INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO AMAZONAS
CAMPUS MANAUS DISTRITO INDUSTRIAL
CURSO DE TECNOLOGIA EM ELETRÔNICA INDUSTRIAL

PAULO SÉRGIO LIMA FERNANDES

SISTEMA DE DETECÇÃO DE TOMATES (SDT): UMA FERRAMENTA DE GESTÃO PARA AGRICULTORES NAS FASES DE MATURAÇÃO.

MANAUS-AM
2025

PAULO SÉRGIO LIMA FERNANDES

SISTEMA DE DETECÇÃO DE TOMATES (SDT): UMA FERRAMENTA DE GESTÃO PARA AGRICULTORES NAS FASES DE MATURAÇÃO.

Trabalho de Conclusão de Curso aprovado como requisito para obtenção do título de Tecnólogo no curso de Tecnologia em Eletrônica Industrial do Instituto Federal do Amazonas - IFAM.

Orientador: Me. Alexandre Lopes Martiniano

MANAUS-AM

2025

Dados Internacionais de Catalogação na Publicação (CIP)

Fernandes, Paulo Sérgio Lima.

F363s Sistema de detecção de Tomates (SDT): uma ferramenta de Gestão para agricultores nas fases de maturação / Paulo Sérgio Lima Fernandes. — Manaus, 2025.

47f.: il. color.

Monografia (Graduação) — Instituto Federal de Educação, Ciência e Tecnologia do Amazonas, *Campus* Manaus Distrito Industrial, Curso de Tecnologia em Eletrônica Industrial, 2025.

Orientador: Prof.º Alexandre Lopes Martiniano, Me.

1. Inteligência artificial. 2. Visão computacional. 3. YOLOv8. I. Martiniano, Alexandre Lopes. II. Instituto Federal de Educação, Ciência e Tecnologia do Amazonas. III. Título.

CDD 629.892

Elaborada por Oziane Romualdo de Souza (CRB11/ nº 734).

ANEXO 7

ATA DE DEFESA PÚBLICA DO TRABALHO DE CONCLUSÃO DE CURSO

Aos 02 dias do mês de ABRIL, de 2025, às 17:00 h, o(a) discente PAULO SÉRGIO LIMA FERNANDES apresentou o seu Trabalho de Conclusão de Curso para avaliação da Banca Examinadora constituída pelos seguintes integrantes: Prof(a). Me. ALEXANDRE LOPES MARTINIANO (docente-orientador), Prof(a). Me. WENNDISSON DA SILVA SOUZA (Membro 1) e Prof(a). Me. EDUARDO PALHARES JÚNIOR (Membro 2). A sessão pública de defesa foi aberta pelo(a) presidente da banca, que apresentou a Banca Examinadora e deu continuidade aos trabalhos, fazendo uma breve referência ao TCC, que tem como título SDT - SISTEMA DE DETECÇÃO DE TOMATES: UMA FERRAMENTA DE GESTÃO PARA AGRICULTORES NAS FASES DE MATURAÇÃO. Na sequência, o(a) discente teve até 30 minutos para a comunicação oral de seu trabalho. Cada integrante da banca examinadora fez suas arguições após a defesa do mesmo. Ouvidas as explicações do(a) discente, a banca examinadora, reunida em caráter sigiloso, para proceder à avaliação final, deliberou e decidiu pela APROVAÇÃO com média final 9,5 (NOVE VIRGULA CINCO) do referido trabalho.

Foi dada ciência ao(à) discente que a versão final do trabalho deverá ser entregue até o dia 15/04/25, com as devidas alterações sugeridas pela banca. Nada mais havendo a tratar, a sessão foi encerrada às 17 h 45 min, sendo lavrada a presente ata, que, uma vez aprovada, foi assinada por todos os membros da Banca Examinadora e pelo(a) discente.

Prof.(a) Orientador(a)/Presidente: Martiniano

Prof.(a) Avaliador 1: Wenndisson da Silva Souza.

Prof.(a) Avaliador 2: Eduardo Palhares Jr.

Discente: Paulo Sérgio Lima Fernandes

À minha família, minha filha Lara Isabel, e minha esposa Layara Silva, por todo o apoio e incentivo, no decorrer do desenvolvimento desse trabalho eles foram essenciais durante esta jornada.

AGRADECIMENTOS

A realização deste trabalho não teria sido possível sem o apoio e dedicação de diversas pessoas, às quais expresso minha mais profunda gratidão.

Aos meus companheiros de jornada, Carlos Roberto e Emyli Beatriz, agradeço pela parceria, empenho e pelo compartilhamento de ideias ao longo do desenvolvimento deste projeto. A colaboração e o espírito de equipe foram fundamentais para superar desafios e alcançar os resultados obtidos.

Aos meus professores, que ao longo da minha trajetória acadêmica contribuíram para a construção do meu conhecimento, incentivando a busca pela inovação e aprimoramento constante. Suas orientações e ensinamentos foram essenciais para a realização deste trabalho.

À minha família, por todo o suporte incondicional, paciência e incentivo nos momentos mais difíceis. O apoio e a motivação que recebi foram essenciais para que eu pudesse me dedicar e concluir esta etapa tão importante da minha vida.

Ao meu orientador, Alexandre Martiniano, minha imensa gratidão pela dedicação, pelo direcionamento preciso e pelo incentivo ao desenvolvimento deste estudo. Sua orientação foi fundamental para a condução do projeto, proporcionando valiosas contribuições que enriqueceram cada etapa da pesquisa.

A todos que, direta ou indiretamente, contribuíram para a concretização deste trabalho, meu muito obrigado.

“A inovação distingue um líder de um seguidor." - Steve Jobs

RESUMO:

Este trabalho apresenta o desenvolvimento e a aplicação de um sistema de visão computacional para a detecção e classificação de tomates em diferentes estágios de maturação, utilizando o modelo YOLO v8. O sistema foi testado em uma horta montada, onde imagens e vídeos foram analisados para identificar tomates maduros, verdes e podres. Para facilitar a visualização e interpretação dos dados, foi implementado um dashboard estatístico interativo, permitindo o monitoramento da produção agrícola em tempo real. Além da detecção dos frutos, a aplicação foi integrada a um módulo de monitoramento climático, fornecendo informações sobre temperatura, umidade e precipitação para auxiliar na tomada de decisões. Os resultados indicam que o modelo apresentou bom desempenho na identificação de tomates maduros e podres, mas encontrou desafios na detecção precisa dos tomates verdes, devido à variação de iluminação e à menor representatividade dessa classe no dataset. A pesquisa reforça o potencial da inteligência artificial na agricultura, promovendo maior eficiência, automação e controle de qualidade na produção de alimentos.

Palavras-chave: Visão computacional; YOLO v8; Agricultura de precisão; Classificação de tomates; Inteligência Artificial.

ABSTRACT:

This study presents the development and application of a computer vision system for the detection and classification of tomatoes at different ripening stages using the YOLO v8 model. The system was tested in a controlled agricultural environment, where images and videos were analyzed to identify ripe, green, and rotten tomatoes. To facilitate data visualization and interpretation, an interactive statistical dashboard was implemented, enabling real-time monitoring of agricultural production. In addition to fruit detection, the system was integrated with a climate monitoring module, providing information on temperature, humidity, and precipitation to support decision-making processes. The results indicate that the model demonstrated good performance in identifying ripe and rotten tomatoes, but faced challenges in accurately detecting green tomatoes due to lighting variations and the lower representation of this class in the dataset. This research highlights the potential of artificial intelligence in agriculture, fostering greater efficiency, automation, and quality control in food production.

Keywords: Computer vision; YOLO v8; Precision agriculture; Tomato classification; Artificial Intelligence.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

CNN – Convolucional Neural Network

IA – Inteligência Artificial

INMET – Instituto Nacional de Meteorologia

MAP – Mean Average Precision

NLP – Natural Language Processing

PIB – Produto Interno Bruto

R-CNN – Region-Based Convolucional Neural Network

RESNET – Residual Neural Network

YOLO – You Only Look Once

SUMÁRIO

1. INTRODUÇÃO

A tecnologia tem sido fundamental para o desenvolvimento de soluções que facilitam o trabalho humano. Um exemplo desse impacto positivo é no campo da agricultura, onde se pode utilizar ferramentas de machine learning para a resolução de problemas simples, como a seleção de frutos e vegetais através de algoritmos que reconhecem padrões característicos em imagens. Segundo Kamilaris e Prenafeta-Boldú (2018), o uso dessas tecnologias tem potencializado a eficiência e a produtividade no campo, reduzindo o esforço manual e melhorando a qualidade dos produtos agrícolas.

Existem diversos tipos de modelos algorítmicos que reconhecem esses padrões. Esses modelos possuem arquiteturas compostas por sistemas de redes neurais artificiais, mais especificamente, redes neurais convolucionais. As redes neurais convolucionais (CNNs, do inglês Convolutional Neural Networks) são projetadas para tratar exclusivamente do reconhecimento de elementos e da classificação em imagens.

Isso é possível graças às extrações de características realizadas em camadas de convolução por meio de filtros, também conhecidos como kernels. Esses kernels percorrem uma imagem, que é representada como uma matriz de valores de pixels, em busca do objeto que foram designados para encontrar. Esse processo permite que as CNNs sejam altamente eficazes em tarefas de visão computacional, como a detecção de objetos.

A imagem passa por várias camadas convolucionais, onde diferentes filtros são aplicados para extrair características de níveis variados, desde bordas simples até formas complexas. Após várias convoluções e aplicações de kernels, a imagem é processada por uma camada chamada Max Pooling, que reduz a dimensionalidade da imagem, mantendo as características mais relevantes para a detecção de objetos. O Max Pooling é crucial, pois ajuda a tornar o modelo mais robusto e menos sensível a pequenas variações na posição do objeto dentro da imagem (Goodfellow et al., 2016).

A sequência de camadas convolucionais e de pooling resulta na criação de mapas de características que capturam as informações essenciais para a identificação do objeto. Esses mapas são então "achatados" em um vetor unidimensional, que é usado como entrada para camadas

totalmente conectadas (fully connected layers). Nessas camadas, a imagem é finalmente classificada com base nas características extraídas, determinando se o objeto pertence a uma das classes treinadas (LeCun et al., 1998).

Esse vetor serve como entrada para uma camada de rede neural densa, na qual são realizados diversos cálculos responsáveis por ajustar os pesos dos neurônios (ou perceptrons) e minimizar a margem de erro. Os cálculos de ajuste dos pesos são fundamentais para a capacidade do modelo de generalizar a partir dos dados de treinamento, permitindo que ele faça previsões precisas em novos dados.

O ajuste dos pesos é realizado durante o treinamento da rede, onde o modelo é exposto a um grande conjunto de imagens rotuladas, e o erro entre as previsões do modelo e as rotulações verdadeiras é minimizado. O processo de aprendizado ocorre por meio da minimização da função de erro, geralmente utilizando algoritmos como backpropagation e otimização por descida do gradiente (Bishop, 2006), permitindo que a rede neural ajuste seus pesos para melhorar sua capacidade preditiva.

Os cálculos de ajuste dos pesos têm um papel de suma importância, visto que, através deles, pode-se diminuir a margem de erro entre a camada de entrada dos dados e a base de dados na qual a rede neural foi treinada. Existem diversas técnicas para ajuste de pesos dos perceptrons, e essas técnicas influenciam diretamente na capacidade do modelo de aprender a partir dos dados.

O método mais básico e amplamente utilizado é a “Descida de Gradiente”. Neste método, busca-se encontrar o ponto mínimo de um gradiente ao longo de várias iterações do treinamento da rede. A ideia é tomar a direção oposta ao gradiente calculado para minimizar o erro. Durante esse processo, os pesos são atualizados iterativamente até que o modelo apresente uma adequada congruência com os dados do banco de dados de treinamento.

A Descida de Gradiente pode ser acelerada com o uso de variantes como a Descida de Gradiente Estocástica (SGD) e a Descida de Gradiente com Momentum, que visam melhorar a eficiência e a estabilidade do treinamento. Segundo Ruder (2016), "a Descida de Gradiente Estocástica é uma abordagem eficiente para grandes conjuntos de dados, pois atualiza os parâmetros do modelo após cada exemplo de treinamento, permitindo uma convergência mais

rápida em comparação com a descida de gradiente tradicional."

Outra técnica interessante é a do 'Momentum', que é uma abordagem complementar à descida do gradiente projetada para suavizar as oscilações durante o treinamento de redes neurais. Ela minimiza os ruídos nos gradientes ao adicionar uma fração do vetor de atualização anterior ao vetor de atualização atual dos pesos.

Como destacado por Qian (1999), "o método do Momentum melhora significativamente a velocidade de convergência ao longo da direção desejada e reduz a variação dos passos em direções irrelevantes, permitindo que a descida do gradiente se beneficie de informações anteriores para acelerar a otimização." Isso melhora a aceleração de convergência em direção ao mínimo local da função de custo, ajudando a evitar mínimos locais imprecisos e a melhorar a estabilidade durante o processo de otimização.

Além das técnicas de ajuste de pesos, a arquitetura da rede neural também influencia significativamente o desempenho do modelo. Nas últimas décadas, várias arquiteturas avançadas foram propostas para melhorar a precisão e a eficiência dos modelos de redes neurais convolucionais. Entre essas arquiteturas, destaca-se a família YOLO (You Only Look Once), que revolucionou o campo de detecção de objetos ao oferecer um modelo capaz de detectar múltiplos objetos em tempo real com alta precisão.

YOLO é uma abordagem de detecção de objetos que se diferencia das outras técnicas por ser capaz de detectar múltiplos objetos em uma única passagem pela rede neural, o que a torna extremamente eficiente. Enquanto métodos tradicionais, como o R-CNN, dividem a tarefa de detecção em duas etapas (geração de propostas de regiões e classificação), YOLO realiza ambas as tarefas simultaneamente, tratando a detecção como um problema de regressão direto.

Como descrito no site oficial do YOLO, "ao dividir a imagem em células de grade e prever diretamente as coordenadas dos objetos, classes e confiança em uma única rede, YOLO alcança uma velocidade superior em relação a outros métodos de detecção" (YOLO, 2025). Essa abordagem permite que YOLO seja muito mais rápido do que outros métodos, sendo capaz de operar em tempo real.

O YOLOv8 aprimora o desempenho do modelo ao integrar avanços em eficiência

computacional e precisão. Ele utiliza técnicas modernas de aprendizado profundo, como a Cross Stage Partial Network (CSPNet), que melhora o fluxo de informações durante a extração de características, e a Path Aggregation Network (PANet), que facilita a fusão de características em múltiplas escalas, resultando em uma detecção mais robusta.

O YOLOv8 é projetado para ser altamente flexível, permitindo ajustes para diversas aplicações, desde a detecção de pequenos objetos em imagens até a identificação simultânea de múltiplas classes de objetos. Conforme descrito na documentação do YOLOv8, "a arquitetura do YOLOv8 foi otimizada para maximizar a precisão e a eficiência computacional, aproveitando estruturas avançadas como CSP e PANet para melhorar a detecção em múltiplas escalas e reduzir o custo computacional."

O trabalho aqui proposto utiliza o modelo YOLO V8 para a detecção de tomates, visando classificar seu estado de maturação em verde, maduro e podre. Também será criada uma interface simples onde, ao fazer o upload de imagens em uma aplicação web, será realizada a detecção, classificação e contagem dos tomates identificados.

Esse projeto tem por objetivo trazer alternativas para a automatização do controle de qualidade de uma colheita de tomates, com foco na eficácia da produção agrícola. A detecção automatizada e precisa dos diferentes estágios de maturação dos tomates é crucial para garantir que apenas os frutos de qualidade sejam colhidos e enviados para o mercado, reduzindo perdas e aumentando a lucratividade dos produtores.

O uso de uma solução baseada em IA, como o YOLO V8, contribui para a sustentabilidade da produção agrícola. Ao automatizar processos que tradicionalmente exigiriam grande esforço manual, como a classificação e contagem de tomates, é possível otimizar o uso de recursos e reduzir o impacto ambiental. Por exemplo, a identificação precoce de tomates podres ou danificados pode ajudar a prevenir a propagação de doenças na plantação, permitindo que os agricultores tomem medidas corretivas antes que o problema se agrave.

A adoção de tecnologias avançadas na agricultura, como o YOLO V8, também abre portas para futuras inovações e melhorias. A integração de sensores IoT (Internet das Coisas) com sistemas de detecção baseados em IA pode permitir monitoramento contínuo das plantações,

oferecendo dados em tempo real sobre a saúde das plantas e o estado dos frutos. Esse tipo de abordagem pode transformar radicalmente a forma como a agricultura é conduzida, tornando-a mais precisa, eficiente e adaptável às mudanças climáticas e às demandas do mercado.

1.1 PROBLEMA DE PESQUISA

O problema central deste trabalho é a falta de eficiência e precisão no processo de controle de qualidade da colheita de tomates, que é causado por fatores humanos, como cansaço, estresse e sobrecarga de trabalho. Esses fatores levam a uma menor produtividade e à possibilidade de grandes desperdícios devido à falha na detecção de frutas impróprias para consumo.

Como se pode automatizar e melhorar a precisão no processo de controle de qualidade e contagem de tomates, utilizando técnicas de inteligência artificial como o YOLO V8, de modo a minimizar erros humanos e aumentar a eficiência na colheita?

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

- Desenvolver um sistema web integrado com uma solução de inteligência artificial capaz de realizar a detecção e classificação automática de tomates em imagens e vídeos, identificando suas três fases de maturação: Maduro, Verde e Podre.

1.2.2 OBJETIVOS ESPECÍFICOS

- Desenvolver um sistema integrado para o monitoramento e gestão da colheita de tomates, utilizando técnicas de inteligência artificial e análise climatológica.
- Implementar um modelo de inteligência artificial baseado na arquitetura YOLOv8 para realizar a detecção, contagem e classificação automática de tomates, identificando os estágios de maturação: maduro, verde e podre.
- Criar um módulo de web scraping capaz de coletar e armazenar dados climatológicos diretamente do site do Instituto Nacional de Meteorologia (INMET), estruturando essas informações em formato JSON.

- Desenvolver uma interface interativa utilizando a linguagem Python e o framework Streamlit, permitindo a análise de imagens e vídeos contendo os frutos.
- Estruturar uma arquitetura modular e eficiente, promovendo a integração entre os componentes do frontend (dashboard), backend (processamento de IA e dados climáticos) e banco de dados.
- Implementar mecanismos de armazenamento do histórico de inferências realizadas, possibilitando o acompanhamento temporal da produção e a exportação dos dados em formatos CSV e Excel, visando facilitar a análise, o controle e a tomada de decisões.

1.3 JUSTIFICATIVA

O presente trabalho busca desenvolver um sistema web inteligente de monitoramento e gestão da colheita de tomates, integrando inteligência artificial e análise climatológica para proporcionar maior eficiência e precisão no acompanhamento da produção agrícola. A justificativa para a definição dos objetivos específicos baseia-se na necessidade de automatizar o processo de detecção e classificação de tomates, permitindo uma avaliação mais ágil e confiável dos estágios de maturação dos frutos. A escolha da arquitetura YOLOv8 para a detecção e classificação automática justifica-se pelo seu desempenho rápido e preciso, tornando-o adequado para aplicações em tempo real. A implementação de um módulo de web scraping para a coleta de dados meteorológicos do INMET permite a correlação entre condições ambientais e produtividade agrícola, possibilitando tomadas de decisão mais informadas.

A interface interativa, desenvolvida em Python com Streamlit, tem como objetivo tornar o sistema acessível e intuitivo, facilitando o uso por produtores rurais e técnicos agrícolas. A estrutura modular do sistema, integrando frontend, backend e banco de dados, proporciona uma solução escalável e eficiente, garantindo maior organização e processamento dos dados coletados. Por fim, a implementação de mecanismos de armazenamento do histórico de inferências permite acompanhar a evolução da produção ao longo do tempo, possibilitando análises detalhadas e exportação dos dados em formatos CSV e Excel, facilitando o controle e a gestão da colheita. Dessa forma, o sistema proposto representa uma solução inovadora e de grande potencial para a modernização e automação da agricultura de precisão.

1.4 METODOLOGIA

O desenvolvimento deste trabalho foi estruturado em cinco etapas principais, que refletem as atividades detalhadas nos capítulos do TCC. Essas etapas incluem coleta de dados, pré-processamento e rotulação de imagens, treinamento da rede neural, avaliação do modelo e desenvolvimento da interface, garantindo que todas as atividades estejam integradas de forma lógica e coesa.

1.4.1 COLETA DE DADOS

A primeira etapa consistiu na coleta de imagens para compor o dataset de treinamento, detalhada no Capítulo 3, tópico 3. Utilizou-se a plataforma Roboflow Universe (ROBOFLOW UNIVERSE, 2025) que disponibiliza bases de dados para aprendizado de máquina. Foram selecionadas aproximadamente 1400 imagens de tomates em diferentes estágios de maturação (maduro, verde e podre), capturadas sob diversas condições de iluminação, ângulos e distâncias, garantindo maior diversidade e representatividade no conjunto de dados

1.4.2 PRÉ-PROCESSAMENTO E ROTULAÇÃO DE IMAGENS

Após a coleta, as imagens passaram por um processo de rotulação no Roboflow, conforme descrito no Capítulo 3, tópico 4. Nessa etapa, cada imagem foi anotada com caixas delimitadoras (bounding boxes) para indicar a posição dos tomates e categorizá-los conforme sua maturação. O redimensionamento das imagens foi realizado para garantir compatibilidade com o modelo YOLOv8, padronizando a entrada do dataset e garantindo maior consistência nos resultados durante o treinamento.

1.4.3 TREINAMENTO DA REDE NEURAL

O treinamento do modelo YOLOv8 foi realizado no Google Colab, aproveitando a API do Roboflow para importar os dados diretamente, conforme descrito no Capítulo 3 tópico 5. A conexão foi estabelecida via chave de API, permitindo um fluxo contínuo e automatizado do dataset rotulado para o ambiente de treinamento. O modelo foi configurado com ajustes de hiperparâmetros, como taxa de aprendizado, número de épocas e tamanho do lote, visando otimizar a acurácia e a eficiência da detecção de tomates.

1.4.4 AVALIAÇÃO DO MODELO

Durante o treinamento, foi realizada uma avaliação sistemática do desempenho do modelo proposto, com o objetivo de verificar sua eficácia na detecção e classificação de tomates em diferentes estágios de maturação. Essa avaliação, descrita em maior profundidade no Capítulo 4, envolveu o uso de métricas consagradas na área de visão computacional, aplicadas sobre um conjunto de validação separado, garantindo que os resultados refletissem a capacidade de generalização do modelo em dados não vistos.

1.4.5 DESENVOLVIMENTO DA INTERFACE

Por fim, a última etapa consistiu na implementação da interface interativa, conforme descrito no Capítulo 3 tópico 7. Para isso, foi utilizado o framework Streamlit, que permite a visualização em tempo real das detecções realizadas pelo modelo. A interface foi projetada para oferecer:

- Upload de imagens e vídeos para análise automática.
- Exibição gráfica dos resultados utilizando Matplotlib, incluindo gráficos estatísticos sobre a detecção dos tomates.
- Módulo de web scraping, utilizando BeautifulSoup e Selenium, para coletar dados climáticos diretamente do INMET, permitindo a correlação entre variáveis ambientais e produtividade agrícola.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda a fundamentação teórica do estudo, explorando conceitos essenciais para a compreensão do tema. Serão discutidos aspectos da agroindústria e sua relação com a economia, além da aplicação da Inteligência Artificial (IA) e do aprendizado de máquina no setor. Essas abordagens fornecem a base teórica para o desenvolvimento da pesquisa.

2.1 REFERENCIAL TEÓRICO

A agroindústria é um segmento essencial que combina a produção agrícola com processos industriais para transformar matérias-primas em produtos processados. Segundo Silva et al. (2020), "a integração entre a agricultura e a indústria agrega valor aos produtos agrícolas, gerando produtos finais para o consumo humano e animal, além de impulsionar a economia por meio da geração de empregos e exportações". No Brasil, esse setor é um dos mais importantes para a economia, sendo responsável por uma parcela significativa do PIB e pela sustentabilidade de milhões de famílias rurais.

A Inteligência Artificial (IA) refere-se ao desenvolvimento de sistemas computacionais capazes de realizar tarefas que normalmente exigiriam inteligência humana. De acordo com Khaleel e Jebrel (2024), "a IA tem a capacidade de aprender, raciocinar e tomar decisões de forma autônoma com base em dados e algoritmos, tornando-se uma ferramenta indispensável em diversos setores da sociedade". Uma das áreas mais conhecidas da IA é o aprendizado de máquina (machine learning), que permite que sistemas aprendam a partir de dados, identifiquem padrões e façam previsões. Como afirmam Russell e Norvig (2021), "o aprendizado de máquina permite que computadores adquiram conhecimento a partir da experiência, tornando-os mais eficientes em suas funções sem necessidade de programação explícita".

Outra área importante da IA é o processamento de linguagem natural (NLP - Natural Language Processing), que possibilita aos computadores compreenderem e responderem a comandos em linguagem humana. Segundo Goodfellow, Bengio e Courville (2016), "o NLP tem permitido avanços significativos na interação humano-computador, com aplicações que vão desde assistentes virtuais até a análise de sentimentos em redes sociais".

A inteligência artificial tem aplicações em diversos campos, incluindo assistentes virtuais, reconhecimento de voz, veículos autônomos e diagnóstico médico. Segundo Khaleel e Jebrel (2024), "a IA está transformando a forma como interagimos com a tecnologia, impactando a

sociedade e impulsionando mudanças na maneira como trabalhamos e vivemos". No entanto, essa transformação também levanta preocupações éticas e sociais. Como apontam Bostrom e Yudkowsky (2014), "a privacidade, a segurança e o impacto da IA no mercado de trabalho são questões críticas que precisam ser debatidas para garantir um uso responsável dessa tecnologia".

As Redes Neurais Convolucionais (CNNs) são um dos principais modelos de IA utilizados em visão computacional. De acordo com LeCun, Bengio e Hinton (2015), "as CNNs são projetadas para reconhecer padrões visuais, tornando-se altamente eficientes para tarefas como reconhecimento de imagens, detecção de objetos e segmentação de imagens". Uma característica fundamental das CNNs é a camada convolucional, que aplica filtros às imagens para extrair características relevantes. Como explicam Gu et al. (2018), "essas camadas permitem que a rede aprenda automaticamente bordas, texturas e formas, reduzindo a necessidade de engenharia manual de características".

As CNNs têm sido amplamente utilizadas em diversas aplicações, como reconhecimento facial, diagnóstico médico e veículos autônomos. Segundo Khan et al. (2020), "a eficácia das CNNs em lidar com grandes volumes de dados e sua capacidade de aprendizado hierárquico as tornaram uma das principais ferramentas da visão computacional moderna".

A visão computacional se concentra na interpretação e análise de imagens digitais. Conforme Szeliski (2022), "a visão computacional envolve técnicas para a extração de informações a partir de imagens e vídeos, permitindo aplicações como segmentação, rastreamento de objetos e reconstrução 3D". Na agricultura, essa tecnologia tem sido aplicada para a detecção de doenças em plantações, estimativa de safra e automação da colheita. Segundo Sharma et al. (2019), "o uso de visão computacional na agricultura tem permitido o monitoramento automatizado da lavoura, otimizando o uso de recursos e aumentando a produtividade".

Uma das abordagens mais eficientes da visão computacional para a detecção de objetos é a arquitetura YOLO (You Only Look Once). Segundo Redmon et al. (2016), "o YOLO revoluciona a detecção de objetos ao realizar todo o processamento em uma única passagem pela rede neural, tornando-o extremamente rápido e eficiente". O modelo divide a imagem em uma grade e prevê caixas delimitadoras para cada região. Como explicam Bochkovskiy, Wang e Liao (2020), "essa abordagem permite que o YOLO detecte objetos em tempo real, sendo amplamente utilizado em aplicações que exigem alta velocidade e precisão, como segurança e monitoramento de tráfego".

O YOLO apresenta vantagens significativas em relação a outros modelos. De acordo com Ge et al. (2021), "sua capacidade de realizar detecções em tempo real sem comprometer a precisão faz com que seja uma das principais escolhas para aplicações práticas de visão computacional". Essa combinação de velocidade e precisão tem tornado o YOLO uma ferramenta essencial para diversas áreas, incluindo a agricultura de precisão. Como destaca Liu et al. (2023), "ao integrar modelos como o YOLO à análise de imagens agrícolas, é possível otimizar processos produtivos e reduzir perdas, garantindo maior eficiência na gestão da colheita".

2.2 FERRAMENTAS DE SOFTWARE

2.2.1 PYTHON

Figura 1- Python

Fonte: https://www.python.org/, 2025.

Python é uma linguagem de programação interpretada, orientada a objetos e de alto nível, com semântica dinâmica (PYTHON, 2024). Sua sintaxe simples e fácil de aprender enfatiza a legibilidade e reduz o custo da manutenção do programa. Python é conhecido por sua capacidade de oferecer um desenvolvimento rápido de aplicações, sendo também amplamente utilizado como linguagem de script ou de ligação para conectar componentes existentes.

Uma das características marcantes do Python é sua vasta biblioteca padrão, que oferece suporte para uma ampla variedade de tarefas, desde manipulação de arquivos até desenvolvimento web. Python também suporta módulos e pacotes, o que promove a modularidade do programa e a reutilização de código, facilitando o desenvolvimento e a manutenção de projetos de software.

Outro ponto forte do Python é sua ampla aplicabilidade no desenvolvimento de sistemas de visão computacional, aliada à sua capacidade de lidar com erros de forma eficiente. Ao trabalhar com bibliotecas como OpenCV, PIL, TensorFlow e Ultralytics YOLO, erros durante o processamento de imagens, leitura de arquivos ou execução de modelos podem ser facilmente

tratados por meio do sistema de exceções do Python. Isso evita falhas inesperadas e torna o processo de desenvolvimento mais robusto. O Python oferece suporte à depuração em nível de código-fonte, permitindo a inspeção de variáveis, análise do fluxo de execução e avaliação de expressões durante o processamento de imagens ou inferência de modelos. Essa capacidade facilita significativamente a identificação e correção de erros em pipelines complexos de visão computacional.

2.2.1.1 BIBLIOTECAS DE PYTHON

As bibliotecas utilizadas no desenvolvimento deste projeto foram fundamentais para garantir o funcionamento eficiente das etapas de leitura de arquivos, visualização de resultados e integração com dados externos. As ferramentas escolhidas permitiram desde a manipulação básica de imagens e vídeos para inferência, até a coleta automatizada de informações ambientais relevantes para a análise do cultivo. A seguir, são descritas as principais bibliotecas empregadas:

- **OpenCV:**
  A biblioteca OpenCV foi utilizada para realizar a leitura e exibição das imagens e vídeos submetidos ao modelo de detecção. Ela permitiu o carregamento eficiente dos arquivos utilizados nas inferências, possibilitando a visualização dos quadros em tempo real e o acompanhamento do desempenho do modelo YOLOv8 na detecção dos tomates. Essa leitura foi essencial para validar a aplicação prática do sistema com diferentes formatos de entrada.
- **Selenium:**
  O Selenium foi utilizado para automatizar a coleta de variáveis climáticas diretamente do site do INMET (Instituto Nacional de Meteorologia). Por meio da técnica de web scraping, foi possível extrair informações como temperatura, umidade e radiação solar, dados esses relevantes para análise complementar ao desenvolvimento dos frutos. A integração com esses dados permitiu ampliar o contexto de monitoramento, tornando o sistema mais completo e útil ao agricultor.
- **Matplotlib:**
  A biblioteca Matplotlib foi empregada na construção de gráficos que representaram visualmente o número de tomates detectados em cada imagem ou vídeo processado. Essa representação gráfica contribuiu para uma melhor compreensão dos resultados das inferências,

permitindo identificar, de forma clara, a quantidade de frutos classificados por tipo (verde, maduro, podre) e analisando o desempenho do modelo de forma quantitativa.

2.2.2 ULTRALYTICS YOLOV8

Figura 2- Ultralytics

Fonte: https://github.com/ultralytics/ultralytics/blob/main/docs/en/models/yolov8.md, *2025.*

O Ultralytics YOLOv8 é uma das versões mais recentes da arquitetura YOLO (You Only Look Once), voltada para tarefas de detecção de objetos com alta precisão e desempenho em tempo real. Desenvolvido pela empresa Ultralytics, o YOLOv8 traz melhorias significativas em relação às versões anteriores, como uma estrutura de modelo mais leve, rápida e precisa. Essa ferramenta tem sido amplamente utilizada em aplicações práticas por sua facilidade de uso e integração com bibliotecas do ecossistema Python, tornando-se ideal para projetos que envolvem visão computacional, como o monitoramento agrícola.

No contexto deste trabalho, o YOLOv8 foi aplicado para realizar a detecção automática de tomates em diferentes estágios de maturação: verdes, maduros e podres. A escolha por essa versão se deu por sua robustez na detecção de objetos em imagens complexas e por oferecer suporte nativo a tarefas de segmentação e classificação. A ferramenta permite a personalização do treinamento, aceitando datasets rotulados com diferentes formatos, além de possibilitar ajustes finos em hiperparâmetros como taxa de aprendizado, número de épocas e tamanho de lote, otimizando o desempenho conforme a necessidade do projeto.

Outro diferencial do Ultralytics YOLOv8 está em sua integração direta com ferramentas como o Roboflow e sua capacidade de exportar modelos treinados para diferentes formatos (ONNX, TensorRT, CoreML), o que facilita a implementação em ambientes diversos. Durante os testes deste projeto, o YOLOv8 demonstrou bom desempenho mesmo em condições variadas de iluminação e posicionamento dos frutos, comprovando sua aplicabilidade em cenários reais do

setor agroindustrial. A clareza na documentação e a interface de uso simples também contribuíram para sua adoção eficiente ao longo do desenvolvimento do sistema.

2.2.3 STREAMLIT

Figura 3- Streamlit

Fonte: https://streamlit.io/, 2025.

O Streamlit é um framework open-source voltado para o desenvolvimento rápido de interfaces web interativas utilizando a linguagem Python. Ele permite transformar scripts de ciência de dados e aprendizado de máquina em aplicações web com poucos comandos, eliminando a necessidade de conhecimentos avançados em front-end. Por sua simplicidade e integração direta com bibliotecas como Pandas, OpenCV e Matplotlib, o Streamlit tornou-se uma ferramenta ideal para prototipagem e apresentação de projetos de forma acessível e visual.

O Streamlit foi utilizado para criar uma interface interativa capaz de carregar imagens e vídeos contendo tomates, realizar inferências com o modelo YOLOv8 e exibir os resultados diretamente na tela. A interface também foi integrada com gráficos e contadores automáticos, permitindo ao usuário visualizar a quantidade de frutos detectados por classe (verde, maduro e podre), além de acompanhar os valores médios de precisão. Essa aplicação tornou o sistema mais amigável e funcional para uso por agricultores ou pesquisadores.

Outro ponto positivo do Streamlit foi sua facilidade de implantação em ambientes locais e servidores online, o que possibilita que o sistema seja acessado remotamente por qualquer dispositivo conectado à internet. Isso torna a solução escalável e adaptável a diferentes realidades do campo, contribuindo diretamente para a democratização da tecnologia na agricultura. A leveza e a agilidade proporcionadas pelo Streamlit foram fundamentais para a criação de um sistema prático, eficiente e de fácil navegação.

2.2.4 ROBOFLOW

Figura 4 - Roboflow

Fonte: https://roboflow.com/, 2025.

O Roboflow é uma plataforma online projetada para facilitar o gerenciamento completo de datasets voltados para visão computacional. Com ele, é possível realizar desde a anotação manual de imagens até o pré-processamento e exportação para diferentes formatos compatíveis com modelos como o YOLOv8. Sua interface baseada na web permite que usuários de diferentes níveis técnicos possam organizar, rotular e preparar seus conjuntos de dados de forma prática e eficiente para tarefas de detecção, segmentação e classificação de objetos.

No desenvolvimento deste projeto, o Roboflow foi utilizado para rotular tomates nas categorias verde, maduro e podre, além de aplicar técnicas de data augmentation (como rotação, alteração de brilho e inversão horizontal), a fim de tornar o modelo mais generalista. As imagens foram exportadas diretamente no formato YOLOv8, otimizando a integração com o ambiente de treinamento. O Roboflow também forneceu métricas e relatórios úteis para validar a qualidade e diversidade do dataset durante a preparação.

Outro recurso explorado foi o Roboflow Universe, um repositório colaborativo que oferece datasets públicos e prontos para uso em diversos contextos. As imagens utilizadas na etapa de treinamento foram obtidas a partir dessa base, contribuindo com variedade de ângulos, iluminação e condições de cultivo. Essa combinação entre imagens capturadas na horta e outras oriundas do Roboflow Universe enriqueceu significativamente o dataset final, favorecendo a capacidade do modelo em reconhecer tomates com maior precisão em situações reais e variadas.

2.2.5 EXCALIDRAW

Figura 5- Excalidraw

Fonte: https://excalidraw.com/, 2025.

O Excalidraw é uma ferramenta online voltada para a criação de desenhos técnicos, esquemas, fluxogramas e wireframes com aparência de esboço à mão livre. Por ser intuitivo, colaborativo e de fácil uso, tornou-se uma solução prática para quem precisa representar ideias de forma visual sem recorrer a softwares complexos de design gráfico.

No desenvolvimento deste projeto, o Excalidraw foi utilizado para elaborar os esquemas de arquitetura da solução, representando visualmente o fluxo de dados entre os módulos, como a interação entre a interface desenvolvida em Streamlit, o modelo YOLOv8 e os serviços de coleta e análise de dados. Esses diagramas auxiliaram na organização do raciocínio lógico e facilitaram a comunicação da estrutura geral do sistema durante a documentação e apresentação.

A ferramenta também foi empregada na criação de mockups das telas da aplicação, possibilitando o planejamento visual da interface de forma rápida e ajustável antes da implementação. Isso permitiu testar diferentes disposições de botões, menus e áreas de exibição de resultados, resultando em uma experiência de usuário mais clara e funcional.

2.3 PLANTIO DE TOMATES

O cultivo de tomates no Brasil ocupa posição de destaque dentro da horticultura nacional, tanto em volume quanto em importância econômica. Segundo dados apresentados pela Agrishow Digital (2021), com base em informações do IBGE, o Brasil produziu aproximadamente 3,9 milhões de toneladas de tomate em 2019, refletindo a força produtiva dessa cultura no cenário agrícola. No entanto, a condução do processo produtivo do tomateiro exige atenção em todas as etapas, desde o plantio até a pós-colheita, devido à sua sensibilidade a variações climáticas, pragas e manuseio inadequado.

Apesar de toda a tecnologia disponível, a colheita do tomate ainda é realizada, majoritariamente, de forma manual em propriedades de pequeno e médio porte. Esse processo, embora acessível economicamente, acarreta perdas significativas relacionadas a fatores humanos, como a colheita em estágios incorretos de maturação, danos mecânicos causados por manipulação imprópria, e acondicionamento incorreto dos frutos. Essas perdas impactam diretamente na qualidade final do produto e na rentabilidade do produtor, além de aumentarem o desperdício de recursos naturais e insumos envolvidos no cultivo.

A falta de padronização nas decisões humanas durante a colheita evidencia a necessidade de ferramentas tecnológicas de apoio à tomada de decisão, como sistemas de visão computacional

baseados em inteligência artificial. Esses sistemas podem detectar automaticamente o estágio de maturação dos frutos, contribuindo para uma colheita mais precisa e sustentável, com redução de perdas causadas por erros humanos, otimização do tempo e melhoria da eficiência no campo.

3. SDT - SISTEMA DE DETECÇÃO DE TOMATES

Este capítulo apresenta o desenvolvimento do Sistema de Detecção de Tomates (SDT), detalhando sua arquitetura, funcionamento e implementação. Inicialmente, são discutidas as arquiteturas de hardware e software utilizadas no projeto. Em seguida, aborda-se o processo de coleta e pré-processamento dos dados, etapas essenciais para garantir a qualidade das imagens utilizadas no treinamento da rede neural. Também são exploradas as estratégias de treinamento e avaliação do modelo, seguidas pelo desenvolvimento da interface do sistema, incluindo funcionalidades como dashboard, reconhecimento em fotos e vídeos, e monitoramento climático.

3.1 ARQUITETURA DO HARDWARE

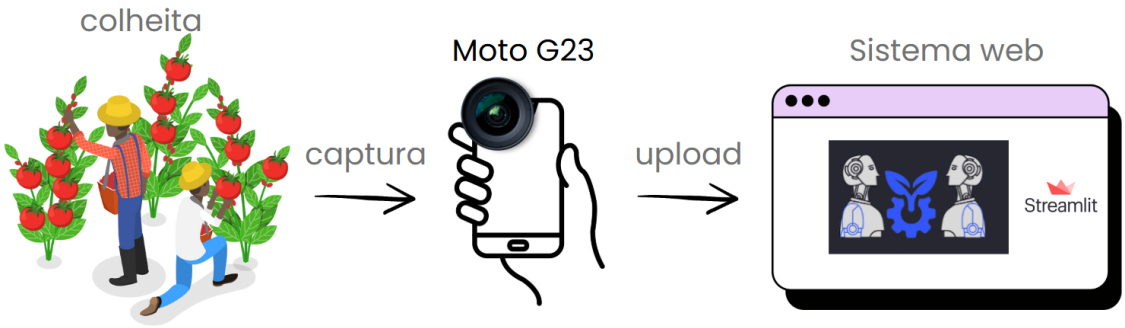

Figura 6 - Arquitetura do hardware

Foi utilizado um Moto G23 como dispositivo de captura de imagens e vídeos para análise no modelo de IA. O Moto G23 é um smartphone da Motorola equipado com um sensor principal de 50 MP e tecnologias avançadas de processamento de imagem, garantindo alta resolução e fidelidade nos registros.

Especificações Técnicas do Moto G23:

- Sensor Principal: 50 MP, f/1.8, PDAF
- Câmera Secundária: 5 MP (ultrawide) + 2 MP (macro)
- Gravação de Vídeo: Full HD (1080p) a 30 fps
- Processador: MediaTek Helio G85 (2x 2.0 GHz Cortex-A75 + 6x 1.8 GHz Cortex-A55)
- Memória RAM: 4 GB ou 8 GB LPDDR4X
- Armazenamento: 128 GB (expansível via microSD)
- Bateria: 5.000 mAh com carregamento rápido de 30W

- Conectividade: USB-C, Bluetooth 5.1, Wi-Fi 802.11ac
- Sistema Operacional: Android 13

3.2 ARQUITETURA DE SOFTWARE

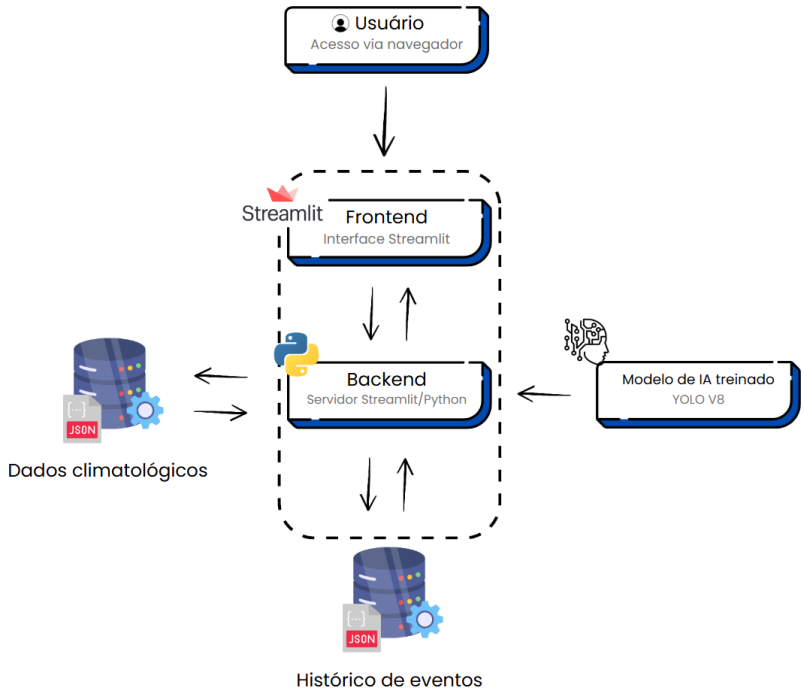

Figura 7 - Arquitetura de software

A arquitetura de software sistema é baseada em Streamlit, integrando o frontend e o backend diretamente no mesmo ambiente. O usuário acessa a interface via navegador, onde pode interagir com o sistema e enviar imagens ou vídeos para análise. O frontend gerencia a interface e envia os dados para o backend, que executa as funções de processamento. O backend, desenvolvido em Python, utiliza um modelo de IA treinado (YOLO v8) para realizar o reconhecimento de imagens. O sistema mantém dois arquivos JSON para armazenamento de informações: um destinado aos dados climatológicos, coletados via web scraping, e outro para o histórico de eventos, registrando as análises processadas pelo modelo de IA.

Dessa forma, a arquitetura permite a organização e persistência dos dados de forma estruturada, garantindo que tanto as condições climáticas quanto os resultados das detecções sejam armazenados e acessados conforme necessário. O fluxo de dados é contínuo, permitindo que os resultados das análises sejam exibidos no frontend em tempo real.

3.2.1 DIAGRAMA DE CASO DE USO

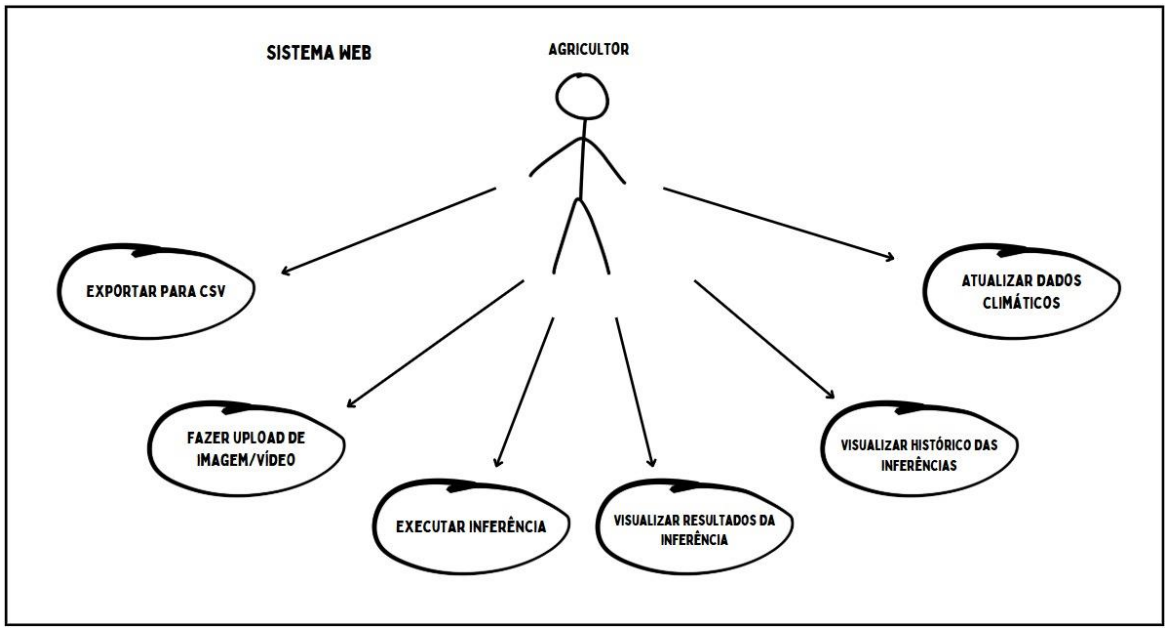

Figura 8 - Diagrama de caso de uso

Este diagrama representa as principais funcionalidades disponibilizadas ao agricultor por meio do sistema web desenvolvido. O ator central, o agricultor, interage diretamente com a interface do sistema para executar ações relevantes no contexto da colheita e monitoramento dos tomates. Dentre essas ações estão: envio de imagens ou vídeos, execução de inferência utilizando o modelo YOLOv8, visualização dos resultados, atualização de dados climáticos coletados por web scraping e exportação dos dados processados em formato csv.

Essas funcionalidades, representadas como casos de uso, reforçam o papel da aplicação como uma ferramenta acessível e funcional, voltada para facilitar a tomada de decisões no campo. A interface do sistema foi construída com o framework Streamlit, permitindo acesso via navegador e simplificando a experiência do usuário. O agricultor, mesmo sem conhecimento técnico avançado, pode facilmente realizar inferências, analisar os dados e acompanhar o histórico de resultados, integrando assim inteligência artificial ao processo agrícola.

A estrutura representada no diagrama complementa diretamente a arquitetura de software apresentada anteriormente, onde o frontend e backend estão integrados em uma única camada com Streamlit. A coleta de dados climáticos é realizada de forma automatizada pelo backend, enquanto a inferência com YOLOv8 ocorre mediante acionamento pelo usuário. Todos os resultados são armazenados localmente em arquivos JSON e podem ser acessados ou exportados conforme

necessário, promovendo uma abordagem robusta, eficiente e interativa para monitoramento da colheita.

3.3 COLETA DE DADOS

A coleta de dados para este trabalho foi realizada por meio da plataforma Roboflow Universe, que oferece uma ampla variedade de conjuntos de dados voltados para aplicações em visão computacional. O processo de seleção foi conduzido de forma criteriosa, com o objetivo de garantir a diversidade, qualidade e representatividade das imagens utilizadas no treinamento e validação do modelo de detecção.

O primeiro passo consistiu na busca e identificação de diferentes datasets públicos no Roboflow Universe contendo imagens de tomates em variados estágios de maturação: maduro, verde e podre. Foram utilizadas palavras-chave relacionadas às classes de interesse, priorizando conjuntos com anotações consistentes, boa resolução de imagem e diversidade de ambientes. A partir dessa triagem, foram escolhidas as imagens mais compatíveis com os critérios definidos para o projeto, totalizando 1.468 imagens.

Essas imagens foram então importadas e organizadas em um ambiente de trabalho próprio criado na plataforma Roboflow, onde compuseram um novo dataset unificado. Esse ambiente permitiu o gerenciamento eficiente dos dados, bem como a aplicação de ajustes, padronizações e pré-processamentos necessários, preparando o conjunto final para ser exportado no formato compatível com o modelo YOLOv8.

3.4 PRÉ-PROCESSAMENTO

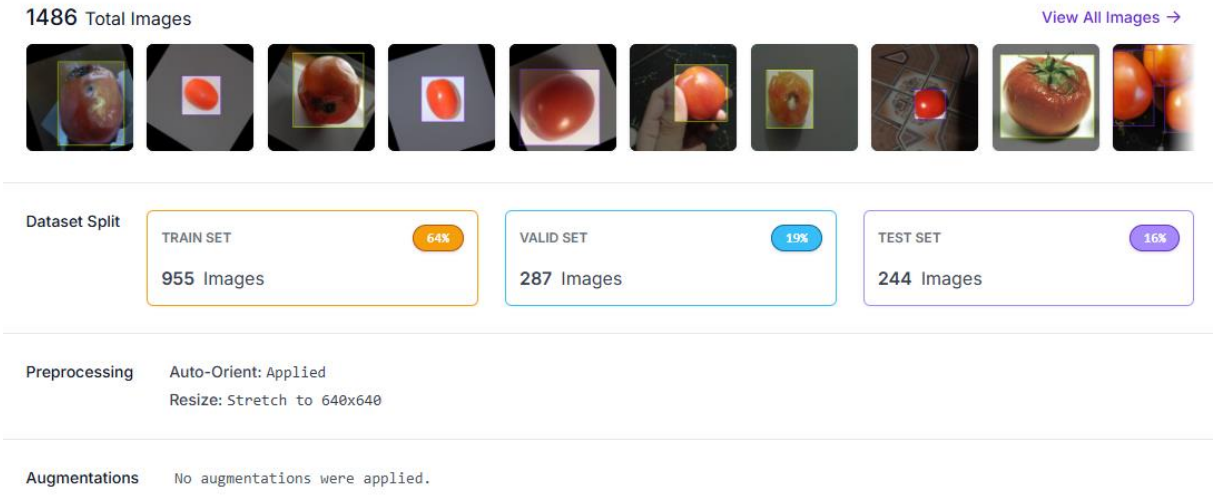

Figura 9 - Etapa de pré-processamento

Após a coleta das imagens, foi realizada a etapa de pré-processamento, com o objetivo de preparar os dados de forma adequada para o treinamento do modelo YOLOv8. Essa etapa incluiu o processo de rotulação (anotação) dos objetos de interesse e o redimensionamento padronizado das imagens, garantindo uniformidade no formato dos dados de entrada e compatibilidade com a arquitetura da rede neural.

Inicialmente, todas as imagens foram rotuladas manualmente dentro do ambiente do Roboflow, utilizando a ferramenta de anotação integrada da plataforma. Cada objeto presente nas imagens foi identificado e marcado com uma caixa delimitadora (bounding box), atribuindo-lhe a classe correspondente: “Maduro”, “Verde” ou “Podre”. Esse processo foi conduzido com atenção para assegurar a precisão das anotações, uma vez que erros nessa etapa poderiam comprometer o desempenho do modelo durante o treinamento.

Após a rotulação, foi realizado o redimensionamento automático das imagens, padronizando todas para uma resolução de 640x640 pixels, compatível com os requisitos da arquitetura YOLOv8. Em seguida, o Roboflow realizou a divisão automática do conjunto de dados em três subconjuntos: 64% das imagens (955) foram destinadas ao treino, 19% (287) para validação e 17% (244) para teste. Essa separação garantiu um fluxo de treinamento adequado, permitindo avaliar o desempenho do modelo de forma confiável tanto durante quanto após o processo de aprendizado. Ao final, o Roboflow gerou o dataset pronto para exportação no formato YOLO, contendo as imagens e seus respectivos arquivos de anotação.

3.5 TREINAMENTO DA REDE NEURAL

O treinamento do modelo YOLOv8 foi realizado no Google Colab, aproveitando a infraestrutura de GPU na nuvem para acelerar o processamento. Para facilitar o gerenciamento do dataset, foi utilizada a API do Roboflow, permitindo a importação das imagens rotuladas e pré-processadas no formato compatível com o YOLOv8.

A conexão com o Roboflow foi estabelecida através de uma chave de API, garantindo um fluxo eficiente e dinâmico dos dados diretamente para o ambiente de treinamento. O dataset foi carregado e organizado dentro do Google Colab, permitindo ajustes diretos antes da execução do treinamento.

O treinamento foi configurado com ajustes de hiperparâmetros estratégicos, incluindo:

- Número de épocas: Definido para 100, garantindo um período suficiente para a rede neural aprender padrões visuais robustos.
- Tamanho da imagem: Ajustado para 640x640 pixels, assegurando compatibilidade com o modelo YOLOv8 e melhor balanceamento entre precisão e desempenho computacional.
- Modelo base: Utilização do YOLOv8m, uma versão (m) intermediária da arquitetura YOLO que oferece um equilíbrio entre precisão e desempenho computacional. Essa escolha foi motivada pela necessidade de uma detecção mais precisa dos diferentes estágios de maturação dos tomates, garantindo melhor generalização sem comprometer excessivamente a velocidade de inferência.
- O parâmetro patience foi configurado com valor 20, indicando que o treinamento poderia ser interrompido automaticamente caso não houvesse melhoria nas métricas de validação durante 20 épocas consecutivas. Essa estratégia, conhecida como early stopping, visa evitar overfitting e reduzir o tempo de treinamento desnecessário, garantindo que o modelo pare de treinar quando atinge um ponto de estagnação em seu desempenho.

A execução do treinamento seguiu um pipeline estruturado, utilizando os comandos necessários para inicializar o modelo e otimizar seus parâmetros. Com isso, foi possível alcançar um equilíbrio entre velocidade de inferência e precisão na detecção, garantindo um desempenho adequado para a aplicação proposta.

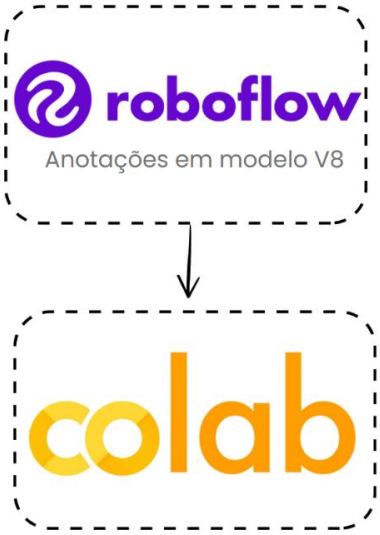

Figura 10 - Fluxo de anotações em formato yolo

Fonte: Próprio Autor, 2025.

```
!yolo task=detect \
mode=train \
model=yolov8m \
data={dataset.location}/data.yaml \
epochs=100 \
imgsz=640
patience=20 \
```

Figura 11 - Parâmetros de treinamento

Fonte: Próprio Autor, 2025.

```
from roboflow import Roboflow

rf = Roboflow(api_key='yWJAirSWZPyZn9Kjrt9S')
project = rf.workspace().project('tomates-4blsk')
dataset = project.version(1).download('yolov8')
```

Figura 12- Código de api key

Fonte: Próprio Autor, 2025.

3.6 AVALIAÇÃO

A avaliação do modelo proposto foi realizada com base em métricas amplamente utilizadas na área de visão computacional, como precisão, c, F1-score, mAP e matriz de confusão. Essas métricas permitem analisar o desempenho do sistema sob diferentes perspectivas, contemplando tanto a capacidade de localização espacial dos objetos quanto a correta atribuição das classes. A análise foi conduzida sobre um conjunto de validação independente, assegurando que os resultados refletissem o comportamento do modelo diante de dados inéditos.

A precisão e a recallfornecem uma visão complementar do desempenho do modelo: enquanto a precisão avalia o quanto das detecções feitas estão corretas, a recall indica o quanto das instâncias reais foram identificadas. O F1-score, por sua vez, representa o equilíbrio entre essas duas métricas, sendo particularmente útil para avaliar o modelo em cenários onde tanto falsos positivos quanto falsos negativos devem ser minimizados. Já o mAP (mean Average Precision) é uma métrica que leva em consideração o desempenho do modelo em diferentes limiares de sobreposição (IoU), refletindo sua eficácia tanto na localização precisa quanto na classificação correta dos objetos.

A matriz de confusão foi utilizada para identificar os padrões de acertos e erros entre as classes, evidenciando quais categorias o modelo consegue distinguir com clareza e onde ocorrem as maiores confusões. Essa análise qualitativa é essencial para compreender os desafios enfrentados pelo modelo, como a semelhança visual entre objetos de classes distintas ou entre objetos e o fundo da imagem. De forma geral, as métricas aplicadas forneceram uma avaliação robusta e abrangente, permitindo identificar os pontos fortes da solução e os aspectos que podem ser aprimorados em ciclos futuros de desenvolvimento.

3.7 DESENVOLVIMENTO DA INTERFACE

Foi desenvolvida uma interface utilizando a biblioteca Streamlit. Essa interface permite que o usuário faça upload de imagens e vídeos, visualize as detecções em tempo real e obtenha estatísticas sobre a classificação dos tomates.

A exibição dos gráficos estatísticos foi implementada utilizando a biblioteca Matplotlib, possibilitando uma análise clara da quantidade de tomates detectados por categoria. Os gráficos incluem representações como gráficos de pizza, que facilitam a compreensão dos resultados pelo usuário. Para obter os dados climáticos em tempo real, foi utilizada a biblioteca Selenium,

permitindo o web scraping de informações climáticas retiradas diretamente do site do INMET. Com essa funcionalidade, a interface exibe dados como temperatura, umidade e previsão do tempo, fornecendo um suporte adicional para o monitoramento da plantação.

3.7.1 TELA DO DASHBOARD

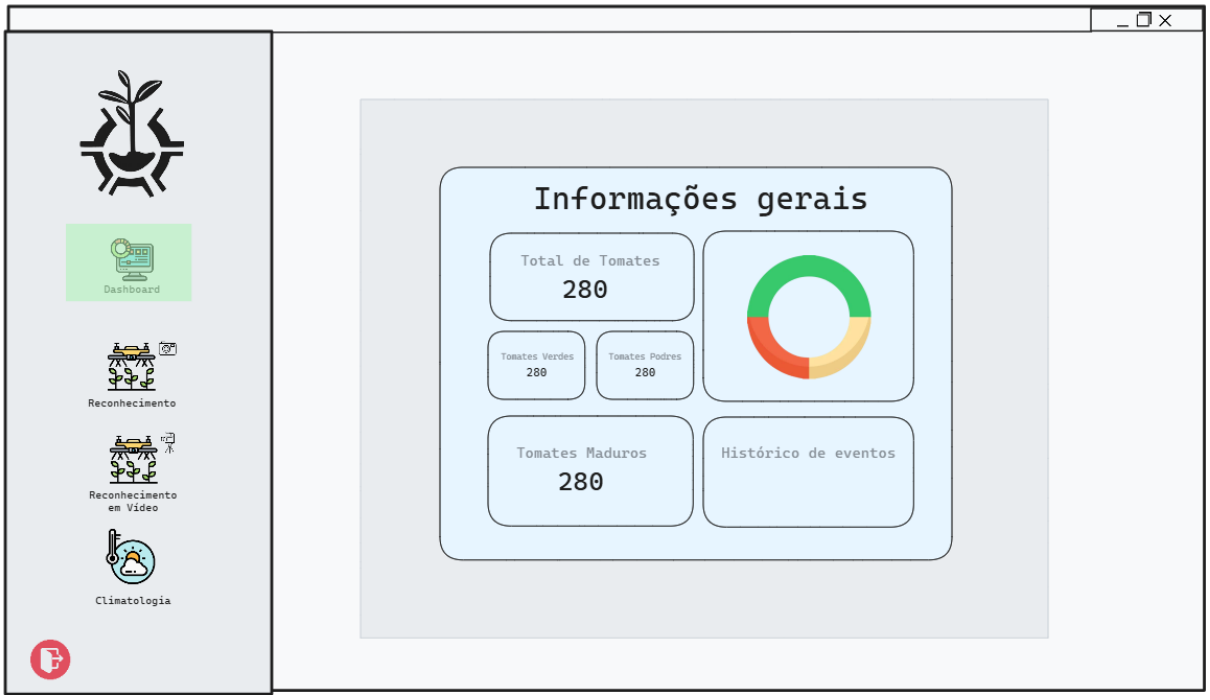

Figura 13- Mockup dashboard

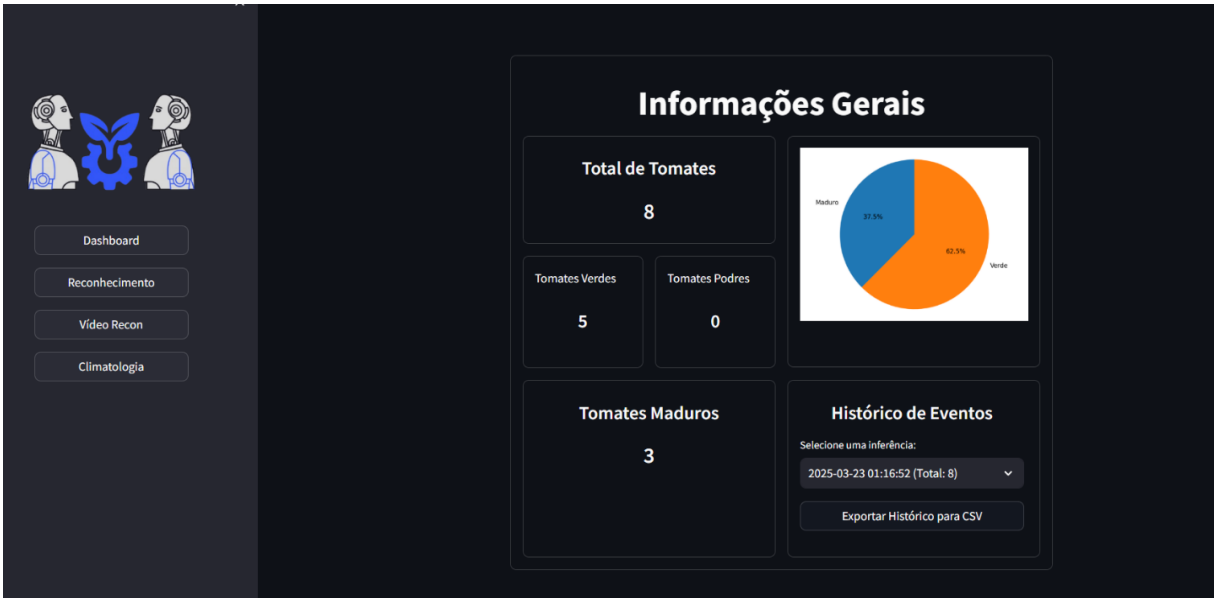

Figura 14 - Tela dashboard

O layout do dashboard possui um menu lateral à esquerda com ícones representando diferentes funcionalidades, como "Dashboard", "Reconhecimento", "Reconhecimento em Vídeo" e "Climatologia". A área principal exibe informações gerais sobre os tomates detectados, incluindo o total de tomates, sua classificação em tomates verdes, maduros e podres, e um gráfico colorido para representar a distribuição visual dessas categorias. Também, há uma seção para o histórico de eventos, para registrar ocorrências ao longo do tempo com possibilidade de exportação em csv.

3.7.2 TELA DO RECONHECIMENTO EM FOTO

Figura 15 - Mockup reconhecimento em foto

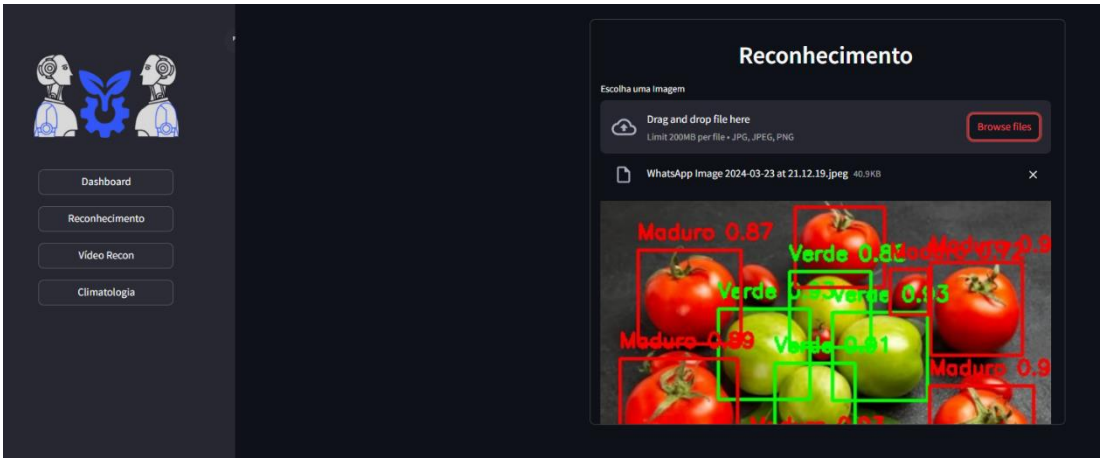

Figura 16 - Tela reconhecimento em foto

Na seção de “Reconhecimento” há um botão de upload no centro da tela, permitindo que o usuário envie imagens para análise. Logo abaixo, o resultado da imagem é exibido. Neste exemplo temos uma imagem com caixas vermelhas ao redor dos frutos, identificando-os como "maduros", acompanhados de valores de confiança variando entre 0.85 e 0.92. No rodapé da tela, há um resumo informando que foram detectados 6 tomates maduros, 0 podres e 0 verdes.

3.7.3 TELA DO RECONHECIMENTO EM VÍDEO

Figura 17- Mockup reconhecimento em vídeo

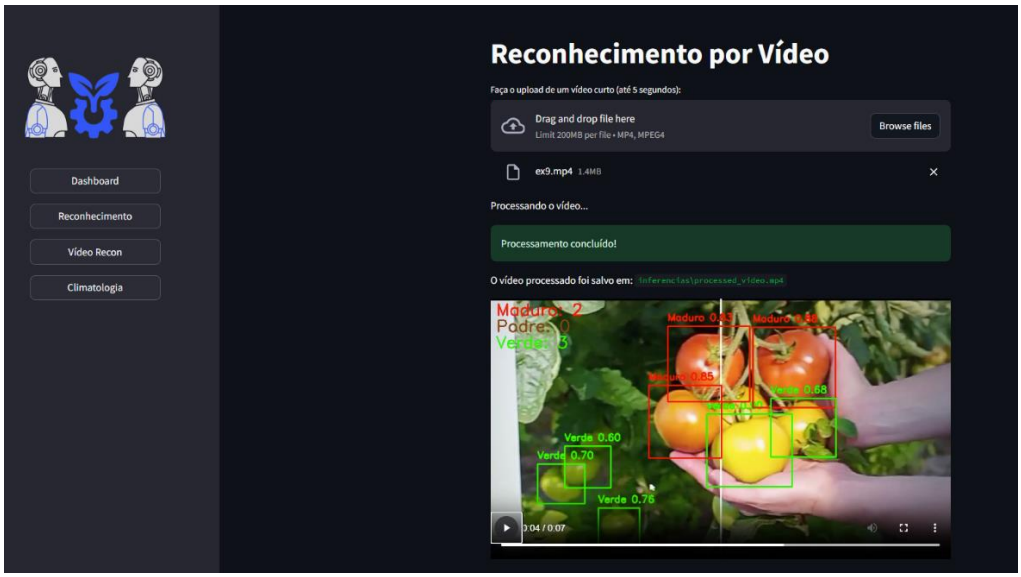

Figura 18- Tela reconhecimento em vídeo

A seção de “Reconhecimento em vídeo” funciona da mesma forma que a seção “Reconhecimento”, temos um botão de upload, permitindo upload de vídeos para uma inferência em vídeo. O detalhe diferencial desta opção é que haverá uma linha no centro do vídeo, quando os tomates passarem pela linha, haverá a contagem dos tomates detectados e após o processamento, ocorrerá o registro de cada classe no rodapé.

3.7.4 TELA DA CLIMATOLOGIA

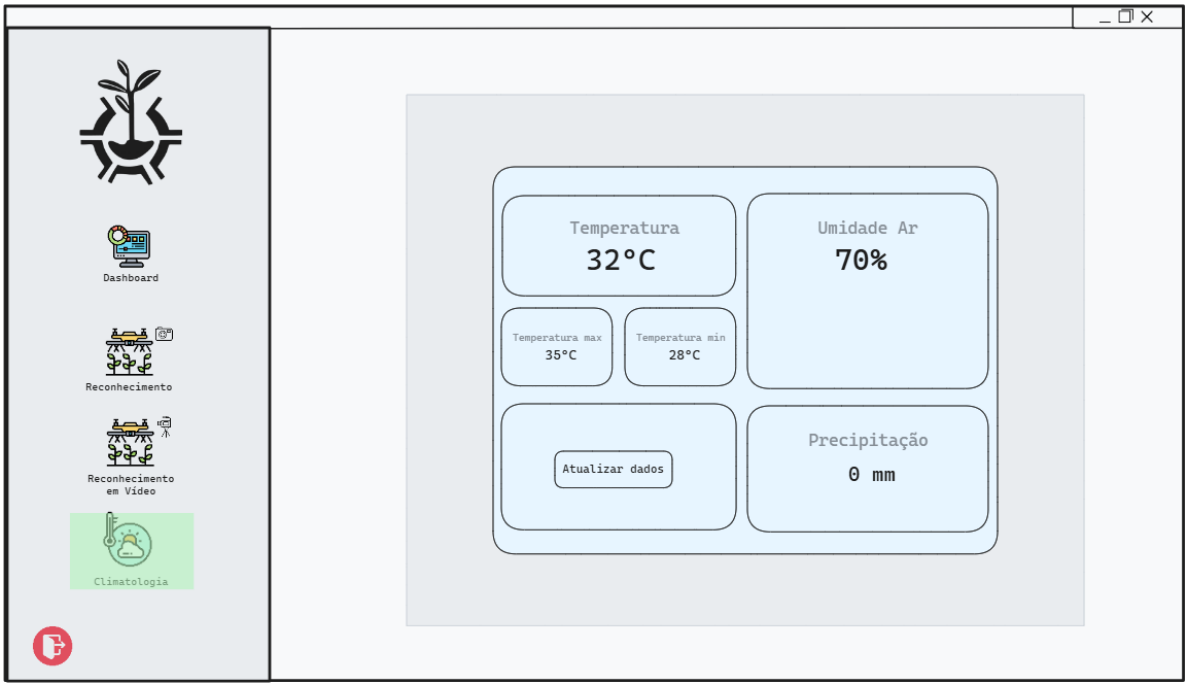

Figura 19 - Mockup climatologia

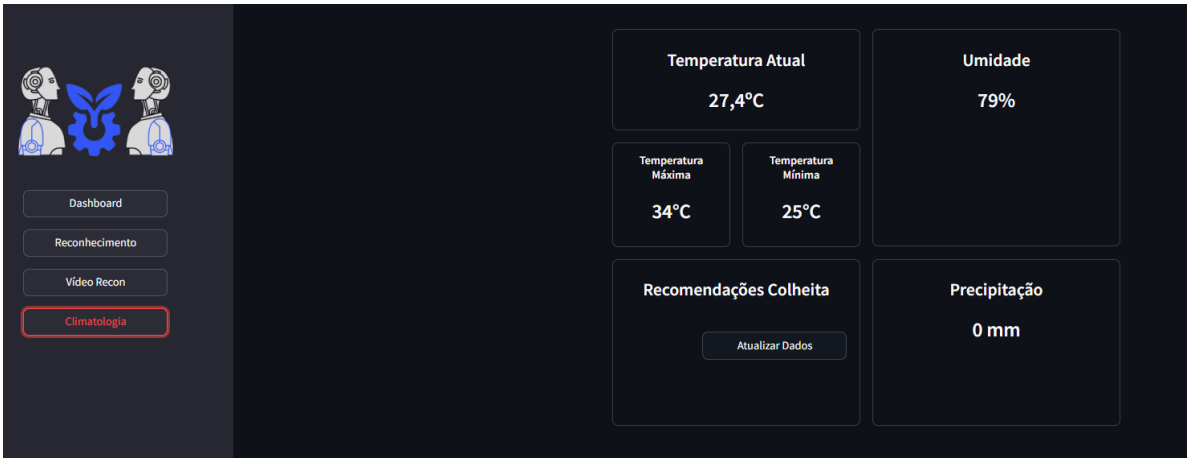

Figura 20 - Tela climatologia

Na seção "Climatologia", a interface exibe informações meteorológicas relevantes para o monitoramento agrícola. O painel central apresenta a temperatura atual de 32°C, acompanhada dos valores de temperatura máxima (35°C) e mínima (28°C) registradas. Ao lado, a umidade do ar é exibida com um valor de 70%, essencial para análise das condições ambientais. Há também um indicador de precipitação, que no momento registra 0 mm, sugerindo ausência de chuvas. Na parte inferior, um botão "Atualizar dados" permite que o usuário obtenha informações climáticas retiradas diretamente do site do INMET.

Figura 21 - Fluxo de dados climatológicos

O diagrama acima representa melhor o fluxo de coleta e armazenamento de dados climatológicos utilizando web scraping em Python. O processo inicia quando o script de web scraping acessa o site do INMET (Instituto Nacional de Meteorologia) para realizar a raspagem de dados, extraindo informações meteorológicas como temperatura, umidade e precipitação. Após a obtenção dos dados, o script processa e entrega os dados extraídos, que são então armazenados em um arquivo JSON. Esse fluxo garante que os dados sejam atualizados manualmente sempre que o usuário aciona a função de atualização, permitindo um monitoramento eficiente das condições climáticas.

A atualização constante dos dados climáticos é essencial no cultivo de tomates, pois variáveis como temperatura, umidade e precipitação afetam diretamente o crescimento dos frutos e a incidência de pragas. Com a função de atualização manual do sistema, o agricultor pode monitorar as condições ambientais em tempo real e ajustar suas ações de forma mais precisa. Isso permite, por exemplo, identificar o melhor momento para irrigar, aplicar defensivos ou realizar a colheita, contribuindo para uma produção mais eficiente e sustentável.

4. EXPERIMENTOS E RESULTADOS

Este capítulo apresenta os testes práticos realizados com o sistema desenvolvido, abordando o desempenho do modelo de detecção de tomates por meio de inferências, assim como a integração com dados climáticos para suporte à tomada de decisão na colheita.

As inferências foram conduzidas utilizando imagens capturadas em uma horta experimental, em diferentes ângulos e condições de iluminação, com o objetivo de simular cenários reais enfrentados pelos agricultores.

Paralelamente, o sistema foi configurado para acessar automaticamente variáveis climáticas do portal INMET, como temperatura, umidade e precipitação, que influenciam diretamente na tomada de decisão agrícola.

Para validar a eficácia do modelo, foram utilizadas métricas amplamente reconhecidas na área de visão computacional, como precisão, recall, F1-score e mAP (média de precisão média), tanto em IoU 0.5 quanto 0.95.

A análise desses resultados permitiu verificar o desempenho do sistema em cada classe e identificar possíveis limitações, fornecendo subsídios para ajustes futuros e destacando o potencial de aplicação da solução no campo.

4.1 CONFIGURAÇÃO DOS EXPERIMENTOS

Figura 22- Horta montada

O experimento foi realizado em uma horta experimental montada especificamente para o cultivo e monitoramento de tomates, com o objetivo de testar o desempenho do sistema de detecção

em um ambiente controlado, porém representativo de condições reais. Para facilitar a captação de imagens e garantir variações nos ângulos e na incidência de luz, foi adotada uma estrutura com arames presos horizontalmente como um varal, nos quais os galhos dos pés de tomate foram cuidadosamente amarrados.

Essa suspensão permitiu que os frutos ficassem visíveis e estáveis durante a captura das imagens, contribuindo para a formação de um banco de dados mais robusto e diversificado. Foram coletadas imagens de tomates em diferentes estágios de maturação verde, maduro e podre, simulando as variações que ocorrem naturalmente na lavoura e servindo como base de entrada para o treinamento e validação do modelo de detecção.

Essa abordagem buscou testar não apenas a eficácia do modelo, mas também sua capacidade de generalização diante de variações de iluminação, perspectiva e distribuição espacial dos frutos.4.2 resultados em imagens estáticas

4.2.1 TOMATES VERDE

Figura 23- Tomates verdes

Na imagem original, observa-se um conjunto de tomates, dispostos verticalmente e ainda presos à planta. Já na imagem processada, o modelo aplicou caixas delimitadoras em torno dos frutos, classificando-os corretamente como "Verde" e atribuindo pontuações de confiança para cada detecção. Essa comparação evidencia a eficiência do sistema na identificação automática dos tomates verdes,

4.2.2 TOMATES MADUROS

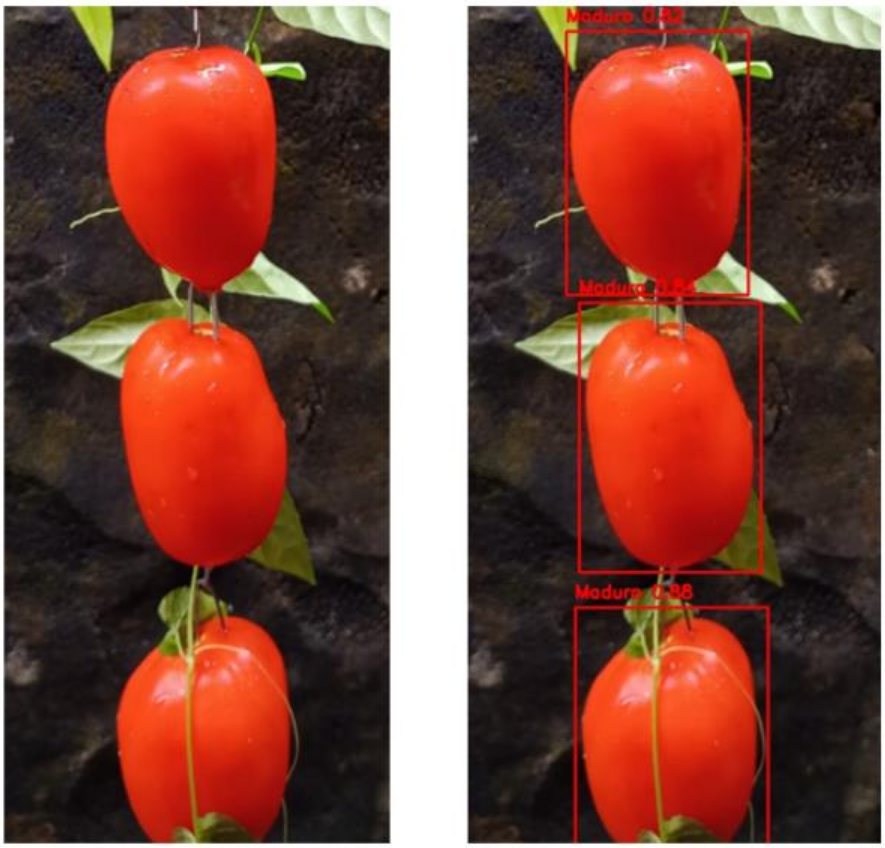

Figura 24-Tomates maduros

Já nesta imagem de tomates maduros, verificamos que o modelo também demonstra alta precisão na detecção. A inferência realizada na imagem processada mostra que todos os tomates foram corretamente identificados como "Maduros", com caixas delimitadoras bem ajustadas e pontuações de confiança elevadas. A acurácia do modelo na classificação reforça sua eficiência para aplicações em monitoramento agrícola, permitindo uma triagem automatizada que pode otimizar a colheita e reduzir erros na separação dos frutos.

4.2.3 TOMATES PODRES

Figura 25 - Tomates podres

Nesta imagem, verificamos que o modelo apresentou um erro de classificação ao identificar um tomate deteriorado como "Maduro" na inferência. Embora a maioria dos tomates podres tenha sido corretamente detectada e marcada com caixas delimitadoras marrons, o último fruto da fileira foi equivocadamente classificado como "Maduro", evidenciado pela caixa vermelha. Essa falha pode ter sido causada por características visuais do fruto, como a coloração avermelhada ainda presente, dificultando a diferenciação entre um tomate maduro e um já em processo de decomposição. Esse resultado indica que o modelo pode ser aprimorado por meio de ajustes na base de dados de treinamento, incluindo mais exemplos de frutos em diferentes estágios de deterioração para minimizar classificações errôneas.

4.2.4 TOMATES VARIADOS

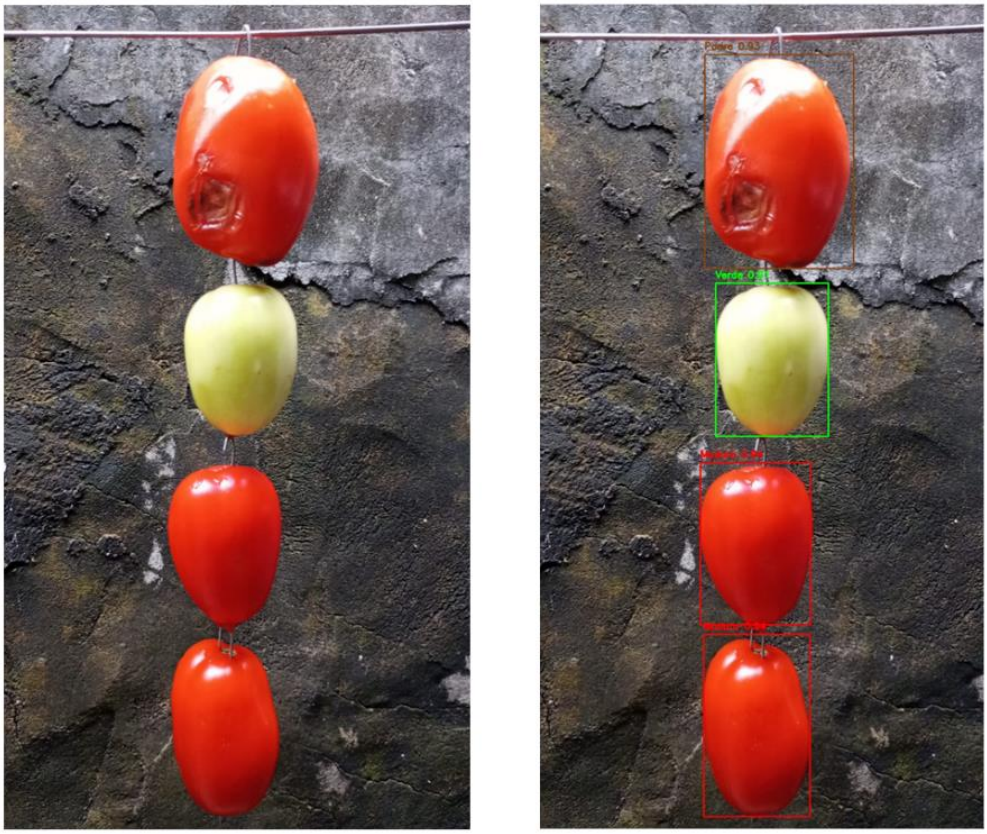

Figura 26 - Tomates variados

Neste caso, analisamos a capacidade do modelo de detecção em identificar corretamente tomates de diferentes classes, incluindo "Podre", "Verde" e "Maduro". A imagem original exibe uma sequência de tomates em distintos estágios de maturação e conservação, enquanto a imagem processada (à direita) apresenta as inferências do modelo. O tomate superior, que apresenta sinais de deterioração, foi corretamente identificado como "Podre", o fruto esverdeado foi classificado como "Verde" e os dois tomates inferiores foram marcados como "Maduros". A presença de diferentes categorias na mesma amostra possibilita avaliar a precisão do modelo ao distinguir cada classe, demonstrando que ele consegue segmentar adequadamente os frutos, mesmo quando dispostos próximos uns dos outros.

Figura 27 - Tomates variados II

Já na abordagem da figura 25, a avaliação do modelo YOLOv8 foi realizada em um cenário mais complexo, onde múltiplos tomates em diferentes estágios de maturação e conservação estão dispostos simultaneamente. O modelo demonstrou uma boa precisão ao diferenciar tomates verdes, destacando-os com caixas verdes, e tomates maduros, marcados com caixas vermelhas. A classificação de tomates podres (caixas marrons) também foi majoritariamente precisa, com alta pontuação de confiança. Entretanto, pequenas inconsistências foram identificadas, como a classificação de alguns frutos em transição entre verde e podre, que podem ter sido categorizados erroneamente devido a variações sutis de coloração.

4.3 RESULTADOS EM VÍDEOS

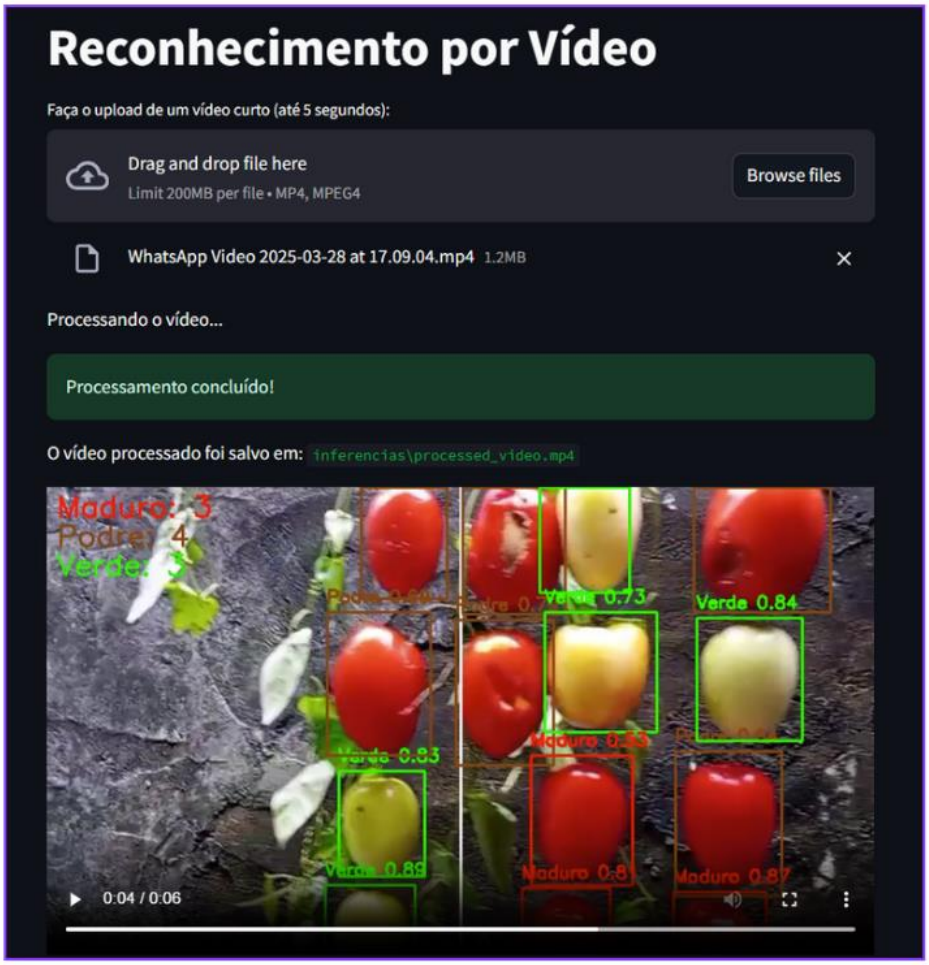

Figura 28 - Reconhecimento da horta em vídeo

Ao contrário das imagens estáticas analisadas anteriormente, a inferência em vídeo apresenta desafios adicionais, como variações na iluminação, movimento da câmera e mudanças na posição dos objetos ao longo dos frames. Esses fatores podem impactar a precisão da detecção, resultando em pequenas inconsistências na classificação.

No exemplo processado, observamos que o modelo conseguiu identificar corretamente a maioria dos tomates, categorizando-os como "Maduro", "Podre" ou "Verde", e exibindo as respectivas pontuações de confiança. No entanto, a precisão geral da inferência em vídeo pode ser afetada por oscilações na qualidade das imagens individuais devido ao desfoque de movimento e à variação da iluminação. Isso pode levar a erros ocasionais na classificação ou à dificuldade em detectar certos objetos com clareza.

Apesar dessas limitações, a inferência em vídeo demonstra a viabilidade da aplicação do modelo em um cenário dinâmico, aproximando-se da realidade operacional de uma colheita automatizada. Melhorias na captura de vídeo, como maior estabilidade da câmera, iluminação controlada e ajustes na calibração do modelo, podem contribuir para um aumento na precisão e confiabilidade da detecção em vídeo.

4.4 VALIDAÇÃO DA ATUALIZAÇÃO CLIMATOLÓGICA

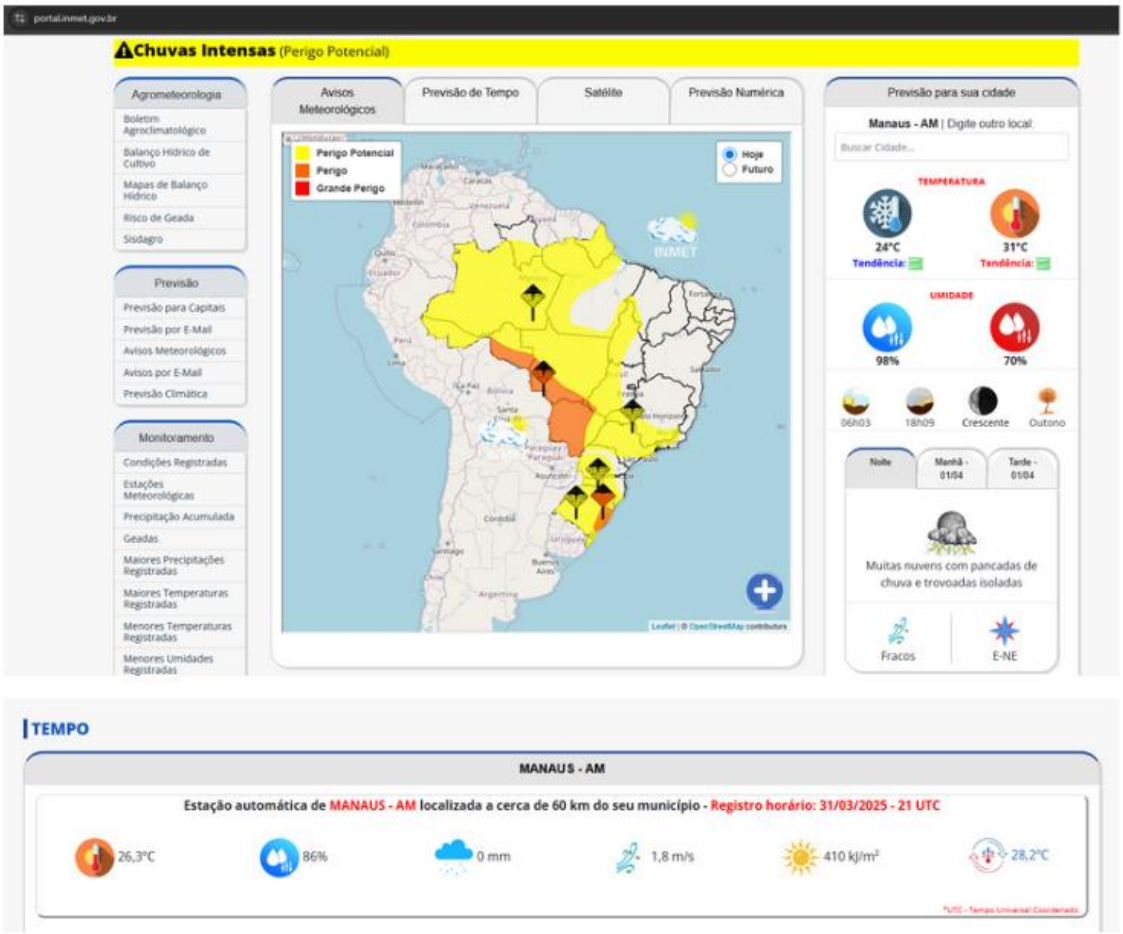

Figura 29- Site do INMET

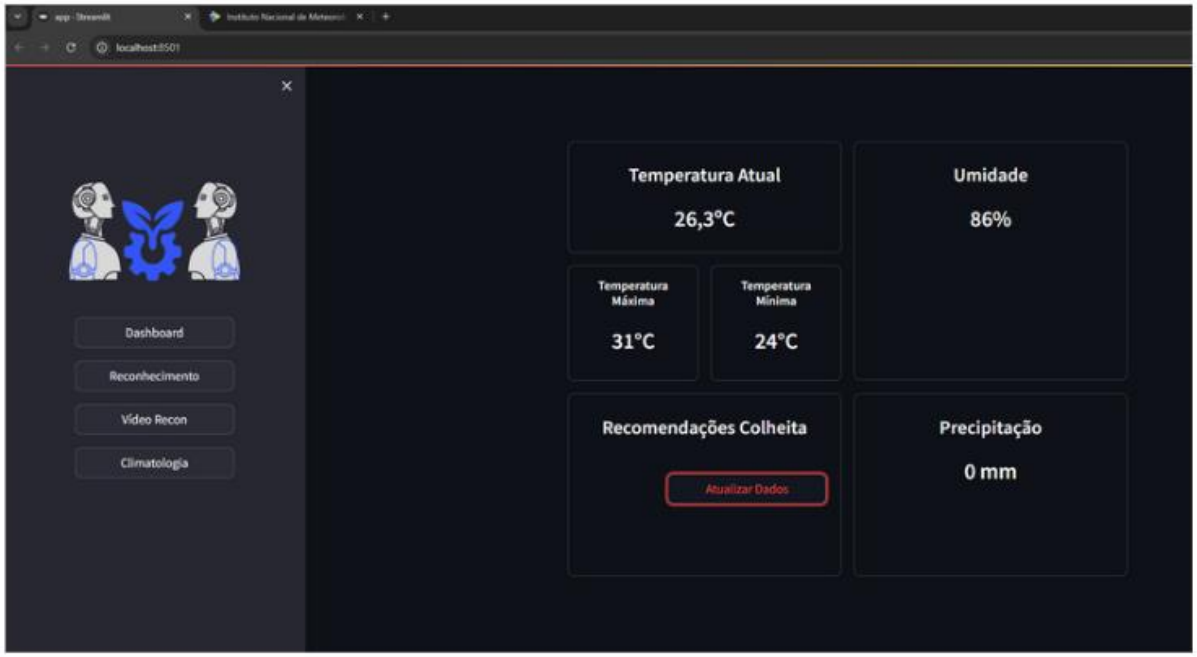

Figura 30 - Seção de climatologia SDT

A validação da atualização climatológica teve como objetivo verificar a consistência entre os dados coletados automaticamente pelo sistema SDT (Sistema de Detecção de Tomates) e as informações divulgadas oficialmente pelo Instituto Nacional de Meteorologia (INMET). Para isso, foi feita uma comparação entre os dados extraídos via automação utilizando Selenium e a interface exibida pelo portal do INMET.

Na Figura 29, observa-se a página oficial do INMET contendo os dados meteorológicos da cidade de Manaus, como temperatura atual de 26,3 °C, umidade relativa do ar em 86%, temperatura mínima de 24 °C e máxima de 31 °C, além de ausência de precipitação. Esses dados servem como referência para validar a precisão da coleta automatizada implementada no sistema.

A Figura 30 mostra a seção de climatologia do sistema SDT, que exibe os mesmos valores obtidos pela automação de web scraping: temperatura atual, temperaturas mínima e máxima, umidade relativa e precipitação. A equivalência dos dados entre o site oficial e o sistema demonstra que a automação está funcionando corretamente, garantindo que os usuários tenham acesso a informações atualizadas e confiáveis diretamente na interface da aplicação, sem necessidade de consulta manual ao site.

4.5 ANÁLISE QUANTITATIVA DETALHADA

Este tópico apresenta uma avaliação quantitativa aprofundada do desempenho do modelo treinado utilizando a arquitetura YOLOv8. Por meio de métricas específicas de detecção, como precisão, recall, F1-score, IoU (Intersection over Union) e mAP (Mean Average Precision), é possível compreender a efetividade do modelo na identificação e classificação das diferentes classes de tomates (maduro, verde e podre), além da classe de fundo.

Serão apresentados os principais indicadores de desempenho, a análise individual do IoU por classe, a matriz de confusão com valores percentuais e, por fim, uma análise da possível ocorrência de overfitting durante o processo de treinamento. Essas análises permitem identificar não apenas o desempenho global do modelo, mas também suas limitações e pontos de melhoria, oferecendo uma base sólida para tomadas de decisão futuras.

4.5.1 INDICADORES PRINCIPAIS

Classe	Precisão (%)	Recall (%)	F1-score (%)
Podre	89.9	84.4	87.1
Maduro	89.0	90.1	89.5
Verde	83.9	83.3	83.6

Figura 31- Tabela de métricas

A Tabela 31 apresenta os principais indicadores de desempenho do modelo treinado, considerando as métricas de precisão, recall e F1-score para as classes "Podre", "Maduro" e "Verde". Esses indicadores não apenas avaliam a performance do modelo em termos estatísticos, como também fornecem insights importantes sobre o impacto prático de cada tipo de erro. Em aplicações reais, como a triagem automatizada de tomates, compreender o equilíbrio entre acertos e falhas é fundamental para garantir a eficiência operacional, reduzir desperdícios e assegurar a qualidade final do produto.

A métrica de precisão se destaca na classe “Podre”, que atingiu 89,9%, indicando que a grande maioria dos tomates classificados como podres realmente estavam deteriorados. Isso é relevante para evitar o descarte indevido de tomates saudáveis, otimizando os custos do processo.

Por outro lado, a classe “Verde” obteve a menor precisão (83,9%), o que pode esté relacionado à semelhança visual com o fundo das imagens.

O recall foi mais elevado na classe “Maduro” (90,1%), o que demonstra que o modelo consegue identificar a maioria dos tomates maduros corretamente, aspecto crucial em aplicações como colheita seletiva ou rotulagem automática. A classe “Podre” apresentou recall de 84,4%, sugerindo que alguns tomates deteriorados não foram identificados — uma falha que pode impactar na segurança alimentar ou na estética do lote.

Já o F1-score, que representa o equilíbrio entre precisão e recall, foi mais alto para a classe “Maduro” (89,5%), evidenciando o bom desempenho do modelo nessa categoria. A classe “Verde”, com F1-score de 83,6%, destaca uma necessidade de aperfeiçoamento, especialmente no treinamento do modelo e na diversidade do conjunto de dados, visando reduzir os erros de classificação relacionados ao fundo da imagem.

4.5.2 MEAN AVERAGE PRECISION COM LIMIARES

Classe	mAP@0.5 (%)	mAP@0.95 (%)
Maduro	89.0	68.2
Verde	83.9	56.0
Podre	89.9	61.4

Figura 32- Tabela de métricas II

Os valores de mAP (mean Average Precision) por classe consideram dois limiares distintos de IoU (Intersection over Union), que é uma métrica utilizada para avaliar a sobreposição entre a caixa predita pelo modelo e a caixa real. O IoU é calculado como a razão entre a área de interseção e a área de união dessas duas caixas, resultando em um valor entre 0 e 1. Quanto maior o IoU, maior é a coincidência entre as caixas. O termo "limiar de IoU" refere-se ao valor mínimo exigido dessa sobreposição para que uma detecção seja considerada correta. No caso do mAP@0.5, utiliza-se um limiar de 0.5, ou seja, uma detecção é considerada correta se a interseção entre as caixas for de pelo menos 50%. Esse limiar é mais tolerante e permite certa imprecisão na localização do objeto.

Já o mAP@0.95 corresponde, na verdade, à média dos mAPs calculados em múltiplos limiares que vão de 0.5 a 0.95 (com incremento de 0.05), exigindo um nível muito maior de

precisão na sobreposição. Esses indicadores são amplamente utilizados na avaliação de modelos de detecção de objetos, como o YOLOv8, pois sintetizam o desempenho do modelo tanto na localização precisa quanto na classificação correta dos objetos detectados.

No limiar de mAP@0.5, todas as classes obtiveram resultados bastante elevados, demonstrando que o modelo é capaz de identificar corretamente a maioria dos objetos quando uma sobreposição mínima é aceita. As classes "Podre" e "Maduro" apresentaram os melhores resultados, com 89,9% e 89,0%, respectivamente, enquanto a classe "Verde" ficou levemente abaixo, com 83,9%. Esse desempenho sugere que, em geral, o modelo consegue localizar bem os tomates nas imagens, especialmente aqueles mais visualmente distintos, como os maduros e podres.

Entretanto, ao analisar o mAP@0.95, observa-se uma queda significativa nos valores para todas as classes, o que é esperado, dada a exigência de maior precisão na correspondência espacial. A classe "Maduro" ainda se destacou, com 68,2%, indicando que as predições dessa classe tendem a estar bem alinhadas com as anotações reais. Já as classes "Podre" e "Verde" obtiveram 61,4% e 56,0%, respectivamente, revelando que o modelo encontra maior dificuldade em gerar caixas perfeitamente ajustadas para essas categorias.

4.5.3 MATRIZ DE CONFUSÃO

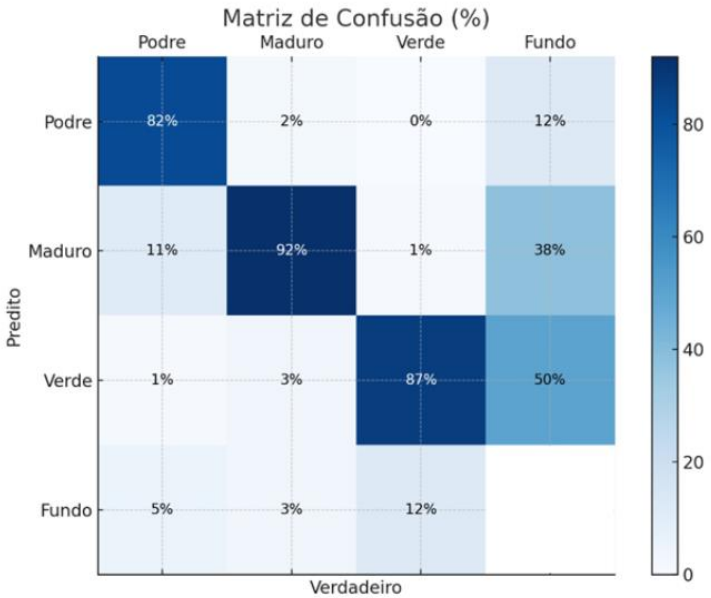

Figura 33 - Matriz de confusão

A matriz de confusão apresentada na Figura 33 fornece uma visão detalhada das classificações corretas e incorretas realizadas pelo modelo para cada classe. A diagonal principal representa as predições corretas, e observa-se um alto percentual de acertos para as classes Maduro (92%), Verde (87%) e Podre (82%). Esses valores confirmam a boa capacidade do modelo em identificar corretamente os tomates nas respectivas categorias. No entanto, também são perceptíveis alguns erros de classificação, como no caso da classe Maduro, que foi confundida com Fundo em 38% das ocorrências, indicando que em algumas situações o modelo falhou em detectar corretamente a presença de tomates maduros.

Outro ponto de atenção é a classe Verde, que apresentou 50% de confusões com o fundo da imagem, o maior índice de erro entre todas as classes. Esse resultado sugere que o modelo encontra dificuldade para distinguir tomates verdes de elementos do ambiente como folhas ou ramos possivelmente devido à similaridade de cores e texturas. Já a classe Podre também sofreu com confusões com o fundo (12%) e em menor grau com a classe Maduro (2%). Essas observações reforçam a importância de estratégias como melhoria do contraste de fundo e balanceamento do conjunto de dados, visando reduzir ambiguidade visual e, consequentemente, elevar a precisão do modelo em situações reais.

É importante destacar que, no contexto da matriz de confusão do YOLOv8, a categoria “Fundo” não representa uma classe rotulada explicitamente no treinamento. Na prática, ela corresponde aos casos em que não houve detecção alguma por parte do modelo, ou seja, quando uma classe de tomate foi completamente ignorada. Assim, as confusões com o fundo indicam falhas na inferência, em que o modelo não foi capaz de identificar a presença de tomates, mesmo quando estavam visíveis. Essa representação é útil para avaliar a sensibilidade do modelo, já que valores elevados de falsos negativos associados ao “fundo” podem sinalizar a necessidade de ajustes no treinamento, seja pela ampliação da base de dados ou pelo refinamento do processo de anotação.

4.5.4 ANÁLISE DE OVERFITTING

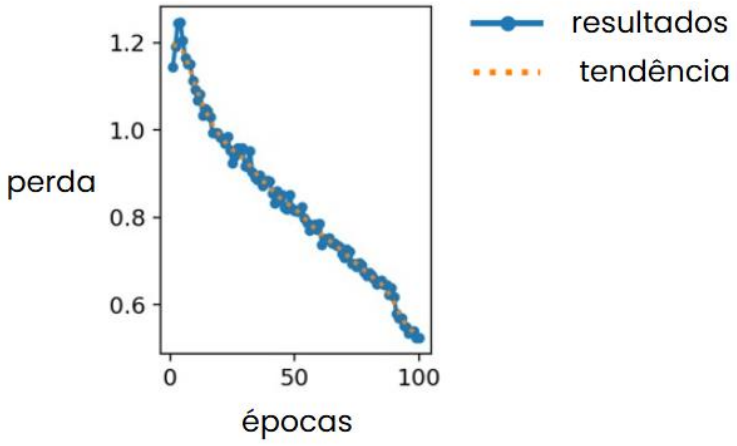

Figura 34- Gráfico de perda por épocas

A figura 34 apresenta a evolução da função de perda do modelo durante as 100 épocas de treinamento. No gráfico, é possível observar duas curvas: uma azul com marcadores, representando os valores reais obtidos a cada época (legenda: resultados), e uma curva laranja tracejada, que indica a tendência suavizada desses valores (legenda: suavizados). Essa distinção entre os dados brutos e a curva suavizada permite uma análise mais clara da trajetória de aprendizado do modelo, reduzindo o impacto visual de pequenas flutuações naturais que ocorrem durante o treinamento.

A curva azul mostra uma redução consistente da função de perda, saindo de um valor inicial superior a 1.2 e chegando abaixo de 0.6 ao final das 100 épocas. Isso indica que o modelo foi, progressivamente, ajustando seus pesos para minimizar os erros de predição, o que é um comportamento esperado em processos de aprendizado bem-sucedidos. A linha laranja, por sua vez, confirma a tendência descendente ao suavizar essas variações e mostrar uma trajetória estável de convergência.

O fato de a curva de perda continuar caindo de forma suave, sem apresentar estagnações ou aumento repentino nas últimas épocas, é um forte indicativo de que não houve overfitting durante o treinamento. Em cenários com overfitting, é comum observar uma queda contínua na perda de treino, acompanhada por um aumento na perda de validação — o que não foi o caso aqui, conforme também verificado em outras métricas como mAP, precisão e recall. O alinhamento entre os resultados reais e os suavizados também reforça a estabilidade do processo de treinamento, sem sinais de flutuações severas.

4.6 DISCUSSÃO DOS RESULTADOS

Os resultados obtidos nas análises quantitativas demonstram que o modelo foi capaz de atingir os objetivos estabelecidos, apresentando desempenho satisfatório tanto em termos de localização quanto de classificação dos objetos, conforme demonstrado pelas métricas de precisão, recall, F1-score e mAP.

As classes “Maduro” e “Podre” se destacaram por apresentarem os melhores indicadores, com F1-scores superiores a 87% e mAP@0.5 acima de 89%. Esses resultados indicam que o modelo conseguiu distinguir essas classes de forma eficaz, mesmo diante de variações de iluminação, ângulos ou qualidade das imagens. A análise da curva de perda ao longo das épocas confirmou que o modelo apresentou um processo de aprendizado estável, sem indícios de overfitting, o que é um aspecto essencial para garantir a capacidade de generalização em cenários reais.

A classe “Verde” apresentou os menores valores em todas as métricas avaliadas, o que sugere maior dificuldade do modelo em identificar corretamente tomates nesse estágio de maturação. A matriz de confusão reforça esse cenário, revelando que boa parte dos erros nessa classe ocorreram devido à confusão com o fundo da imagem. Essa limitação aponta para a necessidade de ajustes no conjunto de dados como o aumento de imagens representativas da classe verde ou a aplicação de técnicas de realce de contraste, além de estratégias de pré-processamento mais eficazes para mitigar ambiguidades visuais.

Conclui-se que o sistema proposto atendeu de forma satisfatória aos objetivos de monitoramento e recomendação inteligente na colheita de tomates, aliando a acurácia de um modelo de visão computacional treinado com YOLOv8 à integração com dados climáticos confiáveis. A sinergia entre inferência visual e dados ambientais permitiu um suporte mais robusto à decisão agrícola, oferecendo ao produtor uma ferramenta de fácil uso, eficiente e com potencial de adaptação contínua. A validação conjunta de desempenho técnico e funcionalidade reforça a aplicabilidade prática do sistema em cenários reais de produção.

5. CONCLUSÃO

O presente trabalho demonstrou a aplicabilidade da visão computacional e da inteligência artificial na agricultura de precisão, por meio do desenvolvimento de um sistema completo para o monitoramento e gestão da colheita de tomates. A utilização do modelo YOLOv8 permitiu a detecção e classificação automática dos frutos em três estágios de maturação maduro, verde e podre, validando o uso de técnicas modernas de aprendizado de máquina para resolver problemas recorrentes na cadeia produtiva agrícola.

A integração do modelo de IA com uma interface interativa desenvolvida em Python utilizando o framework Streamlit possibilitou a análise em tempo real de imagens e vídeos, oferecendo uma solução prática e acessível ao usuário. A contagem e categorização dos frutos foi disponibilizada por meio de um dashboard estatístico, permitindo uma visualização clara e precisa da produção. Esse recurso atendeu ao objetivo de desenvolver um sistema integrado de fácil operação, que auxilia na tomada de decisões em campo.

Adicionalmente, foi implementado um módulo de web scraping responsável pela coleta automática de dados climatológicos diretamente do site do Instituto Nacional de Meteorologia (INMET), armazenando as informações em formato JSON. Essa funcionalidade permitiu correlacionar variáveis ambientais com a produção agrícola, contribuindo para um planejamento mais assertivo.

Outro avanço significativo foi a criação de mecanismos de armazenamento do histórico das inferências realizadas, possibilitando o acompanhamento temporal da produção e a exportação desses dados em formato CSV. Isso permite registrar o desempenho ao longo do tempo e facilita a análise de dados por técnicos e gestores agrícolas.

A estrutura modular do sistema, organizada entre frontend (dashboard), backend (processamento IA e dados climáticos) e banco de dados, garantiu a integração eficiente entre os componentes, validando todos os objetivos específicos propostos neste trabalho.

Embora o sistema tenha apresentado resultados satisfatórios, algumas limitações foram identificadas, como a menor precisão na identificação de tomates verdes atribuída, possivelmente, à baixa representação dessa classe no conjunto de treinamento e às variações de iluminação nas imagens. Melhorias podem ser alcançadas com a ampliação da base de dados, ajuste fino dos hiper parâmetros e a futura implementação de dispositivos automatizados de captura, como drones ou câmeras fixas.

Sua robustez ainda é limitada quando exposto a ambientes externos reais, onde fatores como variações de luz natural, presença de sombras, interferência de outros elementos visuais e movimentações não previstas podem comprometer a precisão das detecções. Esses desafios indicam a necessidade de realizar testes mais amplos em campo, com diferentes condições climáticas e horários do dia.

A implementação do sistema em dispositivos embarcados, como drones agrícolas, câmeras móveis ou microcontroladores com capacidade computacional reduzida, impõe restrições relacionadas ao processamento em tempo real e consumo energético. Para tornar o modelo viável nesses contextos, seria necessário adotar versões otimizadas da arquitetura, reduzir a complexidade do pipeline e empregar estratégias de compressão de modelo ou quantização, sem comprometer significativamente a acurácia.

Para trabalhos futuros, sugere-se a ampliação do sistema para operar com múltiplas culturas agrícolas, permitindo a adaptação do modelo de detecção para diferentes tipos de frutos e vegetais. A implementação de dispositivos embarcados, como drones ou robôs agrícolas equipados com câmeras, poderia possibilitar a coleta autônoma de imagens em grandes áreas, reduzindo a necessidade de intervenção manual. Outra abordagem promissora seria o uso de técnicas de aprendizado por reforço para otimizar a tomada de decisão na colheita, ajustando automaticamente os parâmetros do modelo com base em novas condições ambientais e sazonais.

Conclui-se, portanto, que os objetivos específicos foram plenamente alcançados, e que o sistema desenvolvido representa uma solução promissora para a automação da colheita e o aprimoramento da produtividade agrícola. Espera-se que, com a continuidade das pesquisas e avanços tecnológicos, essa proposta possa ser ampliada para outras culturas, contribuindo com uma agricultura mais inteligente, eficiente e sustentável.

REFERÊNCIAS

BOCHKOVSKIY, Alexey; WANG, Chien-Yao; LIAO, Hong-Yuan Mark. YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv preprint arXiv:2004.10934, 2020. Disponível em: https://arxiv.org/abs/2004.10934. Acesso em: 21 fev. 2025.

BOSTROM, Nick; YUDKOWSKY, Eliezer. The Ethics of Artificial Intelligence. In: Cambridge Handbook of Artificial Intelligence, 2014. p. 44-66. Disponível em: https://www.nickbostrom.com/ethics/artificial-intelligence.pdf. Acesso em: 21 fev. 2025.

GE, Zheng; LIU, Songtao; WANG, Feng; LI, Zeming; SUN, Jian. YOLOX: Exceeding YOLO Series in 2021. arXiv preprint arXiv:2107.08430, 2021. Disponível em: https://arxiv.org/abs/2107.08430. Acesso em: 21 fev. 2025.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. Deep Learning. Cambridge: MIT Press, 2016. Disponível em: https://www.deeplearningbook.org. Acesso em: 21 fev. 2025.

GU, Jiuxiang; WANG, Zhenhua; KAUR, Vijay; ERGUN, Suleyman; SONG, Yi. Recent advances in convolutional neural networks. Pattern Recognition, v. 77, p. 67-85, 2018. Disponível em: https://www.sciencedirect.com/science/article/pii/S0031320318300561. Acesso em: 21 fev. 2025.

KAMILARIS, Andreas; PRENAFETA-BOLDÚ, Francesc X. Deep learning in agriculture: A survey. Computers and Electronics in Agriculture, v. 147, p. 70–90, abr. 2018. ISSN 0168-1699. Disponível em: https://doi.org/10.1016/j.compag.2018.02.016.

KHAN, Asifullah; SOHAIL, Asim; ZAHOOR, Shahid; QURESHI, Bilal. A survey of the recent architectures of deep convolutional neural networks. Artificial Intelligence Review, v. 53, n. 8, p. 1089-1107, 2020. Disponível em: https://link.springer.com/article/10.1007/s10462-020-09825-6. Acesso em: 21 fev. 2025.

KHALEEL, Mohamed; JEBREL, Abdullatif. Artificial Intelligence in Computer Science. International Journal of Electrical Engineering and Sustainability (IJEES), v. 2, n. 2, p. 01–21, jun.

2024. ISSN 2959-9229. Disponível em: https://ijees.org/index.php/ijees/index. Acesso em: 12 fev. 2025.

KHALEEL, Mohamed; JEBREL, Abdullatif. Artificial Intelligence in Computer Science. International Journal of Electrical Engineering and Sustainability, v. 2, n. 2, p. 01–21, 2024. Disponível em: https://ijees.org/index.php/ijees/article/view/80. Acesso em: 21 fev. 2025.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. Nature, v. 521, n. 7553, p. 436-444, 2015. Disponível em: https://www.nature.com/articles/nature14539. Acesso em: 21 fev. 2025.

LECUN, Yann; BOTTOU, Léon; BENGIO, Yoshua; HAFFNER, Patrick. Gradient-based learning applied to document recognition. Proceedings of the IEEE, v. 86, n. 11, p. 2278-2324, 1998. Disponível em: https://ieeexplore.ieee.org/document/726791. Acesso em: 21 fev. 2025.

LIU, X.; ZHANG, J.; WANG, L. Applications of deep learning in precision agriculture: A review. Computers and Electronics in Agriculture, v. 209, p. 2055-2072, 2023. Disponível em: https://www.sciencedirect.com/science/article/pii/S0168169923002072. Acesso em: 21 fev. 2025.

PYTHON. Python. Disponível em: https://www.python.org/. Acesso em: 08 abr. 2024.

REDMON, Joseph; DIVVALA, Santosh; GIRSHICK, Ross; FARHADI, Ali. You Only Look Once: Unified, Real-Time Object Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, p. 779-788, 2016. Disponível em: https://arxiv.org/abs/1506.02640. Acesso em: 21 fev. 2025.

RUSSELL, Stuart; NORVIG, Peter. Artificial Intelligence: A Modern Approach. 4. ed. Harlow: Pearson, 2021. p. 98.

BISHOP, Christopher M. **Pattern Recognition and Machine Learning**. Springer, 2006. Disponível em: https://link.springer.com/book/10.1007/978-0-387-31073-2. Acesso em: 21 fev. 2025.

SHARMA, Anil; GUPTA, Rajeev; PATHAK, Hirdesh. Applications of computer vision in agriculture: A review. Artificial Intelligence in Agriculture, v. 3, p. 134-148, 2019. Disponível em: https://www.sciencedirect.com/science/article/pii/S258972171930011X. Acesso em: 21 fev. 2025.

SILVA, J. R.; OLIVEIRA, M. A.; PEREIRA, L. F. Agroindústria e Desenvolvimento Econômico: Uma Análise Contemporânea. São Paulo: Editora Agrícola, 2020. p. 35.

SZELISKI, Richard. Computer Vision: Algorithms and Applications. 2. ed. London: Springer, 2022. p. 9. Disponível em: https://link.springer.com/book/10.1007/978-1-4471-7397-3. Acesso em: 21 fev. 2025.

RUDER, Sebastian. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747, 2016. Disponível em: https://arxiv.org/abs/1609.04747. Acesso em: 21 fev. 2025.

QIAN, N. On the momentum term in gradient descent learning algorithms. *Neural Networks*, v. 12, n. 1, p. 145-151, 1999. Disponível em: https://doi.org/10.1016/S0893-6080(98)00116-6. Acesso em: 21 fev. 2025.

YOLO. You Only Look Once: Real-Time Object Detection. Disponível em: https://pjreddie.com/darknet/yolo/. Acesso em: 21 fev. 2025.

YOLOV8 ARCHITECTURE. Deep Dive into Its Architecture. Disponível em: https://yolov8.org/yolov8-architecture/. Acesso em: 21 fev. 2025.

ROBOFLOW UNIVERSE. Roboflow Universe. Disponível em: https://universe.roboflow.com/. Acesso em: 08 março 2025.

ROBOFLOW. Roboflow. Disponível em: https:// roboflow.com/. Acesso em: 08 março 2025.

ULTRALYTICS YOLOV8. Ultralytics yolov8. Disponível em: https://github.com/ultralytics/ultralytics/blob/main/docs/en/models/yolov8.md. Acesso em: 08 março 2025.

STREAMLIT. Streamlit. Disponível em: https://streamlit.io/. Acesso em: 08 março 2025.

EXCALIDRAW. Excalidraw. Disponível em: https://excalidraw.com/. Acesso em: 08 março 2025.

AGRISHOW DIGITAL. Cenário do tomate no Brasil: tendências e dificuldades de cultivo. 2021. Disponível em: https://digital.agrishow.com.br/artigos/cenrio-do-tomate-no-brasil-tendncias-e-dificuldades-de-cultivo/. Acesso em: 25 mar. 2025.

APÊNDICE

A.1 CÓDIGO DA APLICAÇÃO.

Repositório do projeto: https://gitlab.com/Paulo-fernds/tcc-project.git

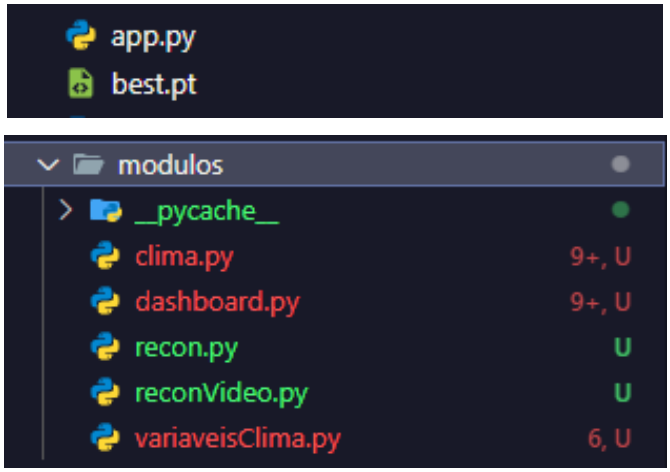

Figura 35- Sistema de pastas do projeto

App.py:

```
import streamlit as st

from modules.dashboard import render_dashboard

from modules.clima import render_clima

from modules.recon import render_recon

from modules.reconVideo import render_recon_video



button_style = """

    <style>

    .stButton > button {

        display: block;
```

```
        margin: 0 auto;
        width: 200px;  /* Defina a largura desejada */
        height: 16px;  /* Defina a altura desejada */
        font-size: 1px;  /* Defina o tamanho da fonte */
    }
    </style>
"""
# Aplica o estilo para centralizar botões
st.sidebar.markdown(button_style, unsafe_allow_html=True)

st.sidebar.image("assets/logo__1_-removebg-preview.png")

# Define a página inicial como o Dashboard ao iniciar o aplicativo
if "module" not in st.session_state:
    st.session_state.module = "Dashboard"

# Botões de navegação
if st.sidebar.button("Dashboard"):
    st.session_state.module = "Dashboard"
```

```
if st.sidebar.button("Reconhecimento"):
    st.session_state.module = "Reconhecimento"

if st.sidebar.button("Vídeo Recon"):
    st.session_state.module = "Vídeo Recon"

if st.sidebar.button("Climatologia"):
    st.session_state.module = "Climatologia"

# Renderizar a página com base no valor de `st.session_state.module`
if st.session_state.module == "Dashboard":
    render_dashboard()  # Renderizar o dashboard

elif st.session_state.module == "Reconhecimento":
    render_recon()

elif st.session_state.module == "Vídeo Recon":
    render_recon_video()

elif st.session_state.module == "Climatologia":
```

```
    render_clima()



elif st.session_state.page == "Climatologia":
    render_clima()
```

dashboard.py:

```
import streamlit as st
import matplotlib.pyplot as plt
import json
import os

def render_dashboard():
    # === Carregar inferências do JSON ===
    json_path = "dados_inferencias.json"
    dados = []
    if os.path.exists(json_path):
        with open(json_path, "r", encoding="utf-8") as f:
            dados = json.load(f)

    # === Definir valores padrão ===
    count_ripe = 0
    count_rotten = 0
    count_green = 0
    total_tomatoes = 0

    # === Se houver dados, selecionar a inferência com base na seleção do selectbox ===
    if dados:
            opcoes = [f"{evento['data_hora']} (Total: {evento['total_tomates']})" for evento in
reversed(dados)]
```

```
    selecionado = st.session_state.get("historico_selector", opcoes[0])  # valor padrão

    # Encontrar o evento correspondente
    idx = opcoes.index(selecionado)
    evento = list(reversed(dados))[idx]

    count_ripe = evento['maduros']
    count_rotten = evento['podres']
    count_green = evento['verdes']
    total_tomatoes = evento['total_tomates']
else:
    st.warning("Nenhuma inferência registrada ainda.")
    count_ripe = st.session_state.get('count_ripe', 0)
    count_rotten = st.session_state.get('count_rotten', 0)
    count_green = st.session_state.get('count_green', 0)
    total_tomatoes = count_ripe + count_rotten + count_green

# === Layout principal ===
with st.container(border=True):
    st.markdown(
        """
        <h1 style="text-align: center;">Informações Gerais</h1>
        """,
        unsafe_allow_html=True
    )

    row1 = st.columns(2)
    row2 = st.columns(2)

    # === Coluna 1: Valores ===
    with row1[0]:
```

```
with st.container():
    row2_1 = st.columns(1)
    row2_2 = st.columns(2)

    with row2_1[0]:
        with st.container(height=140):
            st.markdown(
                f"""
                    <h4 style="text-align: center;">Total de Tomates</h4>
                    <h3 style="text-align: center;">{total_tomatoes}</h3>
                    """,
                unsafe_allow_html=True
            )

    with row2_2[0]:
        with st.container(height=145):
            st.markdown(
                f"""
                <h7 style="text-align: center;">Tomates Verdes</h7>
                <h4 style="text-align: center;">{count_green}</h4>
                """,
                unsafe_allow_html=True
            )

    with row2_2[1]:
        with st.container(height=145):
            st.markdown(
                f"""
                <h7 style="text-align: center;">Tomates Podres</h7>
                <h4 style="text-align: center;">{count_rotten}</h4>
                """,
```

```
                unsafe_allow_html=True
            )

    # === Coluna 2: Gráfico ===
    with row1[1]:
        with st.container(height=300):
            labels = ['Maduro', 'Verde', 'Podre']
            sizes = [count_ripe, count_green, count_rotten]
            explode = (0, 0, 0)

            filtered_labels = [label for label, size in zip(labels, sizes) if size > 0]
            filtered_sizes = [size for size in sizes if size > 0]

            if total_tomatoes > 0 and filtered_sizes:
                fig, ax = plt.subplots()
                ax.pie(filtered_sizes, explode=explode[:len(filtered_sizes)], labels=filtered_labels,
                       autopct='%1.1f%%', startangle=90)
                ax.axis('equal')
                st.pyplot(fig)
            else:
                st.markdown("<h5 style='text-align: center;'>Nenhum tomate encontrado</h5>",
unsafe_allow_html=True)

    # === Linha 2 - Coluna 1: Maduro ===
    with row2[0]:
        with st.container(height=230):
            st.markdown(
                f"""
                <h4 style="text-align: center;">Tomates Maduros</h4>
                <h3 style="text-align: center;">{count_ripe}</h3>
                """,
```

```
            unsafe_allow_html=True
        )

    # === Linha 2 - Coluna 2: Histórico de Eventos com selectbox + botão exportar CSV ===
    with row2[1]:
        with st.container(height=230):
            st.markdown(
                """
                <h4 style="text-align: center;">Histórico de Eventos</h4>
                """,
                unsafe_allow_html=True
            )

            if dados:
                import pandas as pd

                    opcoes = [f"{evento['data_hora']} (Total: {evento['total_tomates']})" for evento in reversed(dados)]
                                selecionado = st.selectbox("Selecione uma inferência:", opcoes, key="historico_selector")

                # Botão para exportar o histórico para CSV
                df = pd.DataFrame(dados)
                csv = df.to_csv(index=False, sep=';').encode('utf-8')

                st.download_button(
                    label=" Exportar Histórico para CSV",
                    data=csv,
                    file_name='historico_inferencias.csv',
                    mime='text/csv',
                    use_container_width=True
```

```
            )

            st.markdown("<hr>", unsafe_allow_html=True)

            ultimos = dados[-5:] if len(dados) > 5 else dados
            for evento in reversed(ultimos):
                st.markdown(
                    f"""
                    <div style="padding: 5px; border-bottom: 1px solid #444;">
                        <strong>{evento['data_hora']}</strong><br>
                        Total: {evento['total_tomates']}<br>
                                Maduro: {evento['maduros']}, Podre: {evento['podres']}, Verde: {evento['verdes']}
                    </div>
                    """,
                    unsafe_allow_html=True
                )
        else:
            st.markdown("<p style='text-align: center;'>Nenhum histórico encontrado.</p>", unsafe_allow_html=True)
```

recon.py:

```
import streamlit as st
from ultralytics import YOLO
from pathlib import Path
from tempfile import NamedTemporaryFile
import cv2
import json
from datetime import datetime
import os
```

```
# Carrega o modelo
model = YOLO('best.pt')

# Dicionários para labels e cores
labels = {0: "Podre", 1: "Maduro", 2: "Verde"}
colors = {0: (19, 69, 139), 1: (0, 0, 255), 2: (0, 255, 0)}

def salvar_em_json(count_ripe, count_rotten, count_green):
    total = count_ripe + count_rotten + count_green
    data_hora = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    novo_registro = {
        "data_hora": data_hora,
        "total_tomates": total,
        "maduros": count_ripe,
        "podres": count_rotten,
        "verdes": count_green
    }

    json_path = "dados_inferencias.json"

    # Se o arquivo existir, carregar e adicionar. Caso contrário, criar novo.
    if os.path.exists(json_path):
        with open(json_path, "r", encoding="utf-8") as f:
            dados_existentes = json.load(f)
    else:
        dados_existentes = []

    dados_existentes.append(novo_registro)

    with open(json_path, "w", encoding="utf-8") as f:
```

```
        json.dump(dados_existentes, f, indent=4, ensure_ascii=False)

def predict(image_path):
    results = model.predict(image_path, save=False, imgsz=640, conf=0.6)

    count_ripe = 0
    count_rotten = 0
    count_green = 0

    img = cv2.imread(image_path)

    for result in results:
        boxes = result.boxes
        for box in boxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            class_id = int(box.cls[0])
            confidence = float(box.conf[0])

            label_text = f"{labels.get(class_id, 'Desconhecido')} {confidence:.2f}"
            color = colors.get(class_id, (255, 255, 255))

            cv2.rectangle(img, (x1, y1), (x2, y2), color, 2)
             cv2.putText(img, label_text, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, color,
2)

            if class_id == 1:
                count_ripe += 1
            elif class_id == 0:
                count_rotten += 1
            elif class_id == 2:
                count_green += 1
```

```
    result_path = 'result.jpg'
    cv2.imwrite(result_path, img)

    st.session_state['count_ripe'] = count_ripe
    st.session_state['count_rotten'] = count_rotten
    st.session_state['count_green'] = count_green

    # Salva os dados no JSON
    salvar_em_json(count_ripe, count_rotten, count_green)

    return count_ripe, count_rotten, count_green

def render_recon():
    with st.container(height=600):
        st.markdown(
            """
            <h2 style="text-align: center;">Reconhecimento</h2>
            """,
            unsafe_allow_html=True
        )
        uploaded_file = st.file_uploader("Escolha uma Imagem", type=["jpg", "jpeg", "png"])

        if uploaded_file is not None:
            with NamedTemporaryFile(delete=False, suffix='.jpg') as tmp_file:
                tmp_file.write(uploaded_file.read())

            count_ripe, count_rotten, count_green = predict(tmp_file.name)

            Path(tmp_file.name).unlink()
```

```
        st.image('result.jpg', caption='Detecções com rótulos', use_column_width=True)
        st.write(f'Número de tomates maduros: {count_ripe}')
        st.write(f'Número de tomates podres: {count_rotten}')
        st.write(f'Número de tomates verdes: {count_green}')
```

reconVideo.py:

```
import cv2
from ultralytics import YOLO
import streamlit as st
import os
import tempfile
import json
from datetime import datetime

# Carrega o modelo YOLO
model = YOLO('best.pt')

# Função para calcular a distância euclidiana entre dois pontos
def calculate_distance(p1, p2):
    return ((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)**0.5

# Função para processar o vídeo
def process_video_with_tracking(video_path):
    # Verifica se a pasta 'inferencias' existe; caso contrário, cria
    output_folder = "inferencias"
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Define o caminho do vídeo de saída
```

```
output_path = os.path.join(output_folder, "processed_video.mp4")

# Carrega o vídeo
cap = cv2.VideoCapture(video_path)
fps = int(cap.get(cv2.CAP_PROP_FPS))
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Configura o escritor de vídeo com codec H.264
writer = cv2.VideoWriter(
    output_path,
    cv2.VideoWriter_fourcc(*'avc1'),  # Codec H.264
    fps,
    (frame_width, frame_height)
)

colors = {0: (19, 69, 139), 1: (0, 0, 255), 2: (0, 255, 0)}
labels = {0: "Podre", 1: "Maduro", 2: "Verde"}
line_x = frame_width // 2

total_ripe, total_rotten, total_green = 0, 0, 0
crossed_ids = set()
tracked_objects = {}
next_object_id = 0
max_distance = 50
max_frames_missing = 5

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
```

```
results = model.predict(frame, save=False, imgsz=640, conf=0.5)
cv2.line(frame, (line_x, 0), (line_x, frame_height), (255, 255, 255), 2)

current_objects = []
for result in results:
    for box in result.boxes:
        x1, y1, x2, y2 = map(int, box.xyxy[0])
        class_id = int(box.cls[0])
        confidence = float(box.conf[0])
        center_x, center_y = (x1 + x2) // 2, (y1 + y2) // 2
        current_objects.append((center_x, center_y, x1, y1, x2, y2, class_id, confidence))

updated_objects = {}
for obj in current_objects:
    center_x, center_y, x1, y1, x2, y2, class_id, confidence = obj
    matched_id = None
    min_distance = max_distance
    for object_id, tracked_obj in tracked_objects.items():
        tracked_center = tracked_obj['center']
        distance = calculate_distance((center_x, center_y), tracked_center)
        if distance < min_distance:
            matched_id = object_id
            min_distance = distance

    if matched_id is not None:
        updated_objects[matched_id] = {
            'center': (center_x, center_y),
            'bbox': (x1, y1, x2, y2),
            'class_id': class_id,
            'confidence': confidence,
```

```
            'frames_missing': 0
        }
        if matched_id not in crossed_ids and line_x - 5 <= center_x <= line_x + 5:
            crossed_ids.add(matched_id)
            if class_id == 1:
                total_ripe += 1
            elif class_id == 0:
                total_rotten += 1
            elif class_id == 2:
                total_green += 1
    else:
        updated_objects[next_object_id] = {
            'center': (center_x, center_y),
            'bbox': (x1, y1, x2, y2),
            'class_id': class_id,
            'confidence': confidence,
            'frames_missing': 0
        }
        next_object_id += 1

for object_id in tracked_objects.keys():
    if object_id not in updated_objects:
        tracked_objects[object_id]['frames_missing'] += 1
        if tracked_objects[object_id]['frames_missing'] <= max_frames_missing:
            updated_objects[object_id] = tracked_objects[object_id]

tracked_objects = updated_objects

for object_id, tracked_obj in tracked_objects.items():
    x1, y1, x2, y2 = tracked_obj['bbox']
    class_id = tracked_obj['class_id']
```

```
            confidence = tracked_obj['confidence']
            color = colors.get(class_id, (255, 0, 255))
            label = f"{labels.get(class_id, 'Desconhecido')} {confidence:.2f}"
            cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
            cv2.putText(frame, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2)

        cv2.putText(frame, f"Maduro: {total_ripe}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
        cv2.putText(frame, f"Podre: {total_rotten}", (10, 60), cv2.FONT_HERSHEY_SIMPLEX, 1, (19, 69, 139), 2)
        cv2.putText(frame, f"Verde: {total_green}", (10, 90), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

        writer.write(frame)

    cap.release()
    writer.release()

    # Atualiza as contagens no `st.session_state`
    st.session_state['count_ripe'] = total_ripe
    st.session_state['count_rotten'] = total_rotten
    st.session_state['count_green'] = total_green

    # Salva os dados no JSON
    salvar_em_json(total_ripe, total_rotten, total_green)

    return output_path

def salvar_em_json(count_ripe, count_rotten, count_green):
    total = count_ripe + count_rotten + count_green
    data_hora = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
```

```
    novo_registro = {
        "data_hora": data_hora,
        "total_tomates": total,
        "maduros": count_ripe,
        "podres": count_rotten,
        "verdes": count_green
    }

    json_path = "dados_inferencias.json"

    # Se o arquivo existir, carregar e adicionar. Caso contrário, criar novo.
    if os.path.exists(json_path):
        with open(json_path, "r", encoding="utf-8") as f:
            dados_existentes = json.load(f)
    else:
        dados_existentes = []

    dados_existentes.append(novo_registro)

    with open(json_path, "w", encoding="utf-8") as f:
        json.dump(dados_existentes, f, indent=4, ensure_ascii=False)

# Renderiza a interface de reconhecimento por vídeo
def render_recon_video():
    st.title("Reconhecimento por Vídeo")
      uploaded_file = st.file_uploader("Faça o upload de um vídeo curto (até 5 segundos):", type=["mp4"])

    if uploaded_file:
        with tempfile.NamedTemporaryFile(delete=False, suffix=".mp4") as temp_input:
```

```
            temp_input.write(uploaded_file.read())
            input_video_path = temp_input.name

        st.write("Processando o vídeo...")
        output_path = process_video_with_tracking(input_video_path)

        st.success("Processamento concluído!")
        st.write(f"O vídeo processado foi salvo em: `{output_path}`")

        # Exibe o vídeo processado
        st.video(output_path)
```

variaveisClima.py:

```
import time
import json
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

def coletar_dados_climaticos():
    """Coleta dados climáticos do site e salva em um arquivo JSON."""
    driver_path = "C:/chromedriver-win64/chromedriver.exe"
    service = Service(driver_path)

    chrome_options = Options()
    chrome_options.add_argument("--headless")
    chrome_options.add_argument("--disable-gpu")
    chrome_options.add_argument("--window-size=1920,1080")
```

```
    driver = webdriver.Chrome(service=service, options=chrome_options)

    try:
        # Acessar o site
        driver.get("https://portal.inmet.gov.br/")

        # Buscar cidade
        busca_cidade = WebDriverWait(driver, 20).until(
                EC.presence_of_element_located((By.CSS_SELECTOR,  "input[placeholder='Buscar Cidade...']"))
        )
        busca_cidade.send_keys("Manaus")

        # Selecionar a sugestão
        sugestao = WebDriverWait(driver, 10).until(
                        EC.presence_of_element_located((By.XPATH,  "//ul[contains(@class,  'ui-autocomplete')]/li"))
        )
        sugestao.click()

        # Aguardar o carregamento da área
        WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.ID, "previsao"))
        )

        # Capturar a linha (row)
        row_elem = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.XPATH, "//div[@class='row' and contains(@style, 'margin-bottom:30px')]"))
        )
```

```
        # Capturar temperatura mínima
        temperatura_minima_elem = WebDriverWait(driver, 10).until(
                                                    EC.presence_of_element_located((By.XPATH,
"//*[@id='previsao']/div[1]/div[2]/div[2]/div[1]/b"))
        )
        temperatura_minima = temperatura_minima_elem.text.strip()

        # Capturar temperatura máxima
        temperatura_maxima_elem = WebDriverWait(driver, 10).until(
                                                    EC.presence_of_element_located((By.XPATH,
"//*[@id='previsao']/div[1]/div[2]/div[3]/div[1]/b"))
        )
        temperatura_maxima = temperatura_maxima_elem.text.strip()

        # Iterar pelas 3 primeiras colunas
        cols = row_elem.find_elements(By.CLASS_NAME, "col")[:3]
        temperatura = cols[0].text.strip() if len(cols) > 0 else None
        umidade = cols[1].text.strip() if len(cols) > 1 else None
        precipitacao = cols[2].text.strip() if len(cols) > 2 else None

        # Salvar os dados em JSON
        dados = {
            "temperatura": temperatura,
            "temperatura_minima": temperatura_minima,
            "temperatura_maxima": temperatura_maxima,
            "umidade": umidade,
            "precipitacao": precipitacao,
            "timestamp": time.strftime("%Y-%m-%d %H:%M:%S")
        }
```

```
        with open("dados_climaticos.json", "w", encoding="utf-8") as json_file:
            json.dump(dados, json_file, ensure_ascii=False, indent=4)

    except Exception as e:
        print(f"Erro ao capturar os dados: {e}")
    finally:
        driver.quit()
```

clima.py:

```
import streamlit as st
import json
from modulos.variaveisClima import coletar_dados_climaticos

def carregar_dados_json():
    """Carrega os dados do arquivo JSON."""
    try:
        with open("dados_climaticos.json", "r", encoding="utf-8") as json_file:
            dados = json.load(json_file)
        return dados
    except FileNotFoundError:
        return {
            "temperatura": "Dados indisponíveis",
            "temperatura_maxima": "Dados indisponíveis",
            "temperatura_minima": "Dados indisponíveis",
            "umidade": "Dados indisponíveis",
            "precipitacao": "Dados indisponíveis"
        }

def render_clima():
    # Carregar os dados do JSON
    dados = carregar_dados_json()
```

```
temperatura = dados.get("temperatura", "Dados indisponíveis")
temperatura_maxima = dados.get("temperatura_maxima", "Dados indisponíveis")
temperatura_minima = dados.get("temperatura_minima", "Dados indisponíveis")
umidade = dados.get("umidade", "Dados indisponíveis")
precipitacao = dados.get("precipitacao", "Dados indisponíveis")

# Construção do layout Streamlit
with st.container():
    row1 = st.columns(2)
    row2 = st.columns(2)

    # Adicionar conteúdo à primeira coluna da primeira linha
    with row1[0]:
        with st.container():
            row2_1 = st.columns(1)
            row2_2 = st.columns(2)
            with row2_1[0]:
                with st.container(height=140):
                    st.markdown(
                        f"""
                        <h4 style="text-align: center;">Temperatura Atual</h4>
                        <h3 style="text-align: center;">{temperatura}</h3>
                        """,
                        unsafe_allow_html=True
                    )
            with row2_2[0]:
                with st.container(height=145):
                    st.markdown(
                        f"""
                        <h6 style="text-align: center;">Temperatura Máxima</h6>
                        <h3 style="text-align: center;">{temperatura_maxima}</h3>
```

```
                """,
                unsafe_allow_html=True
            )
        with row2_2[1]:
            with st.container(height=145):
                st.markdown(
                f"""
                <h6 style="text-align: center;">Temperatura Mínima</h6>
                <h3 style="text-align: center;">{temperatura_minima}</h3>
                """,
                unsafe_allow_html=True
            )
with row1[1]:
    with st.container(height=300):
        st.markdown(
            f"""
            <h4 style="text-align: center;">Umidade</h4>
            <h3 style="text-align: center;">{umidade}</h3>
            """,
            unsafe_allow_html=True
        )
# Adicionar conteúdo à primeira coluna da segunda linha
with row2[0]:
    with st.container(height=230):
        st.markdown(
            """
            <h4 style="text-align: center;">Recomendações Colheita</h4>
            """,
            unsafe_allow_html=True
        )
        row5 = st.columns(3)
```

```
        with row5[0]:
            with st.container():
                st.markdown(
                    """
                    <h4 style="text-align: center;"></h4>
                    """,
                    unsafe_allow_html=True
                )
        with row5[1]:
            with st.container():
                st.markdown(
                    """
                    <h4 style="text-align: center;"></h4>
                    """,
                    unsafe_allow_html=True
                )
                if st.button("Atualizar Dados"):
                    coletar_dados_climaticos()
                    #st.success("Dados climáticos atualizados com sucesso!")
        with row5[2]:
            with st.container():
                st.markdown(
                    """
                    <h4 style="text-align: center;"></h4>
                    """,
                    unsafe_allow_html=True
                )
# Adicionar conteúdo à segunda coluna da segunda linha
with row2[1]:
    with st.container(height=230):
```

```
            st.markdown(
                f"""
                <h4 style="text-align: center;">Precipitação</h4>
                <h3 style="text-align: center;">{precipitacao}</h3>
                """,
                unsafe_allow_html=True
            )
    pass
```

dados_climaticos.json:


```
{
    "temperatura": "26,1ºC",
    "temperatura_minima": "25°C",
    "temperatura_maxima": "34°C",
    "umidade": "88%",
    "precipitacao": "0 mm",
    "timestamp": "2025-02-03 20:53:58"
}
```