontrolador. O objetivo foi superar o tempo de latência e o custo computacional da comunicação sem fio que as abordagens tradicionais com processamento em nuvem possuem. Para enfrentar o problema de detecção de anomalias foi escolhido utilizar uma abordagem de reconstrução das séries temporais. Onde o erro de reconstrução foi utilizado como um *score* de anomalias. A fim de realizar essa reconstrução utilizou-se uma arquitetura de redes neurais chamada de *autoencoder*.

A análise dos resultados demonstrou a eficácia da abordagem. A técnica de quantização completa para inteiros de 8 bits (INT8) foi fundamental para a aplicação do projeto, diminuindo o tamanho do modelo em 47,02% sem introduzir perdas significativas de performance nas métricas de reconstrução (MAE e MSE). Na validação experimental comparativa, o sistema embarcado apresentou um desempenho superior ao sistema tradicional, com um tempo médio de detecção de 4,5 segundos, sendo 1,3 vezes mais rápido que a arquitetura tradicional (5,8 segundos). Essa diferença comprova que a eliminação ou a redução da comunicação sem fio, proporcionada pelo processamento local, é um fator decisivo para aplicações em tempo real.

Embora a implementação embarcada tenha exigido um consumo 20% maior de memória RAM estática (82 Kb) para a alocação das variáveis globais necessárias à inferência do modelo, ela consumiu 62% menos memória Flash (197 Kb) que o sistema tradicional (321 Kb), devido ao alto custo da biblioteca de comunicação *Bluetooth*.

É importante notar que a validação do sistema tradicional utilizou comunicação *Bluetooth*, uma escolha justificada pelas limitações do hardware disponível (ausência de conectividade à internet no microcontrolador). Para trabalhos futuros, sugere-se uma análise comparativa utilizando comunicação via internet (como Wi-Fi), o que aproximaria a validação de um cenário de processamento em nuvem mais realista.

Além disso, embora o modelo *autoencoder* e a técnica de quantização INT8 tenham se provado eficazes para este projeto, o escopo não permitiu uma comparação exaustiva com outras abordagens. Como investigação futura, seria de grande valia comparar o desempenho de outros modelos e técnicas de otimização (como os descritos no Capítulo 2) no mesmo contexto de hardware, a fim de determinar a combinação com a melhor relação entre performance e eficiência de recursos

Conclui-se, portanto, que este trabalho atingiu seu objetivo ao desenvolver e validar um dispositivo portátil, de baixo custo e alta eficiência. O sistema TinyML demonstrou ser uma solução rápida e viável para o monitoramento de anomalias, reforçando as vantagens do processamento na borda em aplicações que demandam respostas imediatas ou que devem limitar a transmissão de dados via rede sem fio.

REFERÊNCIAS

[1] B. Zhang and X. Li, "Optimal computation offloading for industrial IoT," IEEE Access, 2025. Accessed: Apr. 18, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10964287

[2] P. Gupta and R. S. Jadon, "PLANT Detect Net: IoT + Deep Learning for plant disease detection," Evolving Systems, 2025. Accessed: Apr. 18, 2025. [Online]. Available: https://link.springer.com/article/10.1007/s12530-025-09685-x

[3] A. Ahmad and D. Choi, "A wireless power transfer system for IoMT implantable devices using inductive coupling and adaptive frequency control," IEEE Internet of Things Journal, 2025. Accessed: Apr. 18, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10964304

[4] IoT Analytics, "State of IoT—Spring 2024: Number of connected IoT devices growing 16% to 16.7 billion worldwide," IoT Analytics, 2024. Accessed: Apr. 18, 2025. [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/

[5] D. Situnayake and J. Plunkett, AI at the Edge: Solving Real-World Problems with Embedded Machine Learning, 1st ed. Sebastopol: O'Reilly Media, 2023.

[6] G. M. Iodice, TinyML: Combine Artificial Intelligence and Ultra-Low-Power Embedded Devices to Make the World Smarter. Birmingham: Packt Publishing, 2022.

[7] L. Banz et al., "A machine learning-oriented survey on tiny machine learning," 2023. Accessed: Apr. 04, 2025. [Online]. Available: https://arxiv.org/abs/2306.09349

[8] W. de Simoni et al., O Estado da Qualidade do Ar no Brasil. São Paulo: WRI Brasil, 2021. Accessed: Apr. 24, 2025. [Online]. Available: https://www.wribrasil.org.br/publicacoes/o-estado-da-qualidade-do-ar-no-brasil

[9] Companhia Ambiental do Estado de São Paulo (CETESB), "Padrões de Qualidade do Ar," São Paulo: CETESB, [2023?]. Accessed: Apr. 22, 2025. [Online]. Available: https://cetesb.sp.gov.br/ar/padroes-de-qualidade-do-ar/

[10] Companhia Ambiental do Estado de São Paulo (CETESB), "Qualidade do Ar," São Paulo: CETESB, [2023?]. Accessed: Apr. 22, 2025. [Online]. Available: https://cetesb.sp.gov.br/ar/poluentes

[11] S. N. Dapper, C. Spohr, and R. R. Zanini, "Poluição do ar como fator de risco para a saúde: uma revisão sistemática no estado de São Paulo," Estudos Avançados, São Paulo, vol. 30, no. 86, pp. 83–98, 2016. doi: 10.1590/S0103-40142016.00100006.

[12] Redação, "Estado do Rio lidera ranking nacional de monitoramento da qualidade do ar, segundo estudo," O Fluminense, Niterói, Mar. 2, 2024. Accessed: Apr. 23, 2025. [Online]. Available: https://www.ofluminense.com.br/cidades/rio-de-janeiro/2024/03/1272213-estado-do-rio-lidera-ranking-nacional-de-monitoramento-da-qualidade-do-ar-segundo-estudo.html

[13] D. M. Hawkins, Identification of Outliers. Dordrecht: Springer Science+Business Media B.V., 1980. (Monographs on Applied Probability and Statistics). doi: 10.1007/978-94-015-3994-4.

[14] S. K. Adari and S. Alla, Beginning Anomaly Detection Using Python-Based Deep Learning: Implement Anomaly Detection Applications with Keras and PyTorch, 2nd ed. [S.l.]: Apress, 2024. doi: 10.1007/979-8-8688-0008-5.

[15] B. Kennedy, Outlier Detection in Python. Shelter Island: Manning Publications Co., 2025.

[16] P. Boniol, Q. Liu, M. Huang, T. Palpanas, and J. Paparrizos, "Dive into Time-Series Anomaly Detection: A Decade Review," 2024.

[17] D. Colombo, L. Leonardi, A. Panarello, and M. Merlini, "Anomaly Detection for IoT Time-Series Data: A Survey," Sensors, vol. 23, no. 8, p. 3871, 2023. doi: 10.3390/s23083871.

[18] M. A. Samara, I. Bennis, A. Abouaissa, and P. Lorenz, "A Survey of Outlier Detection Techniques in IoT: Review and Classification," Journal of Sensor and Actuator Networks, vol. 11, no. 1, p. 4, 2022. doi: 10.3390/jsan11010004.

[19] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2nd ed. Sebastopol: O'Reilly Media, 2019.

[20] G. M. Iodice, TinyML Cookbook: Combine Artificial Intelligence and Ultra-Low-Power Embedded Devices to Make the World Smarter. Birmingham: Packt Publishing, 2022.

[21] A. A. Cook, G. Misirli, and Z. Fan, "Anomaly detection for IoT time-series data: a survey," IEEE Internet of Things Journal, vol. 7, no. 7, pp. 6481-6494, Jul. 2020. doi: 10.1109/JIOT.2019.2958185.

[22] P. Warden and D. Situnayake, TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers, 1st ed. Sebastopol, CA: O'Reilly Media, 2020.

[23] V. Tsoukas, A. Gkogkidis, E. Boumpa, and A. Kakarountas, "A review on the emerging technology of TinyML," ACM Computing Surveys, vol. 56, no. 10, art. 259, pp. 1-37, Jun. 2024. doi: 10.1145/3661820.

[24] L. Capogrosso, F. Cunico, D. S. Cheng, F. Fummi, and M. Cristani, "A machine learning-oriented survey on tiny machine learning," IEEE Access, vol. 12, pp. 23406-23440, 2024. doi: 10.1109/ACCESS.2024.3365349.

[25] Arduino, "Nano 33 BLE Sense Rev2," 2025. Accessed: Oct. 19, 2025. [Online]. Available: https://docs.arduino.cc/hardware/nano-33-ble-sense-rev2/

[26] Hanwei Electronics, "MQ-7 Semiconductor Sensor for Carbon Monoxide," Datasheet, Zhengzhou, [n.d.]. Accessed: Nov. 1, 2025. [Online]. Available: https://cdn.sparkfun.com/assets/b/b/b/3/4/MQ-7.pdf

[27] Winsen, "MQ-131 Semiconductor Sensor for Ozone," Datasheet, Zhengzhou, [2012?]. Accessed: Nov. 1, 2025. [Online]. Available: https://cdn.sparkfun.com/assets/9/9/6/e/4/mq131-datasheet-low.pdf

[28] Hanwei Electronics, "MQ-2 Semiconductor Sensor for Combustible Gas," Datasheet, Zhengzhou, [n.d.]. Accessed: Nov. 1, 2025. [Online]. Available: https://www.haoyuelectronics.com/Attachment/MQ-2/MQ-2.pdf

[29] Eletrogate, "Fonte Ajustável Para Protoboard," 2025. Accessed: Oct. 19, 2025. [Online]. Available: https://www.eletrogate.com/fonte-ajustavel-para-protoboard

[30] Arduino IDE. Accessed: Oct. 27, 2025. [Online]. Available: https://www.arduino.cc/en/software

[31] "Jupyter Notebook." Accessed: Oct. 27, 2025. [Online]. Available: https://jupyter.org

[32] "Visual Studio Code." Accessed: Oct. 27, 2025. [Online]. Available: https://code.visualstudio.com

[33] "Python." Accessed: Oct. 27, 2025. [Online]. Available: https://www.python.org/

[34] TensorFlow, "TensorFlow," 2025. Accessed: Oct. 27, 2025. [Online]. Available: https://www.tensorflow.org/

[35] Keras, "Keras," 2025. Accessed: Oct. 27, 2025. [Online]. Available: https://keras.io/

[36] TensorFlow, "TensorFlow Lite," 2025. Accessed: Oct. 27, 2025. [Online]. Available: https://www.tensorflow.org/lite

[37] Pandas Development Team, "pandas," 2025. Accessed: Oct. 27, 2025. [Online]. Available: https://pandas.pydata.org/

[38] NumPy Developers, "NumPy," 2025. Accessed: Oct. 27, 2025. [Online]. Available: https://numpy.org/

[39] J. D. Hunter et al., "Matplotlib: A 2D Graphics Environment," Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007. Accessed: Oct. 27, 2025. [Online]. Available: https://matplotlib.org/

[40] C. Liechti et al., "pyserial," GitHub repository, 2025. Accessed: Oct. 27, 2025. [Online]. Available: https://github.com/pyserial/pyserial

[41] Bleak Developers, "Bleak," GitHub repository, 2025. Accessed: Oct. 27, 2025. [Online]. Available: https://github.com/hbldh/bleak

[42] S. Sharma, "Using XXD Command in Linux," Linux Handbook, Mar. 17, 2023. Accessed: Nov. 1, 2025. [Online]. Available: https://linuxhandbook.com/xxd-command/

Apêndice A – REPOSITÓRIO DO CÓDIGO FONTE

O código-fonte completo desenvolvido para este trabalho, incluindo os scripts de aquisição de dados do Arduino e o modelo de Machine Learning, está disponível publicamente no seguinte repositório GitHub: https://github.com/gabriel-victor933/tcc.