Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Edge AI e MLOps na Cafeicultura: Desenvolvimento de um Sistema IoT Embarcado com ESP32-S3 para Análise Inteligente de Dados

Nahan Filipe Duarte Rezende

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
Prof. Igor Muzetti Pereira

Novembro, 2026
João Monlevade–MG

Nahan Filipe Duarte Rezende

Edge AI e MLOps na Cafeicultura: Desenvolvimento de um Sistema IoT Embarcado com ESP32-S3 para Análise Inteligente de Dados

Orientador: Prof. Igor Muzetti Pereira

Monografia apresentada ao curso de Sistemas de Informação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto

João Monlevade

Novembro de 2026

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTACAO E SISTEMAS

FOLHA DE APROVAÇÃO

Nahan Filipe Duarte Rezende

Edge AI e MLOps na cafeicultura: desenvolvimento de um sistema IoT embarcado com ESP32-S3 para análise inteligente de dados

Monografia apresentada ao Curso de Sistemas de Informação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação

Aprovada em 05 de Março de 2026

Membros da banca

Dr. Igor Muzetti Pereira - Orientador (Universidade Federal de Ouro Preto)
Dr. Alexandre Magno de Souza (Universidade Federal de Ouro Preto)
Dr. Eduardo da Silva Ribeiro (Universidade Federal de Ouro Preto)

Igor Muzetti Pereira, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 19/03/2026

Documento assinado eletronicamente por **Igor Muzetti Pereira**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 19/03/2026, às 15:20, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **1078318** e o código CRC **F1B25DD1**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0819 - www.ufop.br

Este trabalho é dedicado aos meus pais, Júlio Cesar e Nildete; aos meus avós, Francisco e Maria; e aos meus tios, Eduardo e Tania, pelo amor, apoio e incentivo em todos os momentos.

Agradecimentos

Agradeço, primeiramente, a Deus, pela vida, pela saúde e pela força para seguir em frente durante toda esta caminhada.

Aos meus pais e à minha família, pelo apoio incondicional, pela compreensão nos momentos de ausência e por sempre acreditarem em mim, mesmo nos períodos mais desafiadores.

Ao meu orientador, Prof. Igor Muzetti Pereira, pela orientação, pela paciência, pela disponibilidade e pelas contribuições fundamentais para o desenvolvimento deste trabalho.

Aos professores do curso, pelos ensinamentos e pela formação acadêmica e humana ao longo da graduação.

Aos colegas e amigos, pelo companheirismo, pelas trocas de conhecimento e pelo incentivo durante esta etapa.

Por fim, agradeço à Universidade Federal de Ouro Preto e ao Instituto de Ciências Exatas e Aplicadas, pela oportunidade de aprendizado e crescimento, que tornaram possível a realização deste Trabalho de Conclusão de Curso.

Resumo

A cafeicultura enfrenta desafios no manejo eficiente da irrigação, especialmente em propriedades com infraestrutura tecnológica limitada, nas quais práticas manuais ainda podem gerar desperdício de água e baixa previsibilidade operacional; nesse contexto, este trabalho teve como objetivo desenvolver e validar tecnicamente um sistema inteligente de irrigação de baixo custo que integra aprendizado de máquina e hardware embarcado para apoiar a decisão de irrigar em cultivo de café. A metodologia combinou coleta e preparação de dados climáticos históricos em pipeline externo (Python), treinamento de um classificador de Regressão Logística, exportação determinística dos parâmetros para `ia_params.h` e execução de inferência local no ESP32-S3 com sensores ambientais, histerese e regras determinísticas de segurança. Como resultados, observou-se a viabilidade técnica da integração ponta a ponta entre treino, empacotamento e inferência embarcada, com acurácia de 0,9951 e F1 de 0,9970 no cenário controlado de validação funcional adotado. Como contribuição, o trabalho apresenta uma arquitetura replicável de Edge AI para irrigação inteligente na cafeicultura, unindo ML e sistema embarcado de baixo custo com operação local e reduzida dependência de conectividade contínua.

Palavras-chave: irrigação inteligente; ESP32-S3; sistemas embarcados; Edge AI; aprendizado de máquina; MLOps; agricultura de precisão.

Abstract

Coffee farming faces challenges in efficient irrigation management, especially in small and medium properties with limited technological infrastructure, where manual practices may lead to water waste and low operational predictability; in this context, this work aimed to develop and technically validate a low-cost intelligent irrigation system that integrates machine learning and embedded hardware to support irrigation decisions in coffee cultivation. The methodology combined historical climate data collection and preprocessing in an external Python pipeline, training of a Logistic Regression classifier, deterministic parameter export to `ia_params.h`, and on-device inference on an ESP32-S3 with environmental sensors, hysteresis, and deterministic safety rules. Results indicate the technical feasibility of end-to-end integration between training, parameter packaging, and embedded inference, achieving 0.9951 accuracy and 0.9970 F1-score in the adopted controlled functional validation scenario. As a contribution, the study provides a replicable Edge AI architecture for smart irrigation in coffee farming, combining ML and low-cost embedded hardware with local operation and reduced dependence on continuous connectivity.

Key-words: smart irrigation; ESP32-S3; embedded systems; Edge AI; machine learning; MLOps; precision agriculture.

Sumário

1 Introdução

A agricultura moderna enfrenta desafios crescentes relacionados ao manejo eficiente da irrigação, especialmente em culturas sensíveis à disponibilidade hídrica, como o café. A variabilidade climática, a limitação de recursos hídricos e a necessidade de otimização de processos tornam a automação um fator estratégico para aumentar produtividade, reduzir desperdícios e garantir sustentabilidade ambiental (MARIN; SENTELHAS, 2020). Nesse contexto, sistemas inteligentes baseados em sensoriamento e modelos de *machine learning* têm se destacado pela capacidade de apoiar decisões críticas no campo, oferecendo respostas rápidas e adaptativas às condições ambientais.

No contexto agrícola brasileiro, particularmente em regiões produtoras de café, a irrigação ainda é frequentemente conduzida de forma manual ou sem apoio de sistemas inteligentes, o que aumenta o risco de desperdício de água, reduz a eficiência produtiva e pode comprometer a sustentabilidade ambiental da cultura (MARIN; SENTELHAS, 2020). Além disso, pequenas e médias propriedades rurais, que representam grande parte dos produtores de café em Minas Gerais, normalmente apresentam infraestrutura tecnológica limitada, com acesso restrito a redes estáveis, servidores externos ou plataformas sofisticadas de automação agrícola.

Esse cenário impõe desafios para a adoção de soluções digitais mais avançadas e reforça a necessidade de abordagens de Edge AI e MLOps voltadas a dispositivos de baixo custo e processamento local, como o ESP32-S3. Tais soluções permitem reduzir a dependência de conectividade contínua e infraestrutura em nuvem, tornando sistemas inteligentes de irrigação mais viáveis para a realidade da agricultura familiar e de pequenos produtores (ZHANG; LI; CHEN, 2022; DEY; ROY; CHAKRABORTY, 2019).

Apesar do avanço de tecnologias embarcadas e da ampliação de bibliotecas e serviços de *Internet of Things* (IoT), ainda existem lacunas importantes na integração entre sensores de baixo custo, modelos de predição e infraestrutura de automação capaz de operar em ambientes com limitações de conectividade. Microcontroladores como o ESP32-S3 representam uma alternativa promissora, pois permitem a execução embarcada de modelos pré-treinados, dispensando servidores externos para inferência e reduzindo dependências de redes externas (Espressif Systems, 2023). Essa independência é particularmente relevante em ambientes rurais, onde falhas de conectividade são comuns.

Além disso, técnicas de MLOps adaptadas para dispositivos embarcados possibilitam um ciclo contínuo de coleta de dados, re-treinamento, validação e implantação de novos parâmetros diretamente no firmware, garantindo evolução do modelo ao longo do tempo. Estudos recentes mostram que práticas de MLOps têm se consolidado como essenciais

para manter modelos em ambientes dinâmicos, tanto em sistemas distribuídos quanto em aplicações embarcadas (KREUZBERGER; KÜHL; HIRSCHL, 2023). A adoção dessas práticas agrega robustez ao processo e aumenta a confiabilidade do sistema (ZHANG; LI; CHEN, 2022), permitindo que o sistema se adapte às mudanças climáticas regionais e às particularidades de diferentes tipos de solo.

Diante desse cenário, esta pesquisa é orientada por uma questão geral e por questões específicas.

Questão de Pesquisa Geral

Como desenvolver um sistema de irrigação inteligente, de baixo custo e com capacidade de inferência embarcada, que integre dados climáticos e sensores locais para apoiar decisões de irrigação em plantações de café?

Questões Específicas de Pesquisa

- Como integrar, de forma reprodutível, um pipeline externo de treinamento de modelo ao firmware embarcado no ESP32-S3?
- Quais são os ganhos e limitações da combinação entre inferência local e regras determinísticas de segurança (como histerese e bloqueio por chuva) na decisão de irrigação?
- Em que medida uma arquitetura inspirada em MLOps pode apoiar atualização controlada de parâmetros e evolução do sistema em cenários com conectividade limitada?
- Quais desdobramentos metodológicos podem originar estudos futuros específicos (por exemplo, validação agronômica em campo, ampliação de variáveis e comparação entre modelos)?

A partir dessas questões, o presente trabalho propôs o desenvolvimento de um sistema inteligente de irrigação baseado no microcontrolador ESP32-S3, integrando sensores ambientais, um modelo preditivo de regressão logística e um pipeline simplificado de MLOps para atualização embarcada (*OTA*). O sistema foi projetado tendo como foco inicial plantações de café no estado de Minas Gerais, mantendo flexibilidade para adaptação a outros contextos agrícolas.

A motivação empírica, técnica e social converge, portanto, na busca por um sistema prático, eficiente e replicável, capaz de combinar ML, hardware embarcado e dados ambientais para apoiar a tomada de decisão de irrigação com maior autonomia operacional no campo.

O restante deste trabalho está organizado da seguinte forma: o Capítulo 2 apresenta a Fundamentação Teórica, discutindo conceitos básicos e trabalhos correlatos relacionados à irrigação inteligente, IoT agrícola, Edge AI e MLOps. O Capítulo 3 descreve detalhadamente a metodologia adotada, incluindo o processo de sensoriamento e aquisição de dados, o pipeline externo de treinamento, a integração ao firmware embarcado e os procedimentos de teste e validação. O Capítulo 4 apresenta o desenvolvimento do sistema proposto, detalhando a arquitetura, a estrutura do firmware na ESP32-S3, a leitura dos sensores, a previsão de chuva, o módulo de inteligência artificial, o controle da bomba de irrigação e os principais *trade-offs* de implementação. Por fim, o Capítulo 5 apresenta os resultados obtidos e a validação funcional da arquitetura proposta.

De forma geral, os resultados mostraram a viabilidade técnica da solução proposta: no pipeline externo, o modelo de Regressão Logística alcançou acurácia de 0,9951 e F1-score de 0,9970 no cenário de validação funcional adotado, e, no ambiente embarcado, foram validadas a leitura cíclica dos sensores, a inferência local no ESP32-S3 e a tomada de decisão com histerese e regras de segurança. Esses resultados, apresentados em detalhe ao longo do texto, evidenciam o potencial da integração entre aprendizado de máquina e hardware de baixo custo para irrigação inteligente na cafeicultura.

2 Fundamentação Teórica

A irrigação inteligente tem se firmado como uma área estratégica dentro da agricultura de precisão, impulsionada por avanços em sensoriamento, conectividade, microcontroladores e técnicas de aprendizado de máquina. O desenvolvimento de sistemas capazes de interpretar variáveis ambientais e automatizar decisões de irrigação contribui diretamente para o uso eficiente da água, aumento de produtividade e sustentabilidade em ambientes agrícolas. Este capítulo apresenta os principais conceitos associados ao tema, bem como trabalhos correlatos que fundamentam a proposta deste estudo.

2.1 Conceitos Básicos

Irrigação Automatizada e Agricultura de Precisão

A agricultura de precisão baseia-se na coleta contínua de dados do ambiente para otimizar processos produtivos. No contexto da irrigação, isso envolve monitorar variáveis como umidade do solo, temperatura, luminosidade e dados climáticos externos. Sistemas inteligentes têm sido empregados para melhorar a eficiência hídrica, reduzindo desperdícios e aumentando o rendimento das culturas (KUMAR; SINGH; GUPTA, 2020). Em regiões semiáridas, soluções baseadas em IoT demonstraram impacto significativo ao integrar sensores com modelos capazes de prever condições futuras (ZHANG; LIU; WANG, 2018).

Sensores Ambientais

Sensores são elementos fundamentais em sistemas de irrigação inteligente. Sensores de temperatura, umidade do ar, luminosidade e umidade do solo fornecem medições essenciais para avaliar o estado hídrico do ambiente. Estudos mostram que sensores capacitivos apresentam maior confiabilidade em ambientes agrícolas devido à menor suscetibilidade à corrosão (ZHANG; LIU; WANG, 2018). Já sensores como o DHT22 permitem medir variáveis atmosféricas com baixo custo e boa precisão.

Sistemas Embarcados e Microcontroladores

Sistemas embarcados desempenham papel central na automação agrícola, permitindo que decisões sejam tomadas localmente. Microcontroladores como ATMEGA e ESP32 têm ampla adoção devido à combinação de baixo consumo energético e conectividade. Marinho (MARINHO, 2019) demonstra a viabilidade de soluções de baixo custo, enquanto Dey et al. (DEY; ROY; CHAKRABORTY, 2019) destacam o potencial do ESP32.

Internet das Coisas (IoT) na Agricultura

A utilização de Internet das Coisas (IoT) na agricultura permite a integração de sensores, atuadores e sistemas computacionais distribuídos, viabilizando o monitoramento contínuo das condições ambientais e o controle remoto de processos produtivos. Hernandes (DIB, 2019) demonstra aplicações desse paradigma em ambientes residenciais, evidenciando sua versatilidade e potencial de adaptação a diferentes domínios.

No contexto agrícola, entretanto, a adoção de IoT impõe requisitos adicionais relacionados à conectividade em áreas remotas, à robustez física dos dispositivos e à confiabilidade das interconexões. Diferentemente de ambientes urbanos ou industriais controlados, sistemas implantados em campo estão sujeitos a vibração mecânica, poeira, umidade, variações térmicas acentuadas e grandes distâncias entre os nós de sensoriamento, o que exige soluções de comunicação e interconexão projetadas especificamente para operação contínua em condições adversas (Mouser Electronics, 2023; Mouser Electronics, 2022). Nesses cenários, a utilização de conectores selados, interfaces resistentes a intempéries e tecnologias de comunicação de longo alcance torna-se essencial para garantir a integridade dos dados e a disponibilidade do sistema ao longo do tempo.

Além da resistência física, a escalabilidade das soluções de IoT é um fator central para a agricultura digital. Arquiteturas modernas têm evoluído para modelos baseados em AIoT (*Artificial Intelligence of Things*), nos quais dispositivos inteligentes realizam processamento local e se integram a plataformas de análise de dados de forma hierárquica e distribuída (Equipe Embarcados, 2023). Essa abordagem permite reduzir a dependência de conectividade contínua com infraestrutura centralizada, diminuir a latência na tomada de decisão e viabilizar a expansão do sistema para grandes áreas agrícolas com múltiplos pontos de monitoramento.

Dessa forma, a IoT aplicada à agricultura não se restringe à simples conexão de dispositivos à internet, mas envolve a construção de uma infraestrutura distribuída, resiliente e escalável, capaz de suportar sensoriamento em larga escala, processamento local e operação autônoma em ambientes com restrições de conectividade.

Modelos Preditivos Aplicados à Irrigação

Modelos de aprendizado de máquina como Regressão Logística, Árvores de Decisão e SVM têm sido empregados para prever a necessidade de irrigação (KUMAR; SINGH; GUPTA, 2020). Pahlavan et al. (PAHLAVAN; ZHANG; LI, 2020) reforçam o impacto positivo de modelos baseados em consumo hídrico.

Neste trabalho, essa abordagem é aplicada por meio do treinamento externo de um classificador de Regressão Logística, responsável por estimar a probabilidade de necessidade de irrigação a partir de variáveis ambientais. O modelo é treinado fora do dispositivo,

utilizando dados climáticos históricos e variáveis representativas das condições do solo e do ambiente, e posteriormente incorporado ao firmware na forma de parâmetros estáticos. Essa estratégia permite utilizar aprendizado de máquina em um microcontrolador de baixo custo, mantendo baixo custo computacional e execução determinística durante a inferência.

MLOps e Edge AI

MLOps integra práticas de engenharia e ciência de dados para gerenciar o ciclo de vida de modelos. Kreuzberger et al. (KREUZBERGER; KÜHL; HIRSCHL, 2023) apresentam uma revisão abrangente sobre ferramentas e desafios. Entretanto, poucos estudos integram MLOps com inferência embarcada, lacuna destacada por Costa et al. (COSTA; MARTINS; ALVES, 2021) e Zhang et al. (ZHANG; LI; CHEN, 2022).

No contexto deste trabalho, a adoção de um pipeline inspirado em MLOps permite separar explicitamente as etapas de coleta de dados, treinamento, avaliação e exportação dos parâmetros do modelo, garantindo reprodutibilidade e consistência entre o ambiente de aprendizado e o ambiente embarcado. Essa abordagem viabiliza a atualização controlada do comportamento do sistema por meio da substituição do artefato `ia_params.h`, sem necessidade de reimplementar a lógica de inferência no dispositivo.

Já o uso de Edge AI, com execução local da inferência no ESP32-S3, proporciona baixa latência na tomada de decisão, operação autônoma em cenários com conectividade limitada e previsibilidade temporal do processamento, características essenciais para aplicações em ambientes agrícolas.

2.2 Trabalhos Correlatos

Diversos trabalhos exploram a automação da irrigação, sensores e modelos preditivos. Marinho (MARINHO, 2019) desenvolveu um sistema para agricultura familiar baseado em ATMEGA, enquanto Jesus (JESUS, 2019) demonstrou ganhos a partir do uso de dados climáticos externos. Batista (BATISTA, 2016) investigou um robô irrigador multifuncional e Hernandes (DIB, 2019) aplicou IoT a sistemas de irrigação doméstica.

No campo do aprendizado de máquina, Zhang et al. (ZHANG; LIU; WANG, 2018) e Kumar et al. (KUMAR; SINGH; GUPTA, 2020) apresentam modelos aplicados em diferentes condições climáticas. Pahlavan et al. (PAHLAVAN; ZHANG; LI, 2020) contribuem com modelos de previsão de consumo hídrico. Na área de MLOps, Costa et al. (COSTA; MARTINS; ALVES, 2021) reforçam a importância de pipelines contínuos para melhorar decisões agrícolas.

No *Seminário Integrado de Software e Hardware* (SEMISH)[1], observa-se a presença

[1] Página do evento no Congresso da Sociedade Brasileira de Computação (CSBC): <https://csbc.sbc.

de sistemas com microcontroladores e conectividade em cenários de "borda" e IoT, incluindo aplicações com ESP32 em soluções de instrumentação e monitoramento (SANTOS et al., 2023), além de estudos que tratam o processamento na borda para reduzir latência e dependência de infraestrutura centralizada (BARBOZA; KNIESS, 2024).

Além dos estudos já apresentados, a literatura recente também reforça o uso de dispositivos embarcados e computação em borda em aplicações agrícolas de baixo custo. Dey et al. (DEY; ROY; CHAKRABORTY, 2019) destacam a viabilidade do ESP32 para automação inteligente, enquanto Silva et al. (SILVA; CARDOSO; GOMES, 2024) apresentam uma proposta de gestão hídrica com foco em acessibilidade e aplicação prática. Em paralelo, trabalhos de sistemas de borda e IoT em eventos nacionais indicam ganhos de autonomia operacional e redução de dependência de infraestrutura centralizada (SANTOS et al., 2023; BARBOZA; KNIESS, 2024), aspecto particularmente relevante para cenários rurais com conectividade limitada.

No contexto de MLOps e Edge AI, Kreuzberger et al. (KREUZBERGER; KÜHL; HIRSCHL, 2023) consolidam conceitos e arquitetura para ciclo de vida de modelos, enquanto Zhang et al. (ZHANG; LI; CHEN, 2022) discutem a transição de abordagens centradas em nuvem para abordagens orientadas a dispositivo. Costa et al. (COSTA; MARTINS; ALVES, 2021) reforçam a importância de pipelines contínuos em aplicações agrícolas. Em conjunto, esses trabalhos sustentam a adoção de práticas de treinamento, avaliação, versionamento e implantação de modelos em ambientes com restrições operacionais.

Essas evidências sustentam a proposta deste trabalho ao indicar que arquiteturas com inferência local e integração com pipelines de atualização/implantação de parâmetros tendem a ser mais adequadas para ambientes rurais, onde a conectividade e a infraestrutura de TI são limitadas.

A análise da literatura revela lacunas importantes: poucos trabalhos integram sensores, inferência embarcada e MLOps em um único sistema; menos ainda utilizam o ESP32-S3 com atualização OTA; e não foi encontrado nenhum trabalho voltado à cultura do café com essa abordagem integrada.

Análise Comparativa: Vantagens, Limitações e Relação com Este Trabalho

Como vantagens, os trabalhos correlatos demonstram: (i) viabilidade de soluções de irrigação e monitoramento com baixo custo; (ii) uso de dados climáticos e variáveis ambientais para apoiar decisões; e (iii) potencial da computação em borda para reduzir latência e dependência de conectividade contínua.

Como limitações, observa-se que, em geral, os estudos tratam apenas parte do

org.br/2024/semish/>. Anais do SEMISH na SBC OpenLib (SOL): <https://sol.sbc.org.br/index.php/semish>.

problema: alguns focam na automação e sensoriamento, outros na modelagem preditiva, e outros em conceitos de MLOps, com menor frequência de integração ponta a ponta entre coleta/preparo de dados, treinamento externo, exportação de parâmetros e inferência embarcada em microcontroladores.

A relação deste trabalho com os demais está justamente na integração dessas frentes em uma única arquitetura aplicada à cafeicultura: sensores de baixo custo, inferência local no ESP32-S3, regras determinísticas de segurança para estabilidade da decisão e pipeline inspirado em MLOps para atualização controlada de parâmetros no firmware.

3 Metodologia

A metodologia adotada neste trabalho foi estruturada para integrar, de forma coerente, três eixos fundamentais: (i) aquisição de dados ambientais por meio de sensores instalados em campo; (ii) modelagem e treinamento de um sistema de decisão formulado como problema de classificação binária por Regressão Logística; e (iii) integração desse modelo ao firmware embarcado no microcontrolador ESP32-S3, responsável por executar inferência em tempo real e acionar o sistema de irrigação. Essa abordagem permite que a irrigação seja realizada de maneira automatizada, fundamentada em dados reais e com capacidade de atualização contínua. A seguir, descrevem-se as bases metodológicas, os procedimentos adotados e a arquitetura do sistema desenvolvido.

3.1 Caracterização da Pesquisa

Quanto à sua natureza, este trabalho caracteriza-se como uma pesquisa aplicada, uma vez que propõe o desenvolvimento de uma solução prática para um problema real no contexto da agricultura de precisão. O objetivo central é a implementação de um sistema inteligente capaz de automatizar o processo de irrigação com base em dados ambientais e técnicas de aprendizado de máquina.

Do ponto de vista dos objetivos, a pesquisa é classificada como exploratória e descritiva, pois envolve o estudo e a aplicação de tecnologias emergentes, além da descrição detalhada das etapas de desenvolvimento, integração e funcionamento do sistema proposto. A abordagem adotada é predominantemente quantitativa, fundamentada na coleta, processamento e análise de dados ambientais.

3.2 Abordagem Metodológica

A abordagem metodológica adotada fundamenta-se no desenvolvimento incremental de um sistema ciberfísico, integrando componentes de hardware, software embarcado e modelagem preditiva em um ciclo iterativo de aprimoramento contínuo.

Essa estratégia possibilitou avaliar continuamente o comportamento do sistema, favorecendo a evolução gradual da solução proposta e a adaptação às limitações computacionais inerentes ao ambiente embarcado.

3.3 Sensoriamento e Aquisição de Dados

A etapa de sensoriamento constitui a base operacional do sistema, fornecendo as variáveis que alimentam o modelo de decisão. Foram utilizados três sensores complementares, amplamente empregados em aplicações agrícolas devido ao baixo custo, facilidade de integração e boa estabilidade em campo.

O primeiro sensor integrado ao sistema foi o **DHT22**, responsável pela medição de temperatura e umidade relativa do ar. Essas variáveis são essenciais na avaliação de estresse hídrico e na estimativa da taxa de evapotranspiração, indicadores fundamentais para determinar a necessidade de irrigação. A Figura 1 apresenta o esquema de ligação deste sensor ao ESP32-S3.

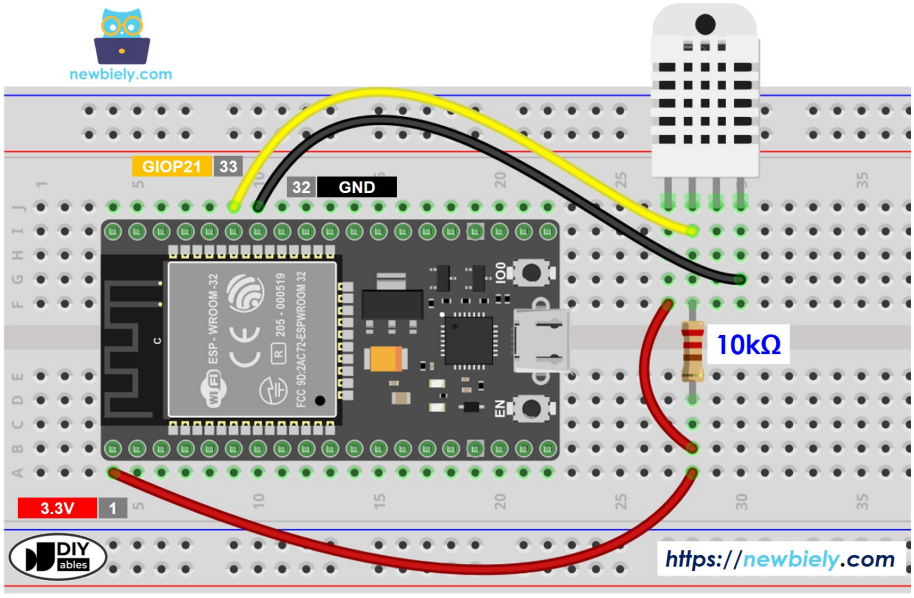

Figura 1 – Ligação do sensor DHT22 (temperatura e umidade). Fonte: ESP32IO.

O segundo sensor é o **LDR**, utilizado para medir a luminosidade incidente. A luminosidade é uma variável relevante, sobretudo porque influencia o comportamento térmico do solo e o consumo hídrico das plantas. Nas horas de maior radiação solar, o solo tende a perder umidade mais rapidamente. A Figura 2 apresenta o circuito de ligação.

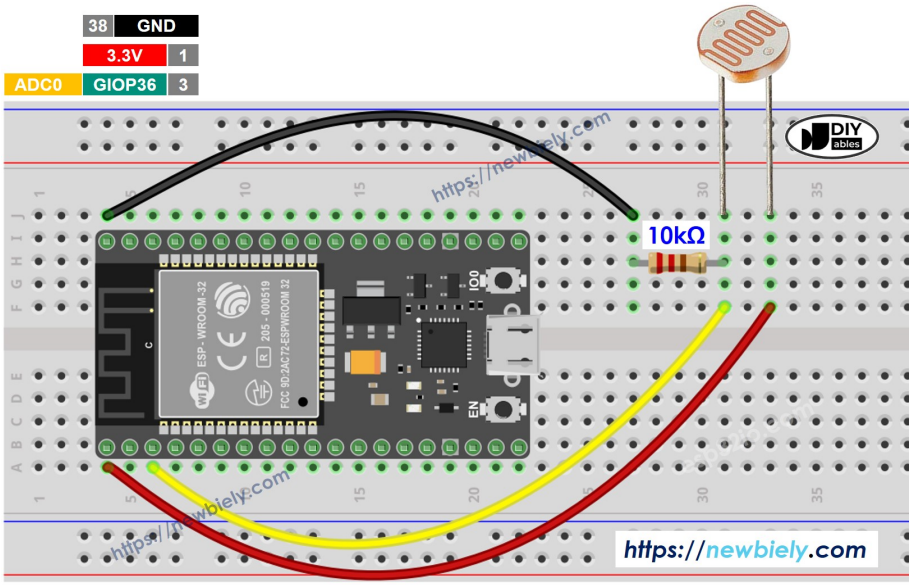

Figura 2 – Ligação do sensor LDR (luminosidade). Fonte: ESP32IO.

Por fim, incorpora-se ao sistema um sensor **capacitivo de umidade do solo**, que permite a leitura direta da umidade do substrato. Esse tipo de sensor apresenta vantagens importantes, como menor corrosão e maior estabilidade em ambientes agrícolas. A Figura 3 apresenta sua ligação ao ESP32-S3.

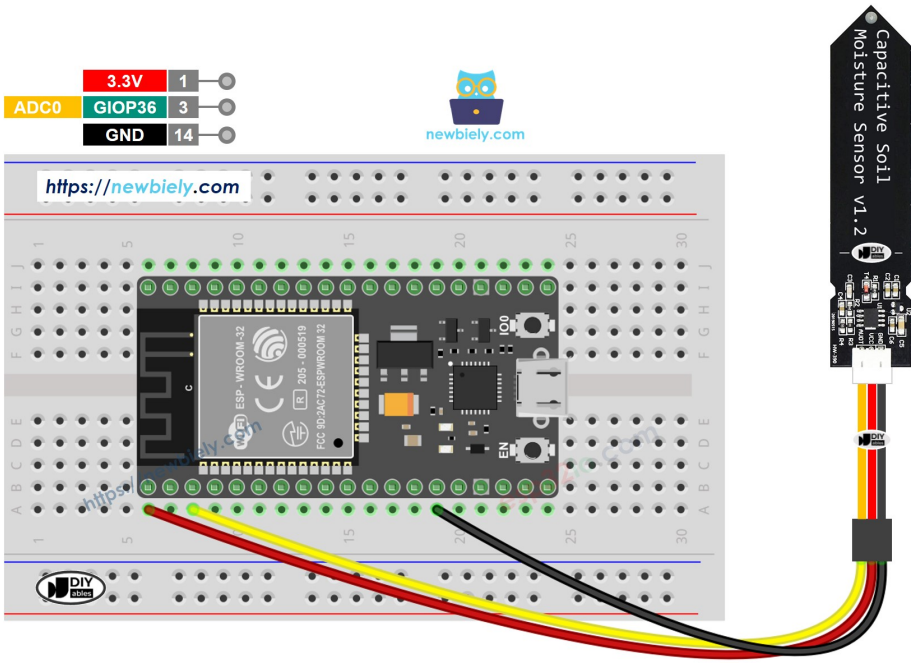

Figura 3 – Ligação do sensor capacitivo de umidade do solo. Fonte: ESP32IO.

A integração desses três sensores fornece um conjunto robusto de variáveis ambientais, permitindo que o modelo preditivo capture condições atmosféricas, incidência solar e umidade efetiva do solo.

Essa combinação de sensores mostrou-se adequada para o monitoramento de condições típicas de lavouras cafeeiras em Minas Gerais.

3.4 Pipeline Externo de Treinamento

O treinamento do modelo foi realizado externamente por meio de um pipeline desenvolvido em Python, com foco em coleta de dados climáticos, preparação do conjunto de dados, treinamento do modelo, avaliação e exportação dos parâmetros para uso no firmware embarcado. Na implementação atual, esse pipeline é estruturado principalmente em scripts (`coleta_dados.py` e `treinar_modelo.py`), o que favorece reprodutibilidade, simplicidade de execução e manutenção incremental.

No escopo deste trabalho, o aprendizado de máquina foi modelado como um problema de **classificação binária supervisionada**, no qual a saída do modelo é a classe *irrigar* (1) ou *não irrigar* (0), além da probabilidade associada a essa decisão.

O script `coleta_dados.py` realiza a coleta de dados climáticos históricos a partir da API externa e gratuita *Open-Meteo Archive* (Open-Meteo, 2024), considerando a região de João Monlevade-MG (latitude -19.82 e longitude -43.17) e séries diárias de temperatura máxima, temperatura mínima e precipitação. Após validação e sanitização das colunas esperadas, os dados são consolidados e exportados no arquivo `dados_climaticos_tratados.csv`.

A etapa de pré-processamento inclui: verificação das colunas esperadas, conversão de tipos, remoção de valores inválidos, limitação de precipitação para valores não negativos, renomeação padronizada de colunas e ordenação temporal. Esse procedimento reduz inconsistências e produz um conjunto de dados compatível com a etapa de modelagem.

Na sequência, o script `treinar_modelo.py` lê o conjunto de dados tratado e executa as etapas de preparação e modelagem. Nessa etapa, são ajustadas as variáveis de entrada utilizadas pelo modelo (`soil_pct`, `temp_c`, `rh_pct` e `chuva_mm_24h`), incluindo a criação de variáveis derivadas e valores *proxy* quando determinadas medições não estão presentes no conjunto histórico. Em particular: `temp_c` é obtida a partir da média entre `temp_max` e `temp_min`; `chuva_mm_24h` é derivada de `chuva_mm`; e, na ausência de medições históricas equivalentes, `soil_pct` e `rh_pct` são preenchidas com valores *proxy* fixos para compatibilização com o vetor de entrada embarcado.

A variável climática de chuva é utilizada por dois motivos: (i) como atributo de entrada do classificador, pois chuva recente influencia diretamente a necessidade de irrigação; e (ii) como suporte a uma regra conservadora de bloqueio no módulo decisor embarcado. Assim, a informação climática ajuda a reduzir acionamentos desnecessários em cenários de umidade já favorecida por precipitação recente. Ressalta-se que, na arquitetura atual, a operação embarcada utiliza série de chuva embutida no firmware (via `ia_params.h`), sem

consulta online de API em tempo de execução.

A Figura 4 apresenta uma representação conceitual das responsabilidades do pipeline externo (coleta, preparação, engenharia de atributos, normalização e exportação), utilizada para fins de documentação arquitetural. Embora o diagrama empregue uma visão modular, a implementação atual no repositório está materializada em scripts e funções, e não em uma hierarquia completa de classes.

O fluxo de execução completo é apresentado no diagrama de sequência da Figura 5, ilustrando a geração progressiva dos artefatos. Além de `dados_climaticos_tratados.csv` e `model.pkl`, o pipeline também produz artefatos auxiliares como `scaler.pkl`, `metrics.txt`, `pesos_bias.json` e o arquivo `ia_params.h`, este último contendo os parâmetros do modelo e metadados necessários à inferência embarcada.

Esse pipeline permite a geração reprodutível de novas versões dos parâmetros do modelo e sua integração ao processo de compilação do firmware por meio do arquivo `ia_params.h`. Assim, o fluxo adotado caracteriza uma abordagem inspirada em práticas de MLOps para dispositivos embarcados, ao separar explicitamente coleta, treinamento, avaliação e empacotamento dos parâmetros utilizados na inferência local (KREUZBERGER; KÜHL; HIRSCHL, 2023).

3.5 Integração ao Firmware Embarcado

Após o treinamento, o arquivo `ia_params.h` é incorporado ao firmware do ESP32-S3 durante o processo de compilação, possibilitando a inferência local. Esse arquivo contém os pesos e o termo de bias do modelo de Regressão Logística, as estatísticas de normalização (média e escala) e os metadados/valores da série de chuva utilizada pelo módulo de previsão embarcado. Os limiares de histerese e demais parâmetros de decisão do sistema permanecem configurados no firmware (por exemplo, em `app_config.h`).

A Figura 6 apresenta uma visão arquitetural dos módulos do firmware. Os módulos de sensores realizam leituras ambientais, o módulo *Forecast* consulta uma série temporal de chuva embutida no `ia_params.h` (indexada em tempo de execução), o módulo *IA* calcula a probabilidade de irrigação a partir das variáveis de entrada, o módulo *Decider* combina essa saída com histerese e regras determinísticas de segurança, e o módulo *Pump* controla o relé responsável pela irrigação.

O ciclo completo de irrigação implementado no dispositivo é detalhado na Figura 7. O sistema executa leituras periódicas dos sensores, monta o vetor de entrada do modelo, calcula a probabilidade de irrigação por inferência embarcada e aplica regras adicionais de decisão, como histerese, bloqueio por chuva e tempos mínimos de acionamento/desligamento (anti-chattering), antes de enviar o comando final ao atuador.

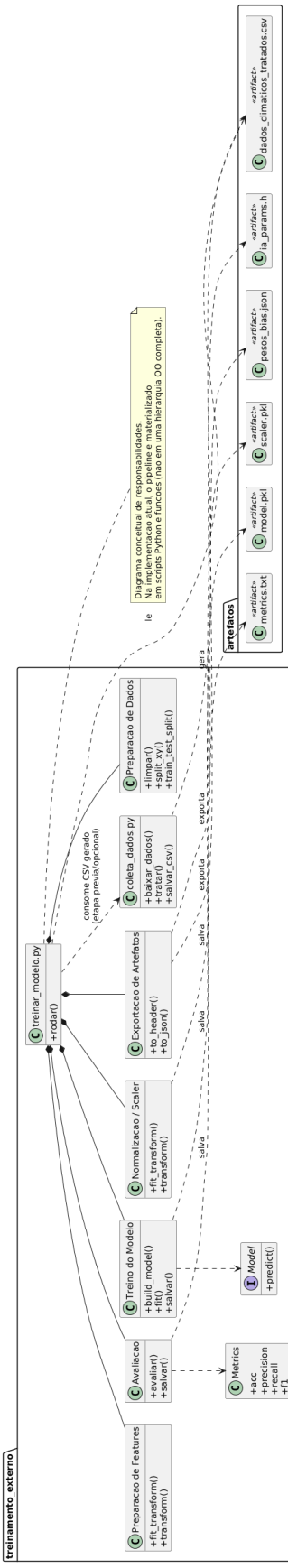

Figura 4 – Representação conceitual das responsabilidades do pipeline externo de treinamento.

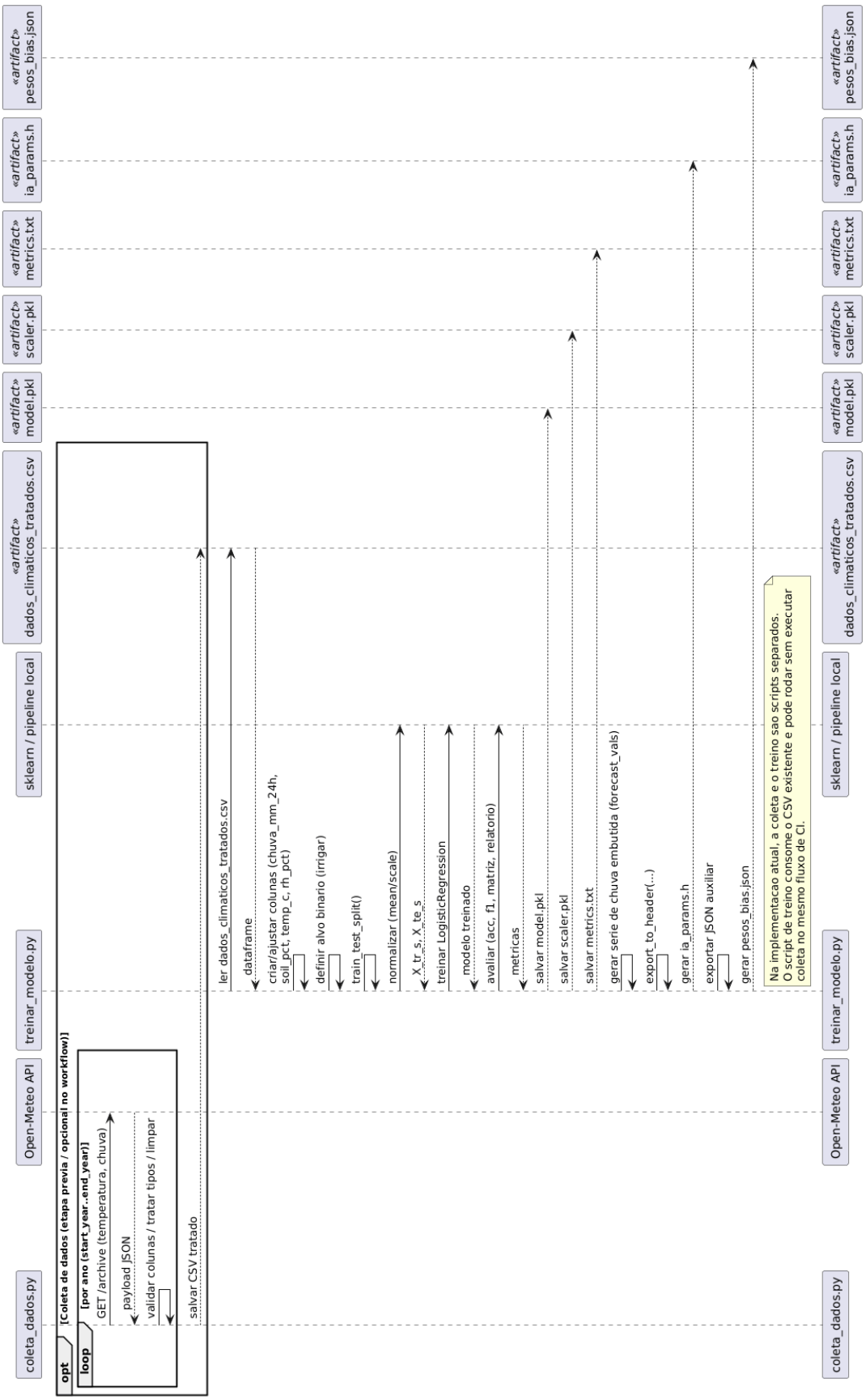

Figura 5 – Diagrama de sequência do processo de treinamento, avaliação e exportação de artefatos.

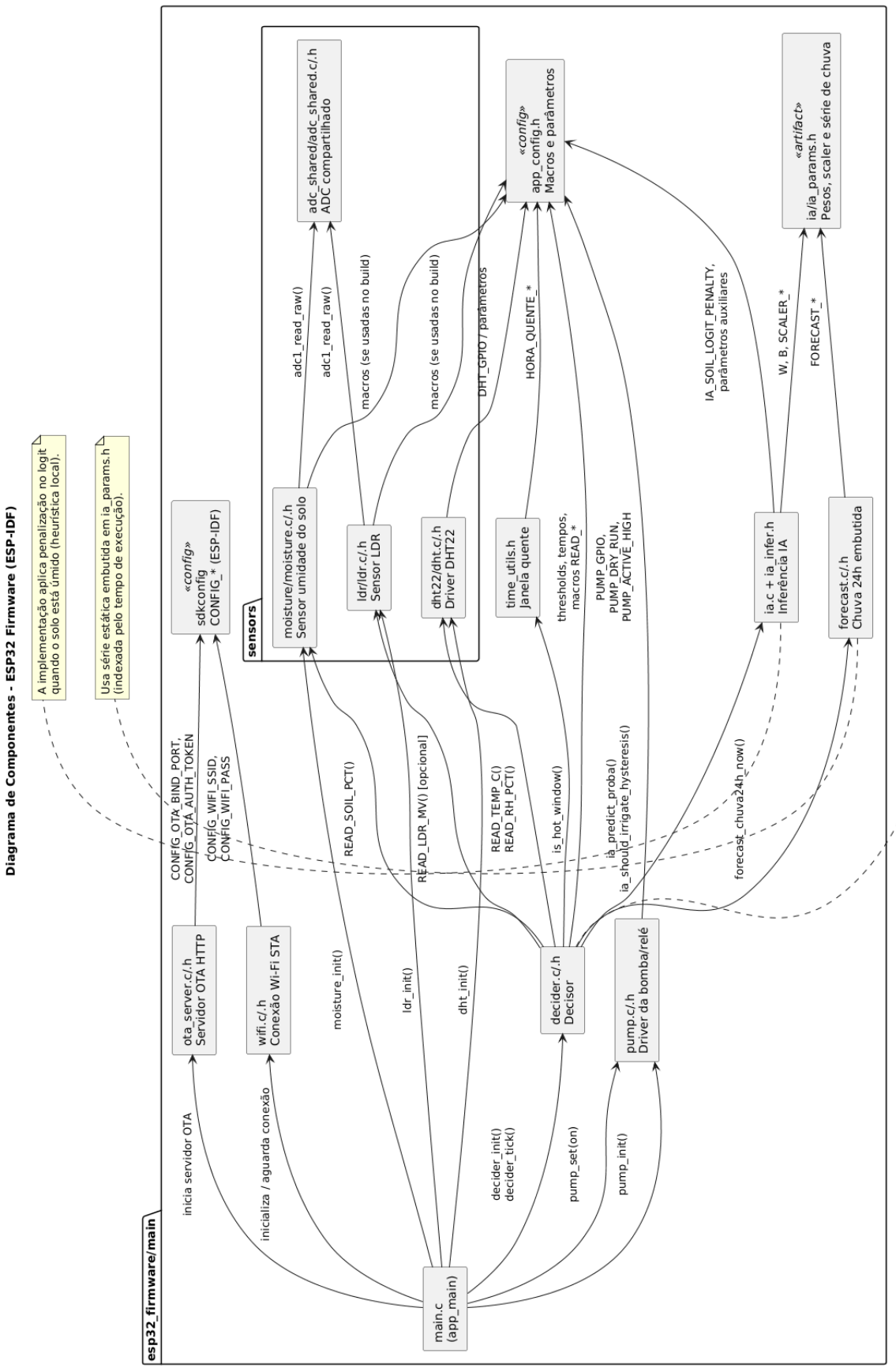

Figura 6 – Visão arquitetural dos módulos do firmware ESP32-S3.

A informação de chuva utilizada no firmware é obtida localmente a partir da série embutida no arquivo `ia_params.h`, o que garante execução determinística e reduz dependência de conectividade durante a operação. Dessa forma, a lógica de controle permanece funcional mesmo em cenários com acesso limitado à rede.

3.6 Organização Modular do Sistema

A Figura 8 apresenta o diagrama de pacotes do projeto, ilustrando a separação clara entre os componentes de treinamento, firmware, sensores, utilidades, IA e comunicação. Essa modularidade facilita manutenção, reuso e evolução incremental do sistema.

3.7 Ambiente de Desenvolvimento

O desenvolvimento do firmware embarcado foi realizado utilizando o framework ESP-IDF, com suporte ao microcontrolador ESP32-S3, permitindo o acesso a recursos de baixo nível e maior controle sobre os periféricos do dispositivo. Os módulos de leitura de sensores, inferência do modelo e controle da bomba foram implementados em linguagem C.

Para demonstrar o funcionamento do sistema em operação, foi produzido um vídeo curto contendo a execução completa da solução embarcada, incluindo a leitura dos sensores e o acionamento da bomba de irrigação. O material encontra-se disponível no endereço: <https://youtu.be/HBBkFBvKrnU> .

O pipeline externo de treinamento foi desenvolvido em ambiente Python, utilizando bibliotecas consolidadas para manipulação de dados e aprendizado de máquina. O versionamento do código foi realizado por meio de sistema de controle de versões, garantindo rastreabilidade, organização e reprodutibilidade do desenvolvimento. O repositório do projeto pode ser acessado em: <https://github.com/NahanRezende/tcc_repository> .

Os experimentos foram conduzidos em um computador com as seguintes configurações: processador Intel Core i7 8500u, 16 GB de memória RAM e sistema operacional Ubuntu 22.04 LTS.

A montagem física do protótipo, contendo o microcontrolador ESP32-S3, os sensores ambientais e o módulo de acionamento da bomba, é apresentada na figura abaixo:

3.8 Procedimentos de Teste e Avaliação

A avaliação do sistema foi conduzida em duas frentes complementares: (i) avaliação do pipeline de aprendizado de máquina no ambiente externo; e (ii) validação funcional do comportamento embarcado no ESP32-S3.

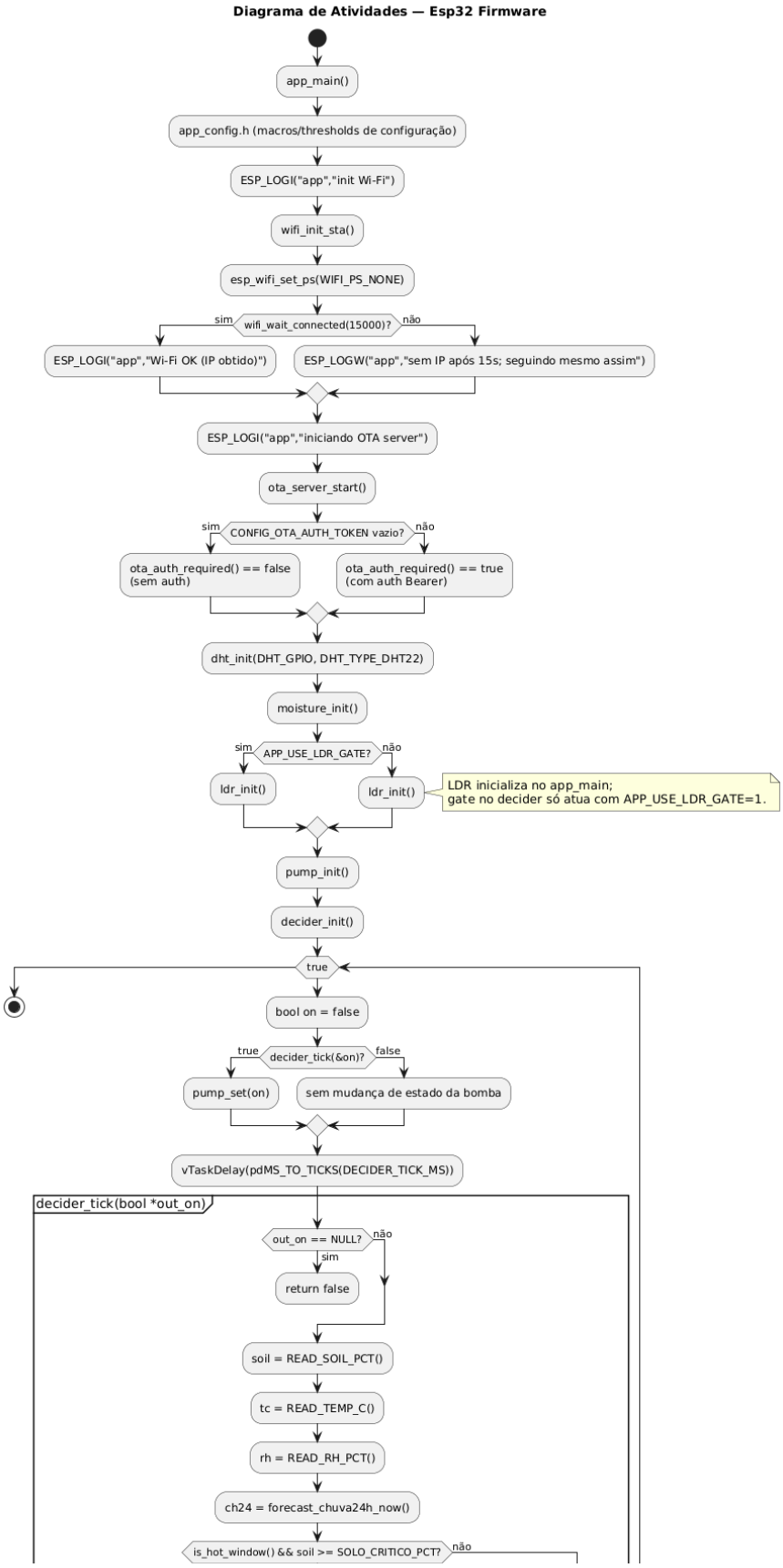

Figura 7 – Diagrama de atividades do ciclo de irrigação no firmware (parte 1/2).

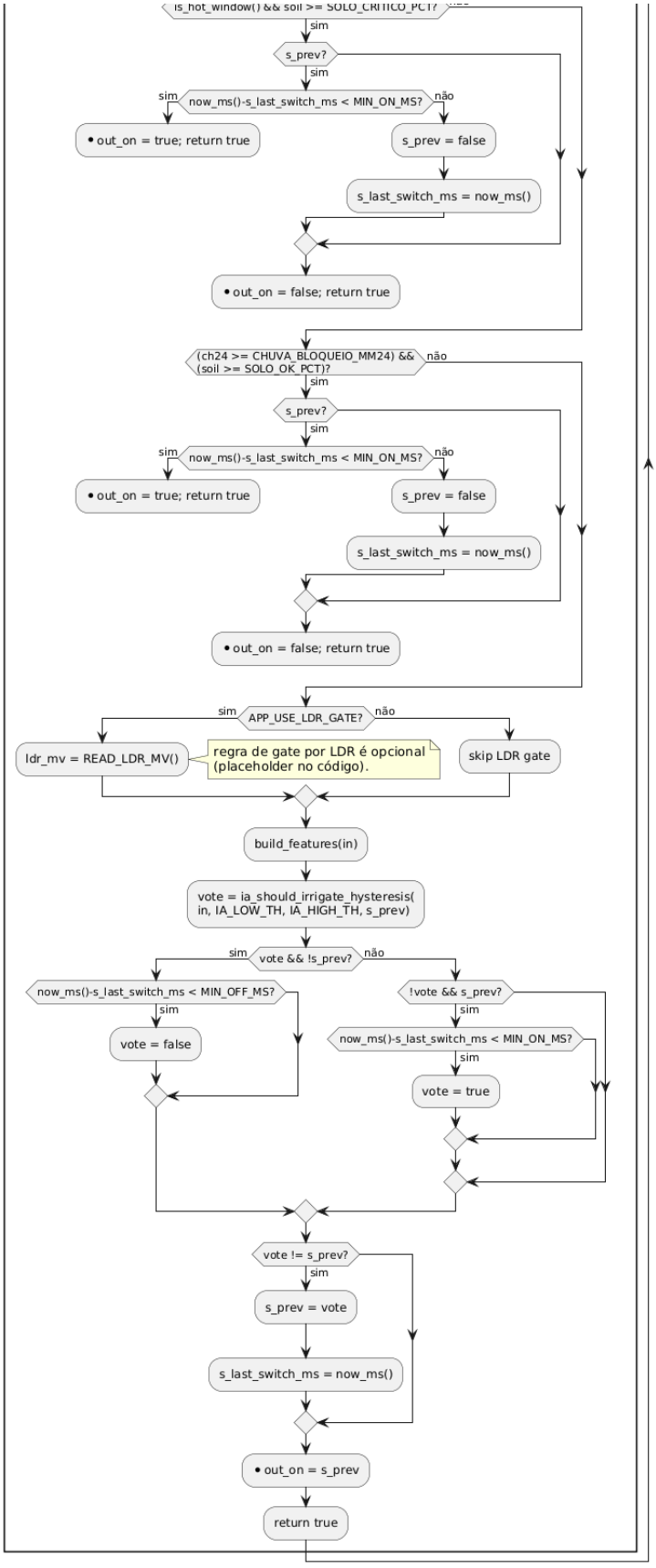

Figura 7 – Diagrama de atividades do ciclo de irrigação no firmware (parte 2/2).

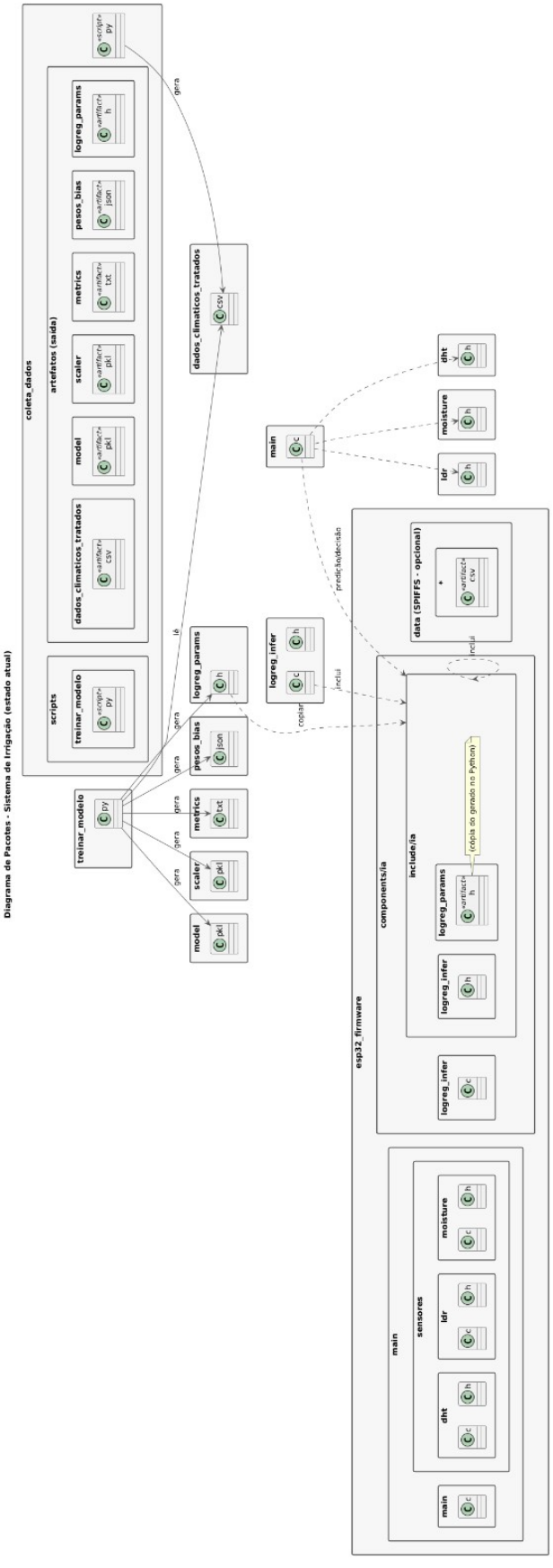

Figura 8 – Diagrama de pacotes do projeto.

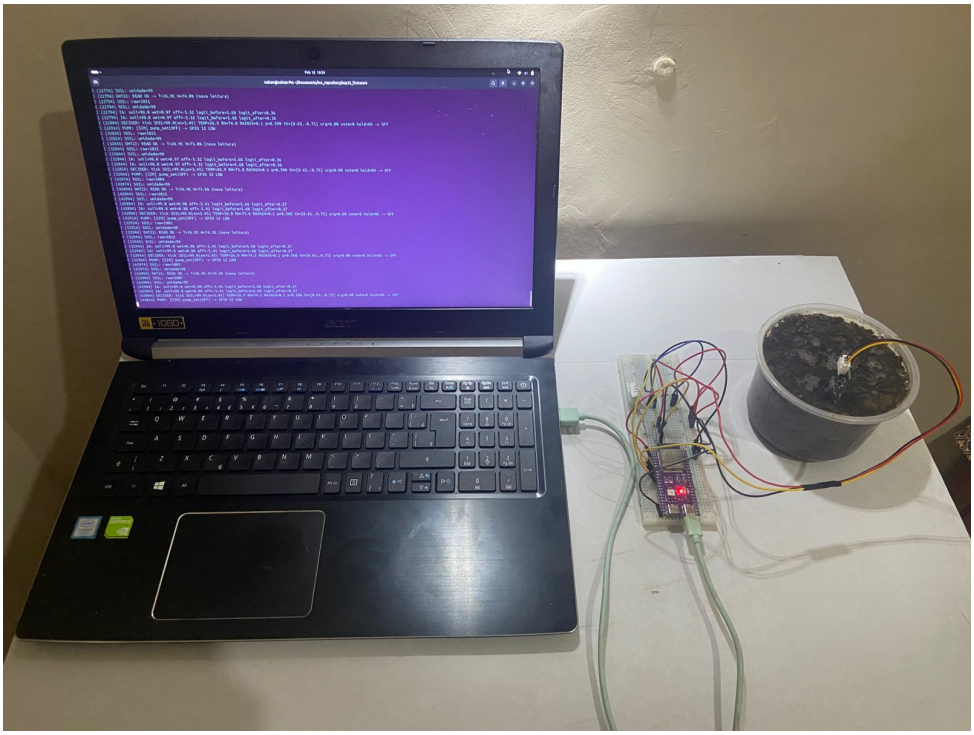

Figura 9 – Protótipo do sistema de irrigação inteligente montado.

Na etapa de aprendizado de máquina, foi adotado o seguinte protocolo experimental, na versão de dados utilizada neste trabalho:

- quantidade total de amostras: 2.054 registros climáticos;
- distribuição de classes: 1.658 amostras da classe *irrigar* (80,72%) e 396 da classe *não irrigar* (19,28%), caracterizando desbalanceamento;
- separação do conjunto de dados: 80% para treino e 20% para teste;
- estratificação: `train_test_split` com `stratify=y`, preservando a proporção entre classes;
- validação cruzada: não realizada nesta versão;
- conjunto separado de validação (ex.: 80/10/10): não adotado nesta versão;
- aumento de dados (*data augmentation*): não utilizado.

As métricas reportadas (acurácia, F1-score, matriz de confusão e relatório de classificação) foram calculadas no conjunto de teste. A avaliação de sobreajuste (*overfitting*) não foi conduzida por análise formal de curva de aprendizado nem por validação cruzada; portanto, não se afirma conclusão definitiva sobre *overfitting* nesta etapa. No entanto, as métricas devem ser interpretadas em conjunto com o caráter simplificado do rótulo e de parte das *features*, conforme discutido no capítulo de resultados.

Na etapa embarcada, inicialmente foram realizados testes individuais dos módulos de leitura dos sensores, com o objetivo de verificar a coerência e a estabilidade das medições coletadas. Em seguida, o sistema integrado foi avaliado em ciclos completos de

operação, analisando-se a coerência das decisões de irrigação frente às condições ambientais observadas e às informações climáticas utilizadas pelo firmware.

A utilização de histerese foi avaliada empiricamente, contribuindo para a redução de acionamentos sucessivos indevidos da bomba de irrigação. Neste trabalho, a histerese consiste na definição de limiares distintos para o acionamento e o desligamento da irrigação, criando uma faixa de operação que evita comutações rápidas e repetitivas quando as medições oscilam próximas ao valor de referência. Na configuração-base avaliada, o sistema utiliza limiar superior para ligar (`high_th`) e limiar inferior para desligar (`low_th`), além de tempos mínimos de permanência ligado/desligado (*anti-chattering*), o que aumenta a estabilidade da decisão final do atuador.

Esses testes buscaram verificar não apenas o funcionamento do sistema, mas também sua robustez frente a variações ambientais e ruídos nas medições dos sensores.

Os resultados desses testes são discutidos no Capítulo 5.

3.9 Limitações Metodológicas

Como limitação metodológica, destaca-se a dependência da qualidade e representatividade dos dados ambientais utilizados no treinamento do modelo, bem como as restrições computacionais impostas pelo ambiente embarcado. Em função dessas limitações, optou-se pela utilização de um modelo de Regressão Logística, que apresenta baixo custo computacional e boa interpretabilidade.

Também se destaca que, na configuração avaliada, parte das entradas foi construída com valores *proxy* para compatibilização entre dados históricos e formato de entrada embarcado, e que o protocolo experimental não incluiu validação cruzada, conjunto de validação independente (80/10/10) nem curva de aprendizado.

Além disso, os testes realizados concentram-se em cenários controlados e períodos limitados de operação, não contemplando avaliações de longo prazo em campo. Essas restrições indicam possibilidades de aprimoramento e expansão do sistema em trabalhos futuros.

4 Desenvolvimento

Este capítulo descreve detalhadamente o processo de desenvolvimento do sistema inteligente de irrigação proposto, abordando sua arquitetura, a implementação do firmware embarcado no microcontrolador ESP32-S3, a integração com sensores ambientais, o uso de informação de chuva embutida no firmware, a aplicação de técnicas de aprendizado de máquina para tomada de decisão e o acionamento do sistema de irrigação. Também são discutidas as decisões de projeto e os principais trade-offs envolvidos na implementação.

4.1 Arquitetura Geral do Sistema

O sistema adota uma arquitetura híbrida que combina treinamento externo do modelo preditivo com inferência local no ESP32-S3, permitindo maior custo computacional na etapa de aprendizado e baixa latência, autonomia e previsibilidade durante a operação em campo.

A arquitetura é composta por dois blocos: (i) o pipeline externo de coleta e treinamento e (ii) o firmware embarcado, responsável pela leitura dos sensores, inferência e controle da irrigação. A integração ocorre pela exportação dos parâmetros do modelo para um arquivo de cabeçalho em C, eliminando dependências dinâmicas no dispositivo.

O fluxo inicia-se com a aquisição de temperatura, umidade do ar, umidade do solo e luminosidade. No firmware, essas leituras são convertidas para unidades físicas e disponibilizadas ao módulo decisor por meio de interfaces padronizadas. Na configuração atual do modelo embarcado, o vetor de características utilizado na inferência contempla umidade do solo, temperatura, umidade relativa do ar e informação de chuva em 24 horas; a leitura de luminosidade (LDR) permanece integrada ao firmware, mas não participa diretamente do vetor de entrada da versão atual do modelo.

A probabilidade de irrigação é calculada por um modelo de Regressão Logística treinado externamente. Nesse contexto, o problema de aprendizado de máquina é tratado como **classificação binária** (*irrigar* vs. *não irrigar*). A informação de chuva em 24 horas, obtida a partir de uma série temporal embutida no `ia_params.h`, é utilizada como variável auxiliar e também como critério determinístico de bloqueio da irrigação.

Após a inferência, são aplicadas regras de segurança, como histerese, restrições por janela de horário, bloqueio por chuva e tempos mínimos de acionamento e desligamento, assegurando estabilidade e evitando chaveamentos frequentes do relé. A decisão validada é então enviada ao módulo de controle da bomba.

O sistema opera de forma cíclica e determinística, executando, a cada iteração, a

leitura dos sensores, montagem do vetor de entrada, inferência, verificação das regras e atualização do atuador, conforme o Algoritmo 4.1.

```
bool decider_tick(bool *out_on) {
    if (!out_on) return false;
    float soil = (float)READ_SOIL_PCT();
    float tc   = (float)READ_TEMP_C();
    float rh   = (float)READ_RH_PCT();
    float ch24 = forecast_chuva24h_now();

    if (is_hot_window() && soil >= SOLO_CRITICO_PCT) {
        if (s_prev) {
            uint64_t n = now_ms();
            if (n - s_last_switch_ms < MIN_ON_MS) { *out_on = true;
               return true; }
            s_prev = false; s_last_switch_ms = n;
        }
        *out_on = false; return true;
    }
    if ((ch24 >= CHUVA_BLOQUEIO_MM24) && (soil >= SOLO_OK_PCT)) {
        if (s_prev) {
            uint64_t n = now_ms();
            if (n - s_last_switch_ms < MIN_ON_MS) { *out_on = true;
               return true; }
            s_prev = false; s_last_switch_ms = n;
        }
        *out_on = false; return true;
    }
    float in[N_FEATS];
    build_features(in);
    bool vote = ia_should_irrigate_hysteresis(in, IA_LOW_TH, IA_HIGH_TH,
        s_prev);

    uint64_t n = now_ms();
    if (vote && !s_prev) {
        if (n - s_last_switch_ms < MIN_OFF_MS) vote = false;
    } else if (!vote && s_prev) {
        if (n - s_last_switch_ms < MIN_ON_MS)  vote = true;
    }
    if (vote != s_prev) {
        s_prev = vote;
        s_last_switch_ms = n;
    }
    *out_on = s_prev;
    return true;
}
```

Listing 4.1 – Fluxo geral de decisão do sistema de irrigação

No código-fonte, esse fluxo é implementado principalmente pelos módulos `decider`, `ia`, `forecast` e `pump`, todos inicializados a partir da função `app_main`. Essa organização reflete diretamente a arquitetura proposta, permitindo que cada etapa do processo de decisão seja isolada, testada e evoluída de forma independente. Ressalta-se que, na implementação final, o módulo decisor também aplica ajustes adaptativos nos limiares de histerese em função da umidade do solo, além das regras explícitas mostradas no fluxo simplificado acima.

4.2 Estrutura do Firmware na ESP32-S3

O firmware do sistema foi desenvolvido utilizando o framework ESP-IDF, adotando sua organização modular e explorando suas abstrações para inicialização de periféricos, conectividade e gerenciamento de tarefas. Todo o código embarcado encontra-se no diretório `esp32_firmware/main/`, que contém o ponto de entrada da aplicação, os módulos funcionais e os arquivos responsáveis pelo processo de build.

A função `app_main`, definida em `main.c`, constitui o núcleo do sistema e coordena a inicialização dos subsistemas. Diferentemente de aplicações baseadas em sistemas operacionais completos, o firmware segue um modelo de execução determinístico, no qual a ordem de inicialização é crítica para o funcionamento adequado.

Inicialmente são configurados os recursos de conectividade, incluindo a interface Wi-Fi e o servidor de atualização remota (OTA), garantindo que o dispositivo esteja apto a receber novas versões de firmware ainda nas fases iniciais de execução. Em seguida, são inicializados os sensores ambientais, o módulo de controle da bomba e o módulo decisor.

Após essa etapa, o sistema entra em um laço de execução contínuo e temporizado no qual as condições ambientais são avaliadas periodicamente e as decisões de irrigação são tomadas sem a utilização de um escalonador complexo de tarefas, aumentando a previsibilidade em ambiente embarcado.

O Código 4.2 ilustra a estrutura simplificada da função `app_main`, evidenciando a sequência de inicialização e o laço principal de execução.

```
1  void app_main(void) {
2      // Wifi
3      ESP_LOGI("app","init Wi-Fi");
4      wifi_init_sta();
5      esp_wifi_set_ps(WIFI_PS_NONE);
6      if (wifi_wait_connected(15000)) {
7          ESP_LOGI("app","Wi-Fi OK (IP obtido)");
8      } else {
9          ESP_LOGW("app","sem IP após 15s; seguindo mesmo assim");
10     }
```

```
11      // Start OTA server
12      ESP_LOGI("app","iniciando OTA server");
13      ota_server_start();
14
15      // Sensores
16      dht_init(DHT_GPIO, DHT_TYPE_DHT22);
17      moisture_init();
18      ldr_init();
19      // Atuador + decisor
20      pump_init();
21      decider_init();
22      while (true) {
23          bool on = false;
24          if (decider_tick(&on)) {
25              pump_set(on);
26          }
27          vTaskDelay(pdMS_TO_TICKS(DECIDER_TICK_MS)); // 10 s por default
28      }
29  }
```

Listing 4.2 – Modelo de execução do firmware embarcado no ESP32-S3

A modularização do firmware foi adotada como princípio de projeto, seguindo o *Princípio da Responsabilidade Única* (*Single Responsibility Principle* – SRP), de modo que cada componente do sistema seja responsável por uma única função bem definida. Essa diretriz reduz o acoplamento e favorece a evolução incremental do código, pois alterações em um subsistema tendem a não se propagar para os demais quando as responsabilidades estão bem delimitadas.

Essa escolha é coerente com os princípios de organização defendidos por Martin em *Arquitetura Limpa*, segundo os quais a separação de responsabilidades e o isolamento de mudanças são fundamentais para manter sistemas compreensíveis, testáveis e sustentáveis ao longo do tempo (MARTIN, 2018). No contexto deste trabalho, o SRP foi materializado por meio da divisão explícita em módulos com fronteiras claras: o subsistema de sensores limita-se à aquisição e conversão das medições; o módulo de inteligência artificial executa a inferência probabilística; o módulo decisor agrega a saída probabilística a histerese e regras determinísticas de segurança (como bloqueio por chuva e tempos mínimos); e o módulo da bomba atua como um *driver* simples responsável apenas pelo acionamento do relé.

Além de melhorar a legibilidade e a manutenibilidade, essa estrutura facilita a extensão do sistema sem comprometer a arquitetura existente. Por exemplo, a inclusão de novos sensores, a substituição do modelo de decisão ou a alteração das regras de segurança podem ser realizadas com impacto localizado, preservando a previsibilidade do firmware e reduzindo a probabilidade de regressões em componentes não relacionados.

As configurações globais do sistema, como pinos de GPIO, limiares de decisão, tempos mínimos de acionamento e parâmetros de histerese, são centralizadas no arquivo `app_config.h`. Essa abordagem permite alterar o comportamento do sistema sem a necessidade de modificar o código-fonte dos módulos, facilitando ajustes e experimentação durante o desenvolvimento.

A separação clara de responsabilidades entre os módulos contribui para a manutenibilidade do firmware e reduz o acoplamento entre componentes. Além disso, essa estrutura facilita futuras extensões, como a inclusão de novos sensores, a substituição do modelo de decisão ou a incorporação de novos mecanismos de comunicação, sem comprometer a arquitetura existente.

4.3 Leitura de Sensores

O sistema realiza a aquisição de dados ambientais por meio de três tipos de sensores, implementados no diretório `esp32_firmware/main/sensors/`. Esses sensores fornecem medições de temperatura, umidade relativa do ar, umidade do solo e luminosidade, utilizadas pelo módulo decisor e, conforme a configuração do modelo embarcado, pelo módulo de inferência.

Cada sensor é implementado como um módulo independente, seguindo o Princípio da Responsabilidade Única. Essa abordagem evita o acoplamento direto entre o módulo decisor e os detalhes de comunicação, temporização ou conversão elétrica dos dispositivos de sensoriamento.

O diretório `esp32_firmware/main/sensors/` contém subdiretórios específicos para cada sensor, organizados conforme o tipo de interface utilizada. Sensores digitais com protocolo próprio, como o DHT22, possuem *drivers* dedicados, enquanto sensores analógicos, como o sensor capacitivo de umidade do solo e o LDR, compartilham uma camada comum de acesso ao conversor analógico-digital (*Analog-to-Digital Converter* – ADC). Essa organização permite a adição ou substituição de sensores com impacto mínimo no restante do firmware, desde que a interface exposta ao módulo decisor seja preservada.

Sensor de Temperatura e Umidade (DHT22)

O sensor DHT22 é utilizado para medir temperatura e umidade relativa do ar, empregando comunicação por temporização (*bit banging*). A leitura é realizada por meio de um protocolo proprietário baseado em controle preciso de tempo, implementado inteiramente por software.

O driver inicia a comunicação enviando um pulso de *start* ao sensor, seguido da leitura sequencial de 40 bits transmitidos pelo dispositivo. Esses bits correspondem aos

valores de umidade, temperatura e a um byte de verificação (*checksum*), utilizado para validar a integridade da transmissão. Caso o *checksum* não seja consistente, a leitura é descartada e um erro é registrado no log do sistema.

A seguir é apresentado um fluxo simplificado da rotina de leitura do sensor DHT22, evidenciando o processo de validação dos dados recebidos.

```
bool dht_read(gpio_num_t pin, float *temperature, float *humidity) {
    int data[5] = {0, 0, 0, 0, 0};
    int laststate = 1;
    int counter = 0;
    int j = 0;
    gpio_set_direction(pin, GPIO_MODE_OUTPUT);
    gpio_set_level(pin, 0);
    vTaskDelay(pdMS_TO_TICKS(20));
    gpio_set_level(pin, 1);
    esp_rom_delay_us(80);  // Pulso de start
    gpio_set_direction(pin, GPIO_MODE_INPUT);
    for (int i = 0; i < MAX_TIMINGS; i++) {
        counter = 0;
        while (gpio_get_level(pin) == laststate) {
            counter++;
            esp_rom_delay_us(1);
            if (counter >= 255) break;
        }
        laststate = gpio_get_level(pin);
        if (counter >= 255) break;
        // Apenas a partir do 4   pulso e nos pares (representam bits)
        if ((i >= 4) && (i % 2 == 0)) {
            data[j / 8] <<= 1;
            if (counter > 40)  // Limiar ajustado para reconhecer bit
               "1"
                data[j / 8] |= 1;
            j++;
        }
    }
    ESP_LOGI(TAG, "Bits lidos: %d", j);
    ESP_LOGI(TAG, "Dados brutos: [%d, %d, %d, %d, %d]", data[0], data
       [1], data[2], data[3], data[4]);
    if (j < 40) {
        ESP_LOGE(TAG, "Dados insuficientes (%d bits)", j);
        return false;
    }
    int checksum = (data[0] + data[1] + data[2] + data[3]) & 0xFF;
    if (data[4] != checksum) {
        ESP_LOGE(TAG, "Checksum incorreto: recebido=%d, calculado=%d",
           data[4], checksum);
        return false;
```

```
39      }
40      *humidity = ((data[0] << 8) + data[1]) * 0.1;
41      *temperature = (((data[2] & 0x7F) << 8) + data[3]) * 0.1;
42      if (data[2] & 0x80) *temperature *= -1;
43      return true;
44  }
```

Listing 4.3 – Fluxo de leitura e validacao do sensor DHT22

Essa abordagem garante que apenas leituras válidas sejam utilizadas pelo sistema decisor, evitando que dados corrompidos influenciem o processo de inferência. Na implementação final do firmware, foram adicionados mecanismos complementares de robustez, como cache da última leitura válida, limitação de taxa de leitura/releitura (*rate-limit/retry*) e proteção por mutex/seção crítica curta durante a captura dos pulsos, reduzindo falhas associadas a jitter e concorrência.

Sensores Analógicos e ADC Compartilhado

A umidade do solo é obtida por meio de um sensor capacitivo conectado ao conversor analógico-digital (ADC) do ESP32-S3. O sensor de luminosidade (LDR), também conectado ao ADC, fornece uma estimativa percentual da intensidade luminosa incidente.

Para evitar conflitos de configuração e duplicação de código, o firmware utiliza uma camada de acesso compartilhado ao ADC, responsável por inicializar o periférico e gerenciar os canais utilizados por múltiplos sensores. O valor bruto retornado pelo ADC, tipicamente no intervalo de 0 a 4095, é convertido para uma escala percentual.

O fluxo geral de leitura utilizando o ADC compartilhado é apresentado a seguir.

```
1   adc_oneshot_unit_handle_t adc1_get_handle(void) {
2       if (!s_adc1) {
3           adc_oneshot_unit_init_cfg_t cfg = { .unit_id = ADC_UNIT_1 };
4           ESP_ERROR_CHECK(adc_oneshot_new_unit(&cfg, &s_adc1));
5       }
6       return s_adc1;
7   }
8
9   esp_err_t adc1_config_channel_once(adc_channel_t channel,
10                                     const adc_oneshot_chan_cfg_t *cfg) {
11      uint16_t bit = (1u << channel);
12      if (s_cfg_mask & bit) return ESP_OK;
13      adc_oneshot_unit_handle_t h = adc1_get_handle();
14      esp_err_t err = adc_oneshot_config_channel(h, channel, cfg);
15      if (err == ESP_OK) s_cfg_mask |= bit;
16      return err;
17  }
18
```

```
19 esp_err_t adc1_read_raw(adc_channel_t channel, int *out_raw) {
20     adc_oneshot_unit_handle_t h = adc1_get_handle();
21     return adc_oneshot_read(h, channel, out_raw);
22 }
```

Listing 4.4 – Fluxo de leitura utilizando ADC compartilhado

No caso do sensor de umidade do solo, valores mais altos indicam menor umidade, sendo necessária a inversão da escala para obter um percentual intuitivo. Na implementação atual, essa conversão utiliza calibração por dois pontos (`SOIL_RAW_SECO` e `SOIL_RAW_MOLHADO`), em vez de um mapeamento fixo 0–4095, permitindo melhor aderência ao sensor e ao ambiente de teste.

```
1 int moisture_get_umidade(void) {
2     int raw = moisture_read_raw();
3
4     int umidade = (SOIL_RAW_SECO - raw) * 100 / (SOIL_RAW_SECO -
          SOIL_RAW_MOLHADO);
5
6     if (umidade < 0) umidade = 0;
7     if (umidade > 100) umidade = 100;
8     return umidade;
9 }
```

Listing 4.5 – Fluxo de conversao calibrada da leitura do sensor de umidade do solo

O sensor LDR utiliza abordagem semelhante, sendo a leitura convertida para uma estimativa percentual de luminosidade, conforme o fluxo apresentado a seguir.

```
int ldr_get_luminosidade(void) {
    int raw = 0;
    esp_err_t err = adc1_read_raw(LDR_ADC_CHANNEL, &raw);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Falha ao ler LDR: %s", esp_err_to_name(err));
        return -1;
    }
    int lum = (raw * 100) / 4095;
    if (lum < 0) lum = 0;
    if (lum > 100) lum = 100;
    return lum;
}
```

Listing 4.6 – Fluxo de leitura e conversao do sensor LDR

Embora o LDR esteja funcionalmente integrado ao firmware, sua leitura não compõe o vetor de características da versão atual do modelo embarcado e o *gate* opcional por luminosidade encontra-se desabilitado na configuração avaliada. Assim, sua presença no protótipo contribui para extensões futuras e testes de instrumentação, sem impactar diretamente a decisão final reportada neste trabalho.

Abstração das Leituras

As leituras dos sensores são abstraídas por macros definidas em `app_config.h`, permitindo que o módulo decisor utilize as medições sem depender diretamente da implementação dos *drivers*. Essa estratégia aumenta a portabilidade do código e simplifica a lógica do módulo decisor.

Não foram implementados filtros digitais, médias móveis ou técnicas de suavização gerais nas leituras. Essa decisão de projeto visa reduzir a complexidade computacional e manter a responsividade do sistema, delegando a estabilidade da decisão final aos mecanismos de histerese, temporização e demais regras determinísticas aplicadas em etapas posteriores do processo.

4.4 Previsão de Chuva

A informação de chuva utilizada pelo sistema é incorporada como variável auxiliar ao processo de decisão de irrigação. Diferentemente de abordagens baseadas em consultas a APIs externas em tempo de execução, o sistema adota uma estratégia embutida no firmware, garantindo funcionamento determinístico e reduzindo a dependência de conectividade durante a operação.

Na implementação atual, a série de chuva acumulada em 24 horas é gerada no pipeline externo e exportada juntamente com os parâmetros do modelo no arquivo `ia_params.h`. Esse arquivo contém tanto os valores da série quanto os metadados necessários à sua indexação em tempo de execução, como o instante de referência inicial e o intervalo entre amostras. Assim, a "previsão de chuva" utilizada no dispositivo corresponde a uma série climática previamente empacotada no firmware, e não a uma consulta meteorológica online em tempo real durante a operação.

O módulo responsável pela manipulação dessa informação no firmware encontra-se em `esp32_firmware/main/forecast/`, sendo composto pelos arquivos `forecast.c` e `forecast.h`, e expõe uma interface simples para obtenção do valor vigente de chuva de 24 horas.

Representação da Série Embutida

A série de chuva é armazenada como um vetor estático, representando valores de chuva acumulada em 24 horas. Essa série é gerada no ambiente de treinamento a partir de dados climáticos históricos pré-processados no pipeline Python e exportada para o firmware como parte do arquivo `ia_params.h`.

Além do vetor principal, o arquivo `ia_params.h` define constantes auxiliares que descrevem a série temporal, incluindo:

- o instante inicial da série (epoch de referência);
- o intervalo entre amostras consecutivas;
- o tamanho total do vetor.

Essas informações permitem que o firmware selecione, em tempo de execução, o índice correspondente ao tempo atual do sistema e obtenha o valor de chuva embutido mais apropriado.

Cálculo do Valor Vigente

O índice da série é calculado com base no relógio do sistema, retornando o valor de chuva acumulada em 24 horas associado ao instante corrente. O índice calculado é limitado ao intervalo válido do vetor, evitando acesso fora dos limites da memória.

O fluxo simplificado desse cálculo é apresentado a seguir.

```
1 float forecast_chuva24h_now(void) {
2     #ifdef FORECAST_LEN
3         time_t now = time(NULL);                 // epoch (s)
4         double step_s = (double)FORECAST_STEP_MIN * 60.0;
```

```
5           double idxf = ((double)now - (double)FORECAST_T0_EPOCH) / step_s
                ;
6           int idx = (int)lrint(idxf);             // arredonda pro inteiro
                mais próximo
7           idx = clampi(idx, 0, (int)FORECAST_LEN - 1);
8           return FORECAST_CHUVA_MM24[idx];
9       #else
10          return 0.0f; // se não existir série embutida no header
11      #endif
12  }
```

Listing 4.7 – Fluxo de calculo do valor de chuva acumulada em 24 horas no firmware

Caso os metadados da série não estejam presentes no arquivo de cabeçalho (por exemplo, em versões reduzidas do `ia_params.h`), o módulo retorna um valor padrão. Essa decisão simplifica o tratamento de erro e evita falhas de execução no firmware.

Uso no Processo Decisório

Na implementação atual, o valor de chuva de 24 horas desempenha dois papéis no processo de decisão de irrigação. Primeiro, ele pode compor o vetor de entrada do modelo de aprendizado de máquina (conforme a configuração e o número de *features* exportadas), influenciando o cálculo da probabilidade de irrigação.

Além disso, essa informação é utilizada como regra determinística de bloqueio. Caso o valor de chuva de 24 horas ultrapasse um limiar configurável e o solo esteja acima de uma faixa considerada adequada, o sistema impede o acionamento da bomba de irrigação, independentemente da probabilidade estimada pelo modelo.

O fluxo simplificado de aplicação da regra de bloqueio por chuva é apresentado a seguir.

```
1  /* trecho do decider_tick() */
2  float ch24 = forecast_chuva24h_now();
3
4  if ((ch24 >= CHUVA_BLOQUEIO_MM24) && (soil >= SOLO_OK_PCT)) {
5      if (s_prev) {
6          uint64_t n = now_ms();
7          if (n - s_last_switch_ms < MIN_ON_MS) { *out_on = true; return
               true; }
8          s_prev = false; s_last_switch_ms = n;
9      }
10     *out_on = false; return true;
11 }
```

Listing 4.8 – Aplicacao da regra de bloqueio por chuva no modulo decisor

Essa abordagem aumenta a segurança operacional, evitando irrigação desnecessária em cenários nos quais a chuva acumulada considerada pelo sistema indica baixa urgência de irrigação.

Considerações de Projeto

A escolha por uma série de chuva embutida no firmware elimina dependências externas em tempo de execução e reduz a complexidade de comunicação em campo. Em contrapartida, essa abordagem exige a atualização do artefato `ia_params.h` e a recompilação do firmware quando se deseja incorporar novos valores.

Na forma atual, essa série deve ser interpretada como um mecanismo auxiliar e conservador para a decisão de irrigação, e não como um serviço de previsão meteorológica em tempo real. Essa limitação foi considerada aceitável no contexto do sistema proposto, no qual a prioridade é demonstrar inferência embarcada determinística e integração ponta a ponta entre pipeline externo e firmware.

4.5 Módulo de Inteligência Artificial

O módulo de inteligência artificial é responsável por estimar a probabilidade de necessidade de irrigação a partir das variáveis ambientais medidas pelo sistema. A inferência é executada localmente no microcontrolador ESP32-S3, utilizando um modelo de Regressão Logística treinado externamente e incorporado ao firmware por meio de um arquivo de cabeçalho em linguagem C.

A escolha pela execução local da inferência visa garantir baixa latência, previsibilidade temporal e independência de conectividade, características fundamentais para aplicações embarcadas em ambientes agrícolas.

Modelo de Regressão Logística

O modelo adotado é uma regressão logística binária, cujo objetivo é estimar a probabilidade $P(y = 1 \mid X)$, onde $y = 1$ representa a decisão de irrigar e X corresponde ao vetor de características ambientais.

Matematicamente, a regressão logística é definida como:

$$P(y = 1 \mid X) = \sigma(z), \quad z = \sum_{i=1}^{n} w_i \cdot x_i + b$$

em que w_i são os pesos do modelo, b é o termo de bias e $\sigma(\cdot)$ representa a função sigmoide. Essa formulação apresenta baixo custo computacional, sendo adequada à execução em dispositivos embarcados com recursos limitados.

Na implementação embarcada, essa inferência é complementada por um ajuste heurístico no *logit* associado ao estado de umidade do solo, com o objetivo de reduzir a probabilidade de irrigação quando o solo já se encontra suficientemente úmido. Esse ajuste é aplicado no firmware após a normalização das entradas e antes da função sigmoide.

Exportação dos Parâmetros do Modelo

O treinamento do modelo é realizado externamente no pipeline em Python. Ao final do processo, os parâmetros aprendidos são exportados para o arquivo `ia_params.h`, que consolida:

- número de variáveis de entrada (`N_FEATS`);
- pesos do modelo;
- termo de bias;
- estatísticas de normalização (média e escala);
- ordem fixa das *features*.

Esse arquivo é incluído diretamente no firmware, garantindo consistência entre o treinamento e a inferência embarcada, sem necessidade de recalcular ou ajustar parâmetros em tempo de execução.

Normalização do Vetor de Entrada e Ajuste do *Logit*

Antes da inferência, o vetor de entrada é normalizado utilizando as estatísticas calculadas durante o treinamento. Essa etapa é fundamental para garantir que os valores processados pelo modelo estejam na mesma escala utilizada na fase de aprendizado.

Na implementação atual, após o cálculo do *logit* da regressão logística, é aplicado um termo de correção dependente da umidade do solo. Esse termo atua como penalização quando o solo está acima de uma faixa de referência configurável, tornando a decisão mais conservadora em cenários de solo úmido.

O fluxo simplificado é apresentado a seguir.

```
1  float ia_predict_proba(const float in[N]) {
2      float x_norm[N];
3      for (int i = 0; i < N; ++i) {
4          x_norm[i] = safe_div(in[i] - SCALER_MEAN[i], SCALER_SCALE[i]);
5      }
6
7      float soil = in[0];
8      float logit_before = dotN(W, x_norm) + B[0];
9
10     float wet = soil_wet_01(soil);                    // 0..1
```

```
11      float off = -IA_SOIL_LOGIT_PENALTY * wet;       // penaliza solo
            umido
12
13      float logit_after = logit_before + off;
14      return sigmoidf(logit_after);
15  }
```

Listing 4.9 – Fluxo de normalizacao e ajuste do logit no modulo de IA

Essa normalização e o ajuste do *logit* são implementados diretamente no módulo de IA do firmware, utilizando os vetores exportados no arquivo `ia_params.h` e parâmetros de configuração definidos no firmware.

Inferência do Modelo

Após a normalização, o módulo de IA calcula o valor do *logit* por meio do produto escalar entre o vetor normalizado e os pesos do modelo, somado ao termo de bias. Em seguida, aplica-se a função sigmoide para obtenção da probabilidade de irrigação.

O fluxo de inferência é apresentado a seguir.

```
1  static inline float dotN(const float *a, const float *b) {
2      float s = 0.0f;
3      for (int i = 0; i < N; ++i) s += a[i] * b[i];
4      return s;
5  }
6
7  static inline float sigmoidf(float x) {
8      if (x > 16.0f)  x = 16.0f;
9      if (x < -16.0f) x = -16.0f;
10     return 1.0f / (1.0f + expf(-x));
11 }
```

Listing 4.10 – Funcoes auxiliares da inferencia da regressao logistica

Para garantir estabilidade numérica, o valor do *logit* é limitado a um intervalo seguro antes da aplicação da função sigmoide, evitando estouros ou subfluxos durante o cálculo exponencial.

Aplicação de Histerese

A probabilidade estimada pelo modelo não é utilizada diretamente para o acionamento da bomba. Em vez disso, o sistema aplica um mecanismo de histerese, definido por dois limiares distintos: um limiar superior para acionamento e um limiar inferior para desligamento.

Esse mecanismo evita comutações frequentes do relé em situações nas quais a probabilidade oscila em torno de um único limiar, aumentando a estabilidade operacional do sistema.

De forma prática, quando o estado anterior é desligado, o sistema exige que a probabilidade ultrapasse o limiar superior (`high_th`) para ligar a bomba. Quando o estado anterior é ligado, a bomba só é desligada se a probabilidade cair abaixo do limiar inferior (`low_th`). Essa "faixa morta" entre limiares reduz o efeito de oscilações pequenas na saída do modelo. Além disso, o módulo decisor aplica tempos mínimos de permanência ligado/desligado (`MIN_ON_MS` e `MIN_OFF_MS`), reforçando a proteção contra chaveamento rápido (*anti-chattering*).

O fluxo de aplicação da histerese é descrito a seguir.

```
1 bool ia_should_irrigate_hysteresis(const float in[N], float low_th,
     float high_th, bool previous_state) {
2     float p = ia_predict_proba(in);
3     if (previous_state) return p > low_th;   // desligar só se cair bem
          abaixo
4     else                return p > high_th;  // ligar só se subir bem
          acima
5 }
```

Listing 4.11 – Fluxo de decisao com histerese no modulo de IA

Os valores de `low_th` e `high_th` são fornecidos pelo módulo decisor. Na implementação atual, além dos valores-base configurados em `app_config.h`, o módulo decisor pode aplicar ajustes adaptativos nesses limiares em função da umidade do solo, preservando a histerese e modulando a sensibilidade da decisão.

Integração com o Módulo Decisor

O módulo de inteligência artificial não atua diretamente sobre o atuador. Sua função é fornecer ao módulo decisor uma estimativa probabilística e uma decisão com histerese a partir das variáveis ambientais.

O módulo decisor combina essa saída com regras adicionais de segurança, como bloqueio por chuva, restrições de horário e tempos mínimos de acionamento/desligamento (anti-chattering), garantindo que o acionamento final da bomba seja coerente com as condições ambientais e operacionais do sistema.

Essa separação reforça a modularidade da arquitetura, permitindo a substituição futura do modelo de aprendizado de máquina com impacto reduzido sobre os demais componentes do firmware.

4.6 Controle da Bomba de Irrigação

O controle da bomba de irrigação é implementado de forma modular, por meio de um *driver* dedicado responsável exclusivamente pelo acionamento do relé conectado ao microcontrolador ESP32-S3. Esse módulo não contém lógica de decisão; sua função é executar comandos de liga/desliga conforme solicitado pelo módulo decisor.

Essa separação de responsabilidades reduz o acoplamento entre a lógica de controle e o hardware, facilitando testes, manutenção e eventuais substituições do atuador.

Configuração do Atuador

As configurações diretamente relacionadas ao módulo da bomba são centralizadas no arquivo `app_config.h`. Entre os parâmetros definidos destacam-se:

- pino GPIO utilizado para o relé;
- polaridade elétrica do acionamento (*active high* ou *active low*);
- habilitação do modo de simulação (*dry-run*).

Parâmetros temporais, como tempos mínimos de acionamento e desligamento, também são configurados em `app_config.h`, porém sua aplicação ocorre no módulo decisor (e não no *driver* da bomba).

Essa abordagem permite ajustar o comportamento do sistema sem modificações diretas no código-fonte do módulo de controle da bomba.

Inicialização do Módulo da Bomba

Durante a inicialização do firmware, o módulo da bomba configura o GPIO correspondente como saída digital e posiciona o relé em estado desligado, respeitando a polaridade definida. Caso o modo de simulação esteja habilitado, nenhuma configuração elétrica é realizada, e o sistema apenas registra os comandos em log.

O fluxo de inicialização do módulo é apresentado a seguir.

```
1 void pump_init(void) {
2     #if PUMP_DRY_RUN
3         ESP_LOGW(TAG, "DRY-RUN ATIVO: não configurando GPIO %d; apenas
              simulando.", PUMP_GPIO);
4     #else
5         gpio_config_t io = {
6             .pin_bit_mask = 1ULL << PUMP_GPIO,
7             .mode = GPIO_MODE_OUTPUT,
8             .pull_up_en = GPIO_PULLUP_DISABLE,
9             .pull_down_en = GPIO_PULLDOWN_DISABLE,
```

```
10              .intr_type = GPIO_INTR_DISABLE
11          };
12          gpio_config(&io);
13          int off_level = PUMP_ACTIVE_HIGH ? 0 : 1;
14          gpio_set_level(PUMP_GPIO, off_level);
15          ESP_LOGI(TAG, "GPIO %d configurado (active_%s).", PUMP_GPIO,
                PUMP_ACTIVE_HIGH ? "HIGH" : "LOW");
16      #endif
17  }
```

Listing 4.12 – Fluxo de inicializacao do modulo da bomba

Acionamento do Relé

O acionamento da bomba é realizado por meio de uma função simples que recebe como parâmetro o estado desejado (*ligado* ou *desligado*). O módulo converte esse estado em nível lógico conforme a polaridade elétrica configurada e aplica o valor ao GPIO correspondente.

O fluxo de acionamento do relé é descrito a seguir.

```
1  void pump_set(bool on) {
2      s_on = on;
3      #if PUMP_DRY_RUN
4          ESP_LOGI(TAG, "[SIM] pump_set(%s) -> GPIO %d %s",
5                   on ? "ON" : "OFF",
6                   PUMP_GPIO,
7                   on ? (PUMP_ACTIVE_HIGH ? "HIGH" : "LOW") : (
                        PUMP_ACTIVE_HIGH ? "LOW" : "HIGH"));
8      #else
9          int level = on ? (PUMP_ACTIVE_HIGH ? 1 : 0) : (PUMP_ACTIVE_HIGH
               ? 0 : 1);
10         gpio_set_level(PUMP_GPIO, level);
11         ESP_LOGI(TAG, "pump_set(%s) -> GPIO %d level=%d", on ? "ON" : "
               OFF", PUMP_GPIO, level);
12     #endif
13 }
```

Listing 4.13 – Fluxo de acionamento da bomba

Esse módulo não realiza verificações adicionais de segurança, delegando integralmente a decisão de acionamento ao módulo decisor.

Proteção Contra Comutações Frequentes

Para evitar desgaste prematuro do relé e oscilações indesejadas no sistema hidráulico, o firmware implementa tempos mínimos de acionamento e desligamento, conhecidos como

mecanismos de *anti-chattering*. Esses mecanismos são aplicados no módulo decisor, antes da chamada ao módulo da bomba.

Dois parâmetros principais são utilizados:

- tempo mínimo em que a bomba deve permanecer ligada;
- tempo mínimo em que a bomba deve permanecer desligada.

O fluxo de verificação desses tempos é apresentado a seguir.

```
/* trecho do decider_tick() */
uint64_t n = now_ms();
if (vote && !s_prev) {
    if (n - s_last_switch_ms < MIN_OFF_MS) vote = false;
} else if (!vote && s_prev) {
    if (n - s_last_switch_ms < MIN_ON_MS)  vote = true;
}
```

Listing 4.14 – Fluxo de verificacao de tempos minimos

Essa verificação garante que o relé não seja comutado repetidamente em intervalos curtos, mesmo em cenários nos quais a probabilidade de irrigação oscila devido a ruído nas leituras dos sensores.

Integração com o Modulo Decisor

O módulo da bomba é acionado exclusivamente pelo módulo decisor, que consolida:

- a saída do modelo de aprendizado de máquina;
- o mecanismo de histerese (com limiares potencialmente ajustados de forma adaptativa);
- o bloqueio por chuva (com base na série embutida no firmware);
- restrições de horário e a verificação de tempos mínimos de acionamento.

Somente após a validação por todas essas regras o comando final é enviado ao módulo da bomba. Essa arquitetura garante que o atuador opere sempre em condições controladas, previsíveis e seguras, reforçando a robustez do sistema de irrigação inteligente.

4.7 Integração com o Pipeline de Treinamento

A integração entre o pipeline externo de treinamento e o firmware embarcado constitui um dos principais elementos arquiteturais do sistema proposto. Essa integração garante que o modelo de aprendizado de máquina utilizado na inferência embarcada seja

treinado, validado e parametrizado em ambiente computacional apropriado, mantendo consistência entre treinamento e execução no dispositivo.

O pipeline de treinamento foi implementado em Python, no diretório `treinamento_externo/`, e é responsável pela coleta de dados climáticos, preparação do conjunto de dados, treinamento do modelo preditivo e exportação dos parâmetros necessários para a inferência no microcontrolador.

Coleta e Preparação dos Dados

A etapa inicial do pipeline consiste na coleta de dados climáticos históricos por meio de uma API externa e gratuita (*Open-Meteo Archive*). O script responsável realiza a obtenção de séries temporais diárias de temperatura e precipitação para a região de João Monlevade-MG (latitude -19.82 e longitude -43.17), organizando-as em um conjunto de dados estruturado no formato CSV.

Esses dados são posteriormente preparados para compatibilização com o formato de entrada esperado pelo firmware e pelo modelo, incluindo renomeação de colunas, validação de tipos e tratamento de valores inválidos. Na implementação, são aplicadas verificações de colunas esperadas, conversões numéricas, descarte de linhas inconsistentes e restrição de precipitação para valores não negativos.

O fluxo de coleta e preparação dos dados pode ser descrito conforme apresentado a seguir.

```
for year in range(start_year, end_year + 1):
    start_date = f"{year}-01-01"
    end_date = today.isoformat() if year == end_year else f"{year
        }-12-31"

    url = (
        "https://archive-api.open-meteo.com/v1/archive"
        f"?latitude={latitude}&longitude={longitude}"
        f"&start_date={start_date}&end_date={end_date}"
        "&daily=temperature_2m_max,temperature_2m_min,precipitation_sum"
        "&timezone=America/Sao_Paulo"
    )

    try:
        r = requests.get(url, timeout=30)
        r.raise_for_status()
        data = r.json()
    except Exception as e:
        print(f"Falha HTTP no ano {year}: {e}")
        continue

```

```
21      daily = data.get("daily")
22      if not daily:
23          print(f"Sem bloco 'daily' no ano {year}. Payload: {data.keys()
                }")
24          continue
25
26      df = pd.DataFrame(daily)
27
28      if not {"time", "temperature_2m_max", "temperature_2m_min", "
            precipitation_sum"} <= set(df.columns):
29          print(f"Colunas inesperadas no ano {year}: {df.columns.tolist()
                }")
30          continue
31
32      df["time"] = pd.to_datetime(df["time"], errors="coerce", utc=False)
33      df["precipitation_sum"] = pd.to_numeric(df["precipitation_sum"],
            errors="coerce").clip(lower=0.0)
34      df["temperature_2m_max"] = pd.to_numeric(df["temperature_2m_max"],
            errors="coerce")
35      df["temperature_2m_min"] = pd.to_numeric(df["temperature_2m_min"],
            errors="coerce")
36      df = df.dropna(subset=["time", "temperature_2m_max", "
            temperature_2m_min", "precipitation_sum"])
37
38      if df.empty:
39          print(f"Sem linhas validas no ano {year}")
40          continue
41
42      df_total = pd.concat([df_total, df], ignore_index=True)
43      print(f"Dados de {year} coletados: {len(df)} linhas")
44
45  if df_total.empty:
46      raise RuntimeError("Nenhum dado coletado. Verifique a rede/URL/
            intervalos.")
47
48  df_total = df_total.rename(columns={
49      "time": "data",
50      "temperature_2m_max": "temp_max",
51      "temperature_2m_min": "temp_min",
52      "precipitation_sum": "chuva_mm"
53  })[["data", "temp_max", "temp_min", "chuva_mm"]]
54
55  output_path = Path(__file__).resolve().parent / "
        dados_climaticos_tratados.csv"
56  df_total.to_csv(output_path, index=False, float_format="%.3f")
57  print(f"Dados tratados salvos em: {output_path} | linhas={len(df_total)
        }")
```

Listing 4.15 – Fluxo de coleta e preparacao dos dados

Treinamento do Modelo Preditivo

O treinamento do modelo é realizado a partir do conjunto de dados preparado, utilizando uma Regressão logística binária. Essa escolha foi motivada pelo baixo custo computacional, facilidade de interpretação e viabilidade de implementação em sistemas embarcados.

Durante o treinamento, são definidas explicitamente as variáveis de entrada (*features*) e o alvo binário (*irrigar*). Além disso, são calculadas as estatísticas de normalização (média e escala) que serão reutilizadas na inferência embarcada.

Na configuração atual do pipeline, parte das variáveis de entrada é construída por meio de valores *proxy* (por exemplo, para compatibilizar dados históricos com entradas esperadas pelo firmware), e o alvo binário é definido por um critério simplificado baseado em chuva acumulada em 24 horas. Essa configuração foi adotada com foco na validação técnica da integração treino–exportação–inferência, e não como validação agronômica definitiva do critério de irrigação.

Na etapa de experimentação da versão atual do pipeline, o conjunto de dados possui 2.054 amostras, com distribuição desbalanceada entre classes (*irrigar*=80,72% e *não irrigar*=19,28%). A separação do dataset foi realizada em treino/teste na proporção 80/20 (`test_size=0.2`), com estratificação (`stratify=y`) para preservar a proporção entre classes. Não foram utilizados aumento de dados (*data augmentation*), validação cruzada nem divisão explícita em treino/validação/teste (80/10/10) nesta versão. Também não foi construída curva de aprendizado nesta etapa; assim, a análise formal de *overfitting* permanece limitada ao comportamento observado nas métricas reportadas em teste.

O fluxo simplificado de treinamento do modelo é apresentado a seguir.

```
1  def main():
2      assert CSV.exists(), f"CSV não encontrado: {CSV}"
3      df = pd.read_csv(CSV, parse_dates=["data"]).sort_values("data")
4
5      if "chuva_mm_24h" not in df.columns:
6          if "chuva_mm" in df.columns:
7              df["chuva_mm_24h"] = df["chuva_mm"].astype(float).clip(lower
                   =0.0)
8          else:
9              df["chuva_mm_24h"] = 0.0
10
11     if "soil_pct" not in df.columns:
12         df["soil_pct"] = 30.0
```

```
13      if "temp_c" not in df.columns:
14          if {"temp_max", "temp_min"} <= set(df.columns):
15              df["temp_c"] = ((df["temp_max"] + df["temp_min"]) / 2.0).
                    fillna(25.0)
16          else:
17              df["temp_c"] = 25.0
18      if "rh_pct" not in df.columns:
19          df["rh_pct"] = 60.0
20
21      df["irrigar"] = (df["chuva_mm_24h"] < THRESH_CHUVA).astype(int)
22
23      X = df[FEATURES].astype(float).values
24      y = df["irrigar"].values
25      X_tr, X_te, y_tr, y_te = train_test_split(
26          X, y, test_size=TEST_SIZE, stratify=y, random_state=RANDOM_STATE
27      )
28
29      MEAN  = SCALER_MEAN[:len(FEATURES)].astype(float)
30      SCALE = SCALER_SCALE[:len(FEATURES)].astype(float)
31
32      X_tr_s = (X_tr - MEAN) / SCALE
33      X_te_s = (X_te - MEAN) / SCALE
34
35      scaler = StandardScaler()
36      scaler.mean_  = MEAN
37      scaler.scale_ = SCALE
38
39      clf = LogisticRegression(max_iter=1000, random_state=RANDOM_STATE)
40      clf.fit(X_tr_s, y_tr)
41
42      y_pred = clf.predict(X_te_s)
43      acc = accuracy_score(y_te, y_pred)
44      f1 = f1_score(y_te, y_pred)
45      cm = confusion_matrix(y_te, y_pred)
46      report = classification_report(y_te, y_pred, digits=4)
47
48      joblib.dump(clf, OUT / "model.pkl")
49      joblib.dump(scaler, OUT / "scaler.pkl")
50      with open(OUT / "metrics.txt", "w", encoding="utf-8") as f:
51          f.write("Modelo: regressao_logistica\n")
52          f.write(f"Acuracia: {acc:.4f}\nF1: {f1:.4f}\n")
53          f.write(f"Matriz de confusao:\n{cm}\n\n{report}\n")
```

Listing 4.16 – Fluxo de treinamento do modelo de Regressao Logistica

Os resultados da avaliação do modelo são registrados em arquivos auxiliares, permitindo análise posterior do desempenho do classificador antes de sua incorporação ao

firmware.

Exportação dos Parâmetros para o Firmware

Após o treinamento, os parâmetros aprendidos pelo modelo são exportados para um arquivo de cabeçalho em linguagem C, denominado `ia_params.h`. Esse arquivo concentra as informações necessárias para a inferência embarcada, eliminando dependências externas no dispositivo.

Entre os elementos exportados destacam-se:

- número de variáveis de entrada;
- pesos e bias do modelo;
- vetores de média e escala para normalização;
- ordem esperada das variáveis de entrada;
- série temporal de chuva em 24 horas e seus metadados.

O fluxo de geração do arquivo de cabeçalho é descrito a seguir.

```
def export_to_header(
    clf: LogisticRegression,
    scaler: StandardScaler,
    features,
    forecast_t0_epoch: int,
    forecast_step_min: int,
    forecast_vals: np.ndarray,
    out_path: Path,
):
    W_vec = np.array(clf.coef_, dtype=float).ravel()
    b = float(np.array(clf.intercept_, dtype=float).ravel()[0])

    n = len(features)
    feat_order_str = ", ".join(features)
    ia_version = datetime.now().strftime("%Y%m%d")
    forecast_vals = np.array(forecast_vals, dtype=float).ravel()

    header = f"""/* IA params (gerado por treinar_modelo.py) */
#pragma once

#define IA_VERSION {ia_version}
// FEATURE_ORDER: {feat_order_str}

#define N_FEATS {n}
static const float SCALER_MEAN[{n}]  = {{ {_fmt_list(scaler.mean_)} }};
static const float SCALER_SCALE[{n}] = {{ {_fmt_list(scaler.scale_)} }};
static const float W[{n}]            = {{ {_fmt_list(W_vec)} }};
```

```
static const float B[1]              = {{ {b:.8f}f }};

#define FORECAST_T0_EPOCH  {int(forecast_t0_epoch)}
#define FORECAST_STEP_MIN  {int(forecast_step_min)}
#define FORECAST_LEN       {int(len(forecast_vals))}
static const float FORECAST_CHUVA_MM24[FORECAST_LEN] = {{ {_fmt_list(
    forecast_vals)} }};
"""
    out_path.parent.mkdir(parents=True, exist_ok=True)
    out_path.write_text(header, encoding="utf-8")
```

Listing 4.17 – Fluxo de exportacao do header ia_params.h

Esse mecanismo garante que a inferência embarcada utilize exatamente os mesmos parâmetros empregados durante o treinamento, evitando inconsistências numéricas ou semânticas.

Integração com o Processo de Build

O arquivo `ia_params.h` gerado pelo pipeline é incorporado ao firmware durante o processo de compilação. No fluxo manual, o desenvolvedor copia o arquivo para o diretório do módulo de inteligência artificial antes da execução do `idf.py build`.

Adicionalmente, o repositório contempla a automação desse processo por meio de workflows de integração contínua, nos quais o treinamento (opcional, conforme configuração do *workflow*) e a compilação do firmware podem ser encadeados de forma controlada.

O fluxo geral de integração entre treinamento e build do firmware é apresentado a seguir.

```
- name: Posicionar header de ML (artefato ou fallback do repositório)
  run: |
    set -euo pipefail
    mkdir -p esp32_firmware/main/ia

    if [ -f ml-artifacts/ia_params.h ]; then
      SRC="ml-artifacts/ia_params.h"
      echo "Usando ia_params.h do artefato de treino: $SRC"
    elif [ -f treinamento_externo/ia_params.h ]; then
      SRC="treinamento_externo/ia_params.h"
      echo "Usando ia_params.h versionado: $SRC"
    elif [ -f ia_params.h ]; then
      SRC="ia_params.h"
      echo "Usando ia_params.h na raiz: $SRC"
    else
      echo "::error::ia_params.h não encontrado."
      exit 1
```

```
18      fi
19
20      cp "$SRC" esp32_firmware/main/ia/ia_params.h
21
22  - name: Compilar firmware
23    working-directory: esp32_firmware
24    run: |
25      set -euo pipefail
26      source "$HOME/esp/esp-idf/export.sh"
27      idf.py set-target esp32s3
28      idf.py build
```

Listing 4.18 – Fluxo de integracao treino-firmware

Essa estratégia permite atualizar o comportamento do sistema de irrigação a partir de novos dados ou ajustes no modelo, mantendo a robustez e a previsibilidade exigidas por aplicações embarcadas em ambiente agrícola. Na arquitetura atual, a distribuição para o dispositivo é realizada por atualização OTA do firmware compilado (imagem completa), que já incorpora a versão desejada do `ia_params.h`.

4.8 Decisões de Projeto e Trade-offs de Implementação

O desenvolvimento do sistema inteligente de irrigação envolveu uma série de decisões de projeto que buscaram equilibrar robustez, simplicidade, desempenho computacional e viabilidade em ambiente embarcado. Nesta seção são discutidas as principais escolhas técnicas adotadas, bem como seus impactos positivos e limitações.

Escolha do Modelo de Aprendizado de Maquina

A Regressão Logística foi adotada como modelo preditivo principal do sistema. Essa escolha foi motivada, sobretudo, pelas restrições computacionais do microcontrolador ESP32-S3 e pela necessidade de uma inferência rápida e determinística.

Entre os benefícios dessa escolha destacam-se:

- baixo custo computacional durante a inferência;
- facilidade de implementação em linguagem C;
- interpretabilidade dos pesos associados a cada variável;
- previsibilidade temporal da execução.

Como contrapartida, a Regressão Logística apresenta limitações na modelagem de relações não lineares complexas entre as variáveis ambientais. Essa limitação foi mitigada parcialmente pela combinação do modelo com regras determinísticas adicionais, como histerese, bloqueio por chuva e ajustes heurísticos no processo decisório.

Inferência Local Embarcada

A decisão de realizar a inferência do modelo diretamente no dispositivo embarcado elimina a dependência de conectividade contínua com serviços externos, característica relevante para aplicações em ambientes rurais.

Essa abordagem garante:

- baixa latência na tomada de decisão;
- funcionamento autônomo do sistema;
- maior previsibilidade operacional.

Por outro lado, a inferência local impõe a necessidade de recompilação e redistribuição do firmware sempre que o modelo é atualizado. Essa limitação é atenuada pelo uso de atualização remota via OTA do firmware e pela modularização do pipeline de treinamento.

Combinação de IA com Regras Deterministicas

O sistema não depende exclusivamente da saída do modelo de aprendizado de máquina para decidir o acionamento da irrigação. Em vez disso, a probabilidade calculada pela Regressão Logística é combinada com regras determinísticas de segurança.

Entre essas regras destacam-se:

- histerese para estabilização do acionamento;
- ajuste adaptativo dos limiares de histerese em função da umidade do solo;
- bloqueio por chuva acima de um limiar configurável;
- restrições de horário de operação;
- tempos mínimos de acionamento e desligamento (anti-chattering).

Além disso, a implementação embarcada aplica uma penalização no *logit* quando o solo está em condição mais úmida, tornando a decisão mais conservadora. Em conjunto, essas estratégias reduzem oscilações indesejadas e comportamentos erráticos causados por ruído nos sensores ou pequenas variações na probabilidade estimada. Como desvantagem, o sistema passa a depender de parâmetros de configuração (limiares, tempos e fatores heurísticos), que podem exigir ajustes manuais para diferentes culturas ou condições climáticas.

Uso de Sensores de Baixo Custo

A escolha por sensores amplamente disponíveis e de baixo custo, como o DHT22, o sensor capacitivo de umidade do solo e o LDR, favorece a replicabilidade e o baixo custo do sistema.

Entretanto, esses sensores apresentam maior suscetibilidade a ruídos, imprecisões e leituras esporadicamente inválidas. Optou-se por não aplicar filtros digitais ou médias móveis gerais nas leituras, priorizando simplicidade e responsividade. A estabilidade da decisão final é garantida principalmente pelas regras de histerese e temporização no módulo decisor, além dos mecanismos de robustez adicionados ao *driver* do DHT22.

Série de Chuva Estática Embutida

A informação de chuva em 24 horas é incorporada ao sistema como uma série temporal estática, gerada durante o treinamento e embutida no firmware. Essa decisão elimina dependências externas em tempo de execução e garante funcionamento contínuo mesmo sem acesso à internet.

Como limitação, a série não reflete alterações climáticas em tempo real. A atualização dessa informação exige a regeneração do arquivo `ia_params.h` e a recompilação do firmware, o que foi considerado aceitável dentro do escopo proposto para o sistema.

Modularização do Firmware

A arquitetura modular do firmware, com separação clara entre sensores, inferência, decisão e atuação, facilita a manutenção e a evolução do sistema. Novos sensores, modelos ou estratégias de decisão podem ser incorporados com impacto mínimo nas demais partes do código.

Essa modularização introduz leve sobrecarga estrutural, mas os benefícios em termos de clareza, testabilidade e extensibilidade justificam plenamente essa escolha no contexto de um sistema embarcado orientado a pesquisa e desenvolvimento.

Em conjunto, as decisões de projeto adotadas resultam em um sistema equilibrado, tecnicamente viável e alinhado aos objetivos do trabalho, servindo como base sólida para extensões e aprimoramentos futuros.

5 Resultados

5.1 Resultados do Pipeline Externo de Treinamento

O pipeline externo, implementado em Python, foi executado de forma automatizada por meio de um workflow de integração contínua. A execução local do runner evidenciou o disparo sequencial das etapas de treinamento do modelo, compilação do firmware e deploy OTA (Figura 10).

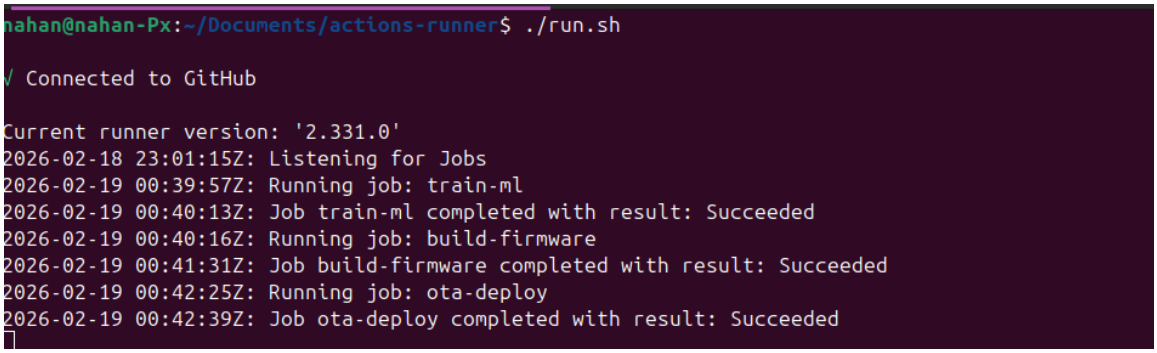

Figura 10 – Execução do runner local do GitHub Actions realizando o treinamento do modelo, compilação do firmware e deploy OTA.

A execução completa do workflow na plataforma do GitHub Actions confirmou a integração ponta a ponta entre as etapas do pipeline, com geração dos artefatos do modelo e do firmware compilado (Figura 11).

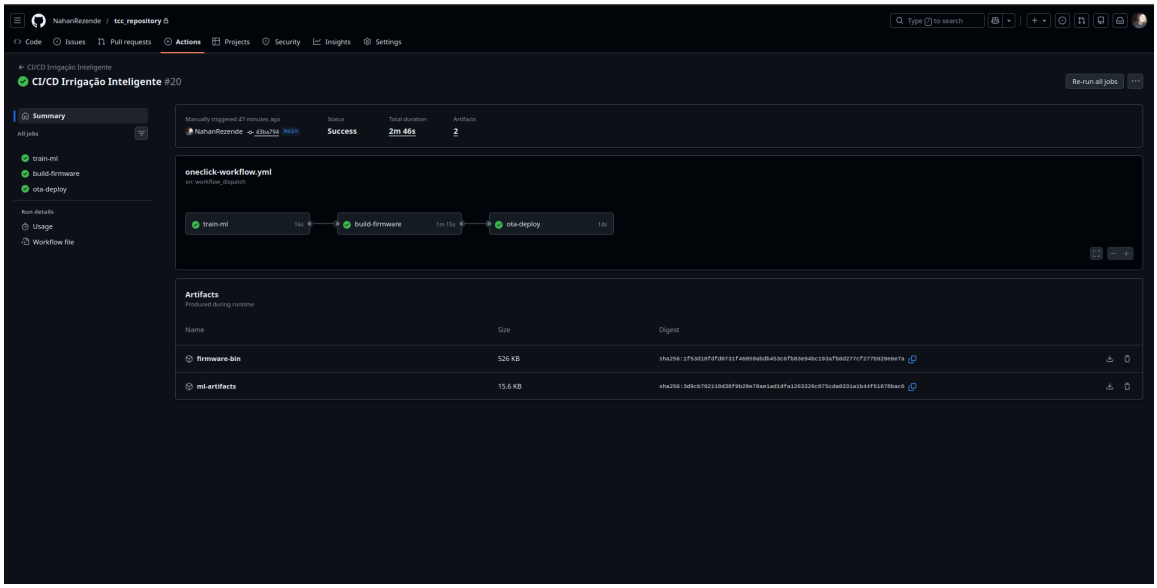

Figura 11 – Execução do workflow de integração contínua com todas as etapas concluídas com sucesso.

Na etapa de treinamento, o modelo de Regressão Logística apresentou acurácia de **0.9951** e F1-score de **0.9970** no cenário de classificação binária adotado (*irrigar* vs. *não irrigar*). A matriz de confusão e o relatório de classificação estão apresentados na Figura 12.

```
Modelo: regressao_logistica
Acurácia: 0.9951
F1: 0.9970
Matriz de confusão:
[[ 77   2]
 [  0 332]]

              precision    recall  f1-score   support

           0     1.0000    0.9747    0.9872        79
           1     0.9940    1.0000    0.9970       332

    accuracy                         0.9951       411
   macro avg     0.9970    0.9873    0.9921       411
weighted avg     0.9952    0.9951    0.9951       411
```

Figura 12 – Métricas de desempenho do modelo e matriz de confusão obtidas na etapa de validação.

Esses resultados indicam consistência do pipeline de preparação, treinamento e exportação de artefatos para o fluxo proposto. Entretanto, as métricas devem ser interpretadas com cautela no contexto deste trabalho, pois a configuração atual do conjunto de dados e do alvo de treinamento tem caráter simplificado para fins de validação técnica da integração treino–firmware.

Em particular, o rótulo de decisão utilizado no treinamento foi definido a partir de um critério baseado em chuva acumulada em 24 horas, e parte das variáveis empregadas no modelo é composta por valores *proxy* para compatibilizar o conjunto de dados histórico com a estrutura de entrada esperada pelo firmware. Assim, os resultados desta etapa não devem ser lidos como validação agronômica definitiva do desempenho da decisão de irrigação em campo, mas como evidência de viabilidade e funcionamento do pipeline externo de treinamento e geração de parâmetros embarcáveis.

5.2 Integração Treino–Firmware via `ia_params.h`

A integração entre treinamento e execução embarcada foi validada por meio da incorporação do arquivo `ia_params.h` ao projeto do firmware no ESP-IDF durante o

processo de build. Esse arquivo consolida os pesos do modelo, o termo de *bias*, as estatísticas de normalização e a série temporal de chuva em 24 horas embutida no firmware, utilizada pelo módulo de previsão/decisão (Figura 13).

```
/* IA params (gerado por treinar_modelo.py) */
#pragma once

#define IA_VERSION 20250818
// FEATURE_ORDER: soil_pct, temp_c, rh_pct, chuva_mm_24h

#define N_FEATS 4
static const float SCALER_MEAN[4]  = { 30.00000000f, 25.00000000f, 60.00000000f, 5.00000000f };
static const float SCALER_SCALE[4] = { 20.00000000f, 7.00000000f, 15.00000000f, 7.00000000f };
static const float W[4]            = { 0.00000000f, -0.36242818f, 0.00000000f, -8.11610546f };
static const float B[1]              = { 0.09847819f };

/* Série de previsão embutida (chuva acumulada nas próximas 24h)
   Índice aproximado em runtime:
     idx ≈ (now_epoch - FORECAST_T0_EPOCH) / (FORECAST_STEP_MIN * 60)
     idx clamp em [0, FORECAST_LEN-1]
*/
#define FORECAST_T0_EPOCH  1754697600
#define FORECAST_STEP_MIN  1440
#define FORECAST_LEN       7
static const float FORECAST_CHUVA_MM24[FORECAST_LEN] = { 0.10000000f, 0.00000000f, 0.00000000f, 2.60000000f, 0.20000000f, 0.00000000f, 0.40000000f };
```

Figura 13 – Trecho do arquivo `ia_params.h` contendo os parâmetros do modelo convertidos para linguagem C.

No fluxo de integração contínua, o arquivo `ia_params.h` é selecionado a partir do artefato de treinamento (quando o treino é executado no workflow) ou a partir da versão mantida no repositório, sendo então posicionado no diretório do módulo de IA do firmware antes da compilação. Esse encadeamento confirma a compatibilidade entre o pipeline externo de treinamento e o processo de build embarcado.

A compilação do firmware com os parâmetros integrados foi concluída com sucesso no ambiente de integração contínua (Figura 14), demonstrando a compatibilidade do artefato gerado com o fluxo de build do ESP-IDF.

```
2026-02-19 00:40:16Z: Running job: build-firmware
2026-02-19 00:41:31Z: Job build-firmware completed with result: Succeeded
```

Figura 14 – Etapa de compilação do firmware executada com sucesso no pipeline de integração contínua.

5.3 Resultados do Firmware: Aquisição de Sensores

No ambiente embarcado, foi verificado o funcionamento do subsistema de aquisição de dados ambientais por meio da inicialização correta dos sensores e da realização de leituras periódicas em tempo de execução. A etapa de inicialização confirma a configuração dos periféricos e o carregamento dos respectivos drivers, conforme evidenciado nos logs do monitor serial (Figura 15). Em seguida, durante a operação contínua do sistema, observam-se as leituras atualizadas de temperatura, umidade relativa do ar e umidade do solo, demonstrando a aquisição cíclica e consistente dos dados utilizados no processo de inferência (Figura 16).

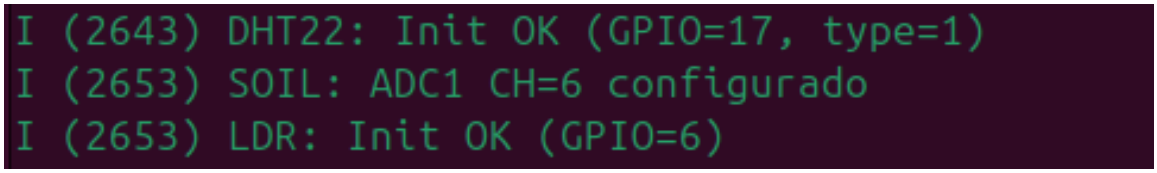

Figura 15 – Inicialização dos sensores de temperatura, umidade relativa do ar e umidade do solo.

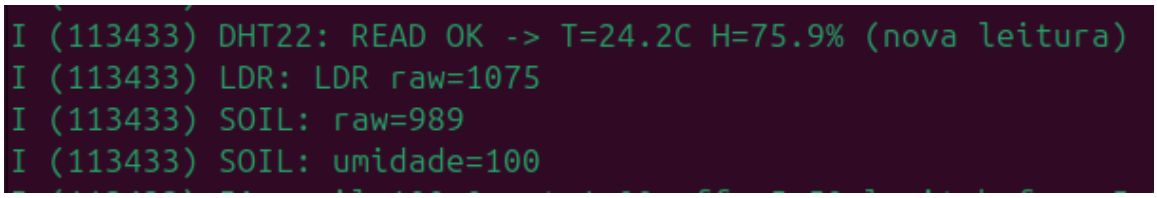

Figura 16 – Leituras periódicas de sensores de temperatura, umidade relativa do ar e umidade do solo.

5.4 Resultados do Firmware: Inferência Embarcada e Tomada de Decisão

A execução da inferência embarcada foi validada por meio do cálculo da probabilidade de irrigação em tempo real a partir das medições dos sensores e dos parâmetros exportados do modelo (Figura 17). Os logs evidenciam a etapa de inferência local no ESP32-S3, incluindo o processamento das variáveis de entrada e a obtenção da probabilidade utilizada pelo módulo decisor.

```
I (2713) DHT22: Init OK (GPIO=17, type=1)
I (2723) SOIL: ADC1 CH=6 configurado
W (2723) PUMP: DRY-RUN ATIVO: não configurando GPIO 12; apenas simulando.
I (2733) DECIDER: config: SOLO_CRIT=15.0 SOLO_OK=25.0 SOLO_REF=40.0 CHUVA_BLOQ=8.0 IA_TH=[0.45
..0.55] ADJ_MAX=0.20 MIN_ON=30000ms MIN_OFF=120000ms TICK=10000ms
I (2743) DECIDER: model: mean=[30.0 25.0 60.0 5.0] scale=[20.0 7.0 15.0 7.0] W=[0.00 -0.36 0.0
0 -8.12] B=0.10
I (2753) SOIL: raw=988
I (2753) SOIL: umidade=100
I (2773) DHT22: READ OK -> T=24.2C H=77.1% (nova leitura)
I (2773) SOIL: raw=999
I (2773) SOIL: umidade=100
I (2773) IA: soil=100.0 wet=1.00 off=-5.50 logit_before=5.82 logit_after=0.32
I (2783) IA: soil=100.0 wet=1.00 off=-5.50 logit_before=5.82 logit_after=0.32
I (2783) DECIDER: tick SOIL=100.0(xn=3.50) TEMP=24.2 RH=77.1 RAIN24=0.1 p=0.580 th=[0.65..0.75
] urg=0.00 vote=0 hold=NO -> OFF
I (2793) PUMP: [SIM] pump_set(OFF) -> GPIO 12 LOW
I (12803) SOIL: raw=1001
I (12803) SOIL: umidade=99
I (12823) DHT22: READ OK -> T=24.1C H=77.0% (nova leitura)
I (12823) SOIL: raw=1010
I (12823) SOIL: umidade=99
I (12823) IA: soil=99.0 wet=0.98 off=-5.41 logit_before=5.83 logit_after=0.42
I (12823) IA: soil=99.0 wet=0.98 off=-5.41 logit_before=5.83 logit_after=0.42
I (12833) DECIDER: tick SOIL=99.0(xn=3.45) TEMP=24.1 RH=77.0 RAIN24=0.1 p=0.603 th=[0.65..0.75
] urg=0.00 vote=0 hold=NO -> OFF
I (12843) PUMP: [SIM] pump_set(OFF) -> GPIO 12 LOW
I (22853) SOIL: raw=1001
I (22853) SOIL: umidade=99
I (22873) DHT22: READ OK -> T=24.2C H=77.0% (nova leitura)
I (22873) SOIL: raw=1007
I (22873) SOIL: umidade=99
I (22873) IA: soil=99.0 wet=0.98 off=-5.41 logit_before=5.82 logit_after=0.41
I (22873) IA: soil=99.0 wet=0.98 off=-5.41 logit_before=5.82 logit_after=0.41
I (22883) DECIDER: tick SOIL=99.0(xn=3.45) TEMP=24.2 RH=77.0 RAIN24=0.1 p=0.602 th=[0.65..0.75
] urg=0.00 vote=0 hold=NO -> OFF
I (22893) PUMP: [SIM] pump_set(OFF) -> GPIO 12 LOW
I (27783) wifi:<ba-add>idx:1 (ifx:0, 62:10:9e:28:02:54), tid:6, ssn:2, winSize:64
```

Figura 17 – Execução da inferência embarcada com cálculo da probabilidade de irrigação.

A decisão final do sistema é realizada pelo módulo decisor, que combina a saída probabilística do módulo de IA com mecanismos de estabilidade e regras determinísticas de segurança. Na implementação avaliada, destacam-se a aplicação de histerese, o bloqueio por chuva, as restrições de acionamento por condições operacionais e os tempos mínimos de acionamento/desligamento (anti-chattering).

Observou-se que, mesmo com novas leituras dos sensores, o estado do atuador permanece estável quando a probabilidade calculada permanece em faixa não suficiente para comutação, evitando chaveamentos rápidos e sucessivos do relé (Figura 18). Essa estabilidade decorre da combinação entre histerese e regras temporais do módulo decisor, e não apenas da probabilidade instantânea estimada pelo modelo.

```
I (12833) DECIDER: tick SOIL=99.0(xn=3.45) TEMP=24.1 RH=77.0 RAIN24=0.1 p=0.603 th=[0.65..0.75] urg=0.00 vote=0 hold=NO -> OFF
I (12843) PUMP: [SIM] pump_set(OFF) -> GPIO 12 LOW
```

Figura 18 – Saída do módulo decisor evidenciando estabilidade do estado do sistema por meio de histerese e regras de temporização.

5.5 Resultados do Atuador: Acionamento em Modo *Dry-run*

O acionamento do atuador foi validado em modo de simulação (*dry-run*), no qual os comandos de liga/desliga são registrados em log sem a ativação física da bomba, permitindo verificar o encadeamento completo entre leitura dos sensores, inferência embarcada, decisão e envio do comando ao módulo de acionamento (Figura 19).

```
W (2723) PUMP: DRY-RUN ATIVO: não configurando GPIO 12; apenas simulando.
```

Figura 19 – Simulação do acionamento da bomba a partir da decisão do sistema em modo *dry-run*.

Esse resultado confirma o funcionamento da integração entre o módulo decisor e o módulo da bomba, preservando segurança durante os testes ao evitar acionamento hidráulico real no ambiente de validação.

Durante a execução do firmware, observou-se também o processo de atualização remota via OTA, incluindo a transferência de uma nova imagem de firmware (`.bin`) e a reinicialização automática do dispositivo ao final do processo (Figura 20). Esse comportamento é consistente com a arquitetura adotada, na qual atualizações de parâmetros do modelo são incorporadas ao firmware em tempo de compilação e distribuídas ao dispositivo por atualização OTA do firmware completo.

```
I (173763) PUMP: [SIM] pump_set(OFF) -> GPIO 12 LOW
I (176573) ota_server: HDR need(Authorization)=71, token_exp_len=64
I (176573) ota_server: HDR get rc=0, got_len=71, first='B'
I (176573) ota_server: cmp recv_len=64 vs exp_len=64
I (176583) ota_server: Token OK
I (176583) ota_server: OTA start: content_len=853424
I (179023) ota_server: OTA progress: 131328 / 853424 bytes
I (179553) ota_server: OTA progress: 264896 / 853424 bytes
I (180043) ota_server: OTA progress: 396032 / 853424 bytes
I (180523) ota_server: OTA progress: 525504 / 853424 bytes
I (181033) ota_server: OTA progress: 657632 / 853424 bytes
I (181523) ota_server: OTA progress: 787328 / 853424 bytes
I (181783) esp_image: segment 0: paddr=00110020 vaddr=3c0a0020 size=26a18h (158232) map
I (181803) esp_image: segment 1: paddr=00136a40 vaddr=3fc97e00 size=04730h ( 18224)
I (181803) esp_image: segment 2: paddr=0013b178 vaddr=40374000 size=04ea0h ( 20128)
I (181813) esp_image: segment 3: paddr=00140020 vaddr=42000020 size=91670h (595568) map
I (181893) esp_image: segment 4: paddr=001d1698 vaddr=40378ea0 size=0eee8h ( 61160)
I (181903) esp_image: segment 5: paddr=001e0588 vaddr=600fe010 size=00004h (     4)
I (181903) esp_image: segment 0: paddr=00110020 vaddr=3c0a0020 size=26a18h (158232) map
I (181933) esp_image: segment 1: paddr=00136a40 vaddr=3fc97e00 size=04730h ( 18224)
I (181933) esp_image: segment 2: paddr=0013b178 vaddr=40374000 size=04ea0h ( 20128)
I (181943) esp_image: segment 3: paddr=00140020 vaddr=42000020 size=91670h (595568) map
I (182023) esp_image: segment 4: paddr=001d1698 vaddr=40378ea0 size=0eee8h ( 61160)
I (182033) esp_image: segment 5: paddr=001e0588 vaddr=600fe010 size=00004h (     4)
I (182073) ota_server: OTA done, rebooting...
I (182373) wifi:state: run -> init (0)
I (182373) wifi:pm stop, total sleep time: 0 us / 181766312 us
```

Figura 20 – Processo de atualização remota do firmware via OTA com reinicialização automática do dispositivo.

6 Trabalhos Futuros

Este trabalho atingiu seu objetivo principal ao demonstrar a viabilidade técnica de uma arquitetura híbrida para irrigação inteligente, combinando: (i) treinamento externo de um modelo leve de aprendizado de máquina; (ii) exportação determinística de parâmetros e estatísticas para um artefato em linguagem C (`ia_params.h`); e (iii) execução de inferência embarcada no ESP32-S3 em um fluxo cíclico, previsível e independente de conectividade. A solução implementada validou, de forma integrada, a leitura de sensores, a normalização consistente entre treino e inferência, o cálculo probabilístico via Regressão Logística, o mecanismo de histerese e as regras determinísticas de segurança (como bloqueio por chuva com base na série embutida no firmware e *anti-chattering*), bem como o encadeamento completo até o atuador em modo de simulação (*dry-run*).

Dessa forma, os trabalhos futuros apresentados a seguir representam evoluções naturais da pesquisa e do protótipo desenvolvido, ampliando gradualmente o nível de validação e a robustez do sistema sem descaracterizar o escopo original (prova de viabilidade técnica) nem tratá-lo como correção de falhas. As extensões propostas partem diretamente das decisões de projeto e das limitações metodológicas assumidas, tais como a execução do atuador em *dry-run*, a ausência de validação agronômica em campo, o uso de sensores de baixo custo, a série de chuva em 24 horas estática embutida no firmware e a adoção de um modelo linear simples.

Uma primeira linha de continuidade consiste na realização de testes em campo com a integração do sistema a um conjunto hidráulico real (bomba, tubulação e emissores), permitindo observar o comportamento do controle sob variações ambientais reais e por períodos prolongados. Essa etapa possibilita avaliar, de maneira mais completa, aspectos operacionais não capturados em cenários controlados, como estabilidade do acionamento em condições de ruído, variações de alimentação, degradação gradual de sensores e interferências do ambiente. Além disso, abre espaço para uma validação agronômica da decisão de irrigação, comparando o impacto do acionamento automatizado sobre variáveis como umidade do solo ao longo do tempo, consumo de água e indicadores de estresse hídrico da cultura. Essa avaliação, embora extrapole o escopo inicial, é uma progressão direta do sistema já funcional, e permite quantificar benefícios práticos a partir da arquitetura comprovada.

Outra extensão relevante refere-se ao enriquecimento do sensoriamento. Embora o protótipo utilize sensores de baixo custo adequados para validar o ciclo de aquisição e decisão, a inclusão de novos sensores pode aumentar a representatividade do vetor de características e apoiar decisões mais robustas. Nesse sentido, sensores como temperatura

do solo, condutividade elétrica, pH, e até sensores de radiação solar ou pluviometria local poderiam complementar o estado ambiental capturado. Adicionalmente, técnicas de calibração periódica, compensação de deriva e validação cruzada entre sensores podem elevar a confiabilidade da leitura, preservando o desenho modular do firmware e permitindo evoluções incrementais sem acoplamento excessivo entre subsistemas.

No componente de informação de chuva, o sistema adotou uma estratégia embutida no firmware, que favorece determinismo e autonomia ao custo de uma série estática após a compilação. Como evolução, pode-se investigar mecanismos de atualização dinâmica da previsão quando conectividade estiver disponível, consumindo dados meteorológicos em intervalos definidos e armazenando-os localmente, ou ainda combinando previsões externas com medições locais para ajuste conservador do bloqueio de irrigação. Alternativamente, pode-se explorar atualizações via OTA em janelas programadas, de modo a manter a previsibilidade do firmware e, ao mesmo tempo, reduzir a defasagem da série embutida. Essa evolução preserva o princípio arquitetural já adotado: a separação entre aprendizagem/parametrização externa e execução determinística embarcada.

No contexto do modelo preditivo, a Regressão Logística mostrou-se apropriada para demonstrar a execução embarcada com baixo custo computacional e boa interpretabilidade. Como trabalhos futuros, propõe-se avaliar treinamento contínuo e re-treinamentos periódicos incorporando dados reais coletados pelo dispositivo em operação, ampliando a aderência do modelo às particularidades microclimáticas e do solo do local monitorado. Essa abordagem pode ser estruturada como um ciclo de MLOps mais completo, incluindo controle de versões de dados, validação automatizada, seleção de modelos e implantação de novos parâmetros (com ou sem atualização completa do firmware). Em paralelo, pode-se investigar modelos mais expressivos, como árvores de decisão compactadas, ensembles leves ou redes neurais pequenas, quantificando o impacto em memória, latência e consumo energético, e mantendo como critério central a viabilidade em microcontroladores e a previsibilidade de execução.

Do ponto de vista de integração e observabilidade, a arquitetura atual pode ser estendida com uma camada de IoT e nuvem para registro histórico de medições, auditoria do processo decisório e suporte a operação remota. A transmissão periódica de telemetria (sensores, probabilidade estimada, estado do atuador e eventos de bloqueio) permitiria a construção de dashboards de monitoramento, contribuindo para análise de desempenho, diagnóstico e ajustes de parâmetros ao longo do tempo. Essa camada deve ser concebida como complementar ao controle local, de forma que a tomada de decisão permaneça funcional mesmo sem conectividade, preservando o caráter de Edge AI do sistema.

Por fim, a evolução do protótipo para cenários reais também exige atenção à eficiência energética. Embora o firmware tenha sido implementado de modo determinístico, a operação em campo pode demandar otimizações adicionais, como uso de modos de

baixo consumo, ajustes no intervalo de amostragem conforme condições ambientais e estratégias de *duty cycling* para sensores e rádio. Essas medidas podem tornar o sistema mais adequado a alimentação por bateria ou energia solar, ampliando sua aplicabilidade em propriedades com infraestrutura limitada.

Em síntese, as extensões propostas avançam em direção a maior robustez, maior fidelidade ambiental e maior capacidade de adaptação, preservando a contribuição central já comprovada: a integração prática entre um pipeline externo de treinamento e uma execução embarcada determinística de inferência no ESP32-S3, com mecanismos de estabilidade e segurança adequados ao contexto de irrigação inteligente.

7 Conclusão

Este trabalho apresentou o desenvolvimento de um sistema inteligente de irrigação baseado em uma arquitetura híbrida, combinando um pipeline externo de treinamento de modelo preditivo com inferência embarcada no microcontrolador ESP32-S3. O objetivo principal foi demonstrar a viabilidade técnica da integração entre sensoriamento ambiental de baixo custo, técnicas de aprendizado de máquina e execução local do processo decisório em um dispositivo com recursos computacionais limitados.

A solução proposta foi implementada de forma modular, contemplando a aquisição de dados por meio dos sensores DHT22, LDR e sensor capacitivo de umidade do solo, o processamento das variáveis ambientais, a incorporação de uma série de chuva em 24 horas embutida no firmware (gerada externamente no pipeline) e a execução de um modelo de Regressão Logística embarcado. O processo de decisão foi complementado por regras determinísticas de segurança, incluindo histerese, bloqueio por chuva e tempos mínimos de acionamento e desligamento, contribuindo para a estabilidade operacional e para a redução de comutações indevidas do atuador.

Os testes realizados em ambiente controlado permitiram validar o funcionamento integrado do sistema, evidenciando a correta leitura dos sensores, a execução da inferência embarcada, a aplicação das regras de decisão e o acionamento do módulo de irrigação em modo de simulação (*dry-run*). Esses resultados demonstram que a abordagem adotada é tecnicamente viável e compatível com as restrições típicas de dispositivos embarcados de baixo custo.

Do ponto de vista arquitetural, destaca-se como principal contribuição a integração consistente entre o pipeline externo de treinamento e o firmware embarcado por meio da exportação determinística dos parâmetros do modelo para um arquivo em linguagem C (`ia_params.h`). Essa estratégia garante reprodutibilidade entre as etapas de treinamento e inferência, elimina dependências de conectividade durante a operação e permite a atualização controlada do comportamento do sistema por meio de novos ciclos de treinamento seguidos de recompilação e atualização OTA do firmware.

Entretanto, o trabalho apresenta limitações. Os testes foram realizados em períodos curtos e em ambiente controlado, não contemplando avaliações de longo prazo em campo nem medições diretas de economia hídrica ou impacto agronômico. Além disso, a informação de chuva utilizada no firmware é estática após a compilação, exigindo a regeneração do artefato `ia_params.h` e nova compilação do firmware para incorporar dados mais recentes. Também se ressalta que a etapa de treinamento foi empregada com finalidade de validação técnica da integração treino–firmware, utilizando configuração simplificada de dados e

variáveis de entrada.

Mesmo com essas restrições, os resultados obtidos indicam que a combinação de Edge AI, pipeline inspirado em MLOps e sistemas embarcados constitui uma alternativa promissora para aplicações em agricultura de precisão, especialmente em cenários com limitações de conectividade e infraestrutura computacional.

Conclui-se, portanto, que a arquitetura proposta é tecnicamente viável, modular e reprodutível, contribuindo como uma abordagem de baixo custo para o desenvolvimento de sistemas inteligentes de irrigação com inferência embarcada.

Referências

BARBOZA, V. G. R. L.; KNIESS, J. Alocação de tarefas com simulated annealing na borda da rede para internet industrial. In: *Anais do Seminário Integrado de Software e Hardware (SEMISH)*. [S.l.: s.n.], 2024. p. 264–275. Citado na página 15.

BATISTA, A. V. de A. *Robô irrigador multifuncional de baixo custo para agricultura familiar (RIRRIG)*. Fortaleza, CE: [s.n.], 2016. Disponível em: <https://repositorio.ufc.br/handle/riufc/18757>. Citado na página 14.

COSTA, D.; MARTINS, P.; ALVES, F. Application of mlops in precision irrigation systems. *Journal of Precision Agriculture*, v. 16, n. 2, p. 123–134, 2021. Citado 2 vezes nas páginas 14 e 15.

DEY, A.; ROY, S.; CHAKRABORTY, S. Embedded systems for intelligent irrigation automation using esp32. *International Journal of Embedded Systems*, v. 18, n. 1, p. 45–53, 2019. Citado 3 vezes nas páginas 9, 12 e 15.

DIB, A. E. H. *Projeto de irrigação de jardim utilizando Internet das Coisas*. São Paulo, SP: [s.n.], 2019. Disponível em: <https://bdta.abcd.usp.br/directbitstream/83b99ccd-efb3-443d-ada0-36da436cd39b/ANA%20EMILIA%20HERNANDES%20DIB%20TCC20.pdf>. Citado 2 vezes nas páginas 13 e 14.

Equipe Embarcados. *Webinar: Inovação em AIoT — soluções escaláveis para agricultura, pecuária e campus inteligentes*. 2023. Disponível em: <https://embarcados.com.br/webinar-inovacao-em-aiot-solucoes-escalaveis-para-agricultura-pecuaria-e-campus-inteligentes/>. Citado na página 13.

Espressif Systems. *ESP32-S3 Technical Reference Manual*. [S.l.], 2023. Documentação oficial do ESP32-S3. Disponível em: <https://docs.espressif.com/>. Citado na página 9.

JESUS, C. D. de. *Automação de sistema de irrigação de baixo custo baseado em dados climáticos*. São Paulo, SP: [s.n.], 2019. Disponível em: <https://sistemas.ifgoiano.edu.br/sgcursos/uploads/anexos_14/2021-08-19-03-35-59Disserta%C3%A7%C3%A3o_Camila%20Dias%20de%20Jesus_Reposit%C3%B3rio.pdf>. Citado na página 14.

KREUZBERGER, D.; KÜHL, N.; HIRSCHL, S. Machine learning operations (mlops): Overview, definition, and architecture. *IEEE Access*, IEEE, v. 11, p. 12035–12054, 2023. Citado 4 vezes nas páginas 10, 14, 15 e 21.

KUMAR, R.; SINGH, P.; GUPTA, A. Intelligent irrigation system using machine learning algorithms. *Journal of Agricultural Technology*, v. 12, n. 4, p. 98–107, 2020. Citado 3 vezes nas páginas 12, 13 e 14.

MARIN, F. R.; SENTELHAS, P. C. Avanços e desafios da irrigação na agricultura brasileira. *Revista Brasileira de Engenharia Agrícola e Ambiental*, v. 24, n. 10, p. 707–715, 2020. Referência adaptada para exemplo de irrigação no Brasil. Citado na página 9.

MARINHO, K. G. de O. F. *Desenvolvimento de sistema de monitoramento e controle de irrigação de agricultura familiar com ATMEGA*. Dissertação

(Mestrado) — Universidade Federal de Pernambuco, Recife, PE, 2019. Disponível em: <https://repositorio.ufpe.br/handle/123456789/47118>. Citado 2 vezes nas páginas 12 e 14.

Mouser Electronics. *Feito para o campo: conectores para agricultura*. 2022. Disponível em: <https://embarcados.com.br/feito-para-o-campo-conectores-para-agricultura/>. Citado na página 13.

Mouser Electronics. *Criadas para resistir: soluções de conectividade robusta para a agricultura autônoma*. 2023. Disponível em: <https://embarcados.com.br/criadas-para-resistir-solucoes-de-conectividade-robusta-para-a-agricultura-autonoma/>. Citado na página 13.

Open-Meteo. *Open-Meteo Weather API*. 2024. <https://open-meteo.com/>. Acessado em: 2024. Citado na página 20.

PAHLAVAN, K.; ZHANG, Z.; LI, J. Water consumption prediction using machine learning for sustainable agriculture. *Agricultural Water Management Journal*, v. 27, n. 1, p. 67–75, 2020. Citado 2 vezes nas páginas 13 e 14.

SANTOS, C. A. P. dos et al. Leitora rfid automática com multiconectividade para pecuária de precisão. In: *Anais do Seminário Integrado de Software e Hardware (SEMISH)*. [S.l.: s.n.], 2023. Citado na página 15.

SILVA, A. T. L.; CARDOSO, G. S.; GOMES, A. S. N. Enabling water management system for agriculture using a low cost approach. In: *Proceedings of the 2024 Brazilian Symposium on Computing Systems Engineering (SBESC)*. [S.l.]: IEEE, 2024. p. 109–114. Citado na página 15.

ZHANG, W.; LI, M.; CHEN, X. Edge ai and MLOps: From cloud-centric to device-centric machine learning. *IEEE Internet of Things Journal*, v. 9, n. 15, p. 13500–13515, 2022. Referência ilustrativa para Edge AI e MLOps em dispositivos. Citado 4 vezes nas páginas 9, 10, 14 e 15.

ZHANG, Y.; LIU, J.; WANG, L. Iot-based automated irrigation system for semi-arid regions using machine learning models. *International Journal of Agricultural Automation*, v. 14, n. 3, p. 205–214, 2018. Citado 2 vezes nas páginas 12 e 14.