Pipeline Interpretável para Detecção de Doenças Foliares em Tomateiro com Inferência em Borda via Rust/ONNX

Felipe Schandler Silva[1], **Victor R. Rodrigues Luz**[1], **Gabrielly de Queiroz Pereira**[1]
Luciano José Senger[1], **Marcella Scoczynski Ribeiro Martins**[2]

[1]Departamento de Informática – Universidade Estadual de Ponta Grossa (UEPG) – Ponta Grossa, PR – Brasil

{20010826,21010526,gqpereira,ljsenger}@uepg.br

[2]Departamento de Engenharia Elétrica – Universidade Tecnológica Federal do Paraná (UTFPR) – Ponta Grossa, PR – Brasil

marcella@utfpr.edu.br

Abstract. *This work presents a low-cost smart indoor garden for tomato cultivation integrating embedded sensing, actuation and artificial intelligence for foliar disease detection. An ESP32-S3 with an OV2640 camera monitors environmental variables and automates irrigation; leaf images go to a Rust/Axum inference server with ONNX Runtime. A 188-dimensional handcrafted vector is extracted after leaf segmentation in the HSV (hue, saturation, value) colour space. Under StratifiedGroupKFold with hash deduplication, XGBoost reached macro F1 96.19% on 4,803 test samples, with 76.7% of its gain attributable to named colour descriptors; MobileNetV3-Small reached 99.02%. Rust cut median latency to 13.2 ms from 24.5 ms ($p < 0.001$), with a 14× faster inference step.*

Resumo. *Este trabalho apresenta uma horta indoor inteligente de baixo custo para o cultivo de tomateiro, integrando sensoriamento embarcado, atuação e Inteligência Artificial para detecção de doenças foliares. Um ESP32-S3 com câmera OV2640 monitora variáveis ambientais e automatiza a irrigação; imagens foliares seguem para um servidor Rust/Axum com ONNX Runtime. Extrai-se um vetor de 188 atributos após segmentação da folha no espaço de cor HSV (matiz, saturação e valor). Sob StratifiedGroupKFold com deduplicação por hash, o XGBoost obteve macro F1 96,19% em 4.803 amostras de teste, com 76,7% do ganho atribuível a descritores de cor nomeados; a MobileNetV3-Small, 99,02%. O Rust reduziu a latência mediana de 24,5 ms para 13,2 ms ($p < 0{,}001$), com inferência 14× mais rápida.*

1. Introdução

A agricultura contemporânea enfrenta o desafio de elevar a produtividade com uso mais eficiente de água e energia, o que motiva a adoção da agricultura digital e de precisão [Bassoi et al. 2019]. No tomateiro, essa necessidade é ampliada pela elevada exigência hídrica [Campos 2019] e pela suscetibilidade a doenças foliares [Zimmermann et al. 2025], exigindo monitoramento contínuo do ambiente e do estado

fitossanitário. Muitas soluções disponíveis apresentam infraestrutura complexa, o que limita sua adoção por pequenos produtores e pela agricultura urbana [Navarro et al. 2020].

A convergência entre automação agrícola, Internet das Coisas (IoT) e visão computacional — com redes neurais convolucionais (CNN) e Inteligência Artificial (IA) aplicadas ao diagnóstico foliar — é revisada na Seção 2 [Azevedo 2025, p. 106–109].

Diante desse cenário, este trabalho propõe uma horta inteligente para o cultivo de tomateiro integrando automação agrícola, IoT e IA. O sistema utiliza um ESP32-S3 com câmera, sensores ambientais, atuação automática de irrigação, e um *pipeline* de classificação foliar implantado com inferência em ONNX Runtime (ONNX, do inglês *Open Neural Network Exchange*) sobre um *backend* Rust. O objetivo é oferecer uma solução reprodutível para monitoramento ambiental e apoio à decisão no manejo do cultivo de tomateiros, conciliando interpretabilidade, desempenho e viabilidade de implantação em *hardware* de baixo custo.

2. Trabalhos Relacionados

Classificação de doenças foliares: descritores manuais e aprendizado profundo. No subconjunto de tomateiro do PlantVillage, a literatura recente é dominada por redes convolucionais e arquiteturas híbridas, com acurácias reportadas acima de 97% [Sharma et al. 2025]. O trabalho mais próximo é o de [Bouni et al. 2024], que funde 43 descritores manuais com atributos profundos de uma MobileNetV2 sobre 7.311 imagens em dez classes, atingindo 88,84% de acurácia. Diferimos em três pontos: usamos 188 atributos, acrescentando Zernike, padrões binários locais (LBP) uniformes e histogramas em Lab; mantemos os paradigmas raso e profundo *separados* em vez de fundidos, para quantificar o custo da interpretabilidade; e reportamos o custo de implantação, ausente naquele estudo.

Viés de captura e protocolo de avaliação no PlantVillage. A confiabilidade dos números publicados nesse conjunto de dados é objeto de crítica documentada. [Mohanty et al. 2016] relatam que a acurácia de 99,35% obtida no PlantVillage cai para cerca de 31% em imagens de outras origens, e [Barbedo 2018] atribui parte desse abismo às condições controladas de aquisição. [Noyan 2022] quantificou o fenômeno: um classificador treinado *apenas* com oito pixels do fundo atinge 49,0% de acerto em 38 classes, contra 2,6% do acaso; a origem dominante, porém, não é o fundo, mas o *viés de captura*, que contamina também o primeiro plano — de modo que removê-lo **não** elimina o viés. Soma-se um risco de vazamento raramente tratado: a base contém múltiplas capturas da mesma folha física. Entre os trabalhos comparáveis que levantamos, nenhum declara particionamento por grupo, deduplicação ou exclusão de derivadas sintéticas do teste; vários aumentam os dados *antes* da partição, o que insere variantes de amostras de teste no treino (Tabela 1).

Os valores da Tabela 1 não são diretamente comparáveis. As demais linhas reportam *acurácia*, métrica mais otimista que a *macro F1* em conjuntos desbalanceados (8,6:1 aqui), e nenhuma declara o controle de vazamento adotado neste trabalho. Nossos resultados devem, portanto, ser lidos como estimativas conservadoras sob protocolo mais restritivo, e não como desempenho inferior.

Sistemas integrados e inferência em borda. A integração de sensoriamento embarcado,

Tabela 1. Resultados publicados para classificação de doenças foliares de tomateiro no PlantVillage (10 classes) e o protocolo de partição declarado.

Trabalho (ano)	Abordagem	Partição	Métrica
[Bouni et al. 2024] (2024)	Manuais + MobileNetV2 (fusão MI)	Aleat. 75/25 + SMOTE	Acur. 88,84%
[Laiskodat and Rahardi 2025] (2025)	Textura + cor + forma, XGBoost	Não declarada	Acur. 97,63%
[Sharma et al. 2025] (2025)	*Ensemble* ResNet50 + MobileNetV2	80/10/10	Acur. 99,91%
Este trab. (raso)	188 atrib. manuais + XGBoost	**Grupo + *dedup.***	**F1 96,19%**
Este trab. (profundo)	MobileNetV3-Small	**idem**	**F1 99,02%**

atuação e conectividade para o cultivo protegido é campo consolidado [Azevedo 2025]. O trabalho mais próximo nesse eixo é o de [Saxena et al. 2025], que combina nós ESP32 com um *gateway* Raspberry Pi executando YOLOv8 — topologia igual à nossa, porém aplicada à maturação do fruto e não ao diagnóstico fitossanitário (mAP 52,8%). A execução do diagnóstico próximo ao ponto de captura é requisito reconhecido para operação sem conectividade estável. [Malche et al. 2025] demonstraram TinyML inteiramente dentro de um ESP32-S3 (94,6% em cinco classes); Nenhum deles reporta conjuntamente latência e memória absolutas para uma pilha de serviço completa, nem documenta a validação da transição entre implementações — lacuna característica da distância entre protótipo e sistema em operação. Não localizamos trabalho revisado por pares que empregue Rust como servidor de inferência fitossanitária; a evidência disponível limita-se a comparações de linguagens em microcontroladores [Plauska et al. 2022].

Lacuna e contribuições. O panorama acima indica que classificar doenças foliares de tomateiro no PlantVillage não é, isoladamente, um problema em aberto. A lacuna está na *combinação*: os trabalhos que alcançam os maiores índices não declaram controle de vazamento nem custo de implantação; os que medem custo de implantação não empregam pipelines auditáveis; e os que integram IoT com visão computacional avaliam outra tarefa ou reportam apenas ganhos relativos. Diante disso, este trabalho contribui com: **(i)** um protocolo de avaliação com controle explícito de vazamento — partição por grupo, teste composto *exclusivamente* por imagens-fonte, deduplicação por *hash* e auditoria do vazamento residual por identidade de folha, quantificado em 3,21% do teste e com efeito medido de $+0{,}04$ ponto percentual (p.p.) na macro F1 —, sob o qual reportamos 96,19% de macro F1, valor que deve ser lido como estimativa conservadora e não diretamente comparável às acurácias obtidas com partição aleatória; **(ii)** a caracterização do custo de implantação de um *pipeline* interpretável, com decomposição de latência que identifica a extração de atributos, e não a inferência, como gargalo dominante (12,5 de 13,9 ms); **(iii)** a migração de Python/Flask para Rust/Axum com ONNX Runtime, avaliada com teste de hipótese ($p < 0{,}001$); e **(iv)** a atribuição quantitativa da decisão do modelo aos grupos de descritores, que sustenta a interpretabilidade reivindicada.

3. Material e Métodos

O desenvolvimento abrangeu a definição do *hardware*, a implementação do *firmware* embarcado, a construção do *pipeline* de aprendizado de máquina e a integração com o serviço de inferência.

3.1. Hardware e Firmware

O sistema embarcado foi baseado na placa ESP32-S3, equipada com câmera OV2640, selecionada por sua conectividade Wi-Fi integrada e compatibilidade com aplicações de visão computacional. O conjunto de sensores incluiu um sensor analógico de umidade do solo, um sensor de luminosidade do tipo LDR (*light-dependent resistor*) e um sensor de temperatura e umidade do ar DHT22. O atuador sob controle do *firmware* é uma mini-bomba submersa de 5 V, acionada por módulo relé optoacoplado, que garante isolamento elétrico entre o circuito lógico de 3,3 V e a carga; a bomba e uma ventoinha de circulação de ar são alimentadas por fonte de 5 V independente da placa, de modo que a corrente de acionamento não trafega pelos pinos do microcontrolador. A ventoinha permanece energizada de forma contínua e não é comandada por *software*. A umidade do solo e o LDR ocupam entradas analógicas, o DHT22 um pino digital dedicado e o relé da bomba uma saída digital ativa em nível baixo; os demais pinos servem ao barramento paralelo e ao *clock* da câmera. A Figura 1 apresenta o diagrama de ligação completo.

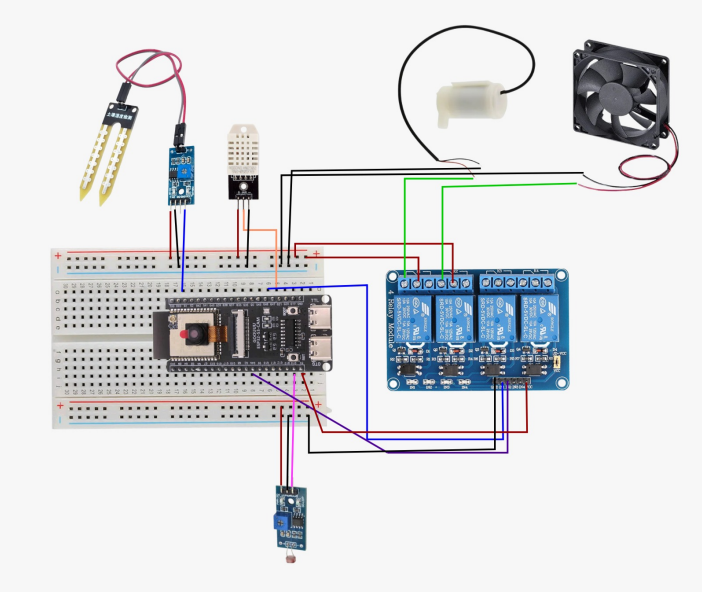


Figura 1. Diagrama de ligação do protótipo. O módulo relé optoacoplado isola o circuito lógico de 3,3 V da carga de 5 V, alimentada por fonte independente; a ventoinha, embora conectada ao módulo, permanece energizada de forma contínua e não é comandada pelo *firmware*.

O *firmware*, em C++ sobre PlatformIO, faz leitura periódica dos sensores (2 s), controle automático da irrigação e transmissão de imagens via HTTP (*Hypertext Transfer Protocol*). A irrigação opera por limiar de umidade do solo com histerese temporal: a bomba é acionada por duração fixa quando a umidade cai abaixo do limiar e um *cooldown* mínimo já decorreu. As demais grandezas têm papéis explícitos: temperatura e umidade do ar (DHT22) são *exclusivamente telemetria*, anexadas como cabeçalhos HTTP para rastreabilidade sem participar de decisão de atuação; a luminosidade (LDR) não comanda atuação climática, mas suprime a captura abaixo de um limiar configurável, evitando inferência sobre imagem subexposta. O nó incorpora uma interface web para acompanhamento operacional, visualização em tempo real, histórico de predições e configuração do servidor de inferência, com armazenamento persistente em memória não volátil (NVS,

non-volatile storage), reduzindo a necessidade de *reflashing* durante testes e migrações de servidor.

3.2. Base de Dados e Pré-processamento

O conjunto de dados utilizado foi o subconjunto de tomateiro do *PlantVillage* [Hughes and Salathé 2015, Mohanty et al. 2016], na variante *color*, com 16.011 imagens RGB de 256×256 px em 10 classes (nove doenças ou pragas e folhas sadias). Não foram utilizadas as variantes *segmented* ou *grayscale*, nem qualquer versão pré-aumentada do conjunto: todo o aumento de dados descrito adiante foi gerado neste trabalho. A distribuição por classe é desbalanceada, com razão de 8,6:1 entre a classe mais frequente (TYLCV, 3.208 imagens) e a menos frequente (ToMV, 373), o que motiva o uso de macro F1, *balanced accuracy* e coeficiente de correlação de Matthews (MCC) como métricas principais, em vez da acurácia global. A Tabela 2 detalha a distribuição em treino e teste.

O PlantVillage não fornece metadados de proveniência (planta, sessão ou lote). Adotou-se como variável de agrupamento o identificador da imagem-fonte — `classe/UUID`, com o UUID extraído do nome do arquivo —, herdado por toda variante derivada. Designamos por *imagem-fonte* a imagem tal como distribuída no PlantVillage, por oposição às cópias sintéticas produzidas neste trabalho pelo aumento de dados. Antes da divisão, imagens-fonte idênticas byte a byte (mesmo *hash* MD5) dentro de uma classe foram fundidas em um grupo (14 ocorrências) e verificou-se que nenhum *hash* ocorre em mais de uma classe. Os 15.997 grupos restantes foram divididos por amostragem estratificada **no nível de grupo** (70/30, semente fixa). O aumento de dados foi aplicado **apenas ao treino e após a partição**: embora as variantes sejam materializadas em disco antes da divisão, o conjunto de teste é formado **exclusivamente** pelas 4.803 imagens-fonte dos grupos de teste, e as 24.015 cópias geradas a partir delas são descartadas. As 67.248 amostras de treino são, portanto, 11.208 imagens-fonte mais 56.040 cópias (cinco por imagem-fonte). Como salvaguarda, comparou-se o *hash* de cada imagem de teste com o de todas as imagens-fonte de treino (0 remoções). A disjunção de grupos é asserção que interrompe o treinamento em caso de falha, com resultados registrados em `split_groups.json`, `leakage_check.json` e `dataset_integrity.json`, publicados junto aos manifests.

O aumento foi produzido com Albumentations 1.4.24 e semente derivada do caminho de cada arquivo-fonte, gerando cinco variantes por imagem por rotação, deslocamento e escala, espelhamento horizontal, perturbação fotométrica (brilho, contraste, gama e HSV) e, com $p = 0{,}25$, desfoque.

O pré-processamento é idêntico no treino e na inferência e consiste em redimensionamento por *letterbox* seguido de segmentação foliar no espaço HSV (matiz, saturação e valor). A máscara retém os pixels com matiz $H \in [15, 95]$ na escala de 8 bits do OpenCV — equivalente a $30°$–$190°$ na escala convencional —, saturação $S \geq 25$ e valor $V \geq 25$ (ambos em $[0, 255]$), aos quais se aplicam abertura e fechamento morfológicos com elemento estruturante elíptico 5×5 e descarte de componentes conexas com área inferior a 1% da imagem; caso nenhuma componente sobreviva, a máscara é mantida integral e a imagem segue sem segmentação. O limiar de saturação é o parâmetro determinante: o fundo acromático padronizado do PlantVillage — tipicamente cinza-neutro — apresenta $S \approx 0$ e é eliminado por qualquer limiar baixo, sendo $S \geq 25$ ($\approx 10\%$ da escala)

uma margem conservadora que tolera reflexos e sombras suaves sem remover tecido foliar. Os dois pipelines partem das mesmas imagens e da mesma partição, mas operam em resoluções distintas e independentes. No *pipeline* de atributos manuais, cada imagem é reduzida a 128×128 px; a resolução foi escolhida por ser suficiente para os descritores empregados — histogramas de cor, momentos de Hu e de Zernike e LBP são pouco sensíveis à resolução, e a matriz de coocorrência de Haralick satura em resoluções baixas — e por reduzir em cerca de quatro vezes o custo de extração, fator determinante para a latência no servidor de borda. A MobileNetV3-Small opera na resolução nativa de seus pesos pré-treinados, 224×224 px, obtida por reamostragem bilinear das mesmas imagens de 128×128 px; não há, portanto, compartilhamento de atributos entre os pipelines, e a rede não recebe informação de alta frequência além da presente em 128×128 px — seu resultado deve ser lido como um limite inferior do que a arquitetura alcançaria na resolução original de 256×256 px.

Tabela 2. Distribuição por classe. "Fonte" são as imagens do PlantVillage; o treino recebe ainda cinco cópias sintéticas por imagem-fonte (56.040 no total). O teste contém *apenas* imagens-fonte.

Classe	Fonte	Tr.	Te.	Classe	Fonte	Tr.	Te.
Mancha bacteriana	2.127	1.489	638	Mancha alvo	1.404	983	421
Pinta preta	1.000	700	300	TYLCV	3.208	2.245	963
Requeima	1.909	1.337	572	ToMV	373	261	112
Mofo foliar	952	666	286	Saudável	1.591	1.114	477
Septoriose	1.771	1.240	531	**Total**	**16.011**	**11.208**	**4.803**
Ácaro rajado	1.676	1.173	503				

3.3. Extração de Características e Modelagem

O *pipeline* de atributos manuais — assim designado por oposição ao *pipeline* profundo fim a fim, que aprende suas próprias representações — baseia-se em descritores explícitos, totalizando 188 dimensões por imagem, organizadas em dez grupos: histogramas HSV (48), momentos de Hu (7), média e desvio-padrão em HSV (6), descritores básicos de forma (6), estatísticas médias em Lab (4), histogramas Lab (16), atributos de cromaticidade (4), Haralick sobre a matriz de coocorrência de níveis de cinza (GLCM, 13), momentos de Zernike (25) e histograma LBP-NRI uniforme (59).

Três classificadores rasos foram treinados sobre o mesmo vetor de 188 atributos, padronizado por *z-score* ajustado apenas no treino: XGBoost [Chen and Guestrin 2016], máquinas de vetores de suporte (SVM) com núcleo de função de base radial (RBF) [Cortes and Vapnik 1995] e floresta aleatória [Breiman 2001]. A seleção de hiperparâmetros usou busca em grade com validação cruzada *StratifiedGroupKFold* de cinco dobras **sobre o conjunto de treino**, com o mesmo `group_id` da partição, de modo que nenhuma variante aumentada de uma imagem apareça simultaneamente na dobra de ajuste e na de validação. O melhor macro F1 de validação cruzada do XGBoost foi 0,9199, *inferior* ao macro F1 de teste (0,9619); a diferença é esperada, pois as dobras de validação contêm imagens aumentadas — rotacionadas, desfocadas e com perturbação fotométrica — enquanto o teste contém apenas imagens-fonte. Esse padrão, de validação mais difícil que o teste, é o oposto do que se observaria sob vazamento entre as partições.

Como *baseline* de visão profunda, treinou-se uma MobileNetV3-Small [Howard et al. 2019] inicializada com pesos pré-treinados no **ImageNet-1k** (*torchvision*, `MobileNet_V3_Small_Weights.IMAGENET1K_V1`), com a última camada do classificador substituída por uma camada linear de dez saídas e ajuste fino de todos os parâmetros, otimizador AdamW com taxa de aprendizado constante de 3×10^{-4}, lotes de 32 e 10 épocas, sobre a mesma partição de grupos do *pipeline* de atributos manuais. O ponto de parada foi escolhido pela melhor macro F1 no próprio conjunto de teste, o que confere caráter otimista a essa estimativa; a MobileNetV3-Small é reportada apenas como referência comparativa, e não como o modelo implantado.

Os modelos selecionados foram exportados para ONNX, com verificação de paridade numérica entre a implementação Python e os artefatos exportados: sobre as 4.803 amostras de teste, a concordância de rótulo é integral e a macro F1 coincide nas duas implementações, com diferença média de probabilidade de $1{,}6 \times 10^{-6}$, atribuível à quantização para *float32* na fronteira de entrada. O grafo exportado embute o próprio normalizador, o que permite ao servidor alimentá-lo diretamente com o vetor de atributos.

3.4. Serviço de Inferência e Avaliação

O serviço de inferência foi implementado em Rust com Axum e ONNX Runtime. As imagens recebidas passam pelo mesmo pré-processamento e extração do treinamento e são classificadas pelo modelo ONNX; a resposta traz classe, pontuação, lista *top-k*, tempos por etapa, indicadores de qualidade e sinalização de confiança.

A inferência diretamente no ESP32 é viável para modelos suficientemente compactos — [Malche et al. 2025] executam TinyML de doenças foliares de tomateiro inteiramente no ESP32-S3 (94,6%, 1,1 MB) sobre cinco classes. A restrição aqui é específica do *pipeline* adotado: as dependências de extração dos 188 atributos (*mahotas*, *scikit-image*) empregam extensões C sem equivalente embarcado, e a MobileNetV3-Small, mesmo quantizada, excede a memória da placa. A arquitetura cliente-servidor preserva ainda a flexibilidade de atualizar o modelo sem *reflashing*.

Foram implementados *quality gates* para reduzir respostas pouco confiáveis — cobertura da máscara e variância do Laplaciano, indicador de desfoque — e as requisições registram metadados do dispositivo e tempos por etapa. A avaliação considerou macro F1, *balanced accuracy*, coeficiente de correlação de Matthews e latência. Para a transição entre implementações foi especificado um protocolo de *shadow mode*: execução paralela dos dois servidores, com cada requisição do nó replicada para a implementação candidata e as respostas comparadas par a par, admitindo-se o corte apenas após taxa de concordância e volume mínimos.

4. Resultados e Discussão

O subsistema físico operou de forma autônoma e estável durante os ensaios: o *firmware* executou leituras periódicas e acionou a bomba quando a umidade do solo cruzou o limiar configurado, com histerese temporal suprimindo reativações; a interface web embarcada permitiu inspecionar sensores, histórico de predições e configuração do servidor sem console serial. Os resultados quantitativos são organizados em três frentes: (i) ganho operacional da migração do servidor; (ii) inferência com imagens reais e sensibilidade

à iluminação; e (iii) comparação entre os classificadores rasos, a *baseline* profunda e a contribuição de cada grupo de descritores.

4.1. Migração do servidor: Python/Flask para Rust/Axum

A migração do *backend* para Rust foi avaliada em *benchmark* controlado com $N = 100$ requisições por servidor (imagens 128×128 px, rede local, 10 requisições de aquecimento descartadas), reportando média $\pm$ desvio padrão, mediana e percentis. Ambos os servidores executaram em contêineres na mesma estação de trabalho (AMD Ryzen 5 7600X, Linux); a pilha Python é gunicorn com um *worker gthread* e quatro *threads* — não o servidor de desenvolvimento do Flask —, e seu caminho de inferência usa *scikit-learn*, sem ONNX Runtime instalado. O servidor Python apresentou latência *wall-clock* de $25{,}1 \pm 5{,}2$ ms (mediana 24,5 ms; p_{95} 30,5 ms), enquanto o Rust apresentou $13{,}9 \pm 2{,}2$ ms (mediana 13,2 ms; p_{95} 18,3 ms). A diferença foi estatisticamente significativa ($p < 0{,}001$, Mann-Whitney U), e o desvio padrão do Rust é 2,4$\times$ menor, indicando latência mais previsível: o 95° percentil do Rust (18,3 ms) é inferior à própria mediana do Python.

A decomposição da latência por etapa mostra que a extração de atributos é similar nas duas implementações (Python: 15,4 ms; Rust: 12,5 ms), pois ambas executam código nativo nessa etapa — o Python delega o cálculo pesado a extensões C (*mahotas*, *scikit-image*, *opencv-python*), evitando o *overhead* do interpretador, enquanto o Rust acessa os mesmos algoritmos via *bindings* C++ (*opencv crate*). O gargalo nessa etapa é algorítmico, não de *runtime*. O ganho do Rust concentra-se inteiramente na inferência: ONNX Runtime (0,4 ms) é 14$\times$ mais rápido que o *scikit-learn* (5,5 ms).

A decomposição evidencia que a extração de atributos tornou-se o gargalo dominante no servidor Rust: os 12,5 ms de extração representam 90% da latência mediana total (13,9 ms). Otimizações adicionais devem, portanto, focar nessa etapa — por exemplo, paralelização de grupos de descritores independentes (Haralick, Zernike e LBP podem ser computados concorrentemente) ou substituição parcial dos descritores manuais por representações aprendidas de uma CNN leve.

Além do ganho de latência, a pilha Rust é distribuída como um binário único de $\sim$27 MB, que dispensa a gestão de um *runtime* interpretado e de suas dependências no destino. Combinado à *cross-compilation* para ARM, isso simplifica a atualização remota (OTA, *over-the-air*) de frotas de nós distribuídos. O consumo de memória residente não difere de forma apreciável entre as duas implementações e, portanto, não constitui argumento a favor de qualquer uma delas neste cenário; a vantagem operacional do Rust concentra-se na latência, na sua previsibilidade e no formato de distribuição.

4.2. Inferência com imagens reais e sensibilidade à iluminação

A câmera OV2640 operou com captura local via Wi-Fi e envio de imagens ao servidor em Rust via HTTP. Em imagens capturadas *in situ* pelo protótipo sob condições de iluminação distintas, o serviço retornou a classe predita, a pontuação, a lista *top-k*, os tempos por etapa e os indicadores de qualidade, com tempo de resposta compatível com aplicações próximas ao tempo real em rede local.

Esses ensaios *in situ* tiveram caráter qualitativo: destinaram-se a validar o caminho completo de captura, transmissão, inferência e resposta, e não a medir acurácia em

campo. Não foi executado um protocolo controlado de exposição de plantas com diferentes doenças à câmera, e a avaliação quantitativa por classe restringe-se ao conjunto de teste do PlantVillage. Em imagens mais escuras ou com menor nitidez, observou-se maior ambiguidade no *top-k*, consistente com a dependência dos descritores de cor e textura em condições fotométricas desfavoráveis; nesses casos, os *quality gates* — cobertura de máscara e variância do Laplaciano — sinalizam a necessidade de reaquisição.

A sensibilidade à iluminação foi quantificada em avaliação controlada: com escurecimento simulado ($\gamma = 1{,}4$), a macro F1 caiu de 96,19% para 90,11%; com redução de brilho de 20 unidades, para 91,57%. Mitigam-na iluminação auxiliar com exposição travada, *data augmentation* fotométrica e reaquisição condicionada pelos *quality gates*.

4.3. Comparação entre classificadores rasos e a baseline profunda

Para contextualizar a escolha do XGBoost, os três classificadores rasos foram treinados sobre o *mesmo* vetor de 188 atributos, com o *mesmo* protocolo de particionamento (*StratifiedGroupKFold*) e avaliados no mesmo conjunto de teste independente de 4.803 amostras.

O XGBoost superou o SVM em macro F1 por $0{,}37$ p.p. e a floresta aleatória por $4{,}9$ p.p., confirmando que a escolha é empiricamente justificada dentro do espaço de classificadores rasos para este conjunto de atributos. O SVM ($C = 10$, `gamma=auto`) apresenta desempenho próximo ao do XGBoost e constitui alternativa viável quando a interpretabilidade das árvores não é requisito. A floresta aleatória, apesar da macro F1 de 91,29%, oferece vantagem em velocidade de treinamento ($\sim 6\times$ mais rápida), tornando-a adequada para iterações exploratórias.

Os resultados mostram dois perfis complementares. O *pipeline* de atributos manuais (XGBoost: 96,19%) permite análise direta dos sinais de cor, textura e forma associados às classes, com custo computacional compatível com atualização frequente do modelo. A MobileNetV3-Small (99,02%) obtém macro F1 superior ao aprender representações fim a fim, porém com menor transparência interna. As duas abordagens oferecem, ademais, formas distintas de interpretabilidade: a MobileNetV3-Small admite explicações *post hoc* por mapas de ativação (por exemplo, Grad-CAM), que indicam *onde* a rede olhou mas não *qual* grandeza física sustentou a decisão, enquanto o *pipeline* de atributos manuais expõe diretamente *quais* descritores nomeados — e em que proporção — determinaram a classe (Tabela 3), permitindo associar o diagnóstico a sinais agronômicos como clorose, necrose e heterogeneidade de textura.

A análise por classe reforça essa leitura. As maiores dificuldades do XGBoost concentram-se em *Early Blight* (90,43%) e *Target Spot* (94,95%), consistente com a maior sutileza visual de lesões iniciais ou necrosantes. Classes como *Healthy* (98,44%) e *Yellow Leaf Curl* (98,13%) mantiveram desempenho elevado.

4.4. Interpretabilidade: contribuição dos grupos de descritores

Cada uma das 188 dimensões corresponde a uma grandeza fotométrica ou geométrica nomeada, o que permite atribuir a decisão do modelo a propriedades verificáveis da folha — diferentemente das ativações internas de uma CNN. A Tabela 3 agrega a importância pelos dez grupos, por duas medidas complementares: o ganho de *split* acumulado pelo *booster* e a queda de macro F1 ao permutar o bloco inteiro no conjunto de teste ($n =$

4803, 10 repetições). Os descritores cromáticos concentram 76,70% do ganho, seguidos pelos de textura (13,82%) e de forma (9,48%). A permutação acentua essa hierarquia: permutar o histograma HSV reduz a macro F1 de 96,19% para 32,93%, enquanto Hu e cromaticidade produzem queda inferior a $0{,}2$ p.p. Individualmente, os atributos de maior ganho são os primeiros *bins* do histograma de matiz (até 4,43% cada) e o desvio-padrão do canal a do espaço Lab (3,65%) — exatamente as grandezas que descrevem clorose, necrose e heterogeneidade cromática das lesões. Isso explica a degradação de 6,08 p.p. sob escurecimento simulado: o modelo apoia-se no sinal mais sensível à variação fotométrica, o que reforça a necessidade dos *quality gates* e da iluminação auxiliar como mitigações de campo.

Tabela 3. Importância dos atributos do XGBoost agregada pelos dez grupos de descritores. *Ganho* é a soma do ganho de *split* do *booster* normalizada para 100%; *Ganho/atr.* é a densidade (ganho do grupo dividido pelo número de atributos); a última coluna é a queda de macro F1 quando o bloco inteiro de atributos é permutado no conjunto de teste ($n = 4803$, 10 repetições, média $\pm$ desvio).

Grupo de descritores	Atr.	Ganho (%)	Ganho/atr. (%)	Δmacro F1
Histograma HSV	48	54,23	1,130	$0{,}633 \pm 0{,}009$
Média/desvio Lab (a, b)	4	8,00	2,000	$0{,}029 \pm 0{,}003$
Histograma Lab (a, b)	16	7,24	0,453	$0{,}015 \pm 0{,}002$
Haralick (GLCM)	13	6,95	0,535	$0{,}039 \pm 0{,}005$
LBP (nri-uniform)	59	6,87	0,116	$0{,}024 \pm 0{,}002$
Momentos de Zernike	25	5,43	0,217	$0{,}015 \pm 0{,}003$
Média/desvio HSV	6	5,28	0,881	$0{,}005 \pm 0{,}001$
Forma (básica)	6	3,00	0,500	$0{,}009 \pm 0{,}001$
Cromaticidade rg	4	1,95	0,486	$0{,}001 \pm 0{,}001$
Momentos de Hu	7	1,06	0,151	$0{,}002 \pm 0{,}001$
Total	**188**	**100,00**	—	—

5. Conclusão

Este trabalho apresentou o desenvolvimento e a validação de uma horta *indoor* que integra automação agrícola, IoT e IA para monitoramento ambiental e diagnóstico de doenças foliares em tomateiro. A solução combinou sensores e atuadores controlados por ESP32-S3, *firmware* com interface web e configuração persistente, e um serviço de inferência em Rust/Axum com ONNX Runtime.

Entre os classificadores rasos avaliados sobre o mesmo vetor de 188 atributos, o XGBoost obteve o maior macro F1, 96,19% (MCC 0,9607); a MobileNetV3-Small atingiu 99,02% como teto com aprendizado fim a fim.

No plano operacional, o *benchmark* controlado ($N = 100$ requisições por servidor) demonstrou que o *backend* Rust reduz a latência mediana para 13,2 ms (contra 24,5 ms do Python), com ganho de $14\times$ na etapa de inferência e dispersão $2{,}4\times$ menor. A decomposição por etapa mostrou ainda que o gargalo dominante deixou de ser o modelo e passou a ser a extração de atributos, responsável por 90% da latência mediana — resultado que redireciona o esforço de otimização nessa família de soluções. A distribuição como binário único de 27 MB, sem *runtime* interpretado a gerenciar no destino, favorece a implantação em nós de borda e a atualização remota de frotas.

Como limitação principal, a validação dos modelos depende integralmente do PlantVillage, conjunto com características controladas cujo viés de captura está documentado [Noyan 2022, Barbedo 2018] e não é integralmente eliminado pela segmentação do fundo. Imagens coletadas diretamente pelo protótipo em condições naturais de campo são ainda insuficientes para avaliação estatística robusta da validade externa dos modelos. Soma-se a isso o vazamento residual por identidade de folha discutido na Seção 2, cujo efeito medido é de $+0{,}04$ p.p. na macro F1, e o caráter otimista da estimativa da MobileNetV3-Small, cujo ponto de parada foi escolhido no próprio conjunto de teste.

Como trabalhos futuros destacam-se: (i) avaliação *cross-dataset*, única forma de verificar se a segmentação foliar mitiga de fato o viés de captura; (ii) *cross-compilation* para ARM com caracterização de latência e energia no próprio *hardware* de borda, já que os tempos aqui reportados foram medidos em estação de trabalho; e (iii) paralelização dos grupos independentes de descritores — Haralick, Zernike e LBP não têm dependência entre si —, atacando o gargalo identificado na Seção 4.1.

Referências

Azevedo, A. T. (2025). *Algoritmos para manejo da irrigação aplicados às metodologias de balanço hídrico e sensores de umidade do solo*. Tese (doutorado em engenharia de sistemas agrícolas), Universidade de São Paulo. DOI: 10.11606/t.11.2025.tde-04062025-103533.

Barbedo, J. G. A. (2018). Factors influencing the use of deep learning for plant disease recognition. *Biosystems Engineering*, 172:84–91. Disponível em: `https://doi.org/10.1016/j.biosystemseng.2018.05.013`.

Bassoi, L. H., Inamasu, R. Y., Bernardi, A. C. d. C., Vaz, C. M. P., Speranza, E. A., and Cruvinel, P. E. (2019). Agricultura de precisão e agricultura digital. *TECCOGS: Revista Digital de Tecnologias Cognitivas*, 20:17–36. DOI: `https://doi.org/10.23925/1984-3585.2019i20p17-36`.

Bouni, M., Hssina, B., Douzi, K., and Douzi, S. (2024). Synergistic use of handcrafted and deep learning features for tomato leaf disease classification. *Scientific Reports*, 14(1):26822. Disponível em: `https://doi.org/10.1038/s41598-024-71225-5`.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32. DOI: 10.1023/A:1010933404324.

Campos, H. d. M. (2019). Automação da irrigação no cultivo de tomate cereja (Solanum lycopersicum var. Cerasiforme) sob conteúdos volumétricos de água no solo. Dissertação (mestrado), Instituto Federal Goiano, Ceres. Sem DOI registrado. Disponível em: `https://sistemas.ifgoiano.edu.br/sgcursos/uploads/anexos_6/2020-02-17-04-35-52Disserta%C3%A7%C3%A3o%20Final%20-%20Hugo%20de%20Moura%20Campos.pdf`.

Chen, T. and Guestrin, C. (2016). XGBoost: a scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York. ACM. DOI: 10.1145/2939672.2939785.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297. DOI: 10.1007/BF00994018.

Howard, A., Sandler, M., Chen, B., Wang, W., Chen, L.-C., Tan, M., Chu, G., Vasudevan, V., Zhu, Y., Pang, R., Adam, H., and Le, Q. (2019). Searching for MobileNetV3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324. IEEE. DOI: 10.1109/ICCV.2019.00140.

Hughes, D. P. and Salathé, M. (2015). An open access repository of images on plant health to enable the development of mobile disease diagnostics. arXiv:1511.08060 [cs.CY]. Disponível em: `https://arxiv.org/abs/1511.08060`. Preprint (v1: 25 nov. 2015; v2: 12 abr. 2016). Artigo que descreve o repositório PlantVillage.

Laiskodat, F. A. and Rahardi, M. (2025). Multiclass classification of tomato leaf diseases using GLCM, color, and shape feature extraction with optimized XGBoost. *Journal of Applied Informatics and Computing*, 9(6):3594–3600. Disponível em: `https://doi.org/10.30871/jaic.v9i6.11273`.

Malche, T., Joshi, M., Upadhyay, G. M., and Soni, P. K. (2025). Automated tomato leaf disease detection and alert system using internet of things and TinyML. *Discover Internet of Things*, 6(1):8. Disponível em: `https://doi.org/10.1007/s43926-025-00257-8`.

Mohanty, S. P., Hughes, D. P., and Salathé, M. (2016). Using deep learning for image-based plant disease detection. *Frontiers in Plant Science*, 7:1419. DOI: 10.3389/fpls.2016.01419.

Navarro, E., Costa, N., and Pereira, A. (2020). A systematic review of IoT solutions for smart farming. *Sensors*, 20(15):4231. DOI: 10.3390/s20154231.

Noyan, M. A. (2022). Uncovering bias in the PlantVillage dataset. Disponível em: `https://arxiv.org/abs/2206.04374`.

Plauska, I., Liutkevičius, A., and Janavičiūtė, A. (2022). Performance evaluation of C/C++, MicroPython, Rust and TinyGo programming languages on ESP32 microcontroller. *Electronics*, 12(1):143. Disponível em: `https://doi.org/10.3390/electronics12010143`.

Saxena, A., Agarwal, A., Nagrath, B., Jayavanth, C. S., Thulasidoss, S., Maheswari, S., and Sasikumar, P. (2025). Deep learning-driven IoT solution for smart tomato farming. *Scientific Reports*, 15(1):31092. Disponível em: `https://doi.org/10.1038/s41598-025-15615-3`.

Sharma, J., Al-Huqail, A. A., Almogren, A., Doshi, H., Jayaprakash, B., Bharathi, B., Rehman, A. U., and Hussen, S. (2025). Deep learning based ensemble model for accurate tomato leaf disease classification by leveraging ResNet50 and MobileNetV2 architectures. *Scientific Reports*, 15(1):13904. Disponível em: `https://doi.org/10.1038/s41598-025-98015-x`.

Zimmermann, G. B., Pellenz, M. E., Costa, Y. M. G., and Britto Jr., A. d. S. (2025). Enhancing disease and pest detection in greenhouse tomato cultivation using advanced machine learning on new dataset of images. *Journal of the Brazilian Computer Society*, 31(1):187–202. DOI: 10.5753/jbcs.2025.4581.