Revista ft

www.revistaft.com.br

Revista FT | ISSN 1678-0817 | v. 30, n. 157, 2026 | DOI: 10.69849/0tbhmj55

# Sistema embarcado para inspeção visual em indústria de beneficiamento do coco

## Embedded system for visual inspection in the coconut processing industry

Ian Lucas Périgo Vianna[1]; Janyelison Rodrigo Marques Ferreira[2]; Luís Guilherme de Souza Coutinho[3]; Ramon Vinícius Ferreira de Souza[4]; Jefferson Igor Duarte Silva[5]; Leonardo Gomes de Paiva Amorim[6]; Diego da Silva Pereira[7]

## Resumo

Este artigo apresenta o desenvolvimento de um sistema embarcado para inspeção visual automatizada aplicado à indústria de beneficiamento do coco, com ênfase nos processos de empacotamento e enfardamento. A pesquisa tem como objetivo automatizar a contagem de pacotes e a identificação de fardos com não conformidades estruturais, reduzindo erros operacionais e aumentando a eficiência do controle de qualidade em ambientes industriais. A solução proposta emprega técnicas de visão computacional e aprendizado de máquina em *edge computing*, executados na *single-board computer* (SBC) Raspberry Pi. Foram desenvolvidos dois dispositivos distintos: um destinado à contagem e verificação da quantidade correta, durante o processo de enfardamento, a partir de uma vista superior, e outro posicionado ao final da linha de produção de leite de coco para detecção de fardos defeituosos. A metodologia envolveu a coleta de imagens em ambiente industrial real, a anotação manual dos dados, o treinamento de modelos baseados na arquitetura YOLO e a validação do desempenho em tempo real. Os resultados obtidos indicam a viabilidade técnica do sistema, com detecção consistente de objetos e identificação eficaz de não conformidades, apesar de limitações impostas pelo hardware embarcado. Conclui-se que a solução apresenta potencial para otimizar processos de inspeção visual industrial, servindo como base para futuras melhorias e integração com sistemas de monitoramento e rastreabilidade.

**Palavras-chave:** Sistema embarcado; *Edge computing*; Visão computacional; Automação industrial.

[1] Discente do Curso de Formação Inicial e Continuada (Curso FIC) em Residência Tecnológica em Software Embarcado, na modalidade a distância. e-mail: ian20240027015@alu.uern.br

[2] Discente do Curso de Formação Inicial e Continuada (Curso FIC) em Residência Tecnológica em Software Embarcado, na modalidade a distância. e-mail: nome@provedor.com.br

[3] Discente do Curso de Formação Inicial e Continuada (Curso FIC) em Residência Tecnológica em Software Embarcado, na modalidade a distância. e-mail: luis.coutinho@cs.unipe.edu.br

[4] Discente do Curso de Formação Inicial e Continuada (Curso FIC) em Residência Tecnológica em Software Embarcado, na modalidade a distância. e-mail: ramon.ferreira.072@ufrn.edu.br

[5] Docente do Curso de Formação Inicial e Continuada (Curso FIC) em Residência Tecnológica em Software Embarcado, na modalidade a distância. e-mail: jefferson.duarte@ifrn.edu.br

[6] Docente do Curso de Formação Inicial e Continuada (Curso FIC) em Residência Tecnológica em Software Embarcado, na modalidade a distância. e-mail: leonardo.amorim@ifrn.edu.br

[7] Docente do Curso de Formação Inicial e Continuada (Curso FIC) em Residência Tecnológica em Software Embarcado, na modalidade a distância. e-mail: diego.pereira@ifrn.edu.br

**ABSTRACT**

This article presents the development of an embedded system for automated visual inspection applied to the coconut processing industry, with emphasis on packaging and baling processes. The research aims to automate package counting and the identification of bales with structural nonconformities, reducing operational errors and increasing the efficiency of quality control in industrial environments. The proposed solution employs computer vision and machine learning techniques in edge computing, executed on the single-board computer (SBC) Raspberry Pi. Two distinct devices were developed: one intended for counting and verifying the correct quantity during the baling process from a top-view perspective, and another positioned at the end of the coconut milk production line for the detection of defective bales. The methodology involved image collection in a real industrial environment, manual data annotation, training of models based on the YOLO architecture, and real-time performance validation. The obtained results indicate the technical feasibility of the system, with consistent object detection and effective identification of nonconformities, despite limitations imposed by the embedded hardware. It is concluded that the solution has the potential to optimize industrial visual inspection processes, serving as a basis for future improvements and integration with monitoring and traceability systems.

**Keywords:** Embedded system; Edge computing; Computer vision; Industrial automation.

# 1. INTRODUÇÃO

A indústria de processamento de derivados do coco possui relevância econômica no Brasil, impulsionando o agronegócio com a geração de milhões de empregos diretos e indiretos, especialmente no Nordeste, que concentra 81,3% da produção nacional de 2,1 milhões de toneladas em 2024 (EMDAGRO, 2025). Segundo dados recentes do setor (EMDAGRO, 2025), o Brasil ampliou sua produção de coco de 1,56 milhão de toneladas em 2019 para 2,1 milhões de toneladas em 2024 (+34%), mesmo com área colhida estável entre 185-189 mil hectares.

Contudo, em linhas de produção da indústria de processamento de derivados de coco, especialmente nas etapas de empacotamento e enfardamento, atividades manuais de inspeção visual e contagem estão sujeitas a falhas humanas, variabilidade de desempenho e dificuldade de padronização. Essas limitações impactam diretamente o controle de qualidade, a eficiência produtiva e os custos operacionais. O uso de técnicas de automação industrial manifesta-se,

neste sentido, como alternativa viável para aumentar a confiabilidade dos processos e reduzir perdas.

Com o avanço da Indústria 4.0, caracterizada pela integração de tecnologias digitais no ambiente fabril (OLIVEIRA, 2022), sistemas computacionais aliados a técnicas de aprendizado de máquina passaram a desempenhar papel central na inspeção automatizada, permitindo a execução de algoritmos inteligentes sobre um conceito denominado borda da rede ou *edge computing*, processando dados em dispositivos próximos à fonte de geração de dados, como sistemas embarcados, em vez de enviá-los para servidores centrais na nuvem, reduzindo latência e aumentando a autonomia em aplicações industriais (LEITE, 2023).

Segundo Dhiman (2025), o avanço da indústria de *microchips*, em especial o desenvolvimento de novos tipos de arquiteturas para uso em hardware para inteligência artificial, possibilitou a execução de modelos de visão computacional em dispositivos com recursos limitados, ampliando sua aplicação em ambientes industriais.

Nesse contexto, a visão computacional surge como um ramo da inteligência artificial que permite interpretar imagens em tempo real para identificar padrões e anomalias, podendo ser integrada em sistemas embarcados (TAVARES; SANTOS, 2025). No caso específico da empresa Coco e Cia (Indústria e Comércio Mendonça Barreto Ltda.), a implementação dessa tecnologia tem como objetivo mitigar gargalos de produção na linha de montagem de fardos, onde a detecção precoce de não conformidades é fundamental para garantir a eficiência do processo produtivo.

Atualmente, foram identificados dois cenários críticos: no primeiro, relacionado ao fechamento de fardos de pacotes de coco ralado, ocorrem falhas ocasionais em que unidades deixam de ser inseridas ou são adicionadas em excesso durante o processo automatizado; no segundo, referente à linha de fardos de leite de coco, observam-se situações em que há falta de garrafas ou desalinhamento dos fardos, comprometendo a qualidade final do produto e exigindo intervenções manuais.

O presente trabalho tem como objetivo desenvolver e avaliar um sistema de inspeção visual automatizada para identificação, contagem e detecção de não conformidades em fardos de produtos derivados do coco na empresa em estudo. A justificativa deste trabalho fundamenta-se na necessidade de aprimorar os processos de controle de qualidade, reduzir a dependência de inspeções manuais suscetíveis a falhas humanas e auxiliar a tomada de decisão em tempo real na linha de produção, contribuindo para a modernização dos processos industriais e para o aumento da eficiência e competitividade do setor.

# 2. FUNDAMENTAÇÃO TEÓRICA

## 2.1. Sistemas embarcados

Segundo Almeida et al. (2016), sistemas embarcados são sistemas eletrônicos microprocessados projetados para executar uma função específica, diferindo de computadores de propósito geral, capazes de desempenhar múltiplas tarefas. Esses sistemas estão presentes em uma ampla variedade de aplicações, que vão desde eletrodomésticos até equipamentos industriais.

Nesse contexto, os microcontroladores se destacam como o principal elemento desses sistemas. São circuitos integrados que reúnem processador, memória e periféricos em um único chip, oferecendo controle eficiente, baixo consumo de energia e custo reduzido. Essas características os tornam amplamente utilizados em dispositivos embarcados.

Entretanto, microcontroladores possuem restrições significativas de processamento e memória, exigindo um gerenciamento muito mais rigoroso do que em sistemas de propósito geral (ALMEIDA et al., 2016, p. 15). Essas limitações dificultam a execução de tarefas computacionalmente intensivas.

Por essa razão, aplicações mais complexas — como visão computacional, aprendizado de máquina e *deep learning* — têm migrado para dispositivos *System on a Chip* (SoC), que oferecem maior capacidade computacional. Embora sejam capazes de executar sistemas operacionais completos, esses dispositivos continuam desempenhando papel de sistemas embarcados quando dedicados a uma função específica dentro de um equipamento.

## 2.2 *System on a Chip* (SoC) na computação de borda e visão computacional

Microcontroladores apresentam limitações de processamento que tornam inviável o deploy de sistemas de visão computacional baseados em aprendizado profundo, já que muitos operadores exigem hardware dedicado, como *Neural Processing Units* (NPUs) ou *Graphics Processing Units* (GPUs). Uma alternativa é o uso de dispositivos *System on a Chip* (SoC), que representam a principal solução intermediária para computação de borda. Esses dispositivos geralmente executam Linux embarcado, o que abstrai o acesso ao hardware e permite o desenvolvimento em linguagens de alto nível, como Python (SITUNAYAKE; PLUNKETT, 2023, p. 125).

A adoção de SoCs, contudo, envolve *trade-offs*. Há maior consumo energético e aumento da complexidade do sistema, já que a presença de um sistema operacional adiciona milhares de linhas de código em execução paralelamente à aplicação, reduzindo a previsibilidade e a confiabilidade (SITUNAYAKE; PLUNKETT, 2023, p. 126).

Mesmo com essas limitações, a presença de sistemas operacionais em dispositivos de borda tem crescido. Entre os fatores que impulsionam o uso do Linux embarcado estão (WEBASHA TECHNOLOGIES, 2025):

- A expansão da computação de borda e a necessidade de processamento próximo à fonte dos dados;
- A adoção crescente de placas baseadas em RISC-V, arquitetura aberta cujo sistema operacional padrão costuma ser Linux;
- A redução de custos associada ao uso de software livre.

O Linux embarcado também se destaca frente aos sistemas operacionais de tempo real (RTOS), oferecendo um ecossistema robusto com interface de usuário, protocolos de rede, sistema de arquivos e suporte a conteinerização, facilitando a implantação de microsserviços na borda (SECO S.p.A., 2026). Entre suas desvantagens, destacam-se a maior latência, a ampliação da superfície de ataque e o consumo superior de recursos.

Dessa forma, a escolha entre microcontroladores, SoCs, arquiteturas e sistemas operacionais deve considerar cuidadosamente o cenário de aplicação, ponderando desempenho, consumo energético, segurança e confiabilidade.

### 2.3 Inteligência artificial em computação de borda

A computação em borda (*edge computing*) representa uma mudança em relação ao modelo centralizado de processamento em nuvem. Após a trajetória histórica que vai dos *mainframes* aos terminais, da computação pessoal à nuvem, observa-se agora um movimento de retorno do processamento para a borda da rede (SITUNAYAKE; PLUNKETT, 2023). Nessa abordagem, dispositivos que compõem a Internet das Coisas (IoT) — tipicamente sistemas embarcados equipados com sensores — processam dados próximos de sua fonte.

Por serem projetados para funções específicas, esses dispositivos operam sob restrições severas de recursos, como memória limitada, baixo poder de processamento e, frequentemente, dependência de bateria. A integração de inteligência artificial nesses equipamentos, denominada *edge AI*, permite realizar inferências diretamente no dispositivo,

reduzindo a necessidade de enviar dados para servidores remotos e possibilitando respostas imediatas (SITUNAYAKE; PLUNKETT, 2023).

Bruges (2019) propõe o acrônimo BLERP como critério para avaliar quando a adoção de *edge AI* é vantajosa. O modelo abrange cinco dimensões:

- *Bandwidth* (Largura de banda)*:* o processamento local evita transmitir grandes volumes de dados brutos.
- *Latency* (Latência): elimina o tempo de envio à nuvem, permitindo respostas em tempo real.
- *Economics* (Economia): reduz custos de conectividade e infraestrutura remota.
- *Reliability* (Confiabilidade): o dispositivo continua operando mesmo com conexão instável ou inexistente.
- *Privacy* (Privacidade): dados sensíveis permanecem no dispositivo, evitando seu trânsito pela rede.

A análise desses cinco fatores é essencial para determinar se a computação de borda é a arquitetura mais adequada para uma solução de inteligência artificial.

### 2.4 Detecção de defeitos e controle de qualidade industrial

A detecção de defeitos é uma das aplicações mais relevantes do *edge AI* na Indústria 4.0, sustentando tanto o controle de qualidade automatizado quanto estratégias de manutenção preditiva. Enquanto a inspeção humana é suscetível a fadiga e inconsistências, sistemas de visão de máquina integram hardware e software para capturar e processar imagens em tempo real, permitindo verificar se cada peça na linha de montagem atende aos padrões estabelecidos (KHANG et al., 2025, p. 27).

Esses sistemas podem ser implementados como dispositivos embarcados equipados com câmeras capazes de identificar falhas superficiais — como riscos, deformações ou variações de cor — diretamente no fluxo de produção. Além disso, a inspeção visual pode ser complementada por sensores industriais, que ampliam a capacidade de detecção para além da análise óptica. Sensores de pressão, por exemplo, são essenciais no monitoramento de circuitos hidráulicos e pneumáticos, identificando vazamentos ou anomalias de compressão que antecedem falhas mecânicas (KHANG et al., 2025, p. 314).

### 2.5 Visão computacional em dispositivos de borda

A visão computacional em dispositivos de borda utiliza sensores de imagem para extrair e interpretar informações do ambiente. Apesar do alto custo computacional desse processamento, avanços em redes neurais artificiais (RNAs) — modelos formados por camadas capazes de aprender padrões — permitiram que técnicas antes restritas a servidores fossem aplicadas em sistemas embarcados. Entre suas arquiteturas, as Redes Neurais Convolucionais (CNNs) destacam-se por serem especialmente adequadas ao processamento de imagens.

No cenário de borda, a visão computacional exige soluções adaptadas a dispositivos com recursos limitados. Os principais desafios incluem:

- Restrições de hardware: modelos tradicionais podem ocupar gigabytes, enquanto muitos dispositivos embarcados possuem apenas alguns kilobytes ou megabytes disponíveis.
- Arquiteturas eficientes: redes otimizadas, como a MobileNet, equilibram precisão e baixo uso de memória e processamento (SITUNAYAKE; PLUNKETT, 2023, p. 173).
- Otimização de modelos: são aplicadas técnicas como:
    - Quantização: redução da precisão dos pesos para 8 bits, diminuindo significativamente o tamanho do modelo com impacto mínimo na acurácia (SITUNAYAKE; PLUNKETT, 2023, p. 183).
    - Fusão de operadores: combinação de operações para maior eficiência computacional.
    - Modelos especializados, como *Spiking* e *Binary Neural Networks*, projetados para operar em hardware altamente restrito.

### 2.6 Arquitetura *You Only Look Once* (YOLO) na visão computacional

O YOLO introduziu uma mudança de paradigma na visão computacional ao substituir detectores de dois estágios por um modelo capaz de prever, em uma única passagem, caixas delimitadoras e probabilidades de classe. Essa abordagem reduz a latência e aumenta o desempenho, tornando o YOLO especialmente adequado a sistemas embarcados e aplicações de *edge AI* (SAPKOTA et al., 2026).

A arquitetura também se destacou por sua versatilidade, podendo ser utilizada para detecção, classificação e estimativa de pose, graças ao uso do *backbone* como extrator de características (SAPKOTA et al., 2026). Dentre as versões recentes, o YOLOv8 e o

YOLOv11 representam marcos importantes. A escolha entre eles é particularmente relevante neste estudo, que exige modelos leves para execução em hardware de borda.

### 2.6.1 YOLOv8: *anchor-free* e módulo C2f

Lançado em 2023, o YOLOv8 introduziu mudanças significativas na arquitetura da série, adotando uma abordagem *anchor-free* (sem âncoras) que simplifica o processo de detecção e melhora a generalização para objetos de formatos variados. A principal inovação estrutural do v8 foi a introdução do módulo C2f (*Cross Stage Partial with 2 convolutions*). Esse módulo foi projetado para enriquecer o fluxo de gradientes e a extração de características, combinando as vantagens das arquiteturas ELAN (*Efficient Layer Aggregation Networks*) com blocos de gargalo (HIDAYATULLAH et al., 2025).

Além disso, o YOLOv8 separa as cabeças de detecção (*decoupled head*) para processar independentemente as tarefas de classificação de objetos e regressão de caixas delimitadoras (SENGUN; AKSOY; SERTEL; FRANSSON, 2025).

### 2.6.2 YOLOv11: refinamento arquitetural com C3k2 e atenção espacial

O YOLOv11, lançado em 2024, representa um avanço focado na eficiência de parâmetros e na capacidade de extração de características complexas (KHANAM; HUSSAIN, 2024). Embora mantenha a espinha dorsal da família YOLO, o v11 substitui o módulo C2f pelo novo bloco C3k2. Este bloco é uma evolução mais compacta e eficiente, permitindo o uso de *kernels* customizáveis para capturar detalhes mais finos com menor custo computacional.

Outra inovação crítica no v11 é a introdução do bloco C2PSA (*Cross Stage Partial with Spatial Attention*) logo após o módulo SPPF (*Spatial Pyramid Pooling - Fast*). O C2PSA incorpora mecanismos de atenção espacial que permitem ao modelo focar em regiões críticas da imagem, melhorando a detecção de objetos parcialmente ocluídos ou de difícil distinção, uma capacidade que estava ausente na arquitetura padrão do v8 (KHANAM; HUSSAIN, 2024).

### 2.6.3 As versões Nano (v8n e v11n) para dispositivos de borda

Para hardware de borda com recursos reduzidos, como o Raspberry Pi, as versões Nano são fundamentais.

- YOLOv8n: É a versão mais leve da arquitetura v8, obtida através da poda (*pruning*) e simplificação dos canais da rede, mantendo os princípios do design C2f. É amplamente utilizada como *baseline* para sistemas que exigem inferência rápida (LIU, 2024).
- YOLOv11n: A versão Nano do v11 demonstra uma eficiência superior. Com aproximadamente 2,6 milhões de parâmetros e 6,5 GFLOPs, o YOLOv11n consegue superar seus antecessores em precisão (mAP) mantendo uma latência extremamente baixa (MAKKAR, 2025). A arquitetura da cabeça de detecção do v11 utiliza convoluções em profundidade (*depthwise convolutions*), o que reduz drasticamente a quantidade de parâmetros e operações de ponto flutuante em comparação com as camadas convolucionais padrão usadas no v8. Testes de *benchmark* indicam que o YOLOv11n pode atingir velocidades de inferência superiores a 170 FPS em GPUs modernas, tornando-o ideal para aplicações de tempo real (KHANAM; HUSSAIN, 2024).

### 2.6.4 A modalidade de classificação (CLS)

Além da detecção de objetos (que fornece localização e classe), tanto o YOLOv8 quanto o YOLOv11 suportam nativamente a tarefa de Classificação de Imagens (sufixo -cls). Nesta modalidade, a rede é modificada para atribuir uma categoria a uma imagem inteira, eliminando a regressão de caixas delimitadoras.

- Arquitetura CLS: nos modelos -cls (como yolov8n-cls e yolo11n-cls), o *backbone* extrator de características é preservado, mas a "cabeça" de detecção complexa é substituída por uma camada linear simplificada. O modelo processa a imagem para gerar *logits*, que representam a probabilidade de cada classe.
- Aplicações práticas: a versão CLS é computacionalmente mais leve que a versão de detecção. O YOLOv8n-cls, por exemplo, é otimizado para cenários em que a localização espacial do objeto não é necessária, oferecendo convergência e predições mais rápidas em função do menor número de parâmetros na camada de saída (LIU, 2024). Devido a essa característica de leveza, a arquitetura de classificação mostra-se especialmente adequada para aplicações em *edge AI* e sistemas embarcados, tornando-se uma solução potencial para o problema analisado. Entretanto, conforme

destacado por Liu, seu uso é recomendado apenas em contextos nos quais a localização do objeto não é um requisito. No caso em estudo, a identificação do pacote pode ser realizada no momento em que ele é capturado pelas garras de retenção, sendo suficiente acompanhar se o objeto foi devidamente acomodado no fundo antes do acionamento do mecanismo de fechamento. O YOLOv11 expande essa capacidade, permitindo não apenas classificação simples, mas servindo como um extrator de características robusto para tarefas secundárias (KHANAM; HUSSAIN, 2024). Para a triagem de produtos em uma esteira, onde o objeto ocupa a maior parte do quadro, o uso de um modelo CLS (ex: yolo11n-cls) pode oferecer uma alternativa mais veloz e performática do que o modelo de detecção completo.

## 2.7 *Dataset* para treinamento de modelos de inteligência artificial

A qualidade e o volume do conjunto de dados (*dataset*), definido como o conjunto estruturado de amostras utilizadas para treinamento, validação e teste de modelos, constituem fatores determinantes para o sucesso de projetos de Inteligência Artificial, especialmente em aplicações de *machine learning*, *deep learning* e *edge AI*. Tanto em abordagens clássicas de aprendizado de máquina quanto em modelos modernos baseados em redes neurais profundas, o desempenho final do sistema está diretamente condicionado à representatividade, consistência e confiabilidade dos dados utilizados no processo de treinamento.

Essa dependência crítica reflete uma mudança de paradigma na engenharia de software, onde o foco se desloca do aperfeiçoamento exclusivo dos algoritmos para a priorização dos dados, movimento conhecido como IA centrada em dados (*data-centric AI*, em inglês). Conforme destacam Whang et al. (2023), "sem bons dados, até mesmo os melhores algoritmos de aprendizado de máquina não conseguem ter um bom desempenho". Os autores explicam que, embora as abordagens recentes de *deep learning* reduzam a necessidade de engenharia de características manual (*feature engineering*), elas exigem, em contrapartida, volumes muito maiores de dados de treinamento para serem eficazes. Além disso, devido ao alto poder expressivo das redes neurais profundas, existe o risco de o modelo "memorizar" ruídos e erros de anotação se a qualidade dos dados não for rigorosamente tratada, tornando a validação e a limpeza dos dados etapas que consomem a maior parte do tempo de desenvolvimento de um projeto.

Os autores de AI at the Edge utilizam uma metáfora afirmando que os dados funcionam simultaneamente como os “tijolos” e o “termômetro” de um projeto de

aprendizado de máquina. Como “tijolos”, os dados constituem a base estrutural do modelo: se determinado padrão não está presente no conjunto de treinamento — ou se está sub-representado — o modelo não será capaz de generalizá-lo ou fazê-lo de forma confiável. Como “termômetro”, o *dataset* de validação e teste é responsável por medir o desempenho do modelo; caso esse conjunto não represente fielmente o cenário real de operação, as métricas obtidas serão enganosas, levando a uma falsa percepção de eficiência (SITUNAYAKE; PLUNKETT, 2023, p. 84).

Essa dependência torna-se ainda mais crítica em aplicações de visão computacional, devido à elevada variabilidade do mundo real (SITUNAYAKE; PLUNKETT, 2023, p. 305). Fatores como iluminação, ângulo de captura, oclusões, contexto de fundo e combinações inesperadas de objetos impõem desafios significativos à generalização dos modelos. Um exemplo emblemático citado em AI at the Edge é o acidente envolvendo um veículo autônomo da Uber, no qual o sistema falhou ao classificar corretamente uma pedestre que atravessava a via empurrando uma bicicleta com sacolas plásticas (SITUNAYAKE; PLUNKETT, 2023, p. 304). A falha foi atribuída, em grande parte, à ausência ou escassez de exemplos semelhantes no *dataset* de treinamento, evidenciando as consequências práticas e éticas de conjuntos de dados inadequados.

Nesse contexto, ganha destaque a abordagem conhecida como aprendizado de máquina centrado em dados (*data-centric machine learning*, em inglês), que propõe uma mudança de paradigma: em vez de concentrar esforços principalmente na otimização de arquiteturas e hiperparâmetros, prioriza-se a melhoria contínua do *dataset*. Isso inclui a correção de rótulos, o balanceamento entre classes, a remoção de *outliers* e a inclusão de exemplos representativos de casos limítrofes. Argumenta-se que um conjunto de dados menor, porém bem curado, pode produzir resultados superiores a um *dataset* volumoso, porém ruidoso e mal rotulado (SITUNAYAKE; PLUNKETT, 2023, p. 307).

## 3. METODOLOGIA

A presente pesquisa caracteriza-se como um estudo de desenvolvimento tecnológico de natureza aplicada, fundamentado na integração de sistemas embarcados e visão computacional para a otimização de processos na indústria de beneficiamento do coco. A metodologia foi dividida em um fluxo incremental, seguindo algumas fases como concepção

de arquitetura, seleção de hardware, desenvolvimento de software, procedimentos de coleta de dados em campo, treinamento do modelo e testes locais.

### 3.1 Arquitetura e estratégia de processamento

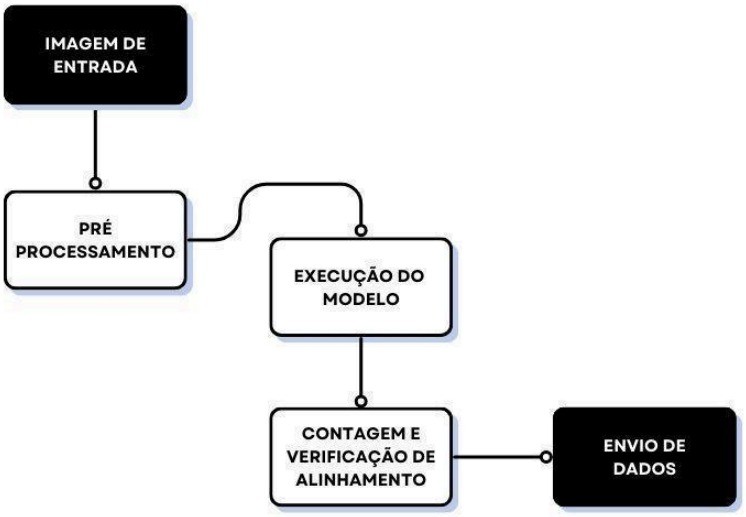
A arquitetura da solução baseia-se no paradigma de *edge computing*, priorizando o processamento de dados na borda da rede para atender aos requisitos de baixa latência e autonomia exigidos pelo ambiente fabril da Coco & Cia. A estratégia de implementação será orientada pelos critérios do acrônimo BLERP (*Bandwidth, Latency, Economics, Reliability, Privacy*), com foco especial na latência e na disponibilidade do sistema independentemente de conectividade estável com a nuvem.

#### 3.1.1 Processamento dos fardos de leite de coco

No contexto da linha de produção de fardos de leite de coco, a arquitetura proposta contempla um pipeline de visão computacional embarcado responsável pela aquisição contínua de imagens, seguido por um módulo de detecção de objetos voltado à identificação das tampas das garrafas, utilizadas como referência visual para a contagem das unidades presentes em cada fardo.

A partir das detecções, o sistema realiza a verificação da quantidade de garrafas, comparando o número identificado com o padrão esperado, além da análise do alinhamento dos fardos, avaliando a disposição espacial relativa das tampas para identificar desalinhamentos ou configurações fora do padrão. Esse processamento local permite a identificação em tempo quase real de não conformidades, como ausência de garrafas, excesso de unidades ou fardos desalinhados, fornecendo subsídios para ações corretivas imediatas na linha de produção por meio de alerta visual.

Figura 1 - Fluxograma da solução para a linha de produção de fardos de leite de coco.




Fonte: dos autores.

### 3.1.2 Processamento dos pacotes de derivados de coco

Para solucionar o problema de identificação de falhas na queda dos pacotes na garra de retenção, foi desenvolvida uma arquitetura de software híbrida. Esta combina a inferência visual em tempo real, realizada por um modelo YOLO na modalidade de Classificação (*Classification* - CLS), com uma Máquina de Estados Finitos (FSM - *Finite State Machine*) determinística. Diferentemente da detecção de objetos tradicionais, que busca localizar itens na imagem, a abordagem CLS foi adotada para categorizar o status global da cena observada pela câmera fixa no teto.

### 3.1.3 Classificação de estados visuais

O modelo YOLO (avaliado nas versões v8n-cls e v11n-cls) foi treinado para identificar classes que representam os diferentes momentos do ciclo de produção, e não apenas o objeto em si. As classes monitoradas incluem:

- Produtos específicos: exemplo: *ralado_umid_adoc_1kg* ou *flocado_int_umido...*, indicando a presença de um pacote na zona de leitura.
- Estado de transição: a classe *Garra_com_pacote*, que representa o momento em que o produto está nas garras de retenção e aguarda queda para o fardo quando a lógica da máquina responsável determinar.

- Estado de vazio: essa classe é um marco onde faz a confirmação que o ciclo anterior foi concluído e que a garra está livre para receber novos itens.
- Algoritmo de controle e detecção de falhas: a lógica de negócio foi implementada para obedecer regras pré-definidas para cada produto.
- Sinalização de conformidade: ao final do ciclo, o sistema compara a contagem física validada com a regra do produto. Se a contagem for inferior à meta (devido a uma falha de queda), o sistema irá emitir um alerta visual e registra o fardo como "INCOMPLETO". Se a contagem bater com a regra e a garra estiver vazia, o fardo é validado como "COMPLETO".

Essa arquitetura garante que a visão computacional não atue apenas como um sensor passivo, mas como um verificador ativo da sincronia mecânica da máquina de enfardamento.

### 3.1.4 Dados obtidos dos sensores

Como extensão da arquitetura proposta, prevê-se a integração do sistema de inspeção visual com um *dashboard* de monitoramento e visualização de dados, responsável por consolidar e apresentar, de forma clara e intuitiva, as informações geradas. Nesse ambiente, seriam disponibilizados indicadores como quantidade de fardos produzidos, quantidade de ocorrências de não conformidades e estatísticas agregadas ao longo do tempo, permitindo o acompanhamento do desempenho da linha de produção.

### 3.1.5 Otimização energética

Para otimizar o consumo energético e a vida útil do hardware, será implementada uma lógica de despertar (*wake-up*) via sinal externo (CLP ou sensor não intrusivo). Este mecanismo permite que o algoritmo de visão computacional seja executado apenas durante a passagem dos produtos, minimizando a utilização do processador em períodos de inatividade da esteira.

## 3.2 Materiais e hardware utilizado

O sistema embarcado deve ser construído em torno da plataforma Raspberry Pi 3 Model B, selecionada por seu equilíbrio entre custo, dimensões reduzidas e suporte a bibliotecas de inteligência artificial. Os principais componentes físicos incluem:

- **Unidade de processamento:** Raspberry Pi 3 Model B com processador quad-core ARM Cortex-A53 e 1 GB de RAM.
- **Sensores de Imagem:** câmeras USB 2.0 (Webcam) full HD de modelo genérico, integradas via protocolo UVC, permitindo captura de vídeo em tempo real compatível com a biblioteca de processamento de imagem OpenCV.
- **Encapsulamento:** gabinetes personalizados projetados pela equipe e fabricados via impressão 3D em filamento de plástico ABS (acrilonitrila butadieno estireno), garantindo proteção contra poeira, umidade e vibrações industriais.

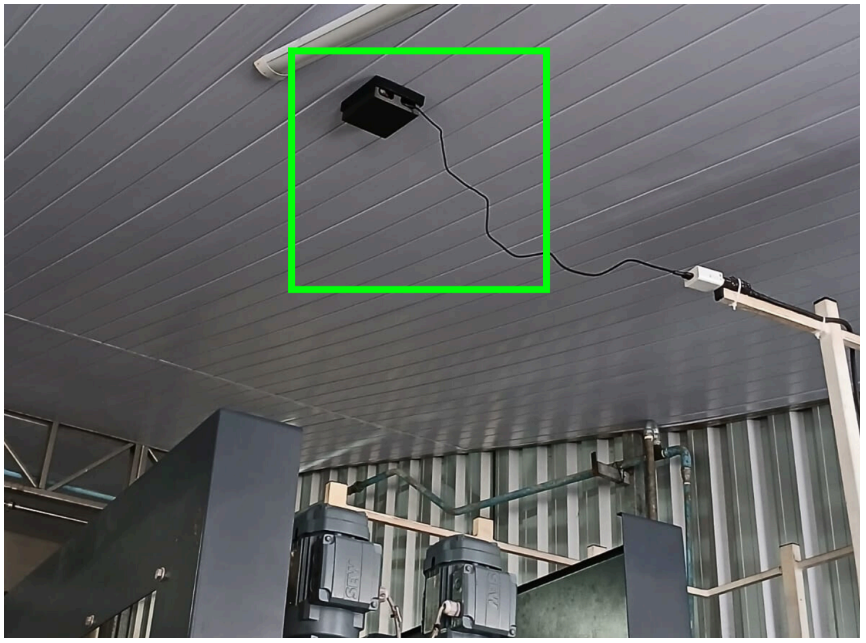
### 3.3 Coleta de dados

A coleta de dados para o sistema ocorreu em duas etapas principais: testes controlados em laboratório para calibração de resolução e taxa de quadros (*frame rate*), seguida pela instalação de dois dispositivos em pontos críticos da linha de produção da Coco & Cia:

- Monitoramento de enfardamento: o sistema de captura foi instalado no teto, diretamente acima da esteira de enfardamento, proporcionando uma visão superior ortogonal da disposição dos pacotes. Essa configuração permite a detecção individual dos pacotes, possibilitando a contagem automática de unidades por fardo e a identificação de fardos incompletos, com excesso de pacotes ou com unidades danificadas, além de auxiliar na verificação da uniformidade da disposição dos pacotes durante o processo de enfardamento.
- Monitoramento de garrafas de leite de coco: o sistema de captura foi posicionado na grade de resfriamento, com vista superior dos fardos durante o deslocamento na esteira, possibilitando a identificação de não conformidades como garrafas ausentes, em excesso ou dispostas de forma desalinhada. Tais condições podem comprometer a padronização do produto e dificultar etapas subsequentes do processo produtivo e logístico.

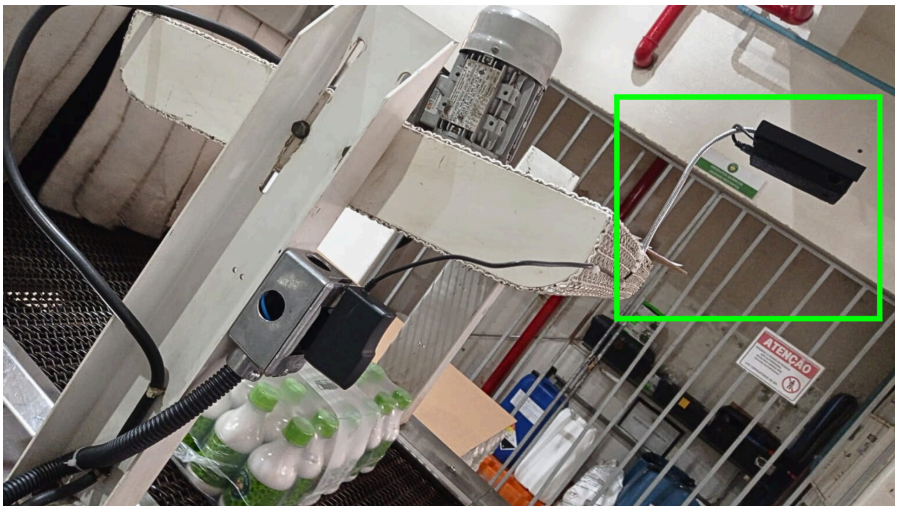
Os dados coletados em ambiente real de produção foram utilizados para o ajuste fino (*fine-tuning*) de um modelo baseado na arquitetura YOLOv11n, escolhida por seu equilíbrio entre desempenho e eficiência computacional. Imagens do hardware de captura para a construção do conjunto de dados acima da máquina de empacotamento de produtos como coco ralado e coco flocado (Figura 1), e na grade de resfriamento, acima da esteira de enfardamento de garrafas de leite de coco (Figura 2), de modo a garantir um campo de visão adequado e representativo das condições reais de operação:

Figura 2 - Hardware de coleta de imagens de pacotes.



Fonte: dos autores.

Figura 3 - Hardware de coleta de imagens de fardos de garrafas de leite de coco.




Fonte: dos autores.

### 3.4 Desenvolvimento de software e modelagem para sistema de garrafas

O ecossistema de software foi desenvolvido em linguagem Python (3.10.x), utilizando bibliotecas especializadas para visão computacional e aprendizado de máquina. As principais ferramentas utilizadas foram:

- OpenCV (4.12.x) e Numpy (2.2.x): para captura, pré-processamento de imagens e manipulação matricial dos quadros.
- Ultralytics (8.4.8): *framework* utilizado para a implementação da arquitetura YOLO (*You Only Look Once*), permitindo a detecção e classificação de objetos em tempo real com eficiência em hardware restrito.
- Plataformas de treinamento: a anotação do *dataset* e a organização dos dados foram realizadas na plataforma Roboflow. Segundo Matuck et al. (2023), o Roboflow é uma plataforma de visão computacional voltada para facilitar o treinamento e a implantação de modelos de detecção de objetos em tempo real. Ela foi desenvolvida para reduzir a complexidade técnica envolvida na criação de sistemas de reconhecimento de imagens, tornando esse processo mais acessível. Já o treinamento dos modelos de detecção foi executado em ambiente Google Colaboratory para aproveitar recursos de aceleração por GPU. O Google Colaboratory (ou Google Colab) é um ambiente de notebooks Jupyter baseado em nuvem, disponibilizado pelo Google, que permite escrever e executar código Python diretamente no navegador, com acesso a recursos computacionais gratuitos como GPUs, sem necessidade de configuração local (DA SILVA, 2020).

### 3.5 Desenvolvimento de software e modelagem para sistema de pacotes

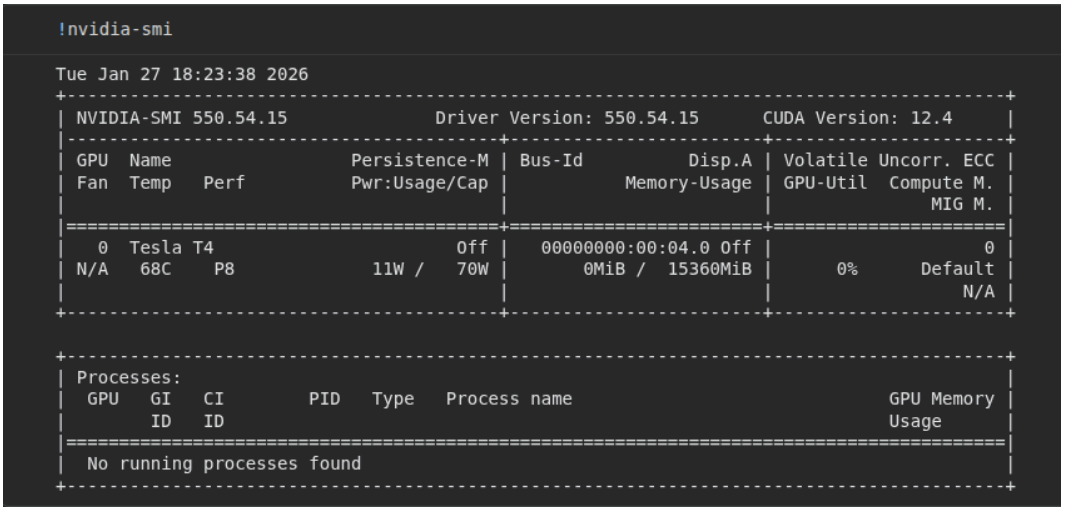
Devido à alta demanda computacional necessária para o ajuste dos pesos das redes neurais profundas, a etapa de treinamento não foi realizada no hardware embarcado (Raspberry Pi), mas sim em ambiente de nuvem de alto desempenho. Utilizou-se a plataforma Google Colaboratory, que disponibiliza recursos de aceleração de hardware dedicados à Inteligência Artificial. Ambos os modelos V11n e V8n foram treinados com o mesmo *dataset*, mesma quantidade de classes (quatro), os mesmos hiperparâmetros e o mesmo ambiente de nuvem.

#### 3.5.1 Ambiente computacional

O ambiente de treinamento foi configurado sobre o sistema operacional Linux virtualizado pelo Google Colab, utilizando a linguagem Python 3.12.12 e o *framework* de aprendizado profundo PyTorch (versão 2.9.0+cu126) integrado à biblioteca Ultralytics 8.4.8. Para acelerar o processo de retropropagação e ajuste de pesos, foi alocada uma GPU

(*Graphics Processing Unit*) modelo NVIDIA Tesla T4, equipada com 15.360 MiB (aproximadamente 15 GB) de memória de vídeo (VRAM) e suporte à arquitetura CUDA 12.4, garantindo a execução eficiente das operações matriciais complexas exigidas pela arquitetura YOLO.

Figura 4 - Monitoramento da GPU com NVIDIA-SMI.




Fonte: dos autores.

### 3.5.2 Definição de hiperparâmetros

O processo de treinamento foi parametrizado para otimizar a convergência do modelo e evitar o sobreajuste (*overfitting*), considerando o tamanho limitado do *dataset*. As configurações aplicadas ao método model.train() foram:

- Épocas (*Epochs*): o treinamento foi limitado a 30 épocas, ciclo suficiente para a estabilização da perda (*loss*) em datasets de transferência de aprendizado.
- Tamanho do lote (*Batch Size*): definiu-se o valor de 16 imagens por lote, adequado à memória da GPU T4 para manter a estabilidade do gradiente.
- Dimensão da imagem (*ImgSz*): as imagens de entrada foram redimensionadas para 384×384 pixels, um equilíbrio entre resolução para detecção de detalhes (como tampas de garrafa) e custo computacional.
- Otimizador: utilizou-se o algoritmo AdamW (*Adaptive Moment Estimation with Weight Decay*), com uma taxa de aprendizado inicial (*lr0*) de 0,001 e decaimento de

peso (*weight_decay*) de 0,0005, favorecendo uma generalização melhor do que o otimizador SGD padrão.

### 3.5.3 Técnicas de aumentação de dados (*data augmentation*)

Para aumentar a variabilidade do conjunto de dados e tornar o modelo resiliente às mudanças no ambiente industrial (como vibração da câmera e posição dos pacotes), foram aplicadas técnicas de aumentação de dados em tempo real durante o treinamento:

- Geométricas: rotação aleatória de até ±10 graus (*degrees=10*), espelhamento horizontal com 50% de probabilidade (*fliplr=0.5*) e escalonamento da imagem em 20% (*scale=0.2*).
- Regularização: aplicação de *dropout* de 10% (*dropout=0.1*) e suavização de rótulos (*label_smoothing=0.05*) para impedir que o modelo se torne excessivamente confiante em previsões ruidosas.
- Oclusão: utilizou-se a técnica de apagamento aleatório (*erasing=0.015*) para simular oclusões parciais, forçando a rede a aprender características mais robustas do objeto.

### 3.5.4 Resultados do Treinamento e Exportação

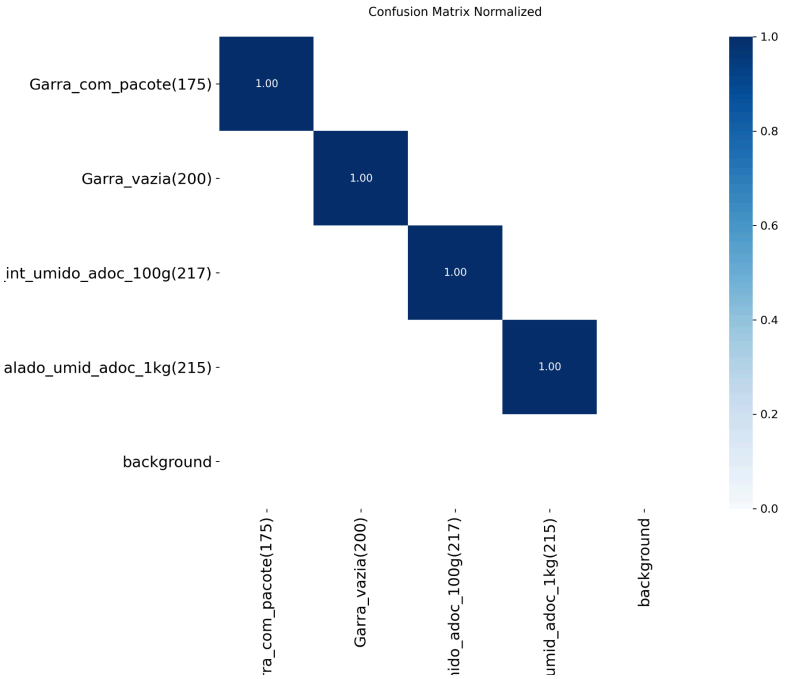
O treinamento foi aplicado especificamente à arquitetura YOLO11n-cls e versão YOLO8N-CLS (versão Nano para classificação), resultando em modelos compactos com o v11n apresentando 47 camadas, aproximadamente 1,53 milhão de parâmetros e um custo computacional de 3,2 GFLOPs e o v8n apresentando 30 camadas, 1,44 milhão de parâmetros e um custo computacional de 3.3 GFLOPs. Após a validação, os pesos do modelo (*best.pt*) foram exportados para o formato NCNN (*best_ncnn_model*), uma biblioteca de computação neural de alto desempenho otimizada para dispositivos móveis e sistemas embarcados ARM, visando a implantação final no Raspberry Pi.

Os resultados do treinamento demonstram que ambas as arquiteturas alcançaram uma convergência ideal, atingindo 100% de acurácia (Top-1) no conjunto de validação ao final das 30 épocas, conforme confirmado pelas matrizes de confusão que mostram pontuação perfeita (1.00) para todas as classes. Entretanto, o YOLOv8n apresentou uma estabilidade superior: ele atingiu a precisão máxima de 100% mais cedo (na época 19) do que o YOLOv11n (na época 21) e manteve uma curva de perda de validação (*val*/*loss*) mais consistente, evitando

oscilações como o pico de erro observado na época 13 do modelo v11n, onde a perda subiu momentaneamente para 0.1059.

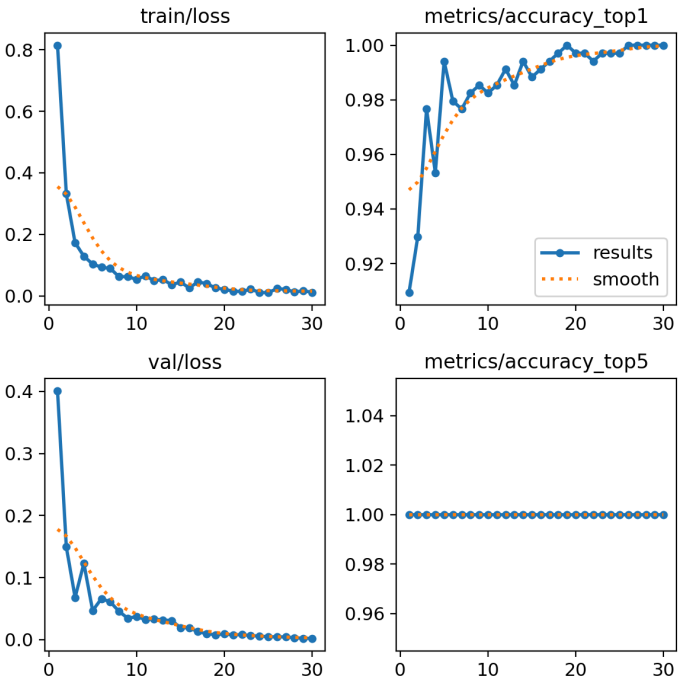
Embora o modelo v11n tenha iniciado o processo com uma perda de treinamento (*train*/*loss*) menor que a do v8n (0.54 contra 0.81), ambos finalizaram o treinamento com valores de perda de validação extremamente baixos e praticamente idênticos (~0.002). Isso indica que a complexidade adicional da arquitetura v11n não resultou em uma capacidade de generalização superior para este conjunto de dados, validando a escolha do v8n pela sua convergência mais rápida e linear.

Figura 5 - Matriz de confusão do modelo classificador YOLO v11n




Fonte: dos autores.

Figura 6 - Matriz de confusão do modelo classificador YOLO v11n




Fonte: dos autores.

### 3.6 *Deploy* do modelo: análise comparativa de desempenho dos modelos YOLOv8n *versus* YOLOv11n

Para determinar a arquitetura mais adequada às restrições do projeto e aos requisitos de precisão, foi conduzido um estudo comparativo entre as versões Nano dos modelos YOLOv8 e YOLOv11. O experimento visou avaliar a melhor relação entre eficiência computacional (FPS e consumo de recursos) e eficácia na detecção (acurácia e confiança) em um cenário controlado.

#### 3.6.1 Ambiente de experimentação e hardware

Os experimentos de *benchmark* comparativo foram executados em uma estação de trabalho para garantir a coleta de métricas sem gargalos severos de hardware e para validar os modelos em arquitetura x86_64 antes da portabilidade. O equipamento utilizado foi um computador portátil Dell Inspiron 15 3511, equipado com processador Intel® Core™ i3-1115G4 (11ª geração, 3.00 GHz, 2 núcleos físicos e 4 *threads*) e instruções vetoriais AVX2/AVX-512.

O sistema contou com 8 GB de memória RAM DDR4 operando a 2667 MT/s e armazenamento via SSD NVMe de 256 GB. O processamento gráfico e a inferência foram realizados exclusivamente via CPU (Intel® UHD Graphics G4 utilizada apenas para vídeo), caracterizando um cenário CPU-*bound* que simula as limitações de dispositivos de borda sem aceleração dedicada. O sistema operacional utilizado foi o Linux Ubuntu 24.04.2 LTS sem virtualização.

### 3.6.2 Protocolo de coleta e aquisição de imagens

A aquisição de imagens foi realizada através de uma webcam integrada (*driver* UVC), configurada com resolução de 640×480 pixels e taxa de 30 quadros por segundo (FPS) no formato YUYV 4:2:2. Para garantir a consistência dos dados de entrada durante o teste comparativo, as classes de produtos foram exibidas visualmente através de um tablet Samsung Galaxy Tab S6 Lite (tela de 10,4 polegadas), posicionado de forma fixa frente à câmera, eliminando variações de iluminação ambiente e ângulo que poderiam ocorrer com objetos físicos em movimento aleatório.

### 3.6.3 Algoritmo de *benchmark* e métricas avaliadas

Foi desenvolvido um algoritmo em linguagem Python para automatizar o processo de *benchmarking*, que consiste em medir e comparar o desempenho de sistemas, modelos ou algoritmos usando métricas padronizadas. O *script* foi projetado para iterar sobre uma lista pré-definida de classes, que incluiu: "*ralado_umid_adoc_1kg*", "*Garra_vazia*", "*flocado_int_umido_adoc_100g*" e "*Garra_com_pacote*".

O procedimento experimental seguiu as seguintes etapas para cada modelo (v8n e v11n):

1. Inferência sequencial: o sistema realizou a captura e inferência de 100 frames consecutivos para cada classe alvo, totalizando um conjunto de testes padronizados.
2. Monitoramento de recursos: utilizando a biblioteca *psutil*, foram coletados dados em tempo real sobre o uso de CPU (normalizado pelo número de núcleos) e o consumo de memória RAM (em MB) a cada quadro processado.
3. Métricas de desempenho: foram registrados o tempo de inferência (em milissegundos) e a taxa de quadros por segundo (FPS) real obtida durante a execução.

4. Métricas de precisão: foram armazenados a classe predita, a classe real (*ground truth*) e o nível de confiança (*confidence*) da detecção para posterior cálculo de acurácia.

### 3.6.4 Processamento e visualização dos dados

Os dados brutos coletados foram exportados para arquivos CSV e processados utilizando as bibliotecas Pandas e Seaborn. A análise comparativa baseou-se nos seguintes indicadores consolidados:

- Acurácia global e por classe: comparação percentual de acertos entre o valor predito e o valor verdadeiro para cada modelo.
- Matriz de confusão: geração de matrizes normalizadas e absolutas para identificar confusões frequentes entre classes similares.
- Estabilidade de FPS: análise da distribuição de FPS (via boxplot e linha temporal) para verificar a fluidez do processamento.
- Eficiência energética/computacional: comparação do uso percentual de CPU e alocação de memória RAM ao longo do tempo.

Essa metodologia permitiu uma validação quantitativa robusta para suportar a decisão sobre qual versão da arquitetura YOLO oferece o melhor equilíbrio para a aplicação final na indústria de beneficiamento de coco.

# 4. RESULTADOS E DISCUSSÕES

## 4.1 Modelo de detecção de garrafas

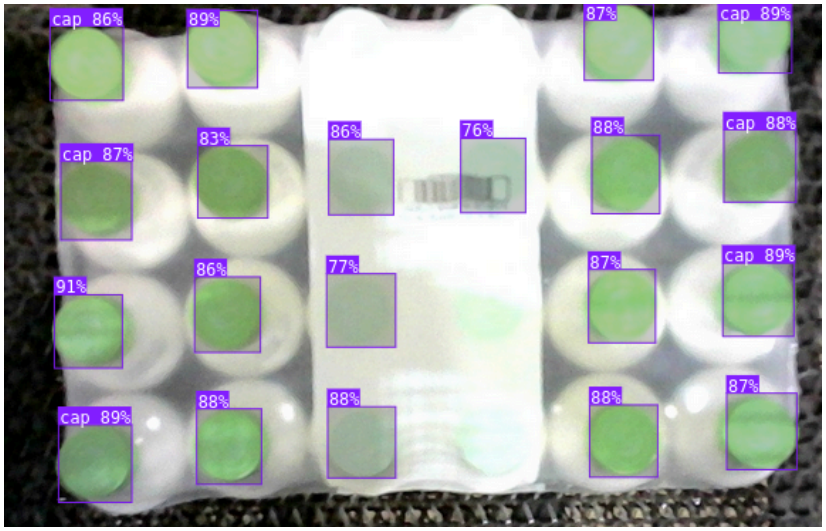
O modelo treinado para a detecção das tampas das garrafas de leite de coco apresentou resultados satisfatórios. Utilizando um *dataset* com aproximadamente 300 imagens, após a aplicação de técnicas de *data augmentation* (aumentação de dados) — empregadas para aumentar artificialmente a quantidade e a diversidade dos dados de treinamento — o modelo alcançou uma precisão de 97,1%, métrica que indica a proporção de detecções corretas entre todas as detecções realizadas, refletindo uma baixa taxa de falsos positivos. O *recall* de 99.8%, por sua vez, mede a capacidade do modelo de identificar corretamente todos os objetos presentes nas imagens, demonstrando que poucas tampas deixaram de ser detectadas.

Os testes foram realizados em uma máquina equipada com o processador Intel i7-13620H de 10 núcleos e 16 GB de memória RAM, sem GPU dedicada. Durante a execução do modelo em um vídeo capturado na linha de produção, o sistema apresentou um desempenho estável, com taxa de processamento variando entre 20 e 25 quadros por segundo (FPS).

O modelo demonstrou capacidade de identificar corretamente os objetos de interesse em condições controladas. No entanto, variações de iluminação e reflexos típicos do ambiente industrial impactaram negativamente o desempenho das detecções. Em particular, a incidência de iluminação intensa em determinadas regiões da linha de produção provoca a superexposição das tampas, fazendo com que elas apresentem aparência excessivamente clara ou esbranquiçada nas imagens, o que dificulta sua correta identificação pelo modelo, como pode ser visto na Figura 3.

Esses fatores evidenciam a necessidade de melhorias nas etapas de pré-processamento das imagens, bem como a ampliação e diversificação do conjunto de dados de treinamento, contemplando diferentes condições de iluminação e posicionamento, além da adoção de estratégias voltadas ao aumento da robustez do modelo frente às variações e adversidades do ambiente real de operação.

Figura 4 - Detecção das tampas de garrafas de leite de coco.




Fonte: dos autores.

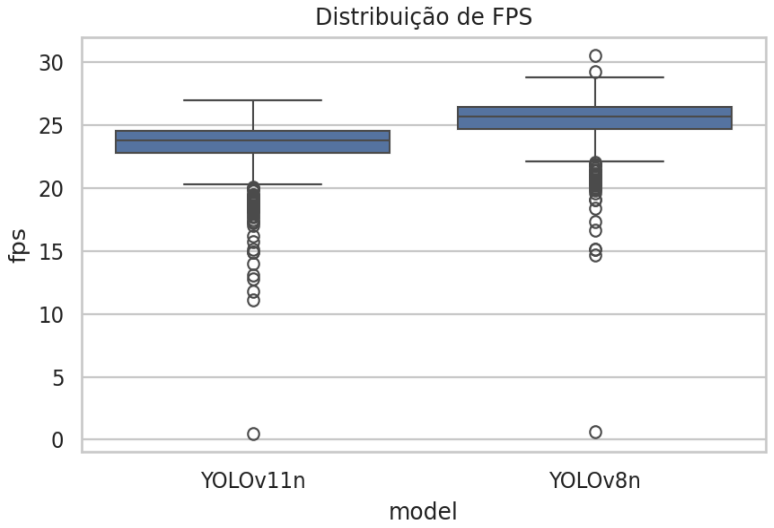
### 4.2 Modelo de detecção de pacotes

A avaliação comparativa entre os modelos YOLOv8n e YOLOv11n foi conduzida focando em duas vertentes principais: o desempenho computacional (velocidade de inferência e consumo de recursos) e a eficácia na detecção de objetos (acurácia e distinção entre classes). Os dados foram obtidos em um cenário de *benchmark* controlado, utilizando hardware de arquitetura x86_64 sem aceleração dedicada (CPU-*bound*).

### 4.2.1 Análise de Desempenho Computacional

A taxa de quadros por segundo (FPS) é uma métrica crítica para sistemas de monitoramento industrial em tempo real. Conforme ilustrado na distribuição de FPS, observou-se uma vantagem de desempenho do modelo YOLOv8n.

Figura 7 - *Boxplot* de distribuição de frames por segundo entre modelos

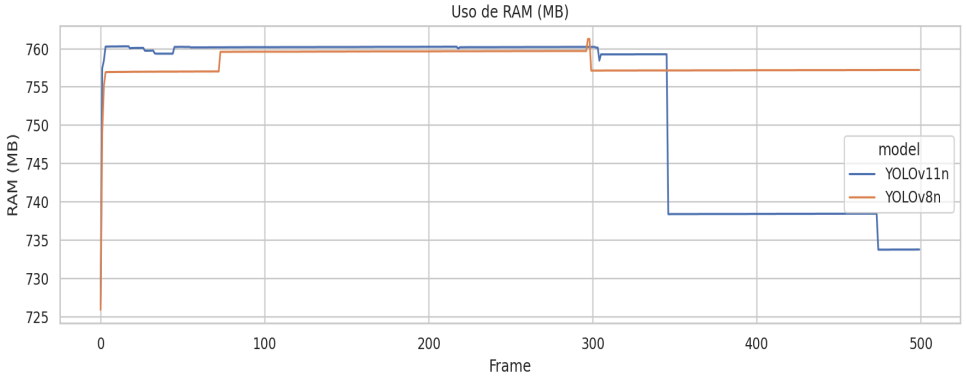



Fonte: dos autores.

- Velocidade de inferência (FPS): o YOLOv8n apresentou uma mediana de desempenho superior, situando-se acima de 25 FPS, enquanto o YOLOv11n operou com uma mediana na faixa de 23 a 24 FPS. Analisando os logs de inferência, o YOLOv8n manteve tempos frequentemente abaixo de 40ms (ex: 37-39ms), enquanto o v11n oscilou na faixa de 40-45ms.
- Estabilidade: ambos os modelos apresentaram *outliers* de baixa performance, mas mantiveram uma consistência operacional adequada.

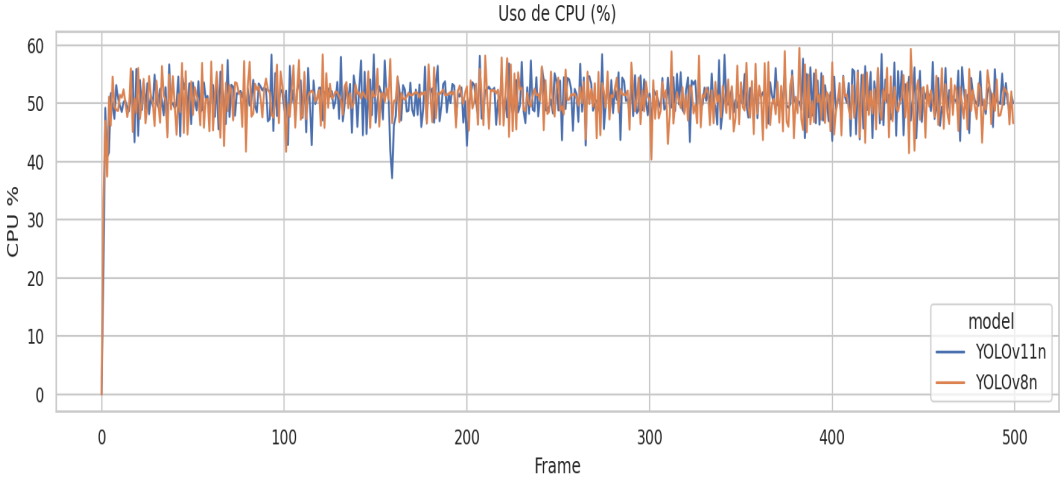
No que tange ao consumo de recursos de hardware, observou-se um comportamento dinâmico distinto entre as versões:

Figura 8 - Gráfico de linha do consumo de memória dos modelos




Fonte: dos autores.

Figura 9 - Gráfico de linha do consumo de uso de CPU




Fonte: dos autores.

- Memória RAM: inicialmente, ambos os modelos exigiram aproximadamente 760 MB. Contudo, a análise temporal revelou uma gestão de memória diferenciada pelo YOLOv11n: após o quadro 340, seu consumo caiu para cerca de 738 MB, mantendo-se mais eficiente que o YOLOv8n, que permaneceu estável em torno de 757 MB até o final do teste. Porém, devem ser realizados novos testes para confirmação se não foi apenas um evento isolado.

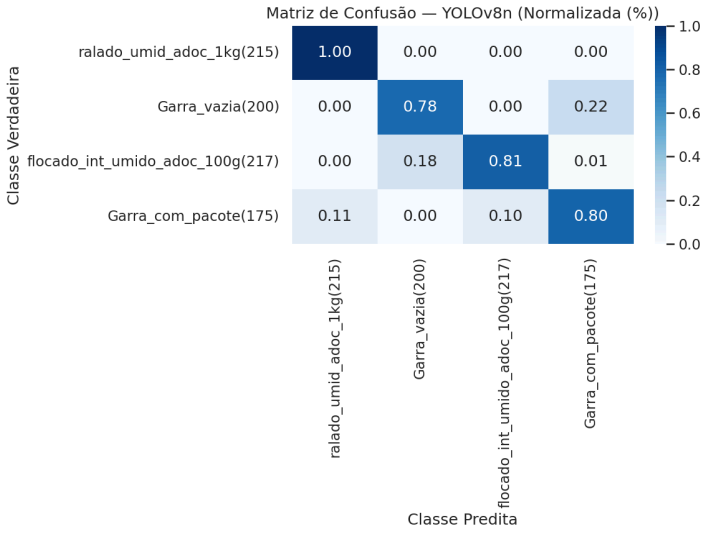
- Uso de CPU: O consumo de processamento manteve-se estável em torno de 50% para ambos os modelos, indicando similaridade na exigência de processamento central.

### 4.2.2 Análise de Eficácia de Detecção (Matriz de Confusão)

A precisão na classificação das classes "*Garra_vazia*" e "*Garra_com_pacote*" é fundamental para evitar a contagem de falsos positivos. A análise das matrizes de confusão normalizadas revelou um cenário de grande paridade técnica, contrariando a expectativa de superioridade significativa de uma versão sobre a outra:

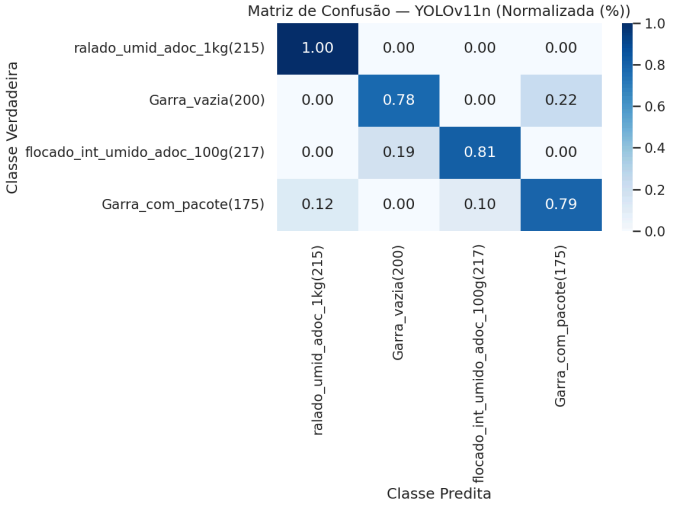
- Detecção de produtos sólidos: Para a classe *ralado_umid_adoc_1kg*, ambos os modelos atingiram 100% de acerto, demonstrando robustez ideal. Para o produto *flocado_int_umido_adoc_100g*, houve um empate técnico, com ambos os modelos acertando 81% das amostras e apresentando taxas de confusão idênticas.
- Discriminação de garras (vazia vs. cheia):
    - *Garra_vazia*: diferente das hipóteses iniciais, ambos os modelos obtiveram exatamente o mesmo desempenho, acertando 78% das amostras e confundindo 22% das vezes com "*Garra_com_pacote*". Não houve vantagem do YOLOv11n neste critério crítico.
    - *Garra_com_pacote*: o YOLOv8n apresentou uma ligeira vantagem, com 80% de acerto, comparado a 79% do YOLOv11n. O modelo v11n apresentou uma tendência marginalmente maior (12%) de confundir esta classe com o produto "ralado" do que o v8n (11%).

Figura 10 - Matriz de confusão normalizada do modelo v8n




Fonte: dos autores.

Figura 11 - Modelo yolo v8n classificando corretamente imagem do pacote




Fonte: dos autores

Figura 11 - Modelo yolo v8n classificando corretamente imagem do pacote


Fonte: dos autores.

### 4.2.3 Discussão e seleção do modelo

Os resultados quantitativos indicam que o YOLOv8n oferece o melhor equilíbrio para a aplicação. Em termos de precisão global, o v8n obteve 83.6% contra 83.4% do v11n.

Embora o YOLOv11n tenha demonstrado uma gestão de memória RAM mais eficiente em longas execuções (reduzindo o consumo em ~20MB na segunda metade do teste), ele não traduziu sua arquitetura mais recente em ganhos de acurácia para este *dataset* específico. Pelo contrário, o YOLOv8n mostrou-se marginalmente superior na detecção de pacotes nas garras e consistentemente mais rápido em inferência (maior FPS).

Considerando que a capacidade de distinguir garras de retenção vazias foi idêntica em ambos (78%) e que a velocidade é um fator relevante para o monitoramento em tempo real, optou-se pela utilização do YOLOv8n para as etapas seguintes. A escolha prioriza a maior velocidade de processamento e a ligeira vantagem na acurácia global, visto que a "robustez arquitetural" do v11n não se materializou em ganhos práticos de detecção neste cenário de teste.

## 5. CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo o desenvolvimento de um sistema embarcado para inspeção visual automatizada aplicado à indústria de beneficiamento do coco, buscando

substituir processos de inspeção manual passíveis de erros. Ao retomar os objetivos iniciais, os resultados iniciais do desenvolvimento do sistema confirmam a viabilidade e escalabilidade da solução proposta. Este estudo também validou a utilização de computação embarcada para o processamento de imagens e a automação de etapas como a contagem de pacotes e a detecção de falhas estruturais, reduzindo assim a dependência de processos de inspeção que são passíveis de erro humano, como a visão.

A pesquisa também confirmou a hipótese inicial de que é possível executar modelos de detecção de objetos (via YOLO) em dispositivos embarcados para o controle de qualidade industrial. As limitações de processamento do hardware durante a fase de coleta de material para treinamento do *dataset* foram observadas várias vezes. Contudo, foi observado, em geral, a viabilidade do sistema como um todo, por isso deu-se prosseguimento ao desenvolvimento do projeto com o hardware e o software embarcado neste artigo.

Um dos principais desafios técnicos foi a instabilidade na conexão remota via protocolo SSH com o Raspberry Pi 3 instalado ao lado das esteiras durante as etapas de captura de imagens em campo. Outro desafio analisado foi a obtenção de recursos (hardware) para desenvolvimento do projeto, viabilizados pelos próprios residentes do programa ou por meio de empréstimo de equipamentos de laboratório de pesquisa, resultado da demora considerável na aquisição dos componentes principais. E houve também uma pequena redução do escopo de materiais, mantendo o essencial para que a empresa pudesse viabilizar a compra. A adequação de escopo deve garantir a implementação do projeto no final da residência, como produto mínimo viável (MVP).

Como passos futuros para este desafio, conclui-se que será necessária a continuação do desenvolvimento do modelo de visão computacional, fazendo a otimização do *dataset*, compreendida como refinamento da rotulagem de imagens, o balanceamento de classes para redução de falsos positivos e a aplicação de técnicas de *data augmentation* para aumentar a robustez do modelo frente a variações de iluminação na fábrica. Além disso, prevê-se a substituição do hardware provisório para captura de imagens pelo hardware do projeto final, além da integração dos dados de aferição do modelo em um *dashboard* com métricas de produtividade das esteiras de leite de coco e pacotes, acessíveis à gestão e análise de dados para uso da empresa.

# 6. REFERÊNCIAS


ALMEIDA, Rodrigo Maximiano Antunes de; MORAES, Carlos Henrique Valério de; SERAPHIM, Thatyana de Faria Piola. **Programação de Sistemas Embarcados**: desenvolvendo software para microcontroladores em linguagem C. 1. ed. Rio de Janeiro: Elsevier, 2016.

BRUGES, Jim. **Edge AI:** Making sense of the noise. [S. l.]: Edge AI and Vision Alliance, 2019. Disponível em: https://pages.edgeimpulse.com/hubfs/Content%20and%20Docs/Edge%20Machine%20Learning%20for%20Industrial%20and%20Manufacturing%20Environments.pdf. Acesso em: 17 jan. 2026.

DA SILVA, Martony Demes. **Aplicação da Ferramenta Google Colaboratory para o Ensino da Linguagem Python.** In: ESCOLA REGIONAL DE ENGENHARIA DE SOFTWARE, 4., 2020, Evento Online. Anais da Escola Regional de Engenharia de Software. Porto Alegre: Sociedade Brasileira de Computação, 2020. p. 67-76. DOI: 10.5753/eres.2020.13717. Disponível em: https://sol.sbc.org.br/index.php/eres/article/view/13717. Acesso em: 26 jan. 2026.

DHIMAN, Jaskaran Singh. Semiconductor in Flux: Past Disruptions and Emerging Trends in Chip Industry. **International Journal of Emerging Trends in Computer Science and Information Technology**, [S. l.], p. 225–234, 2025. DOI: 10.56472/ICCSAIML25-128. Disponível em: https://ijetcsit.org/index.php/ijetcsit/article/view/202. Acesso em: 25 jan. 2026.

EMDAGRO. **Análise conjuntural da cultura do coco.** Aracaju: EMDAGRO, 2025. Disponível em: https://emdagro.se.gov.br/wp-content/uploads/2025/12/COCO-ANALISE-CONJUNTURAL.pdf. Acesso em: 24 jan. 2026.

HIDAYATULLAH, Priyanto; SYAKRANI, Nurjannah; SHOLAHUDDIN, Muhammad Rizqi; GELAR, Trisna; TUBAGUS, Refdinal. **YOLOv8 to YOLO11:** A Comprehensive Architecture In-depth Comparative Review. arXiv, 23 jan. 2025. Disponível em: https://arxiv.org/abs/2501.13400. Acesso em: 29 jan. 2026.

KHANAM, Rahima; HUSSAIN, Muhammad. **YOLOv11:** An Overview of the Key Architectural Enhancements. arXiv, 23 out. 2024. Disponível em: https://arxiv.org/abs/2410.17725. Acesso em: 29 jan. 2026.

KHANG, Alex; HAJIMAHMUD, Vugar Abdullayev; MISRA, Anuradha; LITVINOVA, Eugenia (Eds.). **Machine Vision and Industrial Robotics in Manufacturing:** Approaches, Technologies, and Applications. 1. ed. Boca Raton: CRC Press, 2024. DOI

https://doi.org/10.1201/9781003438137. Disponível em: https://www.taylorfrancis.com/books/edit/10.1201/9781003438137/machine-vision-industrial-robotics-manufacturing-alex-khang-vugar-abdullayev-hajimahmud-anuradha-misra-eugenia-litvinova. Acesso em: 17 jan. 2026.

LEITE, L. L. **Aplicação de Machine Learning em sistemas embarcados de borda:** um estudo de caso. Trabalho de Conclusão de Curso (Graduação em Engenharia de Controle e Automação) – Universidade Federal de Uberlândia, Uberlândia, 2023. Disponível em: https://repositorio.ufu.br/handle/123456789/38588. Acesso em: 25 jan. 2026.

LIU, Xiangchen. Comparison of Four Convolutional Neural Network-Based Algorithms for Sports Image Classification. In: **PROCEEDINGS of the 2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing (DAI 2023)**. Advances in Intelligent Systems Research, [S.l.], v. X, p. 178-186, 14 fev. 2024. DOI: 10.2991/978-94-6463-370-2_20. Disponível em: https://www.atlantis-press.com/proceedings/dai-23/125998112. Acesso em: 29 jan. 2026.

MAKKAR, Naman Balbir Singh. **VajraV1** – The most accurate Real Time Object Detector of the YOLO family. arXiv, 17 dez. 2025. Disponível em: https://arxiv.org/abs/2512.13834. Acesso em: 29 jan. 2026.

MATUCK, Guilherme Rodrigues; CASTRO, Arthur Jackson Abreu; DA SILVA, Lázaro Eduardo; CARVALHO, Eduardo Gomes. Reconhecimento facial com inteligência artificial utilizando a plataforma Roboflow. **Revista Prociências**, Pelotas, v. 6, n. 2, 2023. DOI: https://doi.org/10.15210/prociencias.v6i2.25948. Disponível em: https://periodicos.ufpel.edu.br/index.php/prociencias/article/view/25948. Acesso em: 27 jan. 2026.

OLIVEIRA, J. P. G. de. **Aprendizado de máquina na indústria 4.0:** detecção de anomalias em sistemas embarcados e classificação de substâncias. Tese (Doutorado em Engenharia Elétrica) – Universidade Federal de Pernambuco, Recife, 2022. Disponível em: https://repositorio.ufpe.br/handle/123456789/48267. Acesso em: 25 jan. 2026.

SAPKOTA, Ranjan; CHEPPALLY, Rahul Harsha; SHARDA, Ajay; KARKEE, Manoj. **YOLO26:** key architectural enhancements and performance benchmarking for real-time object detection. Ithaca, NY; Manhattan, KS: Cornell University; Kansas State University, 2026. Disponível em: https://www.arxiv.org/pdf/2509.25164. Acesso em: 18 jan. 2026.

SECO S.P.A. **RTOS vs. Embedded Linux**: A Decision Guide. [S. l.]: SECO, 2024. Disponível em: https://www.seco.com/blog/details/rtos-vs-embedded-linux-a-decision-guide. Acesso em: 17 jan. 2026.

SENGUN, Esra; AKSOY, Samet; SERTEL, Elif; FRANSSON, Johan E. S. Comparative Analysis of YOLOv8 and YOLOv11 on Tree Detection Using UAV RGB and Laser Scanning Data. In: **ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial**

**Information Sciences**, v. X-2/W2-2025, p. 173-179, 2025. Disponível em: https://isprs-annals.copernicus.org/articles/X-2-W2-2025/173/2025/. Acesso em: 29 jan. 2026.

SITUNAYAKE, Daniel; PLUNKETT, Jenny. **AI at the Edge:** Solving Real-World Problems with Embedded Machine Learning. 1. ed. Sebastopol: O'Reilly Media, 2023.

TAVARES, Lucas Bivar Fonsêca; SANTOS, Luís Henrique Lima. **Desenvolvimento de um sistema embarcado para classificação de produtos com visão computacional.** Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação) – Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, Campina Grande, 2025. Disponível em: https://repositorio.ifpb.edu.br/handle/177683/4388. Acesso em: 25 jan. 2026.

WEBASHA TECHNOLOGIES. **Embedded Linux Growth in 2026:** How edge computing, IoT devices, and RISC-V expansion are reshaping the future of open-source operating systems globally. WebAsha Technologies, 12 jul. 2025. Disponível em: https://www.webasha.com/blog/embedded-linux-growth-how-edge-computing-iot-devices-and-riscv-expansion-are-reshaping-the-future-of-open-source-operating-systems-globally. Acesso em: 29 jan. 2026.

WHANG, Steven Euijong; ROH, Yuji; SONG, Hwanjun; LEE, Jae-Gil. Data collection and quality challenges in deep learning: a data-centric AI perspective. **The VLDB Journal**, Heidelberg, v. 32, n. 4, p. 1–25, 2023.