UNIVERSIDADE FEDERAL DE ITAJUBÁ - UNIFEI
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Uso de aprendizado de máquina na borda para um sistema não invasivo de monitoramento de colmeias.

José Anderson dos Reis

Itajubá, 17 de dezembro de 2025

UNIVERSIDADE FEDERAL DE ITAJUBÁ - UNIFEI
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

José Anderson dos Reis

Uso de aprendizado de máquina na borda para um sistema não invasivo de monitoramento de colmeias.

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica como parte dos requisitos para obtenção do Título de Mestre em Ciências em Engenharia Elétrica.

Área de Concentração: Microeletrônica

Orientador: Prof. Dr. Gabriel Antonio Fanelli De Souza
Coorientador: Prof. Marcelo José Rovai

17 de dezembro de 2025
Itajubá

UNIVERSIDADE FEDERAL DE ITAJUBÁ - UNIFEI
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Uso de aprendizado de máquina na borda para um sistema não invasivo de monitoramento de colmeias.

José Anderson dos Reis

Dissertação aprovada por banca examinadora em 12 de Dezembro de 2025, conferindo ao autor o título de **Mestre em Ciências em Engenharia Elétrica.**

Banca Examinadora:
Prof. Dr. Walter Augusto Varella
Prof. Msc. Marcelo José Rovai
Prof. Msc. José Alberto Ferreira Filho
Prof. Dr. Tales Cleber Pimenta
Prof. Dr. Gabriel Antônio Fanelli de Souza

Itajubá
17 de dezembro de 2025

José Anderson dos Reis

Uso de aprendizado de máquina na borda para um sistema não invasivo de monitoramento de colmeias/ José Anderson dos Reis. – Itajubá, 17 de dezembro de 2025-

82 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Gabriel Antonio Fanelli De Souza

Dissertação (Mestrado)
Universidade Federal de Itajubá - UNIFEI
Programa de pós-graduação em engenharia elétrica, 17 de dezembro de 2025.

1. Visão computacional. 2. Computação na borda. I. Fanelli de Souza, Gabriel. II. Universidade Federal de Itajubá. IV. TUso de aprendizado de máquina na borda para um sistema não invasivo de monitoramento de colmeias.

CDU 07:181:009.3

José Anderson dos Reis

Uso de aprendizado de máquina na borda para um sistema não invasivo de monitoramento de colmeias

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica como parte dos requisitos para obtenção do Título de Mestre em Ciências em Engenharia Elétrica.

Trabalho aprovado. Itajubá, 12 de Dezembro de 2025:

Prof. Dr. Gabriel Antonio Fanelli De Souza
Orientador

Prof. Msc. Marcelo José Rovai
Coorientador

Prof. Msc. José Alberto Ferreira Filho

Prof. Dr. Tales Cleber Pimenta

Prof. Dr. Walter Augusto Varella

Itajubá
17 de dezembro de 2025

Agradecimentos

Agradeço, de forma especial, aos meus pais, Amarildo e Suely, meus irmãos Cintia e Everton, e à minha esposa, Monalisa, pelo apoio incondicional e incentivo constante ao longo da realização deste trabalho.

Estendo meus agradecimentos aos amigos pelo companheirismo e pelas contribuições indiretas que enriqueceram esta trajetória. Sou grato também aos colegas de trabalho e pesquisa Luiz Gonzaga, Nedson, Helaine, Fabiano, Anne, Pollyana, Patrícia, Luiz Antônio e Eduardo, cuja colaboração foi de grande importância.

Manifesto minha sincera gratidão ao meu orientador, Professor Dr. Gabriel Fanelli, pela confiança depositada e pela oportunidade de desenvolver esta pesquisa. Agradeço, de modo especial, aos meus coorientadores, Professor Marcelo Rovai e Professor José Alberto, bem como ao aluno Guilherme Fernandes, que tiveram papel fundamental no desenvolvimento deste estudo, compartilhando generosamente seu tempo, experiência e conhecimento ao longo de toda a pesquisa.

"A sabedoria divina está em cada detalhe da criação; compreender a visão é apenas decifrar um reflexo dessa grandeza."
(Inspirado em Provérbios 3:19)

Resumo

As mudanças climáticas representam ameaças significativas às populações de abelhas, tornando essencial o monitoramento contínuo da atividade das colmeias para a gestão da apicultura e para pesquisas ecológicas. Este trabalho apresenta um sistema inovador e não invasivo de monitoramento de abelhas, que combina técnicas de aprendizado de máquina e de visão computacional para a contagem automatizada de abelhas na entrada das colmeias. O sistema utiliza o modelo de detecção de objetos *YOLOv8n*, implementado em uma plataforma embarcada *Raspberry Pi Zero 2W*, permitindo o processamento local, sem necessidade de transmissão para a nuvem. A avaliação de desempenho demonstrou uma Precisão Média (*mAP50*) de 0,987, com precisão de 0,91, revocação (*recall*) de 0,96 e *F1-score* de 0,94 nas tarefas de detecção de abelhas. A validação com padrão-ouro, por meio da análise de correlação de *Pearson*, evidenciou forte concordância (r = 0,93) entre as contagens automatizadas e manuais. A análise de *Bland-Altman* revelou um pequeno viés médio de 3,4 abelhas, com limites de concordância de 95% dentro de ±1,96 DP. A abordagem automatizada elimina a necessidade de observação manual, reduz custos de mão de obra e o distúrbio às colônias, ao mesmo tempo em que mantém alta precisão de medição, oferecendo vantagens significativas para a apicultura de precisão e para iniciativas de monitoramento ecológico.

Palavras-chaves: Abelha-melífera, Análise do comportamento de abelhas, Detecção de insetos, Aprendizado de máquina, Rede neural convolucional, Visão computacional, Análise de imagens, YOLOv8, Computação em borda, Monitoramento não invasivo.

Abstract

Climate change poses significant threats to bee populations, making continuous monitoring of hive activity essential for both apiculture management and ecological research. This work presents an innovative and non-invasive bee monitoring system that combines machine learning and computer vision techniques for automated bee counting at hive entrances. The system employs the YOLOv8n object detection model, implemented on an embedded Raspberry Pi Zero 2W platform, enabling local processing without the need for cloud transmission. The performance evaluation demonstrated a Mean Average Precision (mAP50) of 0.987, with precision of 0.91, recall of 0.96, and an F1-score of 0.94 in bee detection tasks. Validation against the gold standard, through Pearson correlation analysis, revealed a strong agreement ($r = 0.93$) between automated and manual counts. The Bland-Altman analysis showed a small mean bias of 3.4 bees, with 95% limits of agreement within ± 1.96 SD. The automated approach eliminates the need for manual observation, reduces labor costs and colony disturbance, while maintaining high measurement accuracy, offering significant advantages for precision apiculture and ecological monitoring initiatives.

Key-words: Honeybee, Bee behavior analysis, Insect detection, Machine learning, Convolutional neural network, Computer vision, Picture analysis, YOLOv8, Edge computing, Non-invasive monitoring.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AP	*Average Precision*	27
CNNs	*Convolutional Neural Network*	18
DFL	*Distributed Focal Loss*	23
FAO	*Food and Agriuculture organization*	14
FPN	*Feature Pyramid Network*	23
GFLOPs	*Giga Floating-point Operations Per second*	27
GPUs	Unidades de Processamento Gráfico	20
IA	Inteligência Artificial	18
IoU	*Intersection over Union*	28
mAP	*mean Average Precision*	27
ML	*Machine Learning*	18
PAN	*Path Aggregation Network*	23
SPPF	*Spatial Pyramid Pooling – Fast*	23
SSH	*Secure Shell*	42
TPUs	Unidades de Processamento Tensorial	20
YOLO	*You Only Look Once*	22

Sumário

1 Introdução

As abelhas do gênero *Apis mellifera* desempenham um papel essencial na polinização de culturas agrícolas e na manutenção de ecossistemas naturais, sendo reconhecidas como agentes fundamentais para a sustentabilidade ambiental e a economia mundial [1]. No Brasil, esse cenário é amplamente favorecido pela diversidade florística e pelas condições climáticas que permitem a produção contínua de mel durante todo o ano [2]. O desenvolvimento da apicultura nacional foi fortemente impulsionado após a introdução de abelhas africanas na década de 1950, originando as denominadas abelhas africanizadas, híbridos que reúnem alta produtividade e resistência a doenças e pragas [3], representada pela Figura 1.

Figura 1 – *Apis mellifera.*

Fonte: Abelhas.org

A apicultura brasileira destaca-se internacionalmente: em 2020, foram produzidas 51,5 mil toneladas de mel, colocando o país na décima posição no *ranking* mundial de produção. A atividade é liderada por pequenos produtores, concentrados principalmente nas regiões Sul e Nordeste, com forte vocação exportadora. Os Estados Unidos representam o principal mercado de destino, absorvendo 72% das exportações nacionais, que cresceram de US$ 55 milhões em 2010 para US$ 163 milhões em 2021. O mel brasileiro, valorizado pela sua pureza, possui certificações reconhecidas nos mercados europeu e norte-americano, além de potencial para expansão em nichos específicos, como o Halal. Cerca de 89% da produção nacional é destinada ao mercado externo, evidenciando a importância geoeconômica da atividade [2].

No contexto internacional, como ilustrado na Figura 2, a China lidera a produção global de mel (446,9 mil toneladas), seguida pela Turquia (114,1 mil toneladas). O Brasil figura com volume relevante, embora bem abaixo dos principais líderes de mercado, conforme dados da FAO (*Food and Agriuculture organization*) para 2018, quando a produção

mundial atingiu 1.851.546 toneladas [2].

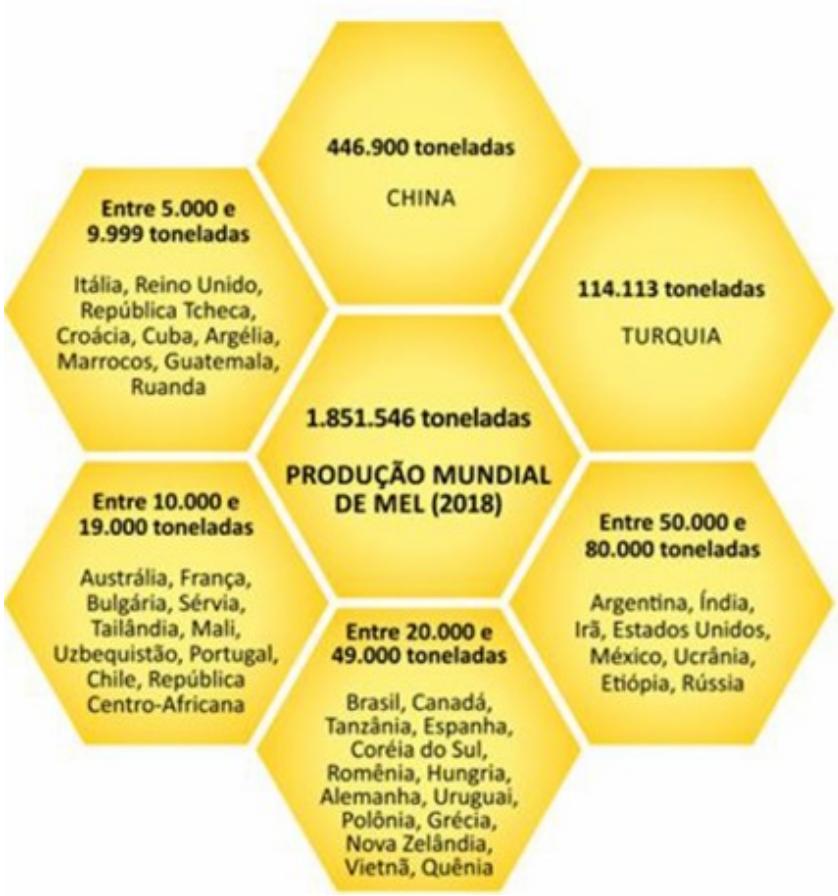

Figura 2 – Produção mundial de mel.

Fonte: FAO, 2020

Além dos aspectos econômicos, o serviço ambiental prestado pelas abelhas é fundamental: sua distribuição e características biológicas são moduladas diretamente por fatores climáticos, especialmente as oscilações sazonais de precipitação e temperatura [4]. Atualmente, profundas alterações nos padrões climáticos globais são observadas, com previsões de intensificação nas próximas décadas, afetando a biodiversidade vegetal e animal, incluindo as abelhas. Estudos recentes indicam que mudanças no clima podem comprometer áreas de forrageamento, dificultar a subsistência das colônias e afetar a produtividade da polinização [5, 6, 7].

A modernização da apicultura foi marcada pela introdução da colmeia Langstroth [3], projetada com base no conceito de espaço-abelha e caracterizada por sua modularidade e praticidade de manejo, mostrado na Figura 3. O alvado, localizado na base frontal da estrutura, é um ponto estratégico para estudos comportamentais e monitoramento automatizado, pois organiza o fluxo de entrada e saída das abelhas.

Atualmente, o monitoramento de fluxo de abelhas na entrada da colmeia é reconhecido como ferramenta essencial para a apicultura de precisão, permitindo avaliações

Figura 3 – Colmeia Langstroth.

Fonte: <https://www.epagri.sc.gov.br/index.php>

não invasivas da saúde e comportamento das colônias e o diagnóstico precoce de distúrbios ambientais ou invasões de predadores [2, 3]. Entretanto, a detecção automática de insetos nessa região enfrenta desafios como variações de iluminação, sobreposições e oclusões frequentes [1]. Tais questões demandam o desenvolvimento de soluções tecnológicas robustas e precisas para geração de dados confiáveis em condições reais de campo.

1.1 Objetivo

Esta dissertação tem o objetivo de descrever um sistema inovador de monitoramento de abelhas que combina técnicas de aprendizado de máquina e de visão computacional para estimar o número de abelhas na entrada das colmeias. Ao realizar todo o processamento localmente, sem necessidade de transmissão em nuvem, o sistema alcança reduções significativas na latência de comunicação, no consumo de banda e nos custos operacionais.

1.2 Organização do trabalho

Além da introdução, o presente trabalho está organizado em cinco capítulos, incluindo a conclusão, visando apresentar de forma estruturada os fundamentos teóricos, a metodologia aplicada e os resultados obtidos. O Capítulo 2 estabelece a base conceitual da pesquisa, abordando os princípios de redes neurais, com ênfase em suas aplicações na visão computacional e na detecção de objetos. O Capítulo 3 descreve a implementação do sistema proposto, detalhando a metodologia de aquisição e pré-processamento de imagens, a construção do banco de dados, o treinamento do modelo de detecção, a descri-

ção e a configuração do hardware embarcado e as estratégias de otimização empregadas para maximizar a precisão e a eficiência computacional. No Capítulo 4, são apresentados os resultados obtidos, incluindo a avaliação do método para monitoramento em tempo real das abelhas, a análise comparativa do desempenho do modelo frente ao método de padrão-ouro desenvolvido e uma análise energética do sistema. Finalmente, o Capítulo 5 sintetiza as contribuições do estudo, evidencia suas limitações e propõe direções futuras, ressaltando o potencial de integração da abordagem desenvolvida em aplicações de monitoramento ambiental de polinizadores.

2 Revisão teórica

2.1 Revisão da Literatura

Esta seção tem como objetivo apresentar os principais fundamentos teóricos e tecnológicos que embasam o desenvolvimento deste trabalho. A revisão da literatura contempla os conceitos essenciais relacionados ao aprendizado de máquina, à visão computacional e aos modelos de detecção de objetos, com destaque para CNNs (*Convolutional Neural Network*) e o algoritmo YOLOv8. Esses tópicos são fundamentais para compreender as bases metodológicas que sustentam o sistema proposto de monitoramento não invasivo de colmeias. Ao longo deste capítulo, são discutidos os avanços científicos que permitiram o aprimoramento de modelos de aprendizado profundo, as aplicações práticas em sistemas embarcados e o papel da computação em borda na execução eficiente de inferências em tempo real. Dessa forma, esta revisão busca contextualizar o estado da arte e justificar as escolhas tecnológicas adotadas no desenvolvimento do presente estudo.

2.2 Redes Neurais Convolucionais

O conceito de IA (Inteligência Artificial) apresenta múltiplas definições e, com frequência, é alvo de interpretações equivocadas. Diante desse cenário, o presente trabalho direciona sua ênfase especificamente ao campo do aprendizado de máquina ML (*Machine Learning*). O aprendizado de máquina pode ser caracterizado como um ramo da IA dedicado ao desenvolvimento de métodos e algoritmos capazes de reproduzir, em alguma medida, a capacidade humana de aprender de forma autônoma a partir de dados, identificando padrões, estruturas e agrupamentos subjacentes.

O aprendizado de máquina constitui um subcampo fundamental da IA, englobando um conjunto abrangente de paradigmas, modelos e algoritmos concebidos para aproximar e generalizar processos cognitivos humanos por meio de métodos computacionais. Essas abordagens baseiam-se em princípios de inferência estatística, técnicas de otimização e modelagem orientada a dados, permitindo que sistemas computacionais identifiquem padrões, adaptem-se a novas informações e aprimorem seu desempenho ao longo do tempo, sem a necessidade de programação explícita baseada em regras rígidas. As aplicações de ML estendem-se a diversos domínios científicos e tecnológicos: na ciência da computação sustenta avanços em visão computacional, reconhecimento de padrões e tarefas de classificação; na física contribui para a modelagem e simulação de sistemas complexos; na farmacologia e na biologia viabiliza a descoberta de fármacos, a caracterização molecular e a modelagem preditiva de interações bioquímicas. Ao oferecer soluções escaláveis para

conjuntos de dados de alta dimensionalidade e heterogeneidade, o aprendizado de máquina consolida-se como uma metodologia central para a pesquisa contemporânea e para a inovação em disciplinas intensivas em dados [8].

Redes neurais, como representadas na Figura 4, são modelos computacionais inspirados na estrutura do cérebro humano, compostos por unidades de processamento simples chamadas neurônios, que se conectam por meio de sinapses artificiais [9]. Essas redes podem ser organizadas em diferentes arquiteturas, como redes de camada única, múltiplas camadas e redes recorrentes, sendo capazes de aprender e generalizar padrões a partir de exemplos, o que as torna especialmente úteis para tarefas complexas e não-lineares. O processo de aprendizagem pode ocorrer de forma supervisionada, quando há um agente externo que apresenta os padrões de entrada e saída, ou não supervisionada, quando a rede busca identificar padrões por conta própria [10].

A principal vantagem das redes neurais artificiais está em sua adaptabilidade e capacidade de reconhecer padrões complexos, o que permite aplicações em áreas como reconhecimento de voz, diagnóstico médico, previsão financeira e controle de processos industriais. Apesar de seu potencial, a implementação prática dessas redes pode apresentar desafios, exigindo pesquisas constantes para aprimorar suas arquiteturas e algoritmos de treinamento, visando maior eficiência e precisão nos resultados.

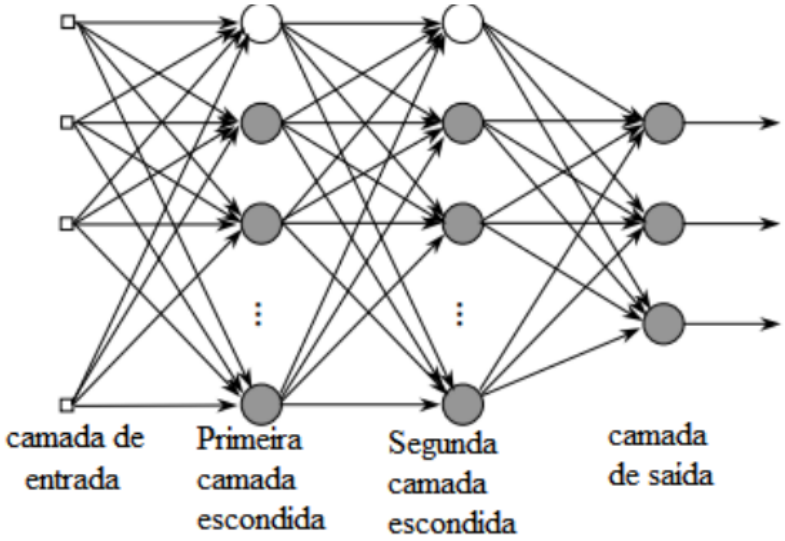

Figura 4 – Exemplo de rede neural.
Fonte: Fleck et al., 2016.

As CNNs constituem uma classe especializada de arquiteturas de aprendizagem profunda *(Deep Learning)*, projetadas para a extração automática e hierárquica de características relevantes a partir de dados de entrada estruturados, notadamente imagens. Sua estrutura é composta por múltiplas camadas convolucionais, responsáveis pela detecção de padrões locais como bordas, texturas e formas; camadas de *pooling*, que realizam a redução dimensional e promovem invariância a translações; e camadas totalmente conec-

tadas, que integram as representações extraídas em níveis mais abstratos para tarefas de classificação ou regressão. A utilização de técnicas complementares, como normalização por lotes *(Batch Normalization)*, funções de ativação não lineares *(ReLU, Leaky ReLU, entre outras)* e mecanismos de regularização *(Dropout, L2 Regularization)*, contribui para a estabilidade do treinamento e para mitigação de problemas como o *overfitting* e o desaparecimento de gradientes. Além disso, a implementação eficiente (principalmente no treinamento) de CNNs depende fortemente de recursos computacionais de alto desempenho, como as GPUs (Unidades de Processamento Gráfico) e as TPUs (Unidades de Processamento Tensorial), que possibilitam a execução paralela de um grande número de operações matriciais. Entretanto, em cenários de inferência em borda, tais unidades não são utilizadas, uma vez que o processamento é realizado em dispositivos embarcados com capacidade computacional, energética e de memória significativamente restrita.

Nesse contexto, torna-se essencial o emprego de técnicas de otimização de modelos, como quantização, poda de parâmetros e arquiteturas leves, a fim de viabilizar a execução de inferências em tempo real, sem dependência de hardware especializado, preservando, na medida do possível, a acurácia e a eficiência do sistema [11, 12]. Dessa forma, as CNNs têm impulsionado avanços significativos em domínios como a visão computacional, o reconhecimento de fala, a análise de séries temporais e o processamento de linguagem natural, consolidando-se como um dos pilares metodológicos mais robustos e amplamente explorados no desenvolvimento de sistemas inteligentes contemporâneos [13].

As CNNs empregam três componentes fundamentais para a extração hierárquica de características, conforme ilustrado na Figura 5, que apresenta uma arquitetura comumente utilizada nesse tipo de rede. O primeiro componente é a camada de convolução, responsável pela detecção inicial de padrões locais na imagem. Nessa etapa, filtros convolucionais de dimensões reduzidas percorrem todo o conjunto de dados de entrada, aplicando a operação de convolução para extrair representações discriminativas. Em seguida, a camada de *pooling* é aplicada, desempenhando o papel de reduzir a dimensionalidade dos mapas de ativação gerados, preservando as características mais relevantes e eliminando redundâncias. Esse processo de redução permite que as camadas subsequentes recebam representações mais compactas e invariantes, facilitando a extração de padrões de ordem superior nas etapas convolucionais seguintes. O propósito fundamental da camada de *pooling* é reduzir a complexidade computacional associada ao processamento de dados, ao mesmo tempo em que preserva as informações mais relevantes da imagem, assegurando que as classes sejam reconhecidas de forma robusta, independentemente de variações de escala, posição ou distribuição espacial dos objetos [14]. Na sequência, a camada totalmente conectada *(fully connected layer)*, situada na etapa final da rede neural, é responsável por integrar e interpretar os atributos extraídos pelas camadas convolucionais e de *pooling*. Nesse processo, os mapas de características são achatados e convertidos em um vetor unidimensional, o que é propagado por uma ou mais camadas densas, culminando em uma

camada *Softmax*. Esta última gera uma distribuição de probabilidades sobre as classes possíveis, permitindo a tomada de decisão no processo de classificação.

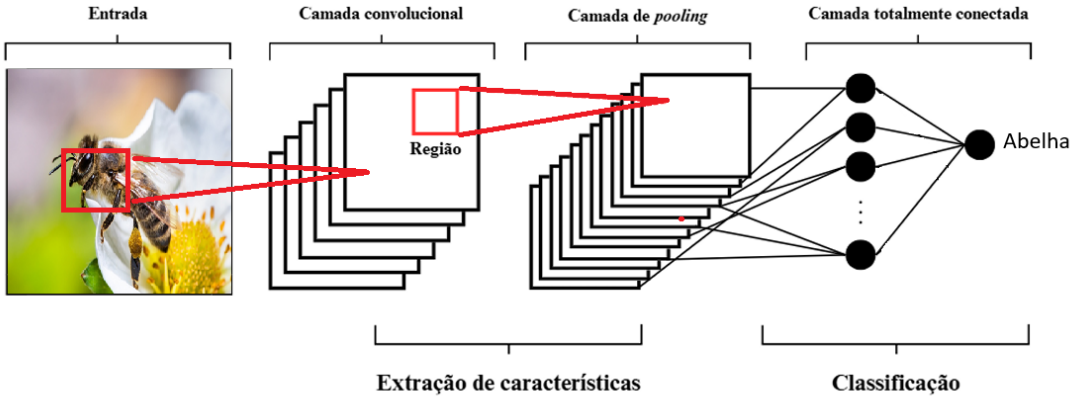

Figura 5 – Arquitetura básica de uma CNN.
Fonte: Do autor com base em Giorgio Venturieri (2024).

Vale mencionar que a CNNs são a base para praticamente todos os avanços modernos em reconhecimento de imagens e visão computacional.

2.3 Visão Computacional

A visão computacional é um campo interdisciplinar relativamente recente, situado na interseção entre a inteligência artificial, a matemática aplicada e a engenharia da computação. O termo foi mencionado pela primeira vez em 1955 por Selfridge, que o descreveu como "os olhos e ouvidos do computador", destacando a perspectiva de dotar as máquinas da capacidade de interpretar estímulos visuais. Originalmente, na década de 1960, as primeiras investigações nessa área estavam fortemente ligadas a modelos simbólicos e heurísticos, que buscavam traduzir imagens em representações geométricas simples, limitadas a tarefas como a detecção de bordas, o reconhecimento de formas básicas e a análise de movimento. Nas décadas seguintes, especialmente entre os anos 1970 e 1990, o avanço de técnicas estatísticas, de processamento digital de imagens e de métodos baseados em transformadas matemáticas (como a transformada de Fourier e a transformada de Hough) consolidou o campo como uma vertente fundamental da inteligência artificial aplicada.

Com a ampliação da capacidade computacional e o crescimento da disponibilidade de dados digitais no início do século XXI, a visão computacional passou por uma transição: deixou de depender majoritariamente de algoritmos projetados manualmente para adotar modelos de aprendizado de máquina e, mais recentemente, de aprendizado profundo *(deep learning)*. CNNs, em particular, representaram um marco nesse processo, ao oferecer uma forma escalável e altamente eficaz de extrair e hierarquizar características

visuais de maneira automática. A Figura 6 representa a diferença da visão do ser humano e a visão computacional. Atualmente, a visão computacional constitui uma área consolidada e em expansão, responsável por possibilitar a interpretação autônoma de dados visuais provenientes de múltiplas fontes — câmeras de vídeo, sensores ópticos, scanners e outros dispositivos [15], sendo aplicada em uma ampla gama de domínios, como sistemas autônomos, diagnóstico médico assistido por imagens, monitoramento ambiental, segurança pública e agricultura de precisão. Essas informações permitem a identificação, análise e extração de atributos relevantes dos objetos presentes em uma imagem. Sob essa perspectiva, a visão computacional pode ser conceituada como um sistema no qual a entrada consiste em dados visuais (imagens ou sequências de imagens) e a saída corresponde à interpretação total ou parcial desses dados, permitindo a compreensão estruturada do conteúdo visual e a tomada de decisões subsequentes baseada nas características extraídas [16]. Algoritmos de visão computacional vêm sendo aplicados em uma ampla gama de funcionalidades, incluindo supervisão de hábitos alimentares em grupos de animais [17], bem como outras tarefas de detecção e sinalização de defeitos em pavimentos rodoviários [18], possibilitando a extração de informações relevantes para sistemas automatizados e tomada de decisão em tempo real.

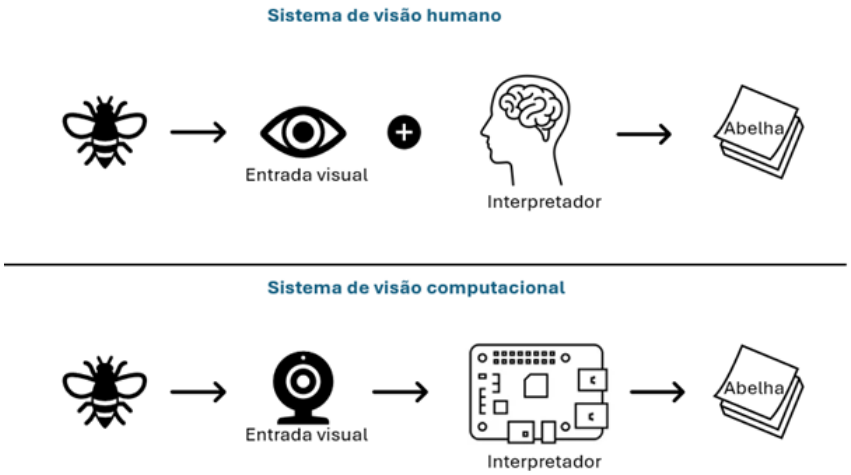

Figura 6 – Visão do ser humano x visão computacional.
Fonte: Do autor com base em Freitas et al.

2.4 YOLO (You Only Look Once)

O presente estudo empregou o algoritmo YOLO (*You Only Look Once*) [19], um algoritmo de detecção de objetos que utiliza uma rede CNNs como núcleo para identificar e localizar objetos em imagens em apenas uma passada, de modo muito eficiente e rápido. O YOLO é classificado como um *framework* de detecção em tempo real, pois divide a imagem em uma grade e prediz, de forma simultânea, *bounding boxes* e probabilidades de

classe para cada célula.

A arquitetura do YOLOv8 (versão do modelo empregada neste trabalho) apresentada na Figura 7 é composta por três módulos principais: *backbone*, *neck* e *head*. Cada um desses componentes desempenha um papel específico e complementar no processo de extração, fusão e predição de características, resultando em um modelo eficiente para tarefas de detecção de objetos em múltiplas escalas.

Backbone

O módulo *backbone* é responsável por extrair características a partir das imagens de entrada. Ele é composto pelos módulos convolucional C2f e SPPF (*Spatial Pyramid Pooling – Fast*).

O componente convolucional emprega núcleos convolucionais para extrair características, minimizando a perda de informações associadas ao fluxo de gradientes.

O módulo C2f no YOLOv8 representa um aprimoramento do conceito ELAN introduzido no YOLOv7, por meio da integração da ideia de fluxo de gradiente. Esse avanço resulta da combinação de princípios do YOLOv5 e do YOLOv7, culminando em um módulo C2f mais eficiente [20]. Essa estrutura permite obter uma maior quantidade de informações sobre o fluxo de gradiente e, consequentemente, melhorar a capacidade de extração de características.

O módulo SPPF, por sua vez, realiza o *spatial pyramid pooling*, convertendo mapas de características de diferentes dimensões em um vetor de tamanho fixo.

Neck

A seção *neck* do modelo é projetada com base na FPN (*Feature Pyramid Network*) e na PAN (*Path Aggregation Network*). Essa configuração integra de forma eficiente os fluxos de informação *top-down* e *bottom-up* na rede, o que permite expandir o campo receptivo e incorporar informações em múltiplas escalas aos mapas de características. Como resultado, o desempenho global da detecção é significativamente aprimorado.

Head

O módulo head utiliza mapas de características em diferentes escalas para extrair simultaneamente informações de categoria e de localização de objetos. O conceito de DFL (*Distributed Focal Loss*) é implementado para reduzir tanto o número de parâmetros quanto a complexidade computacional. Diferentemente da abordagem tradicional baseada em *anchors (anchor-based)*, o YOLOv8 adota a abordagem *anchor-free*, que caracteriza os objetos com base em informações de contorno derivadas de múltiplos pontos-chave, pontos médios e limites. Essa estratégia aumenta a precisão da localização e apresenta desempenho superior em cenários de detecção densa.

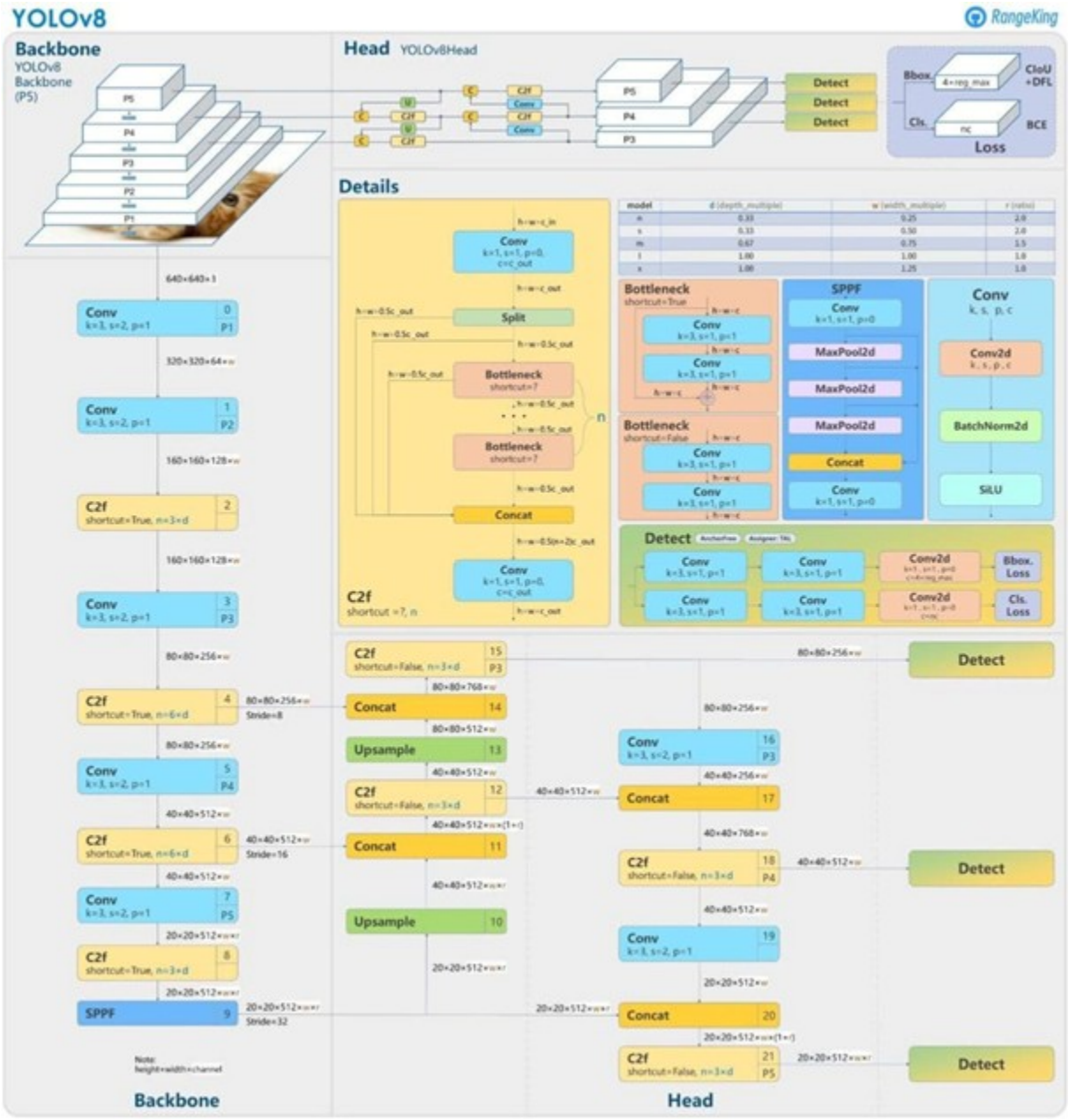

Figura 7 – Arquitetura modelo YOLOv8.
Fonte: GitHub disponível em *RangeKing*.

O algoritmo YOLO modela detecção como um problema de regressão, em que a imagem de entrada é dividida em uma grade de dimensões S×S. Cada célula dessa grade é responsável por prever múltiplas caixas delimitadoras *(bounding boxes)*, associadas às respectivas probabilidades de conter objetos [19]. Para cada caixa delimitadora gerada, calcula-se a probabilidade de existência de um objeto. Quando esse valor ultrapassa um limiar pré-definido, a caixa é considerada uma candidata válida para representar a localização do objeto na imagem de entrada. Esse processo é ilustrado na Figura 8, que exemplifica a estratégia de predição adotada pelo YOLO. O modelo utilizado é composto por uma rede neural com 24 camadas convolucionais, seguidas por 2 camadas totalmente conectadas. Para otimizar o uso do espaço e reduzir a complexidade computacional nas ca-

madas intermediárias, são aplicadas camadas de convolução 1×1 em forma intercalada. As camadas convolucionais da rede YOLO foram previamente treinadas utilizando uma base de dados de classificação de imagens do ImageNet, na qual as imagens de entrada foram processadas com metade da resolução original [19]. Nesse arranjo, as camadas convolucionais iniciais são responsáveis pela extração de características relevantes das imagens, enquanto as camadas totalmente conectadas têm como função prever as probabilidades de classe e calcular as coordenadas das caixas delimitadoras na saída da rede.

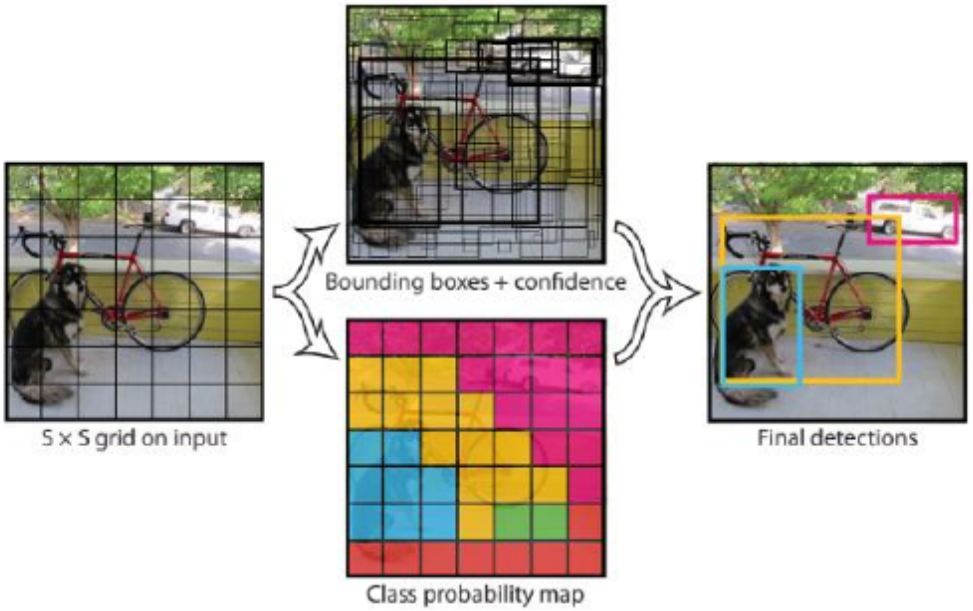

Figura 8 – Estratégia de predição adotada pelo algoritmo YOLO.

Fonte: Adaptado de Redmon et al. (2016).

A seguir, apresenta-se a lista das principais variantes dos modelos pré-treinados de detecção de objetos disponibilizados pelo *Ultralytics* YOLOv8, acompanhada de suas características essenciais:

YOLOv8n (*nano*): Trata-se do modelo mais leve e mais rápido da família YOLOv8. Possui um número reduzido de parâmetros, sendo otimizado para dispositivos com recursos computacionais limitados e aplicações em tempo real, nas quais a velocidade é prioritária, ainda que isso possa implicar em uma leve redução na precisão.

YOLOv8s (*small*): Versão ligeiramente maior em relação ao YOLOv8n, proporciona um equilíbrio entre velocidade e precisão. É adequado para cenários que demandam bom desempenho em dispositivos de capacidade moderada.

YOLOv8m (*medium*): Modelo intermediário que oferece maior precisão em comparação com as variantes menores, mantendo ainda uma boa velocidade de inferência. É indicado para aplicações que necessitam de um equilíbrio consistente entre acurácia e eficiência.

YOLOv8l (*large*): Representa a variante de maior porte dentre as apresentadas, com número significativamente superior de parâmetros. Proporciona a melhor precisão da família YOLOv8, porém com maior custo computacional e menor velocidade de inferência, sendo apropriado para aplicações em que a acurácia se sobrepõe à velocidade de processamento.

YOLOv8x (*extra large*): Esta é a variante mais robusta da família YOLOv8, caracterizada pelo maior número de parâmetros. O YOLOv8x proporciona a mais alta precisão entre todos os modelos disponíveis, porém com aumento significativo na latência. Dessa forma, é mais adequado para aplicações em que a máxima acurácia na detecção é prioridade, ainda que em detrimento da velocidade de processamento.

A Tabela 1 apresenta o desempenho das diferentes variantes dos modelos de detecção de objetos da família YOLOv8. Todos os modelos foram previamente treinados no conjunto de dados *COCO* [21], que contempla 91 classes predefinidas.

Tabela 1 – Desempenho das variantes dos modelos YOLOv8.

Modelo	**Tamanho (pixels)**	**mAPval 50–95**	**Velocidade CPU ONNX (ms)**	**Velocidade A100 TensorRT (ms)**	**Parâmetros (M)**	**FLOPs (B)**
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Fonte: Ultralytics (2025).

Para o desenvolvimento do modelo de detecção de objetos, foi adotado o modelo YOLOv8n, desenvolvido pela Ultralytics [22] e lançada em janeiro de 2023. Entre as variantes disponíveis, optou-se especificamente pela versão YOLOv8n (nano), em razão das limitações do *hardware* utilizado e da necessidade de manter baixo consumo de energia. Essa versão apresenta uma arquitetura otimizada, com menor número de parâmetros, o que reduz significativamente a demanda computacional sem comprometer de forma crítica a acurácia da detecção. Dessa forma, o YOLOv8n mostra-se particularmente adequado para aplicações em dispositivos com recursos limitados, conciliando eficiência energética e desempenho satisfatório. A Figura 9 ilustra a relação entre a complexidade do modelo — mensurada pelo número de parâmetros — e a precisão da detecção, expressa pelo indicador *COCO mAP50-95*. O gráfico à direita apresenta o *trade-off* entre a velocidade de inferência — medida pela latência na GPU NVIDIA A100 em FP16 do TensorRT — e a precisão obtida pelos mesmos modelos.

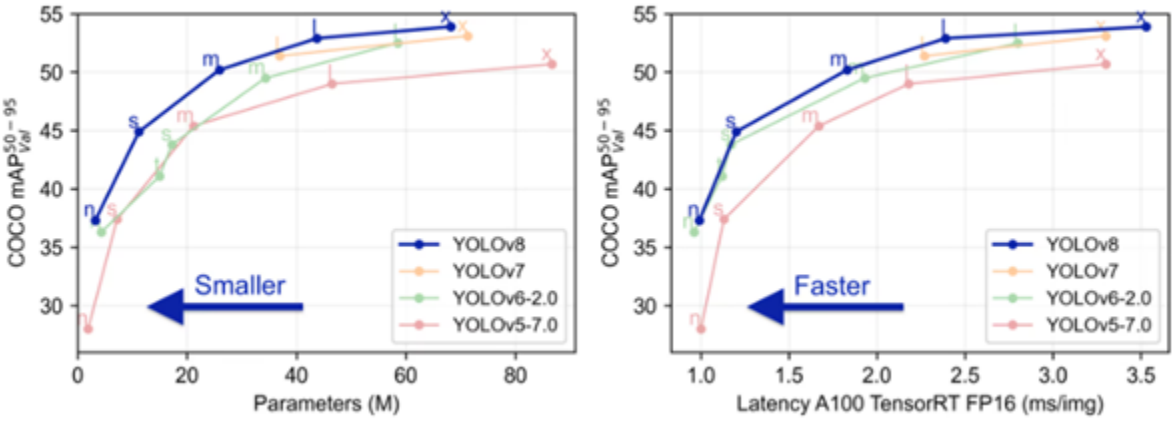

Figura 9 – Desempenho das variantes do modelo YOLOv8: relação entre número de parâmetros, precisão e velocidade de inferência.

Fonte: Adaptado de Ultralytics (2025).

2.5 Métricas de Avaliação

O processo de detecção de abelhas requer a consideração tanto da acurácia quanto da velocidade de detecção [23]. Portanto, este estudo adota as métricas *precision*, *recall*, *F1-Score*, AP (*Average Precision*) e mAP (*mean Average Precision*) para caracterizar o desempenho dos modelos. Adicionalmente, a velocidade de execução dos modelos é avaliada por meio de métricas como GFLOPs (*Giga Floating-point Operations Per second*). A *precision* corresponde à proporção de casos classificados como positivos que, de fato, são positivos dentro de toda a amostra. A acurácia é calculada de acordo com a equação 2.1:

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

O *recall* representa a proporção de casos realmente positivos em relação aos casos previstos como positivos. O *recall* é calculado conforme a Equação 2.2:

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

O *F1-Score* é uma métrica de avaliação amplamente utilizada em tarefas de classificação e detecção de objetos, pois combina de forma harmônica as medidas de precisão (*precision*) e revocação (*recall*) em um único valor representativo do desempenho do modelo. Essa métrica é especialmente útil quando há desequilíbrio entre classes ou quando é necessário avaliar simultaneamente a capacidade do modelo de identificar corretamente os elementos positivos e de evitar falsos positivos. O *F1-Score* é calculado de acordo com a equação 2.3:

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.3}$$

onde:

- TP representa o número de previsões corretas como amostras positivas;
- FP representa o número de previsões incorretas como amostras positivas;
- FN representa o número de previsões incorretas como amostras negativas.

O AP representa a área sob a curva de *precision–recall*, delimitada pela curva e pelos eixos coordenados. Ele é calculado de acordo com a Expressão 2.4:

$$AP = \int_0^1 (\text{Precision} \times \text{Recall})\, dx \tag{2.4}$$

O mAP representa o valor médio do AP e é definido pela Equação 2.5:

$$mAP = \frac{\sum_{i=1}^{n} AP_i}{n} \tag{2.5}$$

O *mAP 0.5:0.95* é a média do mAP calculada com base em dez diferentes limiares de IoU (*Intersection over Union*). Esses limiares variam de 0,5 a 0,95, com um incremento de 0,05.

Além das métricas utilizadas no YOLOv8, também foram utilizados o coeficiente de correlação de *Pearson* (2.6), a equação de regressão linear (2.7) e o método de *Bland-Altman* (2.8).

$$r = \frac{n\sum xy - \sum x \sum y}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} \tag{2.6}$$

$$b = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} \tag{2.7}$$

$$\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i \tag{2.8}$$

Os sistemas de detecção de objetos realizam previsões por meio da geração de caixas delimitadoras (*bounding boxes*) associadas a rótulos de classe, conforme ilustrado na Figura 10. As caixas delimitadoras representam a localização espacial dos objetos dentro da imagem, enquanto os rótulos indicam a categoria à qual cada objeto pertence.

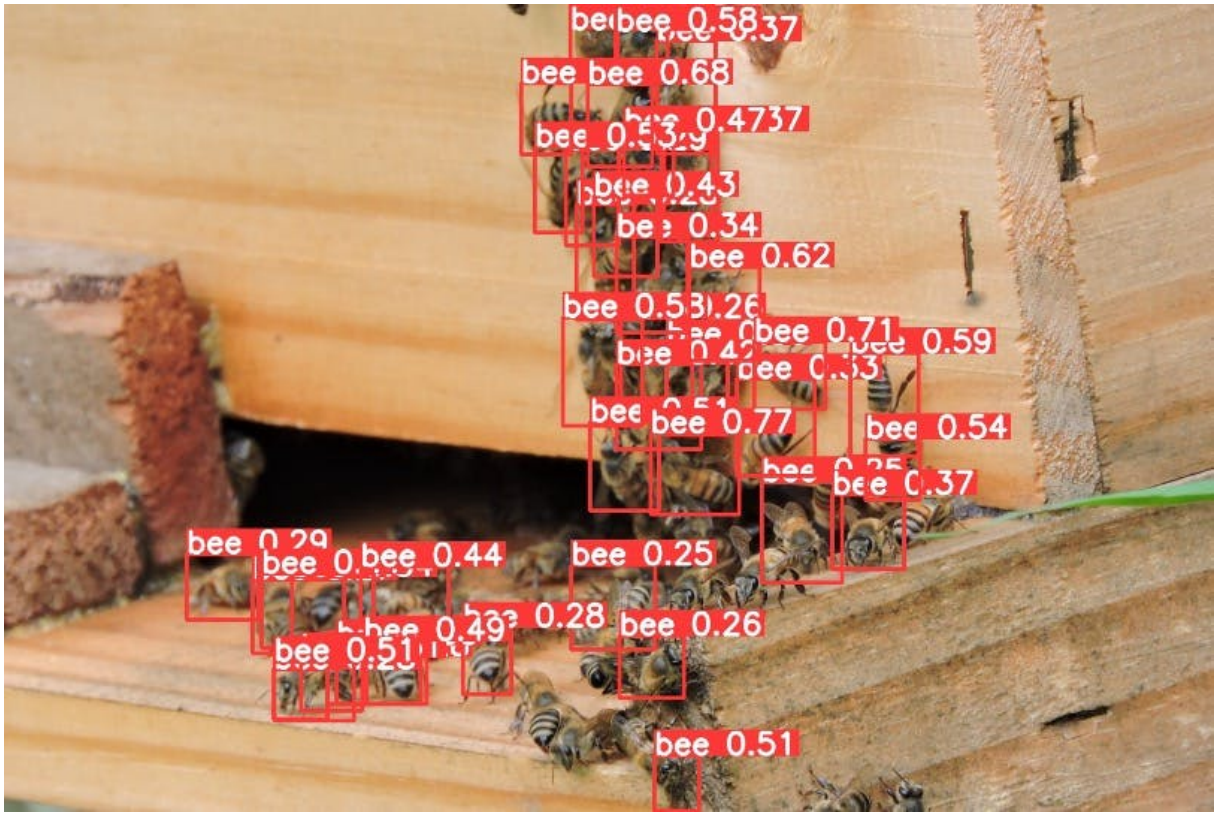

Figura 10 – Exemplo de detecção de abelhas com caixas delimitadoras (*bounding boxes*).

Fonte: Livro professor Marcelo Rovai disponível em *EdgeML Made Easy*.

Na tarefa de detecção de objetos, o desempenho da detecção depende do projeto da função de perda [24]. A função de perda IoU é utilizada para medir a sobreposição entre a caixa prevista pela detecção e a caixa de referência (*ground-truth*). Seu valor varia entre 0 e 1; valores mais próximos de 1 indicam maior sobreposição entre a caixa prevista e a caixa de referência, refletindo uma detecção mais precisa, enquanto valores próximos de 0 indicam menor sobreposição e, consequentemente, maior discrepância entre as caixas.

A função clássica de perda é demonstrada na Figura 11 e é calculada pelas Equações 2.9 e 2.10:

$$IoU = \frac{|A \cap B|}{|A \cup B|} \tag{2.9}$$

$$L_{IoU} = 1 - IoU \tag{2.10}$$

onde:

- O quadrado vermelho representa a área da caixa prevista (*predicted box*);
- O quadrado azul representa a área da caixa de referência (*ground truth*).

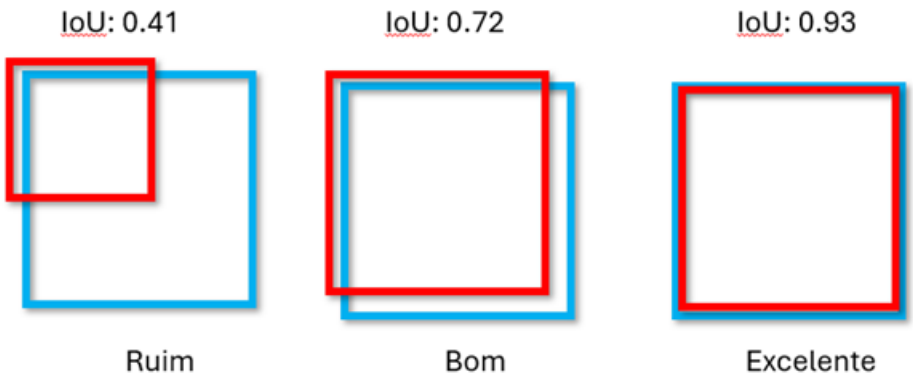

Figura 11 – Ilustração da métrica *Intersection over Union (IoU)* para detecção de objetos.

Fonte: Do autor.

3 Materiais e Métodos

Neste capítulo, apresenta-se detalhadamente a metodologia adotada para o desenvolvimento da pesquisa, conforme ilustrado no fluxograma da Figura 12. Inicialmente, descreve-se o protocolo de coleta de imagens de abelhas nas entradas das colmeias, especificando tanto os equipamentos empregados quanto o contexto experimental em que as gravações foram realizadas. Na sequência, aborda-se o procedimento de elaboração do *Dataset*, incluindo a definição criteriosa da classe de interesse (abelha) a ser considerada no modelo de rede neural.

Posteriormente, discorre-se sobre o treinamento do modelo, utilizando a arquitetura YOLOv8, descrevendo as etapas de parametrização, o ajuste dos hiperparâmetros e os critérios para a seleção dos melhores resultados. Na penúltima etapa, descreve-se a implementação do sistema, detalhando as etapas envolvidas no processo de desenvolvimento e integração dos componentes de *hardware* e *software*. Por fim, apresenta-se a descrição do código-fonte utilizado, acompanhada de explicações sobre sua estrutura e principais funções, de modo a garantir a reprodutibilidade dos resultados.

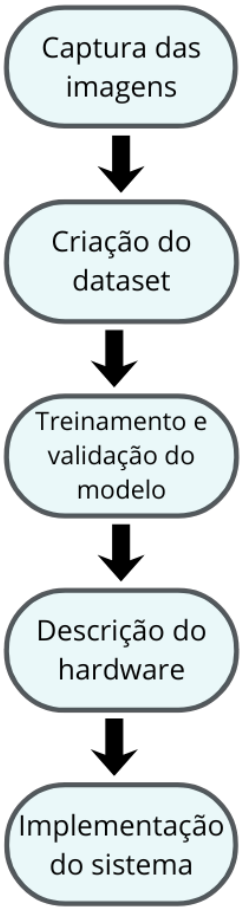

Figura 12 – Fluxograma da metodologia.

Fonte: Do autor.

3.1 Captura das imagens

A captura de imagens da entrada da colmeia foi realizada utilizando o kit *Seeed Studio XIAO ESP32S3 Sense* representado na Figura 13, que integra o processador *dual-core Xtensa*, com frequência de operação de até 240 MHz, acoplado ao sensor de câmera *OV2640*. O referido sensor apresenta um campo de visão horizontal de 53,5°, o que proporciona cobertura espacial adequada para o enquadramento da área de interesse.

Considerando a geometria de uma colmeia padrão com 46 cm de altura, a câmera foi estrategicamente posicionada na parte superior, assegurando que toda a extensão da entrada, com aproximadamente 47 cm de largura, fosse integralmente capturada no campo de visão. Esse posicionamento eliminou a necessidade de recursos ópticos adicionais, como lentes grande-angulares, garantindo registros consistentes e representativos da atividade externa da colmeia.

As imagens foram adquiridas em resolução 1080p (1920 × 1080 *pixels*), com taxa de captura configurada em 1 quadro por segundo, otimizando o equilíbrio entre qualidade e volume de dados gerados. A aquisição ocorreu em intervalos regulares ao longo do dia, sob

condições ambientais naturais, com variações de luminosidade e temperatura típicas do local, visando garantir a representatividade do conjunto de dados em diferentes situações reais de monitoramento.

A iluminação natural foi o principal fator a influenciar a qualidade das imagens, observando-se variações conforme o horário e incidência solar, proporcionando um *dataset* com diversidade suficiente para o desenvolvimento robusto de algoritmos de análise visual [25, 26, 27].

No total, foram capturadas 1.991 imagens, constituindo um banco inicial destinado à criação de um *dataset* estruturado. Esse conjunto de dados será utilizado em etapas posteriores de treinamento e validação de modelos de visão computacional voltados ao monitoramento de comportamentos de abelhas e ao fluxo de entrada e saída da colmeia.

Figura 13 – Seeed Studio XIAO ESP32S3 Sense.

Fonte: <https://www.robocore.net/seeed-studio/seeed-studio-xiao-esp32s3-sense>.

A Figura 14 apresenta uma amostra ilustrativa dos resultados obtidos a partir das capturas de imagens realizadas para a criação do *dataset*. Essa figura evidencia a qualidade das imagens adquiridas, destacando o enquadramento da entrada da colmeia e a nitidez dos detalhes capturados pelo sensor OV2640. Além disso, a ilustração permite visualizar a uniformidade no campo de visão e a adequação da resolução escolhida para os propósitos da análise, evidenciando a consistência dos dados que servirão de base para as etapas subsequentes de processamento e desenvolvimento do projeto.

Figura 14 – Exemplo do resultado das capturas.

Fonte: Do autor.

3.2 Criação do *dataset*

Após a captura, as imagens foram importadas para a plataforma *Roboflow* [28], que oferece um ambiente completo para criação, anotação e gerenciamento de *datasets*. Conforme detalhado no tutorial oficial do *Roboflow*, a plataforma permite criar projetos estruturados, adicionar e anotar imagens com ferramentas intuitivas como *bounding boxes* e polígonos, aplicar pré-processamento e aumentação de dados, e exportar os *datasets* em diversos formatos compatíveis com *frameworks* populares como o YOLO (*Roboflow Docs*). A organização em versões e classes facilita o controle da evolução do *dataset* e adaptações conforme as necessidades do projeto, o que foi evidenciado no trabalho do *CEFET-MG* [29] focado em detecção de objetos em tempo real, que destaca a eficiência da plataforma no ciclo completo de anotação e treinamento. Além disso, o *Roboflow* integra-se diretamente a soluções avançadas de visão computacional como o *Ultralytics YOLO*, possibilitando a transferência rápida dos dados anotados para treinamento com suporte a APIs e SDKs para implantação do modelo treinado. A documentação da *Ultralytics* exemplifica como carregar, rotular, pré-processar e aumentar os dados no *Roboflow* para um treinamento personalizado e posterior *upload* dos pesos do modelo para disponibilização via *API*, tornando o *Roboflow* uma ferramenta estratégica para acelerar o desenvolvimento de soluções de visão computacional. Casos práticos, como os apresentados no *blog Sigmoidal*, também evidenciam o uso da *API* do *Roboflow* para a autenticação, a manipulação e o *download* de *datasets* personalizados para diferentes cenários, reforçando sua flexibilidade e destaque

no ecossistema de visão computacional.

Primeiramente, optou-se por desenvolver um modelo inicial de detecção de abelhas, denominado "*Bees*", composto por 81 imagens em que cada abelha foi anotada individualmente. A Figura 15 ilustra a estratégia utilizada para a criação do *dataset*.

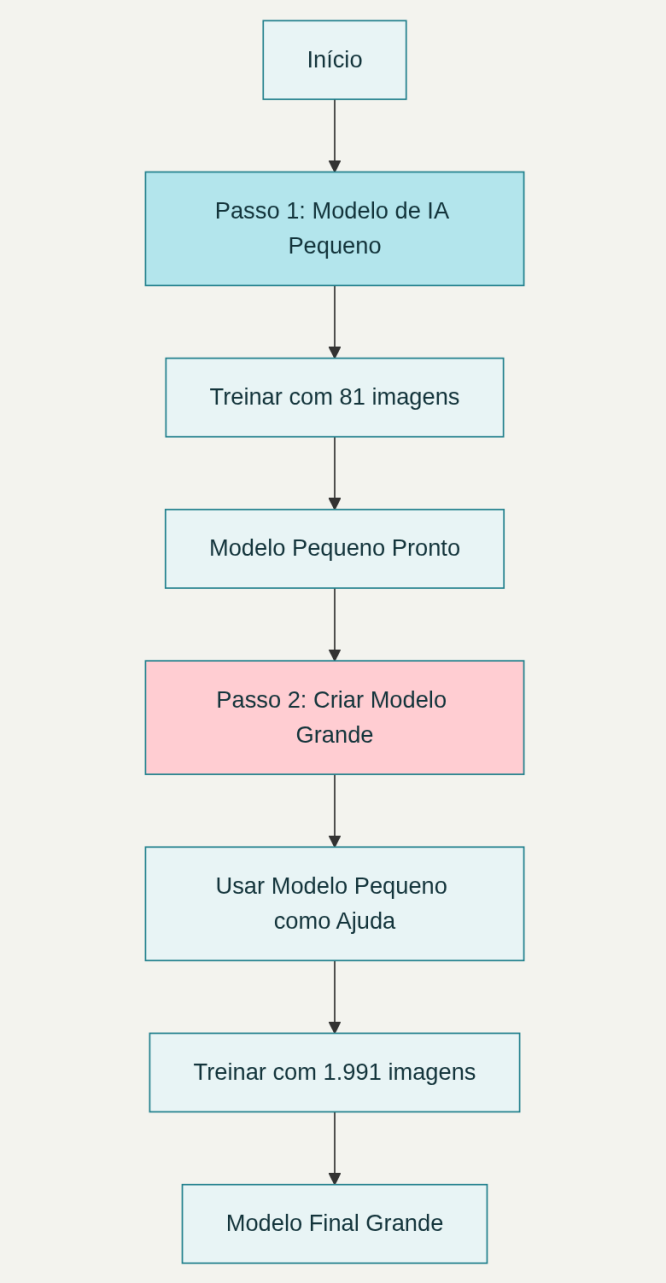

Figura 15 – Fluxograma da estratégia das anotações.

Fonte: Do autor.

O objetivo principal foi facilitar o processo de anotação do *dataset* completo, por meio da criação de um conjunto reduzido e estruturado, que serviu de base para o treinamento do modelo de visão computacional. A metodologia envolveu a importação, anotação e organização das imagens, além da aplicação de técnicas de pré-processamento e de au-

mento de dados (*augmentation*) para otimizar a qualidade do treinamento. A Figura 16 ilustrar o processo de validação do modelo auxiliar.

Figura 16 – Validação do modelo auxiliar.

Fonte: <https://app.roboflow.com/>.

O modelo auxiliar foi treinado por 300 épocas utilizando o modelo disponibilizado pela plataforma, denominado *Roboflow 3.0 Object Detection (Fast)*. Essa configuração resultou em uma *mAP@50* de 93,5%, demonstrando alta precisão na detecção das abelhas após validação ilustrada na Figura 17 onde as curvas de precisão e *recall* atingem patamares elevados ainda nas primeiras épocas e permanecem estáveis ao longo do treinamento.

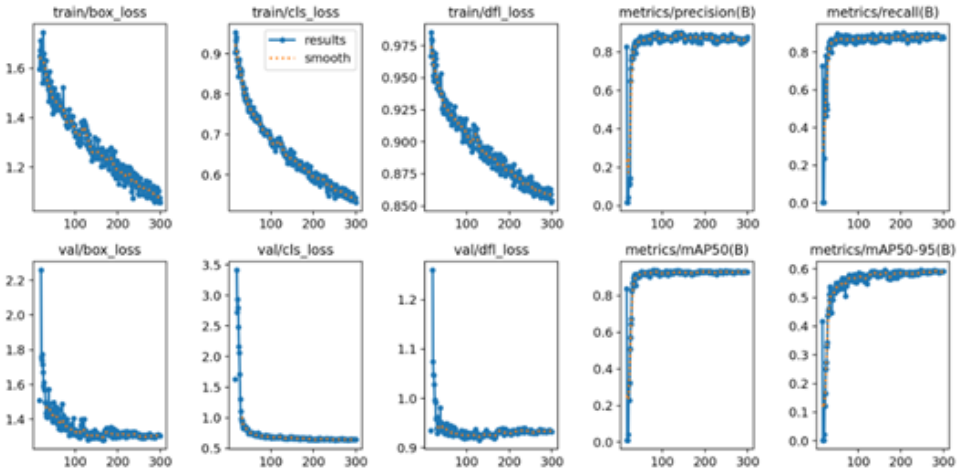

Figura 17 – Métricas do modelo auxiliar.

Fonte: <https://app.roboflow.com/>.

O treinamento do modelo de menor porte facilitou significativamente o processo de anotação, proporcionando um ganho expressivo de tempo, pois foi capaz de reconhecer automaticamente a maioria das abelhas. Contudo, ajustes manuais ainda se fizeram

necessários para assegurar que todas as abelhas fossem devidamente identificadas, como observado na Figura 18.

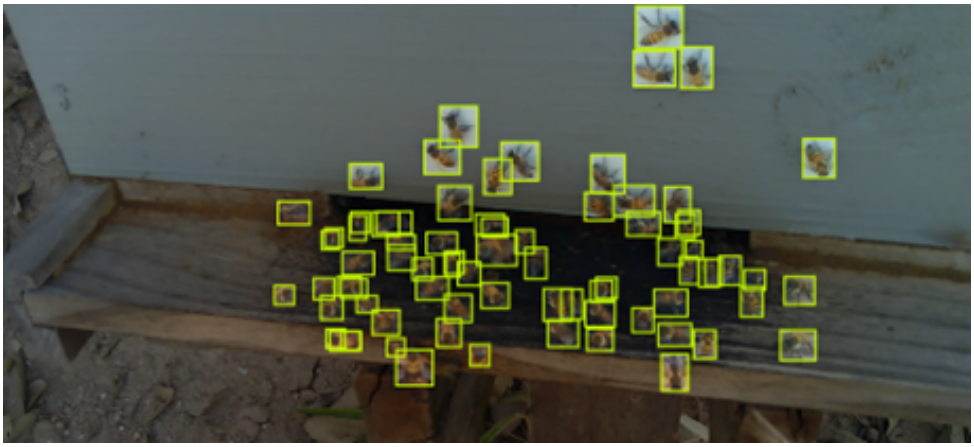

Figura 18 – Processo de anotação utilizando modelo auxiliar.

Fonte: Do Autor

Ao término da fase de anotações, foi realizado o pré-processamento com o redimensionamento das imagens para 640×640 *pixels*. Em seguida, aplicaram-se técnicas de aumento de dados para expandir o conjunto de imagens de entrada da colmeia, aumentando o número de amostras disponíveis. Foram utilizadas transformações de brilho, espelhamento e rotações [30]. Como resultado, o *dataset* foi ampliado para um total de 4.574 imagens, conforme ilustrado na Figura 19. Posteriormente, o conjunto de imagens foi dividido aleatoriamente em três subconjuntos: 87% para treinamento, 9% para validação e 4% para teste. Esses subconjuntos foram empregados no treinamento do modelo, na otimização de parâmetros e na comparação com os resultados de predição para avaliar o desempenho da detecção de abelhas.

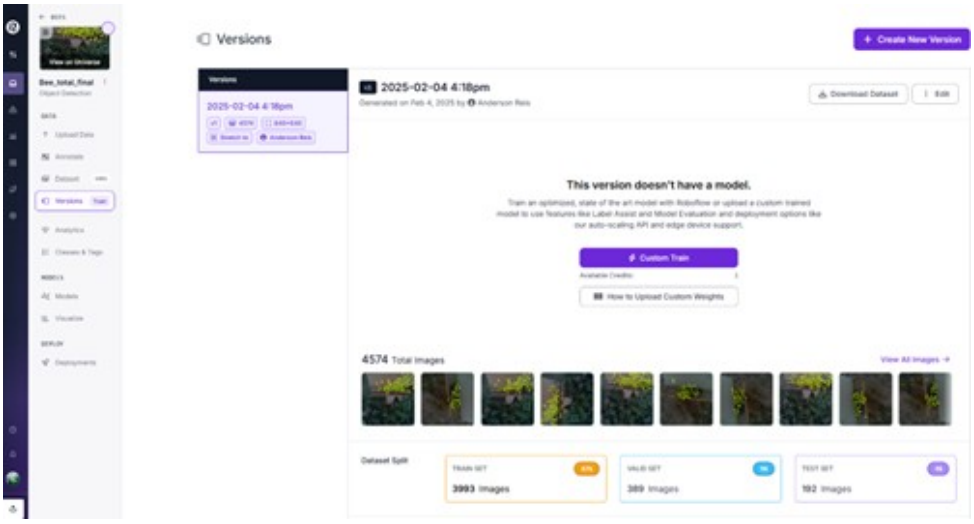

Figura 19 – *Dataset* final utilizado para o treinamento.

Fonte: <https://app.roboflow.com/>.

3.3 Treinamento e validação do modelo

Após a exportação do modelo no formato *YOLOv8*, o conjunto de dados pode ser armazenado em um arquivo compactado para armazenamento local e processamento. Alternativamente, o código gerado pode ser baixado diretamente para execução.

O treinamento do modelo foi realizado em uma *GPU NVIDIA Tesla T4*. Essa plataforma de *hardware* é projetada para cargas de trabalho de aprendizado profundo de alto desempenho, oferecendo um equilíbrio eficiente entre capacidade computacional e escalabilidade. A *Tesla T4* é baseada na arquitetura *Turing* e integra 2.560 núcleos CUDA e 320 *Tensor Cores*. Ela fornece até 65 *TFLOPS* de desempenho em precisão mista e dispõe de 16 GB de memória *GDDR6*. Essa GPU é otimizada para tarefas de treinamento e inferência ao suportar paralelismo em grande escala, mantendo consumo energético relativamente baixo (TDP de 70 W). Devido à sua eficiência, a *Tesla T4* é amplamente utilizada em ambientes de computação em nuvem, incluindo o *Google Colab*, onde oferece recursos acessíveis e confiáveis para treinamento e avaliação de modelos.

O processo de treinamento foi cuidadosamente monitorado para garantir a convergência das métricas de validação. O modelo foi treinado por 100 épocas, que levou aproximadamente 2,4 horas para ser concluído. Essa configuração ofereceu iterações suficientes para atingir a convergência, ao mesmo tempo em que mitigou o risco de sobreajuste. As métricas de validação apresentaram estabilização progressiva durante o treinamento, indicando que o modelo atingiu um equilíbrio ótimo entre acurácia e capacidade de generalização [31, 32].

A Figura 20 ilustra que o modelo apresentou desempenho consistente nos conjuntos de dados de treino e validação, confirmando a robustez e a confiabilidade da versão final. A convergência foi avaliada pela ausência de melhorias adicionais na perda composta de validação. Essa perda foi definida como a soma da perda de localização de *bounding boxes*, da perda de classificação de objetos e da DFL. Após o treinamento, o modelo apresentou um $mAP50 = 0{,}987$.

```
/content
Ultralytics 8.3.203 🚀 Python-3.12.11 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB)
Model summary (fused): 72 layers, 3,005,843 parameters, 0 gradients, 8.1 GFLOPs
val: Fast image access ✅ (ping: 0.0±0.0 ms, read: 2331.0±297.1 MB/s, size: 84.2 KB)
val: Scanning /content/Bee_total_final-1/valid/labels.cache... 389 images, 18 backgrounds,
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
                   all        389      41276      0.952      0.959      0.987      0.695
Speed: 2.1ms preprocess, 5.2ms inference, 0.0ms loss, 5.1ms postprocess per image
Results saved to /content/runs/detect/val
💡 Learn more at https://docs.ultralytics.com/modes/val
```

Figura 20 – Validação do treinamento.

Fonte:Google Colab.

As Figuras 21 e 22 apresentam, respectivamente, a matriz de confusão e as curvas das métricas de desempenho ao longo das épocas de treinamento, permitindo avaliar a evolução da acurácia e a estabilidade do modelo durante o processo de aprendizado onde observa-se que as curvas de precisão e *recall* alcançam valores elevados já nas etapas iniciais do treinamento e mantêm-se estáveis ao longo das épocas.

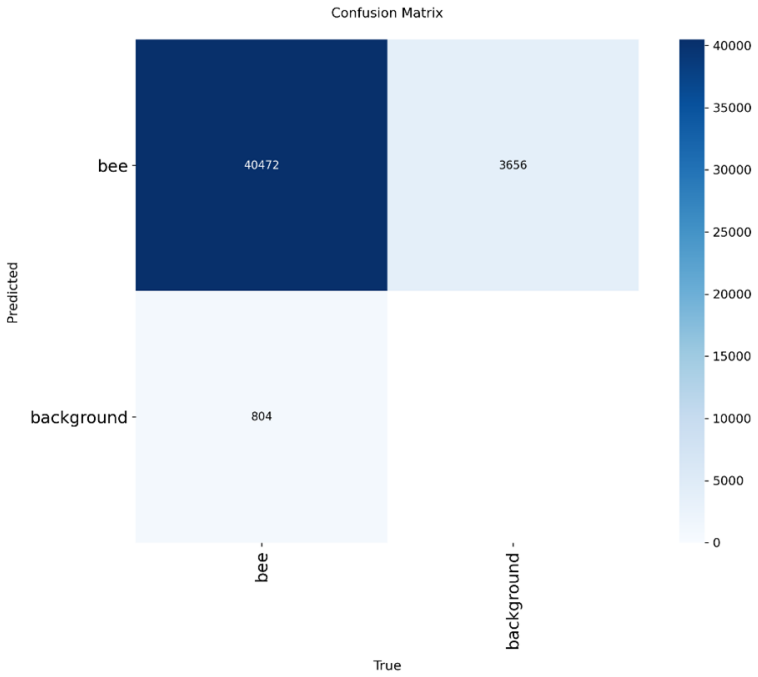

Figura 21 – Matriz de confusão do modelo YOLOv8.

Fonte: Do autor.

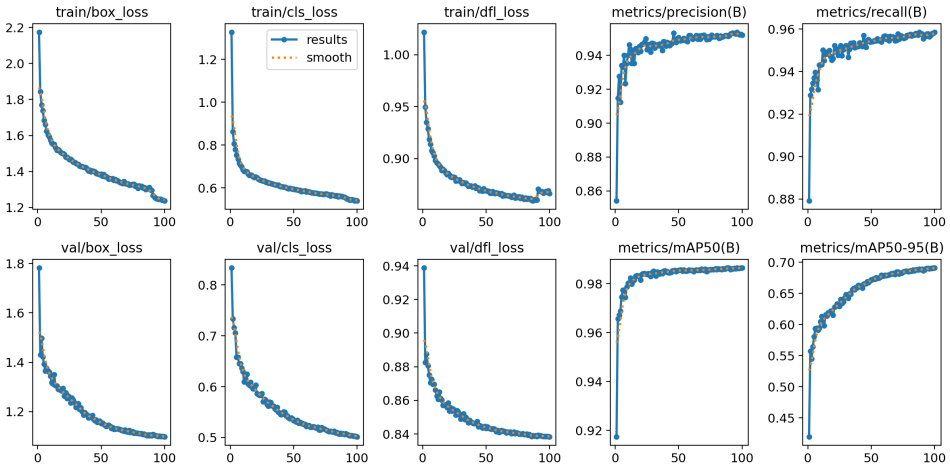

Figura 22 – Curvas de métricas por época durante o treinamento.

Fonte: Do autor.

A Figura 23 apresenta os resultados obtidos no processo de validação utilizando o

conjunto de imagens de teste. Observa-se que o modelo demonstrou alto desempenho na identificação das abelhas, evidenciando sua eficiência e robustez nas tarefas de detecção.

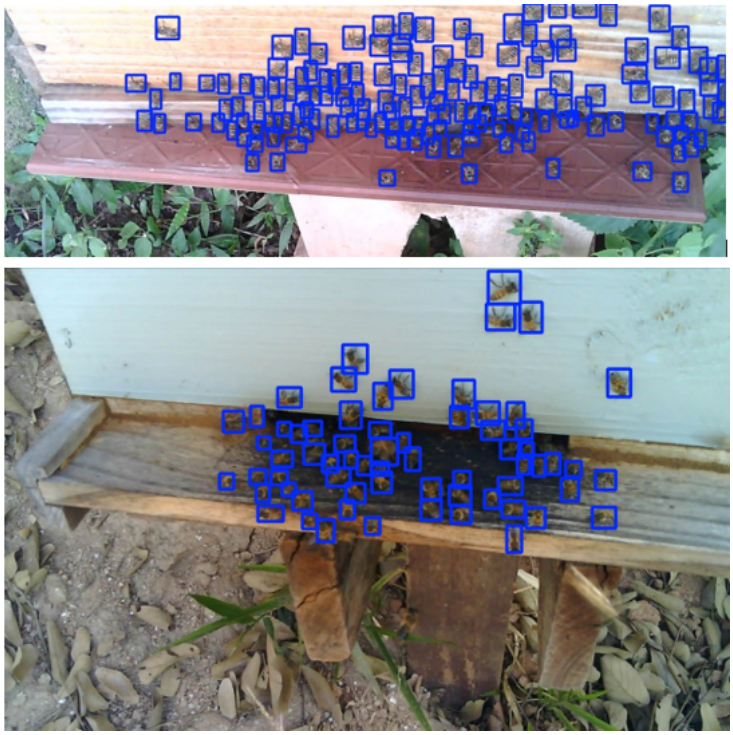

Figura 23 – Resultados do processo de validação a partir de imagens de teste.

Fonte: Do autor.

3.4 Descrição do *Hardware*

A seleção e a integração dos componentes de *hardware* foram realizadas considerando os requisitos de baixo consumo energético, portabilidade, confiabilidade em ambientes externos e capacidade computacional suficiente para a execução local de inferências baseadas em redes neurais convolucionais. O sistema foi projetado para operar de forma autônoma, sem dependência de infraestrutura de comunicação em nuvem, caracterizando uma aplicação típica de computação em borda (*Edge Machine Learning*).

O conjunto de dispositivos empregados garante a aquisição contínua de dados visuais e ambientais da entrada da colmeia, com elevada precisão e com mínima interferência no comportamento natural das abelhas. A Tabela 2 apresenta as principais especificações técnicas dos componentes utilizados na implementação do protótipo de monitoramento.

A montagem final do protótipo foi realizada integrando todos os componentes de hardware descritos na Tabela 2 em uma estrutura compacta, resistente e de fácil manutenção. O sistema foi acondicionado em um gabinete com classificação IP65, garantindo proteção contra poeira, umidade e variações de temperatura, características essenciais para a operação em ambiente externo.

Tabela 2 – Descrição do hardware utilizado.

Componente	**Descrição Técnica**	**Função no Sistema**
Placa de processamento	Raspberry Pi Zero 2W – Processador Quad-core ARM Cortex-A53 de 64 bits a 1 GHz; 512 MB de SDRAM; conectividade Wi-Fi 802.11 b/g/n, Bluetooth 4.2 e BLE; interfaces CSI-2, micro-USB OTG, mini HDMI® e slot microSD; alimentação via micro-USB; dimensões 65 mm × 30 mm.	Unidade central de processamento responsável pela execução do modelo de aprendizado de máquina e pelo gerenciamento dos periféricos.
Módulo de câmera	OV5647 CMOS 5 MP – Resolução máxima 2592 × 1944 pixels; formato óptico 1/4; tamanho de pixel 1,4 µm; lente de foco fixo de 3,6 mm; campo de visão (FoV) 54° × 41°; taxa de quadros ajustável de até 90 fps.	Captura das imagens na entrada da colmeia para análise e contagem automatizada das abelhas.
Sensor de temperatura e umidade	Sensirion SHT20 – Interface I^2C; precisão típica ±3 % UR e ±0,3 °C; faixa operacional de –40 °C a +125 °C.	Monitoramento das condições microclimáticas no entorno da colmeia.
Relógio de tempo real (RTC)	DS1302 – Comunicação serial de três fios; registro de segundos, minutos, horas, dia, mês e ano com compensação automática de anos bissextos.	Fornecimento de marcação temporal precisa para sincronização e armazenamento dos dados coletados.
Gabinete	Gabinete IP65 com proteção contra poeira e respingos de água.	Proteção do sistema em ambiente externo.
Montagem	Tripé ajustável para posicionamento frontal à colmeia.	Estabilização física durante a aquisição de dados.
Alimentação elétrica	Power bank de 5 V via porta micro-USB.	Suprimento de energia estável ao sistema embarcado.

Fonte: Autor.

O módulo de câmera foi fixado na parte frontal do gabinete, alinhado com a entrada da colmeia, de modo a capturar toda a área de interesse sem a necessidade de lentes adicionais. A disposição física dos componentes internos (placa de processamento, sensores e módulo de alimentação) está representada na Figura 24.

Figura 24 – Montagem do sistema

Fonte: Do autor.

3.5 Implementação do sistema

Para a execução do modelo de detecção em ambiente embarcado, foi necessário preparar e configurar a placa de processamento, garantindo a compatibilidade entre o *hardware* e o *software* do sistema. Essa etapa teve como objetivo estabelecer um ambiente estável, leve e seguro para o funcionamento do modelo de aprendizado de máquina na borda, minimizando o consumo de recursos e evitando interferências externas durante a operação.

A preparação da placa para receber o modelo foi realizada por meio da instalação do sistema operacional *Raspberry Pi OS Lite 64 bits*, uma versão otimizada e de baixo consumo de recursos, sem interface gráfica, ideal para aplicações embarcadas. A configuração do dispositivo foi realizada via SSH (*Secure Shell*), um protocolo de rede que permite acesso remoto seguro a computadores e servidores por meio de uma conexão criptografada.

O SSH utiliza mecanismos de criptografia e autenticação baseados em chaves públicas, assegurando a confidencialidade, a integridade e a autenticidade dos dados trans-

mitidos. Essa tecnologia é amplamente empregada em tarefas de administração remota, transferência de arquivos e gerenciamento de sistemas, sendo fundamental para a operação e manutenção de dispositivos embarcados.

Inicialmente, foi configurado um espaço de *swap* de 2 GB no cartão SD, destinado a funcionar como memória RAM virtual, ampliando a capacidade de processamento da placa em tarefas mais intensivas. Em seguida, foi criado um ambiente virtual dedicado à aplicação, com o propósito de isolar e gerenciar as dependências do projeto, assegurando a compatibilidade e a reprodutibilidade das versões das bibliotecas utilizadas durante o desenvolvimento do sistema [33].

Para a execução eficiente do modelo de detecção *YOLOv8n* na borda, o modelo treinado foi exportado para o formato NCNN [34], conforme as recomendações oficiais da *Ultralytics*[1]. O NCNN é um *framework* de inferência otimizado para dispositivos embarcados e sistemas com recursos limitados, desenvolvido para proporcionar alta velocidade de execução, baixo consumo de memória e independência de bibliotecas externas, como o CUDA ou o *TensorRT*. Essa conversão foi essencial para permitir o processamento local em dispositivos sem GPU ou TPU dedicados, garantindo baixa latência e portabilidade do sistema em diferentes plataformas *ARM*.

O processo de exportação foi realizado diretamente a partir da biblioteca *Ultralytics*, utilizando o comando de conversão para NCNN, o que gerou os arquivos necessários para a inferência. A adoção dessa abordagem permitiu que o modelo fosse executado de forma autônoma na *Raspberry Pi Zero 2W*, mantendo a precisão obtida no treinamento e assegurando a viabilidade do sistema proposto em cenários de *Edge Machine Learning*.

O fluxograma apresentado na Figura 25 ilustra o funcionamento geral do sistema de monitoramento não invasivo de colmeias, desenvolvido para operar em ambiente embarcado. O código correspondente a esse fluxograma encontra-se no (Apêndice A). O diagrama descreve, de forma sequencial e estruturada, as etapas executadas pelo *software* implementado, abrangendo desde a inicialização dos módulos até o armazenamento das informações processadas. A lógica do sistema foi projetada para garantir eficiência no processamento local, uso otimizado dos recursos computacionais e robustez diante de falhas durante a execução contínua em campo.

O processo inicia-se com o carregamento do modelo de detecção *YOLOv8n_ncnn_model* e dos parâmetros de configuração definidos para o sistema. Em seguida, são inicializadas duas *threads* principais: a *thread* principal (*main*) e a *thread* de monitoramento (*watchdog*).

A *thread watchdog* tem como função monitorar a integridade operacional do sistema. Ela verifica periodicamente se o programa principal está ativo e responde dentro de

[1] Disponível em: <https://docs.ultralytics.com/pt/integrations/ncnn/>

um intervalo de tempo estabelecido. Caso o sinal de atividade ("alimentação do *watchdog*") não seja detectado, o sistema encerra automaticamente o processo, evitando travamentos e garantindo uma reinicialização segura.

Paralelamente, a *thread main* é responsável pela execução das rotinas centrais. Inicialmente, ela configura o banco de dados e define as estruturas de armazenamento temporário. Em seguida, realiza a captura de imagens na entrada da colmeia, aplicando o modelo de detecção e contagem de abelhas. Após o processamento da imagem, o sistema lê os sensores de temperatura e umidade (*SHT20*) e armazena essas informações em um vetor temporário.

Quando o vetor atinge a quantidade predefinida de registros, os dados são transferidos para o banco de dados principal e as imagens correspondentes são removidas da memória *RAM*, que otimiza o uso dos recursos do microcontrolador. Ao final de cada iteração, a *thread main* envia o sinal de alimentação ao *watchdog*, indicando que o sistema está funcionando corretamente, e o ciclo recomeça de forma contínua e autônoma.

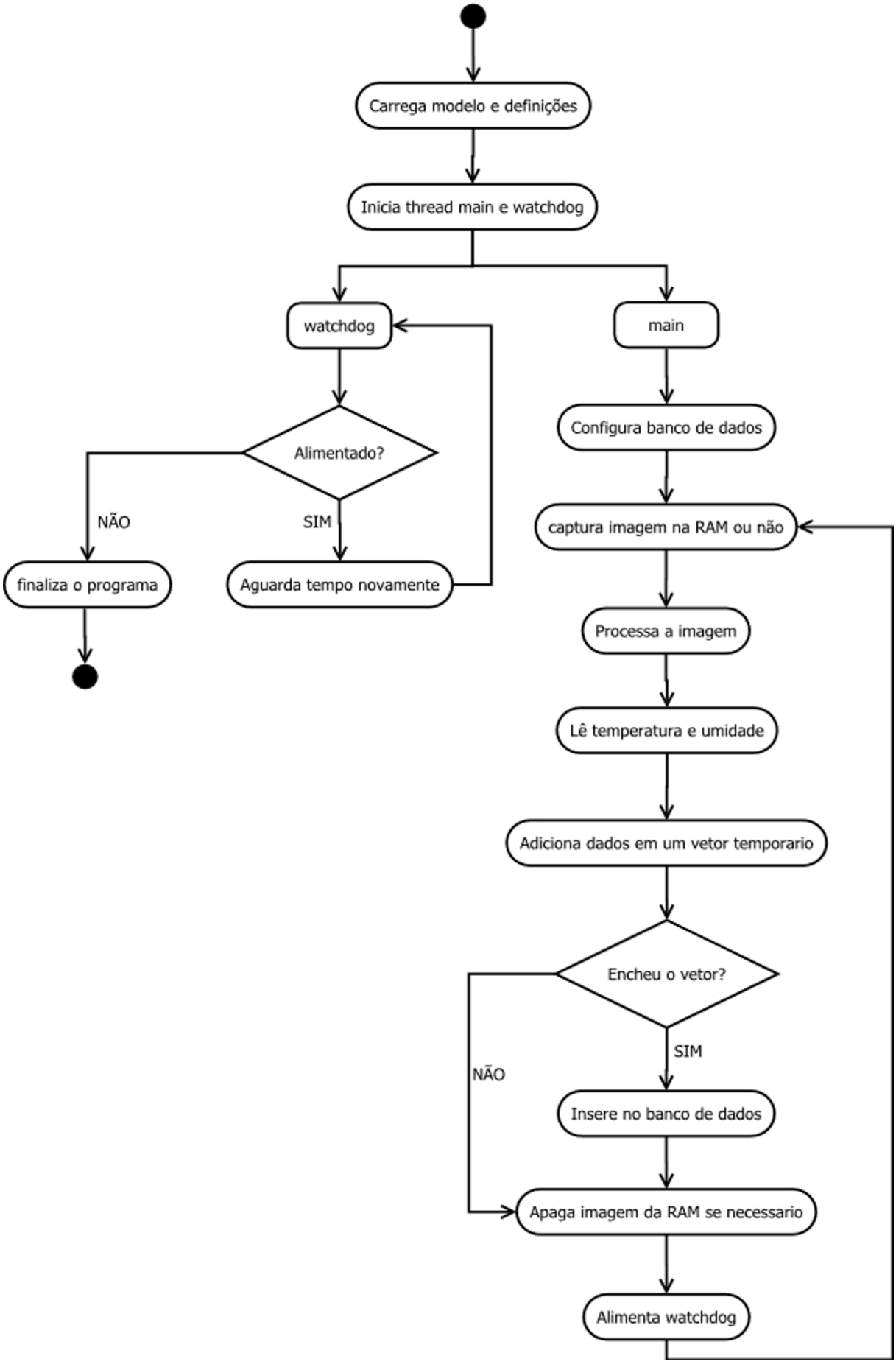

Figura 25 – Fluxogra do funcionamento do sistema embarcado.

Fonte: Do autor.

4 Resultados

4.1 Testes em campo

A Figura 26 ilustra o resultado da montagem do protótipo completo, incluindo o posicionamento sobre o tripé e a orientação em relação à colmeia. Essa configuração assegura a estabilidade do sistema durante o processo de aquisição de imagens e reduz a perturbação da atividade natural das abelhas, permitindo a coleta contínua de dados visuais e ambientais de forma não invasiva.

Figura 26 – Montagem e posicionamento do sistema.

Fonte: Do autor.

A Figura 27 apresenta a comparação visual entre três estágios distintos do processo de inferência do modelo *YOLOv8* aplicado à detecção de abelhas na entrada da colmeia.

- Na imagem à esquerda, observa-se o quadro original capturado pela câmera, sem qualquer processamento, ilustrando a entrada da colmeia em condições naturais de iluminação de presença de abelhas.
- Na imagem central, são exibidas as detecções iniciais do modelo, evidenciadas pelas *bounding boxes* azuis ao redor de cada abelha identificada. Este estágio demonstra o desempenho do algoritmo em um cenário de menor densidade de insetos.
- Na imagem à direita, o modelo realiza a inferência em uma situação com maior concentração de abelhas, mantendo a precisão nas detecções mesmo diante de oclusões parciais e de sobreposição entre indivíduos.

Essa sequência comprova a robustez do modelo em condições ambientais reais, com variações de luminosidade e densidade de objetos, validando sua aplicabilidade para o monitoramento automatizado de colmeias.

Figura 27 – Validação dos resultados.

Fonte: Do autor.

A Tabela 3 apresenta a estrutura de armazenamento dos dados no banco de dados, incluindo os principais parâmetros registrados durante o monitoramento na fase de validação do sistema. Cada linha corresponde a uma imagem processada, identificada pelo nome do arquivo, que inclui a data e o horário da captura. Associados a cada registro, estão também a contagem de abelhas detectadas, o tempo de inferência do modelo e as medições ambientais de temperatura e umidade, coletadas simultaneamente ao processo de inferência. Na prototipagem final do sistema, as imagens não serão armazenadas, ficando apenas armazenadas a contagem, a inferência, a temperatura e a umidade.

Tabela 3 – Exemplo de dados armazenados no banco de dados.

Nome da imagem	**Contagem**	**Inferência (ms)**	**Temperatura**	**Umidade**
2025-08-23_15-29-44	24	841	26,50°C	34,02
2025-08-23_15-29-46	26	852	26,50°C	34,02
2025-08-23_15-29-49	25	847	26,50°C	34,02
2025-08-23_15-29-51	26	841	26,50°C	34,02
2025-08-23_15-29-53	26	841	26,50°C	34,02
2025-08-23_15-29-55	33	845	26,50°C	34,08
2025-08-23_15-29-57	32	846	26,50°C	34,15

Fonte: Do autor.

A Figura 28 apresenta a variação temporal da contagem de abelhas detectadas, da temperatura e da umidade registradas durante o experimento de monitoramento automatizado da colmeia em um período de duas horas. O gráfico em amarelo apresenta o número de abelhas identificadas pelo modelo na entrada da colmeia, evidenciando oscilações na atividade ao longo do período analisado.

O gráfico em vermelho mostra a variação da temperatura ambiente (°C), com tendência de leve declínio ao longo do tempo.

O gráfico em azul ilustra a umidade relativa do ar (%), que apresentou pequenas flutuações naturais ao longo do intervalo de observação, refletindo as condições microclimáticas do entorno da colmeia.

Essa correlação visual entre a atividade das abelhas e as variáveis ambientais reforça a eficácia do sistema proposto em integrar o monitoramento das abelhas de forma contínua e não invasiva, associando à temperatura e à umidade relativa do ambiente.

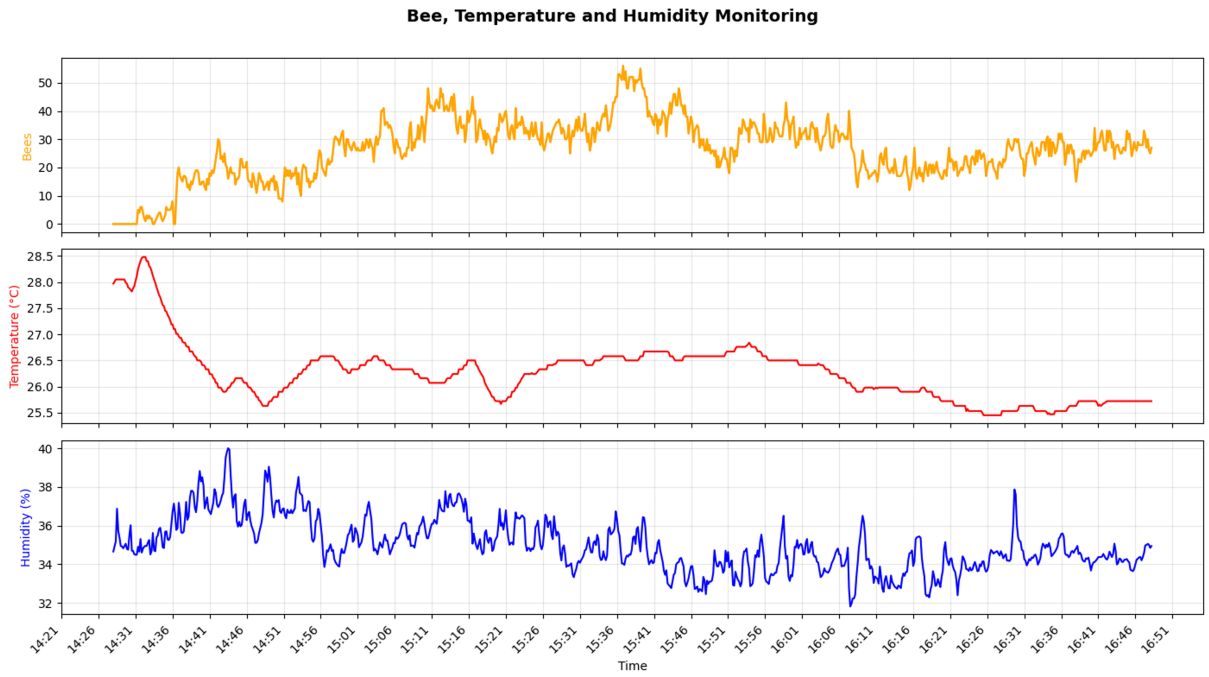

Figura 28 – Validação dos resultados.

Fonte: Do autor.

Durante os testes, também foi possível observar a atuação do *watchdog*, no qual o sistema realizou automaticamente a reinicialização, conforme ilustrado na Figura 29.

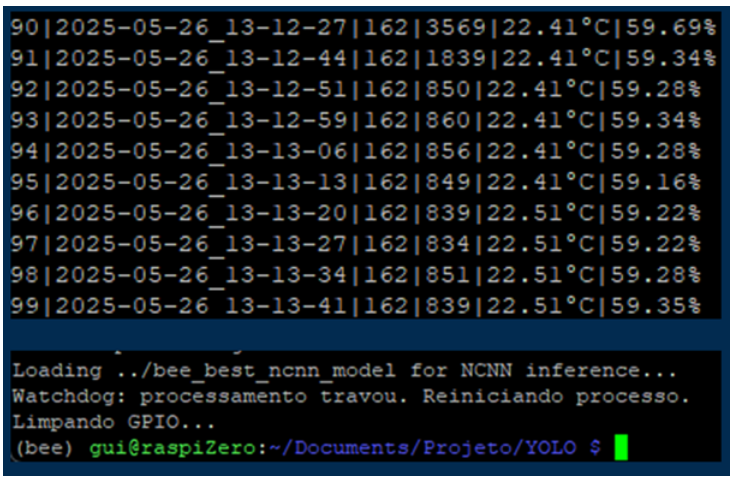

Figura 29 – Atuação do watchdog.

Fonte: Do autor.

Na fase de validação do sistema com fluxo contínuo de amostras, obtiveram-se os seguintes tempos: durante a primeira execução de inferência, o sistema levou 4,018 segundos. O processo completo incluiu: 24,04 segundos para inicialização do sistema, 23,67 segundos para carregamento do modelo, 6,085 segundos para captura da imagem, 0,874 segundos para a etapa de inferência, 6,085 segundos para o salvamento das imagens processadas e não processadas, e 0,196 segundos para gravação dos resultados no banco de

dados. O tempo total médio por ciclo operacional foi de aproximadamente 2,542 segundos, conforme demonstrado na Figura 30.

```
Começando...
tempo para foto 6.085s
Loading bee_best_ncnn_model for NCNN inference...
Results saved to processed_images
tempo para carregar o modelo 23.67s
Inference time 4018
Bee count: 0
Temperature: 21.81°C
Humidity: 73.70%

==============================
Processamento concluído em 30.011s
Começando...
tempo para foto 2.707s
Results saved to processed_images
tempo para carregar o modelo 1.224s
Inference time 1023
Bee count: 0
Temperature: 21.81°C
Humidity: 73.70%

==============================
Processamento concluído em 4.147s
Começando...
tempo para foto 9.986s
Results saved to processed_images
tempo para carregar o modelo 5.063s
Inference time 3995
Bee count: 0
Temperature: 21.81°C
Humidity: 73.59%

==============================
Processamento concluído em 15.656s
Começando...
tempo para foto 1.124s
Results saved to processed_images
tempo para carregar o modelo 1.137s
Inference time 992
Bee count: 0
Temperature: 21.81°C
Humidity: 73.59%

==============================
Processamento concluído em 2.464s
Começando...
tempo para foto 0.854s
Results saved to processed_images
tempo para carregar o modelo 1.277s
Inference time 874
Bee count: 0
Temperature: 21.81°C
Humidity: 73.59%
tempo para salvar no bacno 0.196s

==============================
Processamento concluído em 2.542s
Começando...
tempo para foto 0.981s
Results saved to processed_images
tempo para carregar o modelo 0.953s
Inference time 861
```

Figura 30 – Tempo de funcionamento em fluxo contínuo.

Fonte: Do autor.

Na fase de validação do sistema com 10 segundos de amostragem e sem armazenamento das imagens, os tempos são semelhantes, porém com adição de um *sleep* de 8 segundos, conforme a Figura 31.

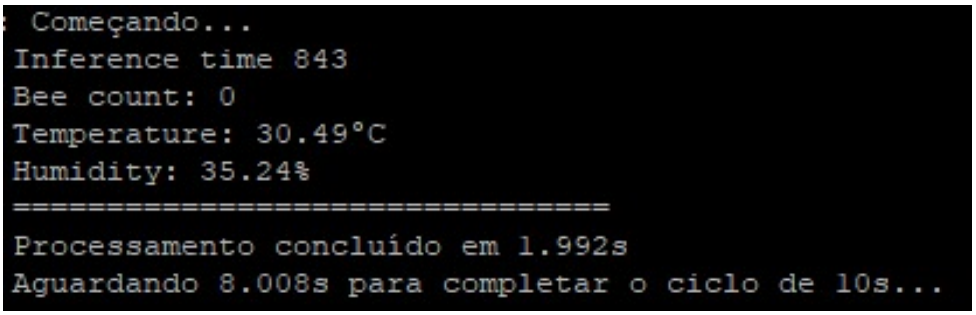

Figura 31 – Tempos de funcionamento com taxa de amostragem de 10 segundos.

Fonte: Do autor.

Durante a etapa de teste em campo, observou-se a ocorrência de falsos positivos e falsos negativos, fenômenos inerentes a sistemas de detecção baseados em aprendizado profundo. A Figura 32 ilustra um desses casos.

Os falsos positivos referem-se às situações em que o modelo prediz a presença de uma abelha em regiões da imagem onde, na realidade, não há o inseto. Esse tipo de erro pode ser atribuído a texturas de fundo complexas, reflexos luminosos ou objetos morfologicamente semelhantes, que induzem a rede neural a classificar erroneamente tais áreas como contagens válidas.

Por outro lado, os falsos negativos correspondem às abelhas presentes na cena e o modelo não identificou. Esses casos ocorrem, principalmente, em situações de oclusão parcial, sobreposição entre indivíduos, movimento rápido das abelhas ou condições de iluminação desfavoráveis, o que resultam em perda de nitidez e dificultam a detecção.

Figura 32 – Exemplos de falsos positivos e falsos negativos detectados pelo modelo.

Fonte: Do autor.

4.2 Análise do método padrão ouro

A validação do modelo proposto foi realizada por meio da comparação entre as contagens automatizadas de abelhas obtidas pelo sistema e as contagens manuais, consideradas o método padrão-ouro (*gold standard*). Essa metodologia é amplamente utilizada em estudos que visam verificar a confiabilidade de métodos automatizados em relação a medições de referência, permitindo mensurar o grau de concordância entre ambos os procedimentos [35]. Foram utilizadas nesse processo 50 imagens para verificação manual dos seguintes parâmetros: contagem do modelo, contagem manual, falsos positivos e falsos negativos (Apêndice B).

O desempenho global do modelo manteve-se elevado, com precisão de 0,91, *recall* de 0,96 e *F1-score* de 0,94, indicando alta confiabilidade nas tarefas de detecção, conforme podemos demonstrar:

$$Precision = \frac{TP}{TP + FP} = \frac{1005}{1005 + 89} = 0{,}91 \tag{4.1}$$

$$Recall = \frac{TP}{TP + FN} = \frac{1005}{1005 + 35} = 0{,}96 \tag{4.2}$$

$$F_1 = 2 \times \frac{(Precision \times Recall)}{Precision + Recall} = 2 \times \frac{(0{,}91 \times 0{,}97)}{0{,}91 + 0{,}97} = 0{,}94 \tag{4.3}$$

Para avaliar a associação linear entre as duas abordagens, aplicou-se a análise de correlação de *Pearson*. O coeficiente de correlação obtido foi $r = 0{,}97$ ($p < 0{,}00001$),

indicando forte correlação positiva e alta significância estatística entre as contagens automatizadas e manuais, conforme ilustrado na Figura 33. Resultados semelhantes foram relatados por [36], que também observaram altos índices de correlação entre métodos de contagem manual e automatizada em sistemas de monitoramento não invasivo de abelhas.

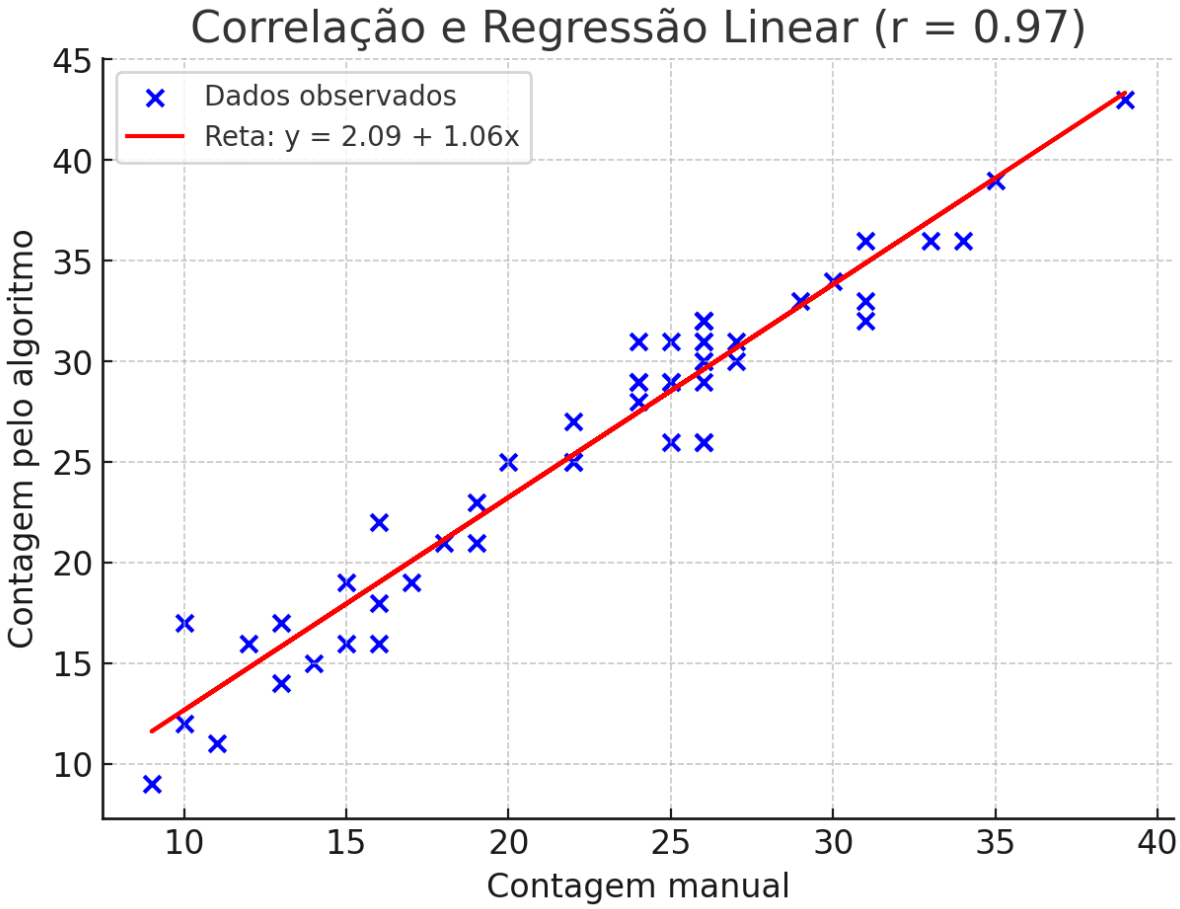

Figura 33 – Gráfico de dispersão entre contagem manual e automatizada.

Fonte: Do autor.

Do mesmo modo, realizou-se a análise de *Bland-Altman*, técnica recomendada para avaliar a concordância entre dois métodos de medição [35]. Essa análise estima o viés médio entre as medições e define os limites de concordância de 95%, dentro dos quais se espera que a maioria das diferenças entre os métodos esteja contida. Os resultados obtidos apresentaram um viés médio de 3,4 abelhas, com limites de concordância de $\pm 1{,}96$ DP, que demonstra que as diferenças entre os métodos permanecem dentro de intervalos estatisticamente aceitáveis, conforme mostrado na Figura 34.

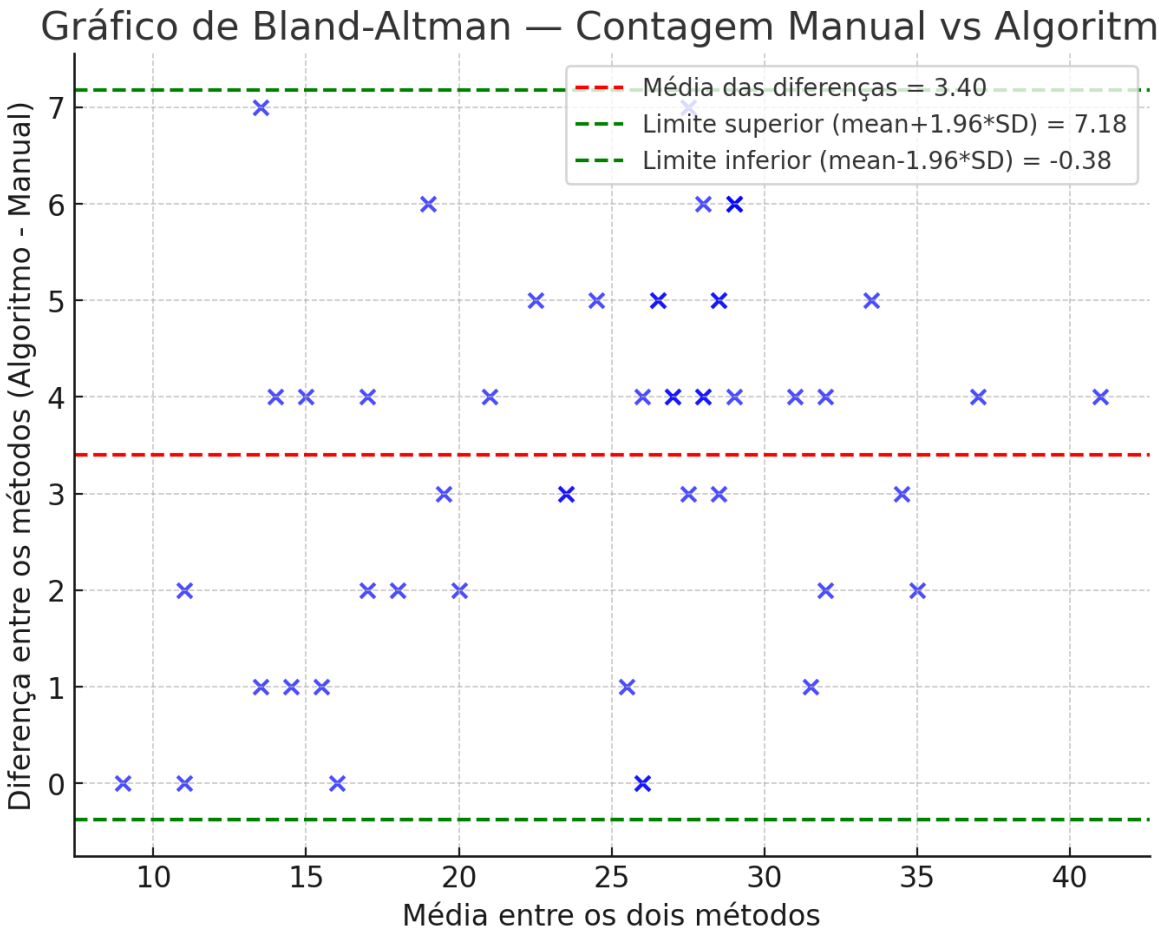

Figura 34 – Gráfico representando o viés e limites de concordância.

Fonte: Do autor.

4.3 Análise energética

4.3.1 Fluxo contínuo de monitoramento

O teste teve como objetivo caracterizar o consumo energético de uma *Raspberry Pi Zero 2W* operando em regime contínuo com execução de inferências sequenciais, visando dimensionar adequadamente um sistema de alimentação autônomo baseado em energia solar.

Configuração do Sistema

- Equipamento: *Raspberry Pi Zero 2W*
- Tensão de Operação: 5,08 VCC
- Modo de Operação: Amostragem sequencial com intervalo de 2 segundos entre inferências

- Duração do Teste: 35 horas contínuas

O teste foi conduzido em ambiente controlado, com a *Raspberry Pi Zero 2W* executando ciclos contínuos de inferência com intervalo de 2 segundos entre cada execução. Durante todo o período, foram monitorados os parâmetros elétricos de tensão e corrente consumida, permitindo calcular a energia total consumida pelo sistema. A Tabela 4 demonstra os resultados obtidos:

Tabela 4 – Resultados do teste de consumo energético da *Raspberry Pi Zero 2W*.

Parâmetro	**Valor**
Carga Total Consumida	11.776 mAh
Energia Total Consumida	59 Wh
Corrente Média	336,46 mA
Potência Média	1,71 W
Consumo Diário Estimado	8,08 Ah/dia
Energia Diária Estimada	40,97 Wh/dia

Fonte: Do autor.

Para o dimensionamento do painel fotovoltaico foram consideradas as seguintes premissas:

- Autonomia da bateria: 5 dias (considerando período nublado)
- Profundidade de descarga (*DoD*): 50% (para maior vida útil)
- Eficiência do controlador de carga: 85%
- Horas de sol útil (*HSP*): 4,5 horas/dia (média para Brasil)
- Margem de segurança: 20%
- Tensão do sistema: 12 V (sistema padrão)

Cálculo da capacidade da bateria:

- Energia diária: 40,97 Wh/dia
- Corrente diária em 12 V: 40,97 Wh $\div$ 12 V = 3,41 Ah/dia

Capacidade necessária para 5 dias:

- Capacidade base: 3,41 Ah/dia $\times$ 5 dias = 17,05 Ah
- Considerando *DoD* de 50%: 17,05 Ah $\div$ 0,50 = 34,10 Ah

- Com margem de segurança (20%): 34,10 Ah × 1,20 = 40,92 Ah
- Bateria recomendada *LiFePO4* 12 V 50 Ah (maior vida útil, menor peso)

Cálculo do Painel Solar levando em consideração a energia diária considerando as perdas:

- Energia consumida: 40,97 Wh/dia
- Eficiência do controlador: 85%
- Energia a ser gerada: 40,97 Wh ÷ 0,85 = 48,20 Wh/dia
- Com margem de segurança (20%): 48,20 Wh × 1,20 = 57,84 Wh/dia

Potência do painel:

- Potência necessária: 57,84 Wh ÷ 4,5 *HSP* = 12,85 Wp
- Painel solar recomendado: 50 Wp (ideal) ou 30 Wp (mínimo)

Especificações do painel:

- Tensão nominal: 12 V
- Corrente de curto-circuito (*Isc*): > 1,5 A
- Tensão de circuito aberto (*Voc*): 18–22 V

Controlador de Carga:

Corrente máxima do painel (considerando 50 Wp): ∼3 A com margem de segurança de 3,75 A. Controlador recomendado: 10 A *PWM* ou *MPPT* / 12 V com as seguintes características:

- Tipo: *PWM* (econômico) ou *MPPT* (maior eficiência)
- Proteções: sobrecarga, descarga profunda, curto-circuito

A tabela 5 exibe a lista de componentes sugeridos.

Tabela 5 – Configuração final do sistema fotovoltaico para operação autônoma da *Raspberry Pi Zero 2W*.

Item	**Componente**	**Especificação**
1	Painel Solar Fotovoltaico	50 Wp (recomendado) ou 30 Wp (mínimo) / 12 V
2	Bateria Estacionária	45–50 Ah / 12 V / Ciclo profundo ou *LiFePO4*
3	Controlador de Carga Solar	10 A / 12 V / *PWM* ou *MPPT*
4	Conversor DC-DC Step-Down	Entrada 12 V / Saída 5 V 3 A
5	Cabeamento e Conectores	Cabos 4 mm^2 e 2,5 mm^2, Fusíveis 5 A e 10 A, *MC4*

Fonte: Do autor.

A Figura 35 apresenta o diagrama de conexão do sistema de alimentação autônoma, ilustrando a integração entre os componentes responsáveis pelo fornecimento de energia à *Raspberry Pi Zero 2W*. O diagrama evidencia o fluxo de energia desde o painel solar fotovoltaico de 50 Wp até o conversor DC-DC responsável por reduzir a tensão de 12 V para 5 V, necessária para a operação da unidade de processamento. Essa configuração assegura o funcionamento contínuo do sistema, garantindo estabilidade elétrica e eficiência energética durante as atividades de monitoramento em campo.

Figura 35 – Diagrma de conexão e ligação.

Fonte: Do autor.

Foram realizadas algumas análises do desempenho do sistema em diferentes cenários de operação, considerando as variações nas condições climáticas e sua influência direta na geração e consumo de energia.

Cenário 1 - Dia Ensolarado (4,5 *HSP*):

- Energia gerada por painel 50 Wp: 225 Wh/dia
- Energia consumida: 40,97 Wh/dia
- Excedente para bateria: 184,03 Wh/dia
- Tempo para carga completa: ∼1,5 dias

Cenário 2 - Dia Parcialmente Nublado (2,5 *HSP*):

- Energia gerada: 125 Wh/dia
- Energia consumida: 40,97 Wh/dia
- Excedente para bateria: 84,03 Wh/dia

Cenário 3 - Período Nublado (0 *HSP*):

- Operação exclusiva por bateria
- Autonomia: 5 dias completos
- Após 5 dias, bateria atinge 50% de carga (*DoD*)

Eficiência Global do Sistema:

- Eficiência do controlador de carga: 85%
- Eficiência do conversor *DC-DC*: 92%
- Eficiência global: 78,2%
- Perdas esperadas: 21,8%

4.3.2 Monitoramento energético com intervalo de amostras de 10 segundos

O presente teste teve como objetivo caracterizar o consumo energético de um sistema embarcado baseado em *Raspberry Pi Zero 2W* operando em regime de inferência sequencial com amostragem periódica.

O sistema foi configurado para executar inferências com os seguintes parâmetros:

- Intervalo entre inferências: 8 segundos
- Taxa de amostragem: 10 segundos
- Duração total do teste: 67 horas (aproximadamente 2,8 dias)

Durante o teste, a placa permaneceu em operação ininterrupta, simulando condições reais de *deployment* em campo para a aplicação de contagem de abelhas. Os resultados são apresentados na Tabela 6.

Tabela 6 – Resultados com taxa de amostragem de 10 segundos.

Parâmetro	**Valor Medido**
Tensão de Alimentação	5V
Consumo Total	10.453 mAh
Energia Total Consumida	52 Wh
Corrente Média (calculada)	156 mA
Potência Média (calculada)	0,805 W
Consumo Diário (calculado)	18,6 Wh/dia

Fonte: Do autor.

Também foi realizada uma análise mais detalhada do consumo energético do sistema observando a potência instantânea durante o período de inferência e taxa de amostragem de 0,01s. As medições foram realizadas no medidor de consumo *WITRN-U3*. A Figura 36 representa o comportamento do sistema para as condições especificadas.

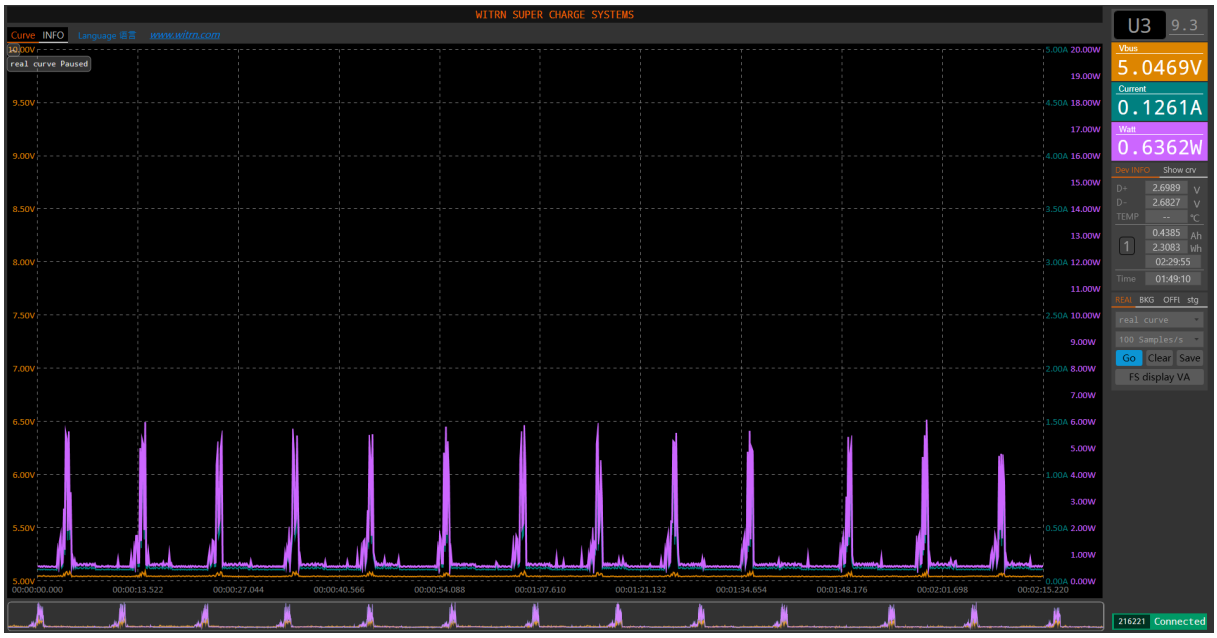

Figura 36 – Potência instantânea do sistema com taxa de amostragem de 0,01s.

Fonte: Do autor.

A partir do medidor foi possível exportar uma planilha no formato CSV com os dados de potência instantânea e calcular o consumo energético durante um período de amostragem de 10s.

$$E = \int_{t_0}^{t_f} P(t)\, dt \tag{4.4}$$

Onde:

- E = Energia total consumida (*Joules* ou *Wh*);
- $P(t)$ = *Potência* instantânea em função do tempo (*Watts*);
- t_0 = Tempo inicial;
- t_f = Tempo final.

Na forma discreta, quando realizamos medições em intervalos de tempo ΔT, podemos expressar o cálculo da energia consumida como:

$$E = \sum_{i=0}^{n-1} P(t_i) \cdot \Delta T \tag{4.5}$$

Ou de forma mais explícita:

$$E = P(t_0) \cdot \Delta T + P(t_1) \cdot \Delta T + P(t_2) \cdot \Delta T + \cdots + P(t_{n-1}) \cdot \Delta T \tag{4.6}$$

$$E = [P(t_0) + P(t_1) + P(t_2) + \cdots + P(t_{n-1})] \cdot \Delta T \tag{4.7}$$

Onde:

- $P(t_i)$ = Potência medida no instante i;
- ΔT = Intervalo de tempo entre medições (segundos);
- n = Número total de amostras.

Assim, a Tabela 7 apresenta os resultados do consumo energético encontrados para duas amostras de 10s, considerando o tempo da inferência, tempo de descanso e tempo total de um ciclo (10s).

Tabela 7 – Consumo energético durante os períodos de inferência e descanso.

Inferência		**Descanso**		**Total – 10s**
mWh	**% total**	**mWh**	**% total**	**mWh**
1,03	43,6 %	1,33	56,4 %	2,36 mWh
1,06	44,9 %	1,30	55,1 %	2,36 mWh

Fonte: Do autor.

Para fazer uma checagem dupla do gasto energético observou-se inicialmente que foram gastos 52 Wh para um período de teste de 67 horas. Por outro lado, cada 10s consumiu cerca de 2,36 mWh. Partindo dos dados de consumo de cada ciclo de amostragem, temos:

$$E = \frac{(67) \times (3600) \times (2,36)}{(1000) \times (10)}$$

$$E = 56,9\ Wh$$

Este resultado representa cerca de 5% de diferença entre a duas metodologias de cálculo o que para a aplicação é aceitável.

Para o dimensionamento do painel fotovoltaico foram consideradas as seguintes premissas:

- Autonomia requerida: 5 dias sem insolação (dias nublados/chuvosos)
- Local de instalação: Brasil (insolação média de 4,5 horas úteis/dia)
- Temperatura de operação: 15°C a 35°C
- Eficiência global do sistema: 75% (considerando perdas no controlador *MPPT*, cabeamento e conversores)

Como margem de segurança, acrescentou-se 20% ao consumo diário estimado, a fim de compensar eventuais variações de carga e perdas no sistema, chegando ao total de 22,3 Wh.

Dimensionamento do painel solar

$$\text{Potência do Painel} = \frac{\text{Consumo Diário}}{\text{Horas de Sol} \times \text{Eficiência}} \tag{4.8}$$

Cálculo:

$$P_{pv} = \frac{22,3 \text{ Wh}}{(4,5 \text{ h} \times 0,75)}$$
$$P_{pv} = \frac{22,3}{3,375}$$
$$P_{pv} = 6,6 \text{ Wp (Watts-pico)}$$

Painel recomendado: 10 a 20 Wp

Justificativa para o sobre-dimensionamento:

- Compensação de dias parcialmente nublados
- Degradação anual do painel (∼0,7% ao ano)
- Variação sazonal de insolação
- Possível sujeira/poeira na superfície
- Ângulo de instalação não-ótimo

A Tabela 8 apresenta o dimensionamento do banco de baterias.

Tabela 8 – Dimensionamento do banco de baterias.

Parâmetro	**Valor**
Energia Necessária (5 dias)	111,5 Wh
Profundidade de Descarga (*DoD* 50%)	÷ 0,50
Energia Nominal Necessária	223 Wh

Fonte: Do autor.

Convertendo para capacidade em Ampere-hora:

Para sistema 12 V:

$$C_{\text{bat}} = \frac{223 \text{ Wh}}{12 \text{ V}} = 18,6 \text{ Ah}$$

Bateria recomendada: *LiFePO4* 12V, 18,6Ah, composto por 48 células na configuração 4S12P de baterias 18650 LiFePO (3,2V, 1500mAh cada).

A Tabela 9 apresenta as especificações técnicas do controlador de carga utilizado no sistema, componente essencial para a gestão do fluxo de energia entre o painel fotovoltaico e o banco de baterias, assegurando o carregamento eficiente e a proteção contra sobrecarga e descarga profunda.

Tabela 9 – Especificação do controlador de carga.

Parâmetro	**Especificação**
Tipo	*MPPT (Maximum Power Point Tracking)* ou *PWM*
Corrente	Mínimo 3 A (para painel de 20 W)
Tensão	12 V

Fonte: Do autor.

Proteções necessárias:

- Sobrecarga
- Descarga profunda
- Curto-circuito
- Inversão de polaridade
- Sobre-temperatura

Para garantir o funcionamento autônomo e a confiabilidade operacional do sistema de monitoramento, foram selecionados componentes adequados às exigências de baixo consumo energético e operação contínua em campo. Os equipamentos recomendados para o monitoramento contínuo do sistema estão descritos na Tabela 10, incluindo o conjunto de dispositivos responsáveis pela aquisição de energia, armazenamento, controle de carga e alimentação do módulo de processamento. Essa configuração assegura estabilidade na coleta de dados, eficiência no carregamento da bateria e capacidade de operação prolongada mesmo em condições de baixa insolação.

Tabela 10 – Componentes recomendados para o sistema de monitoramento autônomo.

Componente	**Especificação**
Painel Solar	20 W monocristalino, 12 V
Bateria	*LiFePO4*, 12 V
Controlador	*MPPT* 12 V / 3 A
Conversor *DC-DC*	*Step-down* 12 V $\rightarrow$ 5 V, 3 A (regulador *buck*)
Autonomia Real	6–7 dias (*DoD* 50%)

Fonte: Do autor.

5 Conclusões e trabalhos futuros

O desenvolvimento deste sistema de monitoramento não invasivo de colmeias demonstrou que o uso de técnicas de aprendizado de máquina na borda não é apenas viável, mas também altamente eficaz para aplicações na apicultura de precisão. Esses resultados evidenciam que o modelo de detecção de abelhas desenvolvido apresenta elevada confiabilidade e precisão, capaz de reproduzir com fidelidade os resultados obtidos por contagem manual, mesmo em condições desafiadoras, como oclusão parcial, sobreposição e variação da iluminação. Trabalhos recentes [36] corroboram a eficiência de abordagens baseadas em aprendizado profundo (*deep learning*) para a análise comportamental e contagem automatizada de abelhas na entrada das colmeias.

A validação estatística, realizada por meio da correlação de *Pearson* e da análise de *Bland-Altman*, confirmou a robustez e a aplicabilidade prática do sistema proposto. Os resultados destacam o potencial da abordagem para aplicações na apicultura de precisão e em programas de monitoramento ecológico não invasivo, reduzindo o tempo de observação manual e minimizando interferências nas colônias. Além disso, a atuação do *watchdog* foi fundamental para a robustez do sistema, pois evitou travamentos que poderiam comprometer o monitoramento contínuo, garantindo a confiabilidade operacional do protótipo em campo.

Em relação ao consumo energético do sistema operando com amostragem de 10s, a análise revelou uma característica crítica: aproximadamente 50% do consumo total ocorre quando a *Raspberry Pi Zero 2W* encontra-se em estado de repouso. Este comportamento evidencia uma ineficiência significativa para aplicações em *Edge AI*, onde a autonomia energética é um requisito fundamental.

Embora seja esperado um consumo elevado durante os ciclos de inferência, o consumo observado durante o modo de repouso ou *sleep* apresentou-se em patamares inadequados para um sistema embarcado destinado à operação contínua em campo. Esta limitação torna-se ainda mais evidente quando se observa o dimensionamento necessário do sistema de energia: um *pack* de baterias de 12V, 18,6Ah, composto por 48 células na configuração 4S12P de baterias 18650 LiFePO (3,2V, 1500mAh cada).

Tais dimensões representam um desafio significativo para a viabilidade prática do sistema, indicando a necessidade de estratégias mais robustas de gerenciamento de energia, como a implementação de modos de *deep sleep* efetivos ou a exploração de plataformas de *hardware* alternativas com melhor eficiência energética em estado de repouso.

Mais do que comprovar o desempenho de um modelo de detecção, esta pesquisa contribui para uma nova perspectiva sobre o monitoramento de polinizadores: demonstra

que é possível realizar inferências complexas em dispositivos de baixo custo e baixa potência, abrindo caminho para redes distribuídas de monitoramento ambiental autônomo. Essa abordagem aproxima a inteligência artificial da borda (*Edge AI*) de problemas reais da biologia e da ecologia, promovendo uma integração concreta entre engenharia e ciências ambientais.

Contudo, como toda inovação, o sistema ainda apresenta limitações. A ocorrência de falsos positivos e falsos negativos em cenários de alta densidade de abelhas, somada às restrições de processamento inerentes à plataforma *Raspberry Pi Zero 2W*, sinaliza oportunidades de aprimoramento. A detecção pode ser refinada por meio da incorporação de algoritmos de rastreamento multiobjeto (*tracking*), calibração adaptativa de limiares de confiança e uso de técnicas de *transfer learning* com conjuntos de dados mais amplos e diversificados.

Como perspectivas para trabalhos futuros, propõe-se:

- Otimização energética do sistema, explorando modelos de gerenciamento dinâmico de energia e integração com fontes renováveis (como painéis fotovoltaicos), visando operação totalmente autônoma.
- A realização de ajustes e otimizações no sistema operacional da *Raspberry Pi*, visando aprimorar o desempenho geral da aplicação. A adoção de uma distribuição mais enxuta e personalizada, com a remoção de serviços desnecessários e ajustes do *kernel* voltados à inferência em tempo real, pode reduzir o tempo de inicialização, o consumo de memória e a latência do sistema.
- Explorar o uso de uma taxa de amostragem mais alta durante o processo de inferência, a fim de obter maior resolução temporal dos eventos observados. Essa modificação permitirá ampliar a capacidade do sistema de fornecer informações detalhadas sobre a dinâmica das colônias.
- Ampliação da arquitetura para permitir comunicação entre múltiplas colmeias via rede *LoRa* ou 5G, possibilitando o monitoramento remoto em larga escala.
- Integração com análise comportamental avançada, utilizando séries temporais e aprendizado não supervisionado para identificar padrões anômalos de forrageamento ou colapso de colônias.
- Desenvolvimento de módulos multimodais, que incorporam áudio e vibração como fontes complementares de informação sobre o estado da colmeia.

Assim, este trabalho não encerra uma investigação, mas inaugura uma nova linha de pesquisa em *Edge Machine Learning* aplicada à conservação de polinizadores. A principal pergunta que emerge desta dissertação é: como a inteligência artificial embarcada

pode evoluir para se tornar uma aliada efetiva na proteção e na sustentabilidade das populações de abelhas em escala global?

Essa questão representa o próximo passo — o ponto de partida para futuras colaborações e avanços científicos que poderão transformar o monitoramento ambiental em um instrumento ainda mais inteligente, acessível e sustentável.

Apêndices

APÊNDICE A – Código implementado

Listing A.1 – Script de instalação do sistema do arquivo setup.h

```
#!/bin/bash
echo "Atualizando pacotes..."
sudo apt-get update && sudo apt-get upgrade -y && sudo apt install git -y


echo "Aumentando memória SWAP..."
sudo dphys-swapfile swapoff
sudo sed -i 's/^CONF_SWAPSIZE=.*/CONF_SWAPSIZE=2000/' /etc/dphys-swapfile
sudo dphys-swapfile setup
sudo dphys-swapfile swapon

echo "Configurando c mera OV5647 e otimizando consumo de energia..."
read -p "Pressione [ENTER] após terminar a edição manual..."
sudo nano /boot/firmware/config.txt
echo "    Edição concluída. Continuando o script..."

echo "Instalando pip..."
sudo apt install -y python3-pip
sudo rm -f /usr/lib/python3.11/EXTERNALLY-MANAGED

echo "Criando ambiente virtual em ~/bee..."
python3 -m venv ~/bee
source ~/bee/bin/activate
echo "    Ambiente virtual criado."

echo "Criando estrutura de pastas..."
mkdir -p ~/Documents/Projeto
cd ~/Documents/Projeto

echo "Copie manualmente a pasta Backup para dentro de ~/Documents/Projeto"
echo "Depois pressione ENTER para continuar..."
read
cd ~/Documents/Projeto/Backup
echo "    Estrutura de pastas criada."

echo "Instalando depend ncias do projeto..."
pip install -r requirements.txt
sudo apt install -y libgl1
```

```
sudo apt install -y sqlite3
echo "    Depend ncias instaladas."

echo "Configurando I2C"
sudo apt-get install -y python3-smbus
sudo apt-get install -y i2c-tools
echo "Com a interface raspi-config, vá em "Interfacing Options" e ative o I2C."
read -p "Pressione [ENTER] após terminar a edição manual..."
sudo raspi-config
echo "    I2C configurado."

echo "Configurando serviço de inicialização..."
read -p "Pressione [ENTER] após terminar a edição manual..."
sudo nano /etc/systemd/system/bee.service
echo "    Edição concluída. Continuando o script..."

echo "Habilitando e iniciando o serviço..."
sudo systemctl daemon-reexec
sudo systemctl daemon-reload
sudo systemctl enable bee.service
sudo systemctl start bee.service

echo "    Configuração concluída. Por segurança, reinicie com: sudo reboot"
echo "Atenção: Após reiniciar, execute source ~/bee/bin/activate para ativar o ambiente virtual."
echo "E cd ~/Documents/Projeto/Backup para entrar na pasta do projeto."
echo "Obrigado por usar o script de configuração!"
```

Listing A.2 – Lista de pacotes versionados do arquivo requirements.txt

```
certifi==2025.4.26
charset-normalizer==3.4.2
coloredlogs==15.0.1
colorzero==2.0
contourpy==1.3.2
cycler==0.12.1
ds1302 @ git+https://github.com/guilhermefernandesk/RTC-DS1302-PY.git@317410c947c83d25ea25c9331133bfda4ba3610d
filelock==3.18.0
flatbuffers==20181003210633
fonttools==4.57.0
fsspec==2025.3.2
humanfriendly==10.0
idna==3.10
Jinja2==3.1.6
kiwisolver==1.4.8
```

```
lgpio==0.2.2.0
MarkupSafe==3.0.2
matplotlib==3.10.1
mpmath==1.3.0
ncnn==1.0.20250503
networkx==3.4.2
numpy==2.2.5
onnx==1.17.0
onnxruntime==1.21.1
onnxslim==0.1.51
opencv-python==4.11.0.86
packaging==25.0
pandas==2.2.3
pillow==11.2.1
portalocker==3.1.1
protobuf==6.30.2
psutil==7.0.0
py-cpuinfo==9.0.0
pyparsing==3.2.3
python-dateutil==2.9.0.post0
pytz==2025.2
PyYAML==6.0.2
requests==2.32.3
rpi-lgpio==0.6
RPi.GPIO==0.7.1
scipy==1.15.2
seaborn==0.13.2
sht20 @ git+https://github.com/guilhermefernandesk/SHT20-PY.
    git@810729a8af24e52052edd1686301b40a7b872667
six==1.17.0
smbus2==0.5.0
sympy==1.14.0
torch==2.7.0
torchvision==0.22.0
tqdm==4.67.1
typing_extensions==4.13.2
tzdata==2025.2
ultralytics==8.3.127
ultralytics-thop==2.0.14
urllib3==2.4.0
```

Listing A.3 – Configuração do sistema no arquivo /boot/firmware/config.txt

```
# === CONFIG OTIMIZADO PARA ECONOMIA DE ENERGIA ===

# Desativa HDMI completamente
hdmi_force_hotplug=0        # Não força saida HDMI se não estiver
    conectada
```

```
hdmi_blanking=2             # Desliga sinal HDMI (economia de energia)

# Desativa vídeo composto (analógico)
enable_tvout=0

# Desativa Wi-Fi, Bluetooth, Usb se não estiver usando
#dtoverlay=disable-wifi
dtoverlay=disable-bt
dtoverlay=disable-usb

# Desativa LEDs
dtparam=act_led_trigger=none
dtparam=act_led_activelow=on
```

Listing A.4 – Configuração do serviço Linux no arquivo bee.service

```
[Unit]
Description=Script Bee - controle de Wi-Fi e sensores
After=multi-user.target
StartLimitInterval=50
StartLimitBurst=3

[Service]
ExecStart=/home/gui/bee/bin/python /home/gui/Documents/Projeto/Backup/
    bee_wat.py
Restart=on-failure
RestartSec=5
User=gui
WorkingDirectory=/home/gui/Documents/Projeto/Backup
StandardOutput=journal
StandardError=journal
Environment=PYTHONUNBUFFERED=1

[Install]
WantedBy=multi-user.target
```

Listing A.5 – Script principal para inferência de abelhas

```
# === BIBLIOTECAS ===
from ultralytics import YOLO
from ds1302 import DS1302
from sht20 import SHT20
import sqlite3
import subprocess
import os
import time
import threading
import RPi.GPIO as GPIO
```

```
11
12 # === CONFIGURAÇÕES ===
13 MODEL_PATH = './bee_best_ncnn_model'
14 CAPTURE_DIR = 'capture/'
15 PROCESSED_DIR = "processed_images/"
16 DB_PATH = 'bee_count.db'
17 BATCH_SIZE = 5
18 CAPTURE_RAM = True
19 SAVE_IMAGE = False
20 WATCHDOG_TIMEOUT = 50  # segundos
21 CYCLE_TIME = 10  # segundos
22
23 BOTAO = 16  # GPIO16 -> Pino 36
24 LED = 19    # GPIO19 -> Pino 35
25
26 # === Configuracao GPIO para BOTAO e LED ===
27 GPIO.setmode(GPIO.BCM)
28 GPIO.setwarnings(False)
29 GPIO.setup(BOTAO, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
30 GPIO.setup(LED, GPIO.OUT)
31
32 event = threading.Event()
33
34 # Criar diretórios se não existir
35 os.makedirs(CAPTURE_DIR, exist_ok=True)
36 os.makedirs(PROCESSED_DIR, exist_ok=True)
37
38 # Definição dos pinos GPIO
39 CLK_PIN = 4   # GPIO4 -> Pino 7
40 DAT_PIN = 5   # GPIO5 -> Pino 29
41 RST_PIN = 6   # GPIO6 -> Pino 31
42
43 # === INICIALIZACAO ===
44 rtc = DS1302(CLK_PIN, DAT_PIN, RST_PIN)
45 sht20 = SHT20(port=1, resolution=SHT20.TEMP_RES_11bit)
46
47 # Carregar modelo YOLO
48 model = YOLO(MODEL_PATH, task="detect")
49
50 # === FUNÇÕES ===
51 def setup_database():
52     '''Cria o banco de dados e a tabela, se não existirem.'''
53     conn = sqlite3.connect(DB_PATH)
54     cursor = conn.cursor()
55     cursor.execute('''
56         CREATE TABLE IF NOT EXISTS bee_counts (
57             id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
58          timestamp TEXT,
59          count INTEGER,
60          inference_time TEXT,
61          temperature TEXT,
62          humidity TEXT
63      )
64  ''')
65  conn.commit()
66  conn.close()
67 def insert_batch(batch):
68  '''Insere a inferencia no banco de dados.'''
69  conn = None
70  try:
71      conn = sqlite3.connect(DB_PATH)
72      cursor = conn.cursor()
73      cursor.executemany('''
74            INSERT INTO bee_counts (
75                timestamp, count, inference_time, temperature, humidity
76            ) VALUES (?, ?, ?, ?, ?)
77        ''', batch)
78      conn.commit()
79  except sqlite3.Error as e:
80      print(f"[ERRO][DB] Falha ao inserir batch: {e}")
81  finally:
82      if conn:
83          conn.close()
84 def getTemperatureHumidity():
85  temp = sht20.read_temp()
86  humid = sht20.read_humid()
87  if temp != sht20.ERROR and humid != sht20.ERROR:
88      return temp, humid
89  else:
90   print("Erro na leitura do sensor.")
91 def capture(ram=False):
92  """Captura uma imagem usando a c mera"""
93  timestamp = rtc.getDateTime("file")
94  if ram:
95      image_path = f"/dev/shm/{timestamp}.jpg"
96  else:
97      image_path = f"{CAPTURE_DIR}{timestamp}.jpg"
98  subprocess.run([
99          "rpicam-jpeg",       # Programa chamado (captura imagem JPEG
                da c mera)
100         "-o", image_path,    # Arquivo de saída (salva a imagem no
               caminho definido por `image_path`)
101         "--width", "640",    # Largura da imagem
102         "--height", "640",   # Altura da imagem
```

```
103             "--quality", "85",    # Qualidade JPEG (0 a 100)
104             "--nopreview",        # Desativa visualização ao vivo
105             "-n",                 # Sem interface de preview (não abre
                    janela)
106             "--timeout", "1"      # Espera 1 ms antes de capturar (na prá
                    tica, captura imediata)
107             ],
108             stdout=subprocess.DEVNULL,    # Ignora saída padrão (oculta
                    prints do terminal)
109             stderr=subprocess.DEVNULL,    # Ignora erros (não imprime no
                     terminal)
110             check=True                    # Lança erro (Exception) se o
                    comando falhar
111             )
112     return image_path
113 def process_image(full_path, save=False):
114     '''realiza inferencia'''
115     result = model.predict(
116         full_path,
117         save=save,
118         imgsz=640,
119         conf=0.3,
120         iou=0.3,
121         show_labels=False,
122         show_boxes=True,
123         verbose=False,
124         project=".",
125         name=PROCESSED_DIR,
126         exist_ok=True
127         )
128     # save the results
129     num_bees = len(result[0].boxes.cls)
130     timestamp = rtc.getDateTime("file")
131     inference_time = int(result[0].speed['inference'])
132     return inference_time, num_bees, timestamp
133 def clearRamImage(image_path):
134     '''Limpa a imagem da RAM.'''
135     if os.path.exists(image_path):
136         os.remove(image_path)
137 def cleanup():
138     '''Limpa os pinos GPIO'''
139     print("Limpando GPIO...")
140     rtc.cleanupGPIO()
141 def watchdog():
142     while True:
143         signaled = event.wait(WATCHDOG_TIMEOUT)
144         if signaled:
```

```
145             event.clear()
146         else:
147             print("Watchdog: processamento travou. Reiniciando processo.")
148             cleanup()
149             os._exit(1)
150 def atualiza_led():
151     GPIO.output(LED, wifi_ativo())
152 def wifi_ativo():
153     # Executa o comando `rfkill list wifi` e captura a saída
154     resultado = subprocess.run(["rfkill", "list", "wifi"],
            capture_output=True, text=True)
155     # Procura a linha com "Soft blocked"
156     for linha in resultado.stdout.splitlines():
157         if "Soft blocked" in linha:
158             status = linha.split(":")[-1].strip()
159             return status == "no"  #WIFI está ativo = LED acesso
160     return False #WIFI está desativado = LED apagado
161 def desliga_wifi():
162     subprocess.run(["sudo", "rfkill", "block", "wifi"])
163 def liga_wifi():
164     subprocess.run(["sudo", "rfkill", "unblock", "wifi"])
165 def toggle_wifi(channel):
166     if wifi_ativo():
167         desliga_wifi()
168     else:
169         liga_wifi()
170     atualiza_led()
171 def pisca_led():
172       """Pisca o LED para sinalização."""
173       GPIO.output(LED, GPIO.HIGH)
174       time.sleep(0.1)
175       GPIO.output(LED, GPIO.LOW)
176       time.sleep(0.1)
177       atualiza_led()
178
179 # Configura interrupção
180 GPIO.add_event_detect(BOTAO, GPIO.RISING, callback=toggle_wifi,
    bouncetime=300)
181
182 atualiza_led()
183
184 # === LOOP PRINCIPAL ===
185 def main():
186     try:
187         setup_database()
188         batch = []
189
```

```
190         while True:
191             print("Começando...")
192             t_start = time.perf_counter()
193 
194             # 1. Captura
195             image_path = capture(ram=CAPTURE_RAM)
196 
197             # 2. Processa a imagem
198             inference_time, num_bees, timestamp = process_image(image_path,
                    save=SAVE_IMAGE)
199             print(f"Inference time {inference_time}")
200             print(f"Bee count: {num_bees}")
201 
202             # 3. L  temperatura e umidade
203             temp, humid = getTemperatureHumidity()
204             print(f"Temperature: {temp:.2f} C ")
205             print(f"Humidity: {humid:.2f}%")
206 
207             # 4. Adiciona os dados ao batch
208             batch.append((timestamp, num_bees, inference_time, f"{temp:.2f}
                    C ", f"{humid:.2f}%"))
209             if len(batch) >= BATCH_SIZE:
210                 insert_batch(batch)
211                 batch.clear()
212 
213             # 5. Apaga a imagem da RAM se necessário
214             if CAPTURE_RAM:
215                 clearRamImage(image_path)
216 
217             # 6. Alimenta o watchdog
218             event.set()
219 
220             # 7. Próxima captura
221             print("\n================================")
222             t_end = time.perf_counter()
223             duracao = t_end - t_start
224             print(f"Processamento concluído em {round(duracao, 3)}s")
225 
226             # 8. Aguarda tempo restante (se houver)
227             if duracao < CYCLE_TIME:
228                 delay = CYCLE_TIME - duracao
229                 print(f"Aguardando {round(delay, 3)}s para completar o ciclo
                        de {CYCLE_TIME}s...")
230                 time.sleep(delay)  # economiza energia colocando CPU em idle
231 
232             pisca_led()
233 
```

```
234     # Permite interromper o programa com Ctrl+C
235     except KeyboardInterrupt:
236         print("Programa interrompido pelo usuário.")
237         if batch:
238             insert_batch(batch)
239     except Exception as e:
240         print(f"Erro inesperado: {e}")
241     finally:
242         cleanup()
243         print("Limpeza concluída. Encerrando o programa.")
244 
245 if __name__ == "__main__":
246     t1 = threading.Thread(target=watchdog, daemon=True)
247     t2 = threading.Thread(target=main)
248 
249     t1.start()
250     t2.start()
251 
252     t2.join()
```

APÊNDICE B – Contagem manual

Tabela 11 – Resultados da validação manual e automática do sistema de detecção.

Nome arquivo	Modelo	Manual	Falso positivo	Falso negativo
2025-08-23_14-37-21	14	13	2	1
2025-08-23_14-39-15	17	13	3	0
2025-08-23_14-41-08	17	10	5	0
2025-08-23_14-43-01	26	26	2	1
2025-08-23_14-44-53	15	14	2	0
2025-08-23_14-46-46	16	16	3	0
2025-08-23_14-48-38	11	11	2	2
2025-08-23_14-50-31	9	9	3	1
2025-08-23_14-52-23	16	15	3	0
2025-08-23_14-54-15	16	12	3	0
2025-08-23_14-56-08	21	18	3	0
2025-08-23_14-58-00	29	25	3	2
2025-08-23_14-59-53	25	22	3	1
2025-08-23_15-01-45	31	26	5	1
2025-08-23_15-03-38	25	22	3	1
2025-08-23_15-07-23	30	26	4	1
2025-08-23_15-09-16	26	26	4	1
2025-08-23_15-15-18	36	31	4	1
2025-08-23_15-17-12	29	26	3	0
2025-08-23_15-19-05	27	22	2	1
2025-08-23_15-20-58	36	33	3	1
2025-08-23_15-22-51	39	35	3	0
2025-08-23_15-24-43	30	27	4	0
2025-08-23_15-26-36	31	26	5	0
2025-08-23_15-28-29	32	31	3	1
2025-08-23_15-30-22	30	26	2	0
2025-08-23_15-32-14	32	26	5	3
2025-08-23_15-36-00	43	39	4	1
2025-08-23_15-41-38	33	31	3	1
2025-08-23_15-45-24	36	34	4	0

Continua na próxima página

Tabela 11 – Continuação da página anterior

Nome arquivo	**Modelo**	**Manual**	**Falso positivo**	**Falso negativo**
2025-08-23_15-47-17	31	25	4	0
2025-08-23_15-49-09	23	19	3	1
2025-08-23_15-51-02	19	15	4	0
2025-08-23_15-52-55	33	29	4	0
2025-08-23_15-56-41	31	27	4	0
2025-08-23_15-58-34	34	30	5	0
2025-08-23_16-00-27	29	24	3	1
2025-08-23_16-03-44	31	24	9	1
2025-08-23_16-08-32	12	10	2	1
2025-08-23_16-12-18	26	25	3	2
2025-08-23_16-16-03	25	20	2	0
2025-08-23_16-21-11	21	19	3	2
2025-08-23_16-24-00	18	16	2	2
2025-08-23_16-26-49	19	17	2	0
2025-08-23_16-29-54	32	26	6	0
2025-08-23_16-33-29	29	25	8	1
2025-08-23_16-37-15	32	26	7	2
2025-08-23_16-41-44	28	24	7	0
2025-08-23_16-44-08	22	16	6	0
2025-08-23_16-46-23	29	24	7	1

Fonte: Do autor.

Referências

1 KHALIFA, S. A. M.; OUTROS. Overview of bee pollination and its economic value for crop production. *Insects*, v. 12, n. 8, p. 688, jul. 2021. 14, 16

2 VISTA do Panorama econômico da produção e exportação de mel de abelha produzidos no Brasil. s.d. 14, 15, 16

3 RODRIGUES, R. C.; TéCNICO, C. E. *Produção de Mel*. [S.l.], 2002. Disponível em: <http://www.foxitsoftware.comForevaluationonly>. Acesso em: s.d. 14, 15, 16

4 ASTUTI, P. K. et al. Climate change and dairy farming sustainability; a causal loop paradox and its mitigation scenario. *Heliyon*, v. 10, n. 3, p. e25200, fev. 2024. 15

5 TABOR, J. A.; KOCH, J. B. Ensemble models predict invasive bee habitat suitability will expand under future climate scenarios in hawai'i. *Insects*, v. 12, n. 5, p. 443, maio 2021. 15

6 MARTíNEZ-LóPEZ, O. et al. Reduction in the potential distribution of bumble bees (apidae: Bombus) in mesoamerica under different climate change scenarios: Conservation implications. *Glob Chang Biol*, v. 27, n. 9, p. 1772–1787, maio 2021. 15

7 LANNER, J.; OUTROS. On the road: Anthropogenic factors drive the invasion risk of a wild solitary bee species. *Science of The Total Environment*, v. 827, p. 154246, jun. 2022. 15

8 LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, v. 521, n. 7553, p. 436–444, maio 2015. 19

9 FLECK, L. et al. Redes neurais artificiais: princípios básicos. *Revista Eletrônica Científica Inovação e Tecnologia*, v. 7, n. 15, p. 47–57, mar. 2016. [Online]. Disponível em: <https://periodicos.utfpr.edu.br/recit/article/view/4330>. 19

10 FERNEDA, E. Redes neurais e sua aplicação em sistemas de recuperação de informação. *Ciência da Informação*, v. 35, n. 1, p. 25–30, abr. 2006. 19

11 HAN, S.; MAO, H.; DALLY, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2016. [Online]. Disponível em: <https://arxiv.org/abs/1510.00149>. 20

12 HOWARD, A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. [Online]. Disponível em: <https://arxiv.org/abs/1704.04861>. 20

13 LI, Z. et al. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, v. 33, n. 12, p. 6999–7019, dez. 2022. 20

14 HONG, J. et al. Design of power-efficient training accelerator for convolution neural networks. *Electronics*, v. 10, n. 7, p. 787, mar. 2021. 20

15 HONORATO, D. de M. L. B. VisÃo computacional. *UNICAMP – Universidade Estadual de Campinas FT – Faculdade de Tecnologia*, v. 10, n. 7, p. 787, mar. 2010. 22

16 MARENGONI, M.; STRINGHINI, S. Tutorial: Introdução à visão computacional usando opencv. *Revista de Informática Teórica e Aplicada*, v. 16, n. 1, p. 125–160, mar. 2010. 22

17 TEFILI, D. et al. Supervisão de hábitos alimentares em grupos de animais com visão computacional em raspberry pi. *Brazilian Journal of Development*, v. 10, n. 5, p. e69849, maio 2024. 22

18 GONçALVES, M. et al. Protótipo de solução para detetar e sinalizar defeitos em pavimentos rodoviários baseado em técnicas de visão computacional. *Revista Ibérica de Sistemas e Tecnologias de Informação (RISTI)*, v. 52, p. 25–44, 2024. 22

19 REDMON, J. et al. You only look once: Unified, real-time object detection. In: *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*. [S.l.: s.n.], 2016. p. 779–788. 22, 24, 25

20 MECOCCI, A.; GRASSI, C. Rtaiaed: A real-time ambulance in an emergency detector with a pyramidal part-based model composed of mfccs and yolov8. *Sensors*, v. 24, n. 7, abr. 2024. 23

21 LIN, T. Y. et al. Microsoft coco: Common objects in context. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 8693, n. PART 5, p. 740–755, 2014. 26

22 ULTRALYTICS. *GitHub - ultralytics/ultralytics: Ultralytics YOLO* . 2025. [Online]. Disponível em: <https://github.com/ultralytics/ultralytics?tab=readme-ov-file>. 26

23 MA, N. et al. Wheat seed detection and counting method based on improved yolov8 model. *Sensors*, v. 24, n. 5, mar. 2024. 27

24 WANG, H. et al. A remote sensing image target detection algorithm based on improved yolov8. *Applied Sciences*, v. 14, n. 4, p. 1557, fev. 2024. 29

25 ULTRALYTICS. *Integração Roboflow - Documentação Ultralytics YOLO*. 2025. [Online]. Disponível em: <https://docs.ultralytics.com/pt/integrations/roboflow/#how-can-i-integrate-and-deploy-yolo11-models-with-roboflow>. 33

26 CHOWDHURY, M. F. S. et al. Deep-learning-based real-time visual pollution detection in urban and textile environments. *Sci*, v. 6, n. 1, p. 5, jan. 2024. 33

27 SHARMA, T. et al. Deep learning-based object detection and scene perception under bad weather conditions. *Electronics (Switzerland)*, v. 11, n. 4, p. 563, fev. 2022. 33

28 DWYER, B. et al. *Roboflow (Version 1.0)*. 2025. [Online]. Disponível em: <https://roboflow.com>. 34

29 LáZARO, O. et al. *Detecção de objetos em tempo real com a plataforma Roboflow*. 2025. [Trabalho técnico]. CEFET-MG. 34

30 XIE, S.; SUN, H. Tea-yolov8s: A tea bud detection model based on deep learning and computer vision. *Sensors*, v. 23, n. 14, p. 6576, jul. 2023. 37

31 POURESKANDAR, R.; RAZZAGZADEH, S. *Improving Object Detection Performance through YOLOv8: A Comprehensive Training and Evaluation Study*. 2025. 38

32 GAMANI, A.-R. A.; ARHIN, I.; ASAMOAH, A. K. Performance evaluation of yolov8 model configurations for instance segmentation of strawberry fruit development stages in an open field environment. 2024. 38

33 M. J. Rovai. *Edge AI Engineering*. 2025. [Online]. Acessado em: 7 de outubro de 2025. Disponível em: <https://mjrovai.github.io/EdgeML_Made_Ease_ebook/>. 43

34 COMO Exportar para NCNN do YOLO11 para Implantação Suave. 2025. [Online]. Acessado em: 7 de outubro de 2025. Disponível em: <https://docs.ultralytics.com/pt/integrations/ncnn/#why-should-you-export-to-ncnn>. 43

35 BLAND, J. M.; ALTMAN, D. G. Statistical methods for assessing agreement between two methods of clinical measurement. *Lancet*, v. 1, n. 8476, p. 307–310, fev. 1986. 52, 53

36 SLEDEVIč, T. et al. Visual recognition of honeybee behavior patterns at the hive entrance. *PLoS One*, v. 20, n. 2, fev. 2025. 53, 64