UNIVERSIDADE FEDERAL DE ITAJUBÁ - UNIFEI

Instituto de Engenharia de Sistemas e Tecnologia da Informação - IESTI

MACHINE LEARNING SYSTEMS ENGINEERING - IESTI05

PlantTalker: Plataforma IoT com assistente conversacional para monitoramento de plantas

Gabriel Batista de Oliveira Vilas Boas – 2024000467

Gabriel Del Monte Schiavi Noda – 2022014552

Gabrielle Gomes Almeida – 2022002758

Professor: Marcelo José Rovai

ITAJUBÁ

07/12/2025

1 Introdução e motivação

Sistemas embarcados (ou sistemas de borda) são sistemas eletrônicos microprocessados que, após serem programados, possuem uma função específica que, geralmente, não pode ser alterada. Eles estão presentes em todos os ambientes, cobrindo uma ampla gama de funcionalidades: controles industriais, sistemas de gerenciamento de aviação, antenas retransmissoras, etc. Outra característica da maioria dos sistemas embarcados é uma restrição de recursos tanto computacionais (memória e processamento) quanto físicos (número de terminais e interfaces de exibição/entrada de dados).

A inteligência artificial de borda (*Edge AI*) é uma derivação do aprendizado de máquina que estuda e implementa modelos de inteligência artificial (IA) nesses dispositivos. Essa técnica se difere dos modelos de IA convencionais, pois foca na otimização dos modelos para que possam operar com uma quantidade mínima de capacidade de processamento, e sem a necessidade de comunicação com a nuvem, realizando toda a operação no *hardware* local.

O problema investigado neste projeto envolve o monitoramento contínuo de plantas em pequenos ambientes domésticos, de laboratório ou estufas experimentais. Plantas dependem de condições ambientais adequadas, como umidade do solo, umidade do ar e temperatura, para manter um bom estado fisiológico. A ausência de monitoramento e intervenção adequada resulta em estresse hídrico, baixo crescimento ou mesmo morte da planta. Em ambientes maiores, esse controle já é realizado com sistemas complexos; porém, em contextos menores ou domésticos, a automação costuma ser limitada ou inexistente.

O contexto de aplicação real deste projeto está no desenvolvimento de um sistema acessível e inteligente capaz de acompanhar, interpretar e comunicar dados ambientais relevantes para o cuidado de uma planta. Para isso, utiliza-se uma arquitetura embarcada híbrida composta por uma Raspberry Pi, responsável pelos sensores de umidade do ar e temperatura, e um ESP32 dedicado à leitura da umidade do solo, com comunicação UART entre os dispositivos. Além disso, o sistema conta com uma interface física (LEDs e botão) para feedback imediato ao usuário, além de uma interface web baseada em modelos de linguagem (LLMs), permitindo ao usuário "conversar com a planta", receber recomendações diretas e explorar dashboards com o histórico dos dados coletados.

O objetivo geral deste projeto é desenvolver um sistema embarcado inteligente capaz de monitorar variáveis ambientais e auxiliar o usuário no cuidado da planta por meio da coleta contínua de dados, análise local e interação natural com um modelo de linguagem. Para isso, o sistema utiliza uma arquitetura híbrida composta por Raspberry Pi e ESP32 para medir umidade do solo, umidade do ar e temperatura, processando essas informações diretamente no dispositivo. Além de interpretar condições como déficit hídrico e temperatura elevada, o sistema oferece feedback imediato por LEDs, permite a irrigação automática mediante acionamento de um botão físico e disponibiliza uma interface gráfica com dashboards e um chat baseado em LLM capaz de responder perguntas e fornecer recomendações contextualizadas, funcionando como um assistente virtual para o monitoramento da

planta.

Os critérios de sucesso do projeto incluem a leitura confiável dos sensores, a comunicação correta entre os dispositivos, a tomada de decisão adequada quanto ao estado da planta e a operação da interface gráfica e do modelo de IA local. Apesar de ser voltado ao cuidado de uma única planta, o sistema possui uma arquitetura modular que pode ser expandida para aplicações de maior escala, como estufas e lavouras inteligentes, demonstrando o potencial da combinação entre sistemas embarcados e técnicas de *Edge AI* para soluções agrícolas autônomas e de baixo custo.

2 Revisão da literatura

Com o recente aumento no uso da inteligência artificial, vários estudos estão sendo feitos no ramo da IA de borda para os mais diversos objetivos. Até a data deste projeto, estão registrados no Google Acadêmico cerca de 432 trabalhos relacionados ao monitoramento de plantas utilizando aprendizado de máquina embarcado publicados em 2025.

O crescimento da IA na borda atraiu a atenção de inúmeros pesquisadores, professores e estudantes para o cenário. Dentre os esforços para avançar os conhecimentos nessa área, surgiram organizações como a *Edge AI Foundation*, responsável por unir estudiosos e criar uma comunidade que compartilha informações sobre esse ramo do aprendizado de máquina. Também sugiram iniciativas como a *TinyML4D*, um projeto colaborativo que democratiza a educação em aprendizado de máquina embarcado, focado em países em desenvolvimento.

Enquanto os chamados grandes modelos de linguagem, ou *large language models* (LLMs), evoluem para modelos com bilhões de parâmetros, centenas de gigabytes de armazenamento necessários, e precisam de *data centers* cada vez mais poderosos para serem executados, os modelos de IA de borda revolucionam por se manter relativamente leves enquanto garantem o equilíbrio entre precisão e eficiência computacional. Optar pelo uso dessa tecnologia em projetos traz vários benefícios relacionados à sua capacidade de processamento local:

- Velocidade: a independência do envio de dados para a nuvem garante agilidade em operações;
- Privacidade: ao manter todos os dados em um armazenamento local, é garantido que apenas o usuário tenha acesso ao que for importante;
- Segurança: eliminar a comunicação com a nuvem evita que as informações fiquem expostas a vazamentos de dados e ciberataques como *sniffing, man in the middle, DoS,* etc;
- Robustez: um sistema local pode tomar decisões por conta própria e manter seu funcionamento em ambientes limitados, como falta de conexão com a internet e ausência de energia (caso possua uma bateria).

A construção de SLMs é possível graças aos novos *frameworks* do mercado voltados especificamente para essa área, como o TensorFlow Lite, PyTorch, TensorRT, Edge Impulse, entre outros. Além dos *frameworks*, boas técnicas de otimização são necessárias para o projeto de um software robusto, confiável e eficiente. Algumas das técnicas mais relevantes para a otimização desses sistemas são: *function calling, prompt engineering, retrieval-augmented generation* (RAG) e *fine-tuning*.

2.1 Function calling

A chamada de funções é uma técnica que permite que SLMs executem mais tarefas além de gerar texto. Ao utilizar recursos como interfaces de programação de aplicações (APIs) ou funções

externar, os modelos são capazes de se comunicar com periféricos externos ao *hardware* onde foram implementados, possibilitando o monitoramento de dados em tempo real, realização de cálculos mais complexos, integração com outros sistemas, etc.

Essa possibilidade de comunicação com componentes externos faz com que o dispositivo se torne mais flexível e capaz de executar tarefas de forma mais fluida.

2.2 Prompt engineering

É uma técnica que consiste em detalhar de forma cuidadosa os *prompts* que serão recebidos pelo modelo, a fim de garantir maior precisão na resposta. Informar ao modelo, por exemplo, em que tipo de tarefa ele deve ser especializado, pode melhorar de forma considerável a qualidade das respostas. Apesar de simples, é uma técnica poderosa que pode ser usada também para modelos e *chatbots* do dia-a-dia para reduzir a margem de erro.

2.3 RAGs

São arquiteturas que combinam duas etapas: um mecanismo de busca em uma base de conhecimento externa, trechos de texto ou documentos relevantes ao prompt fornecido, e um modelo de geração de linguagem que utiliza tanto o prompt original quanto os trechos recuperados como contexto para produzir a resposta final.

Para aplicações de borda, RAGs trazem grandes vantagens. Eles podem reduzir o tamanho do modelo, pois parte do ”pensamento”é responsabilidade de um módulo de recuperação de informações, que costuma ser mais leve e rápido. Essa divisão resulta em menor consumo energético e latência. A utilização de RAGs também garante que o modelo forneça informações mais atualizadas, bastando adicionar novos documentos contendo essas informações à sua base de dados.

2.4 Fine-tuning

É o processo de adaptar um modelo de linguagem já treinado a para realizar uma tarefa específica, re-treinando com um conjunto de dados menor e para a tarefa requisitada. Esse ajuste permite que o modelo mantenha seu conhecimento geral adquirido durante seu projeto, ao mesmo tempo em que se adapta ao novo contexto.

O *fine-tuning* oferece benefícios consideráveis para modelos de borda. Ele possibilita a utilização de modelos menores que já possuem o conhecimento necessário, reduzindo os requisitos de armazenamento e processamento. Essa técnica pode ser realizada diretamente no dispositivo, mantendo os dados sensíveis sob controle do usuário e garantindo a privacidade.

3 Projeto do sistema

3.1 Arquitetura do sistema

O sistema desenvolvido utiliza uma arquitetura embarcada híbrida composta por dois módulos principais: uma Raspberry Pi, responsável pelo processamento local, leitura dos sensores de umidade do ar e temperatura, controle dos LEDs, acionamento do servo e execução do modelo de linguagem; e um ESP32, dedicado exclusivamente à leitura do sensor de umidade do solo. A comunicação entre os dispositivos ocorre por meio de uma interface UART com protocolo simples baseado em envio periódico de mensagens contendo o valor de umidade do solo.

Assim como mostra a Figura 2, a Raspberry Pi desempenha o papel de unidade central de controle: recebe dados do DHT22, interpreta os valores transmitidos pelo ESP32, classifica o estado hídrico da planta, aciona LEDs para feedback visual instantâneo e executa a rotina de irrigação automática por meio de um microservo, a qual será apenas simulada nessa parte do trabalho. Paralelamente, ela alimenta a interface gráfica e o chat baseado em LLM, permitindo que a interação com o usuário ocorra localmente, sem dependência de nuvem.

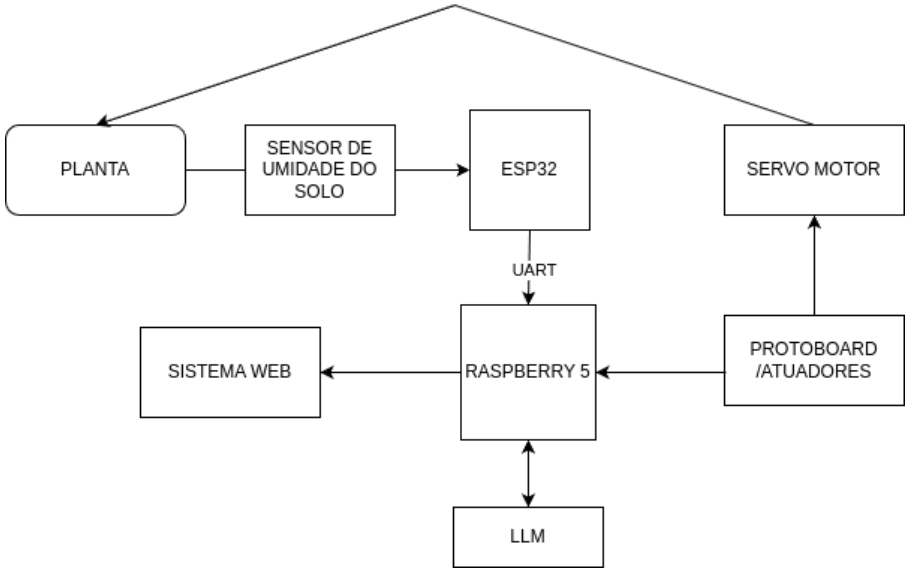

Figura 1: Arquitetura geral do sistema

3.2 Componentes utilizados e conexões

A Tabela abaixo descreve todos os componentes utilizados e seus respectivos papéis no sistema:

Componente	Função	Local	GPIO / Interface
Raspberry Pi 5	Unidade central de processamento, execução do modelo local e controle físico	—	—
ESP32	Leitura da umidade do solo e envio periódico via UART	—	TX/RX conectados à Raspberry Pi
Sensor DHT22	Medição de temperatura e umidade do ar	Raspberry Pi	GPIO 16 (D16)
Sensor de umidade do solo	Medição do conteúdo de água no solo	ESP32	Entrada analógica (ADC)
3 LEDs (Vermelho, Amarelo, Verde)	Indicação visual do estado hídrico da planta	Raspberry Pi	GPIO 13, 19, 26
Botão	Ação manual para iniciar a irrigação	Raspberry Pi	GPIO 20
Microservo (SG90 ou similar)	Abertura da válvula/seringa para irrigação	Raspberry Pi	GPIO 12 (PWM)
Fonte de alimentação 5V	Alimentação dos dispositivos	Ambos	—
Resistores e jumpers	Circuito de proteção e conexão	Ambos	—

Tabela 1: Componentes de hardware e respectivas conexões

Abaixo está listado as conexões de cada componente:

- **Comunicação UART (ESP32 → Raspberry Pi)**
  - ESP32 (TX) → Raspberry Pi (RX / `ttyAMA0`)
  - ESP32 (GND) → Raspberry Pi (GND)
  - Taxa de comunicação: 115200 bps

- **Conexões da Raspberry Pi**
  - **DHT22**
    - DATA → GPIO 16
    - VCC → 3.3V
    - GND → GND
  - **LEDs (via resistores de $\sim 220\,\Omega$)**
    - Vermelho → GPIO 13
    - Amarelo → GPIO 19
    - Verde → GPIO 26
  - **Botão**
    - Um terminal → GPIO 20
    - Outro terminal → GND
  - **Microservo**
    - Sinal (laranja) → GPIO 12
    - VCC (vermelho) → 5V
    - GND (marrom) → GND
- **Conexões do ESP32**
  - **Sensor de umidade do solo**
    - A0 → GPIO 36
    - VCC → 3.3V
    - GND → GND

3.3 Arquitetura de software e fluxo de dados

A organização do software acompanha diretamente a divisão física do sistema. Na Raspberry Pi, o diretório `src/code` reúne tanto o servidor de aplicação, desenvolvido em Python, quanto a interface web construída em React. O servidor, composto pelo arquivo `api_server.py` e pelos módulos internos da pasta `iot/`, é responsável pela comunicação com os sensores, controle dos atuadores e disponibilização dos dados para o frontend. O ESP32, por sua vez, executa um firmware simples que realiza leituras periódicas do sensor de umidade do solo e envia o valor pela UART em formato textual, como em: “Moisture = 45%”.

Na Raspberry Pi, o módulo `UARTHandler` opera em uma thread separada, monitorando continuamente a porta serial. Cada vez que uma nova leitura chega do ESP32, o valor numérico é extraído, verificado e armazenado na estrutura global `SystemState`. Um processo semelhante ocorre com o sensor DHT22: o módulo `DHTSensor` realiza leituras periódicas em outra thread, atualizando temperatura e umidade do ar sem interferir na execução do restante do sistema.

O servidor Flask oferece uma API REST para consultas pontuais do estado da planta (rota `/api/status`) e para o acionamento manual da irrigação (rota `/api/irrigate`). Além disso, o uso do Flask-SocketIO permite manter um canal WebSocket ativo com o navegador, enviando atualizações frequentes com o conteúdo consolidado em `SystemState`. No frontend, o componente principal `App.jsx` recebe esses dados em tempo quase real e os repassa para componentes específicos, como o painel de monitoramento e os gráficos.

De forma resumida, o fluxo de dados do sistema segue quatro etapas principais:

1. coleta das medições pelos sensores (DHT22 na Raspberry Pi e sensor analógico no ESP32);
2. envio e armazenamento das leituras em `SystemState`;
3. disponibilização dessas informações por meio da API REST e do WebSocket;
4. exibição e interação com o usuário na interface web, incluindo a comunicação com o modelo de linguagem local.

3.4 Seleção e justificativa do modelo SLM

Para compor a camada de IA embarcada, optou-se pelo uso de um pequeno modelo de linguagem local (SLM) executado via Ollama na Raspberry Pi 5. O modelo escolhido, `llama3.2:1b`, possui aproximadamente um bilhão de parâmetros, atendendo ao equilíbrio necessário entre qualidade das respostas e o consumo de recursos de processamento e memória.

Modelos maiores, comuns em ambientes de nuvem, demandariam capacidades de hardware e energia incompatíveis com o objetivo de manter o sistema acessível e local. Por outro lado, modelos significativamente menores tendem a apresentar limitações de contextualização e raciocínio, o que comprometeria a utilidade do assistente ao oferecer orientações sobre o estado da planta. Nesse sentido, o `llama3.2:1b` mostrou-se adequado ao entregar desempenho suficiente para operar na Raspberry Pi sem comprometer a qualidade das interações.

A comunicação com o modelo é feita pela classe `LLMInterface`, que monta o *prompt* enviado ao Ollama. Sempre que o usuário envia uma pergunta, o sistema acrescenta ao *prompt* dados atualizados da planta como temperatura, umidade do ar, umidade do solo e mensagens de status. Essa estratégia, inspirada em RAG, permite que o modelo gere respostas alinhadas ao estado real dos sensores, e não apenas ao texto digitado pelo usuário.

4 Implementação

O desenvolvimento do software foi organizado em duas partes principais: o backend, que executa na Raspberry Pi e concentra toda a lógica de controle e comunicação com o hardware; e o frontend, responsável pela interface acessada pelo usuário via navegador. As duas camadas conversam entre si por meio de uma API REST e de uma conexão WebSocket para o envio contínuo de atualizações.

4.1 Estrutura do Código

A estrutura do projeto foi pensada para manter clareza e facilitar manutenção. Os diretórios principais são:

- `src/code/api_server.py`: Arquivo que inicializa o servidor Flask e os módulos principais.
- `src/code/iot/`: Implementação da lógica que interage diretamente com sensores e atuadores.
- `src/code/ui/`: Código da interface web desenvolvida em React.

4.2 Backend e API (Flask)

No backend, o arquivo `api_server.py` centraliza o servidor Flask e a configuração do Flask-SocketIO. Ao iniciar, ele instancia a classe `PlantTalkerAPI`, que por sua vez cria e gerencia os módulos responsáveis pelo hardware e pela lógica do sistema.

Como o sistema depende de atualizações frequentes, uma *thread* separada faz o envio periódico do estado atual para todos os clientes conectados via WebSocket. Assim, a interface permanece sincronizada com as leituras dos sensores e com o status da irrigação. Os principais *endpoints* expostos são:

- `GET /api/status`: Fornece um retrato completo do estado atual, incluindo sensores e atuadores.
- `POST /api/irrigate`: Permite acionar manualmente a irrigação quando necessário.
- `POST /api/chat`: Recebe as mensagens enviadas pelo usuário e encaminha para o modelo de linguagem.

4.3 Módulos de Hardware (IoT)

A camada responsável pela interação com o hardware foi estruturada de forma modular, localizada em `src/code/iot/libs/`. Essa separação facilita testes, manutenção e eventual substituição de dispositivos.

4.3.1 Gerenciamento de Estado (SystemState)

A classe `SystemState` funciona como o ”centro”das informações do sistema. Como vários módulos trabalham simultaneamente como sensores, servidor, lógica de irrigação e a interface com o modelo de linguagem, o acesso ao estado global é protegido por bloqueios. Isso evita inconsistências e garante que leituras e escritas ocorram de forma segura entre *threads*.

4.3.2 Comunicação Serial (UARTHandler)

A comunicação com o ESP32 é realizada pelo módulo `UARTHandler`. Ele opera continuamente em uma *thread* separada, monitorando a porta `/dev/ttyAMA0`. Quando recebe mensagens como “Moisture=XX%”, o módulo extrai o valor, valida o formato e atualiza o estado global. Qualquer erro de transmissão ou formatação é tratado adequadamente para evitar leituras incorretas.

4.3.3 Controle de Sensores e Atuadores

A leitura de temperatura e umidade do ar é realizada pelo módulo `DHTSensor`, que utiliza a biblioteca `adafruit_dht`. O controle do microservo, responsável pelo acionamento da irrigação, é feito pelo `ServoController`. Esse módulo implementa verificações de segurança: caso o sensor de umidade indique que o solo já está suficientemente úmido, ou apresente valores inválidos, o sistema impede a ativação do servo para evitar irrigação excessiva.

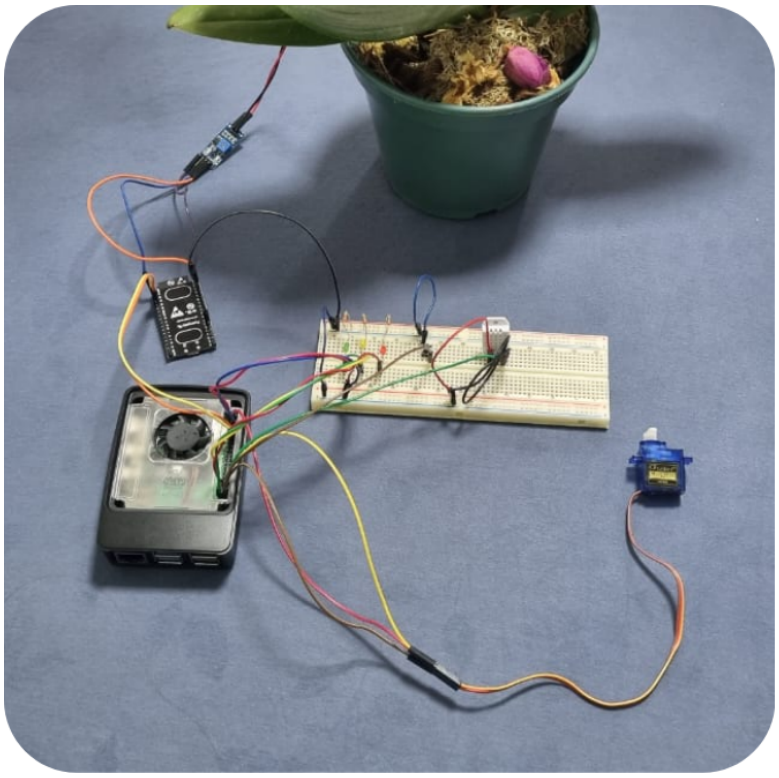

Figura 2: Imagem do circuito, com sensores e atuadores

4.3.4 Interface com LLM (LLMInterface)

A classe `LLMInterface` gerencia a comunicação com o modelo de linguagem executado localmente via Ollama. Antes de enviar a mensagem do usuário ao modelo `llama3.2:1b`, o sistema acrescenta ao *prompt* informações atualizadas dos sensores. Essa abordagem, inspirada em estratégias simplificadas de RAG, garante que as respostas do modelo estejam alinhadas com as condições reais da planta.

4.4 Interface de Usuário (Frontend)

A interface web, conforme Figura 3, foi construída em React. O componente `App.jsx` centraliza a conexão WebSocket e distribui os dados para os demais componentes.

Entre os elementos principais da interface estão:

- Dashboard — Exibe o estado geral da planta e os indicadores principais. Essa tela está representada pela Figura 4.
- ChatInterface — Mostra o histórico de conversas e permite enviar perguntas ao modelo. Essa tela está representada pela Figura 5.
- SensorChart — Responsável pelos gráficos históricos, desenvolvido com a biblioteca Recharts.

Durante o desenvolvimento, ferramentas baseadas em IA foram utilizadas de forma complementar. O Claude Agent colaborou na definição da estrutura dos componentes e no fluxo de atualização em tempo real, enquanto o GitHub Copilot auxiliou com sugestões de código, principalmente na integração com Tailwind CSS e no tratamento de eventos do Socket.IO.

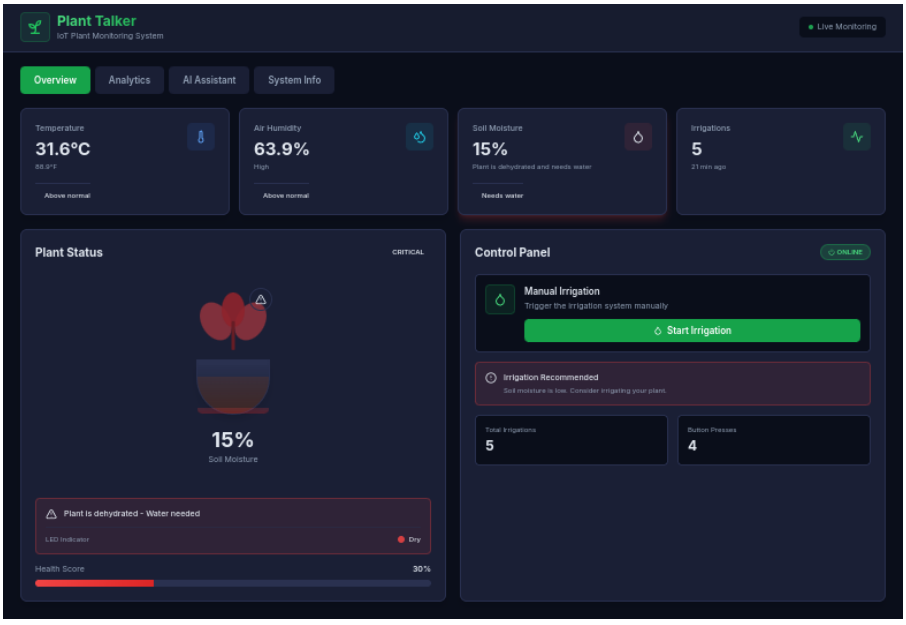

Figura 3: Tela inicial do sistema

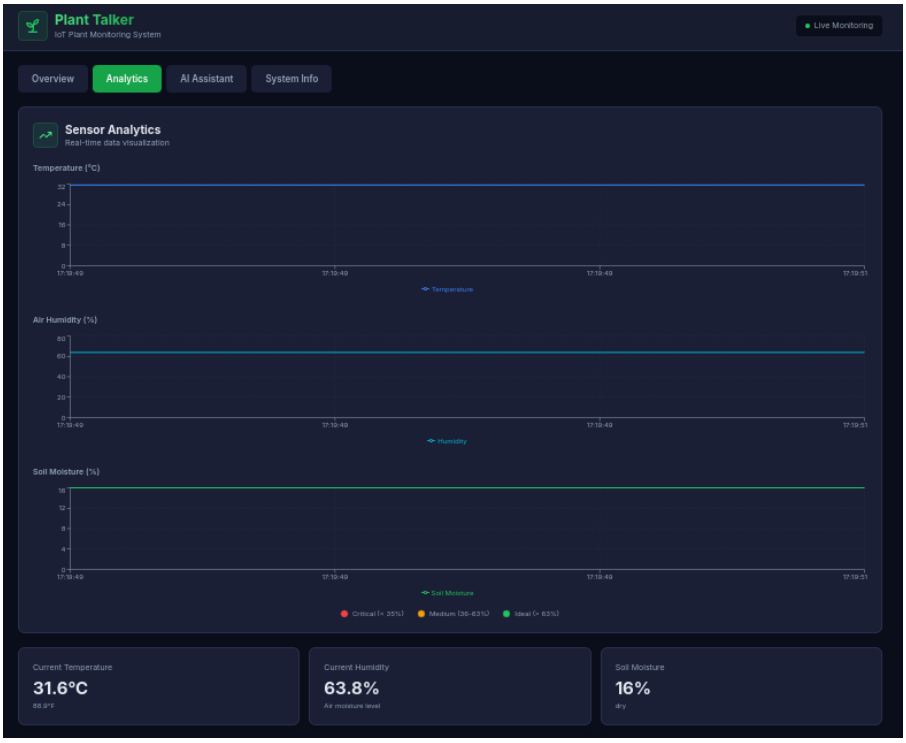

Figura 4: Tela dos dashboards

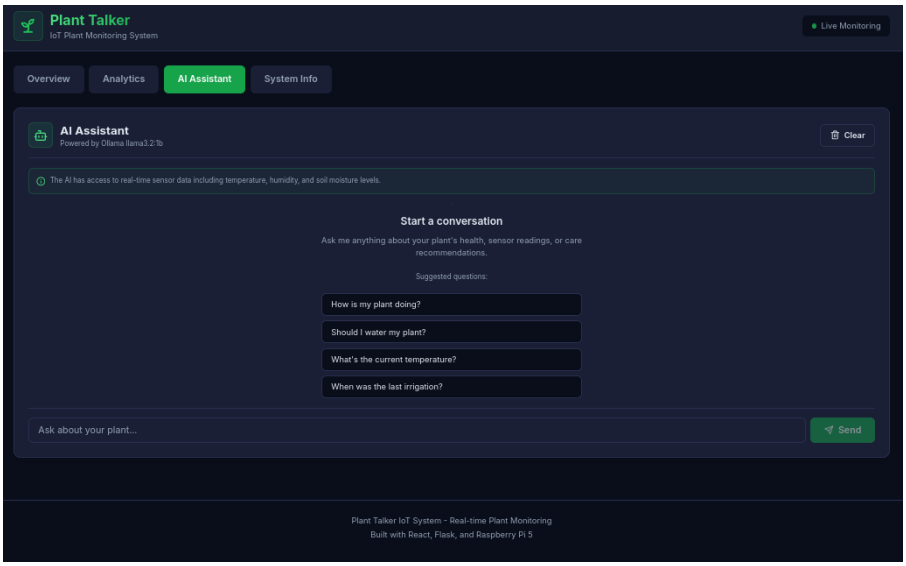

Figura 5: Chat com LLM

5 Teste e validação

O processo de validação do sistema foi conduzido de forma gradual, começando pela verificação individual de cada componente físico e avançando para testes de integração e avaliação do sistema funcionando de ponta a ponta. Essa abordagem permitiu identificar falhas de maneira localizada antes de consolidar o comportamento global da plataforma.

5.1 Testes de Componentes

Cada elemento de hardware foi analisado separadamente para garantir que funcionasse como esperado antes de ser integrado ao restante da aplicação:

- **Sensores**: As medições do DHT22 foram comparadas com um termômetro de referência para verificar a precisão e possíveis desvios. Já o sensor de umidade do solo no ESP32 foi testado tanto em solo seco quanto úmido, confirmando que a conversão analógica estava refletindo corretamente as variações das condições reais.
- **Atuadores**: O microservo passou por testes de movimento para assegurar que seu ângulo de atuação era suficiente para liberar o fluxo de água no mecanismo de irrigação. Os LEDs também foram verificados um a um, garantindo que cada cor correspondesse ao estado hídrico previsto.
- **Comunicação UART**: Realizou-se um teste contínuo de envio de dados do ESP32 para a Raspberry Pi para avaliar a estabilidade da porta serial e a integridade das mensagens recebidas, observando possíveis corrupções ou perdas.

5.2 Validação da Lógica de Controle

A lógica responsável pela irrigação automática foi testada simulando diferentes cenários de umidade. Dessa forma, foi possível confirmar que as decisões do sistema estavam alinhadas com as regras definidas:

- **Umidade $< 35\%$**: O sistema identificou corretamente a condição crítica, autorizando a irrigação.
- **Umidade $> 65\%$**: O solo foi interpretado como suficientemente úmido e a irrigação foi bloqueada.
- **Sensor desconectado**: O sistema reconheceu a ausência de leituras válidas e adotou a postura de segurança, impedindo qualquer acionamento do servo.

5.3 Avaliação da Interface e IA

A interface web foi testada em navegadores e dispositivos distintos para verificar seu comportamento responsivo. A comunicação via WebSocket mostrou-se estável, permitindo atualizações quase

em tempo real, aproximadamente a cada dois segundos.

O modelo de linguagem também passou por avaliações práticas. Perguntas relacionadas ao estado da planta foram feitas repetidamente, e o modelo demonstrou utilizar corretamente as leituras fornecidas no prompt. As respostas geradas foram coerentes ao contexto, comprovando que a estratégia de RAG simplificado cumpriu seu papel ao contextualizar a IA com informações reais do sistema.

6 Resultados e discussão

O sistema PlantTalker apresentou funcionamento estável ao longo dos testes realizados, permitindo avaliar tanto o desempenho dos componentes individuais quanto o comportamento integrado da solução. A interação entre o ESP32, a Raspberry Pi e a interface web operou conforme o esperado, possibilitando a coleta contínua de dados ambientais e sua disponibilização ao usuário em tempo quase real.

6.1 Desempenho do Monitoramento

As leituras dos sensores mantiveram consistência temporal, sem oscilações abruptas que indicassem erros de medição. A visualização histórica disponibilizada na interface facilitou a identificação de tendências, como variações térmicas ao longo do dia e flutuações de umidade associadas ao microambiente. A comunicação UART entre o ESP32 e a Raspberry Pi apresentou estabilidade, sem registros significativos de perda de pacotes ou corrupção de dados durante os testes prolongados.

Observou-se também que a taxa de atualização configurada foi suficiente para representar o comportamento ambiental sem sobrecarregar o barramento de comunicação. Entretanto, em cenários de maior densidade de sensores ou maior frequência de amostragem, seria necessário avaliar se o throughput atual da UART permanece adequado.

6.2 Interação com o Usuário

A interface web construída em React permitiu acesso contínuo ao estado dos sensores e aos indicadores do sistema. A atualização via WebSocket apresentou baixa latência, possibilitando que o usuário acompanhasse em tempo quase real as mudanças nas variáveis monitoradas. Elementos visuais como indicadores de cor e gráficos temporais contribuíram para a interpretação rápida do estado da planta, reduzindo a necessidade de leitura individual de valores numéricos.

A funcionalidade de chat com o modelo de linguagem `llama3.2:1b` demonstrou que o uso de um modelo local é viável em hardware de borda como a Raspberry Pi 5. Os tempos de inferência variaram de acordo com a complexidade da consulta, mas permaneceram dentro de limites aceitáveis para interação assíncrona. O modelo utilizou corretamente os dados inseridos no *prompt*, respondendo às perguntas do usuário de forma contextualizada. No entanto, a qualidade das respostas depende diretamente da atualização e estruturação do contexto fornecido, o que sugere a necessidade de mecanismos mais formais de injeção de dados caso a complexidade do sistema aumente.

6.3 Automação e Controle

O módulo de controle da irrigação executou a lógica de decisão conforme especificado, acionando o servo apenas quando o nível de umidade estava abaixo do limiar definido. Em testes com

valores simulados e reais, o comportamento permaneceu consistente, indicando que a camada de software é capaz de impedir acionamentos incorretos, seja por leituras inválidas do sensor ou por estados intermediários do sistema.

A atuação do sistema também revelou limitações esperadas de soluções baseadas em sensores resistivos de umidade do solo, cuja variabilidade pode afetar decisões automáticas caso não haja filtragem ou calibração adequadas. O comportamento do servo motor apresentou precisão suficiente para o mecanismo utilizado, mas sua durabilidade dependerá do regime de acionamento e das condições mecânicas do conjunto.

7 Conclusão e trabalhos futuros

Este projeto atingiu seu objetivo principal de desenvolver um sistema de monitoramento de plantas inteligente e acessível, utilizando conceitos de sistemas embarcados e inteligência artificial de borda. A arquitetura híbrida, combinando a eficiência do ESP32 para leitura de sensores analógicos com o poder de processamento da Raspberry Pi para a IA e interface web, provou ser uma escolha adequada.

A utilização de modelos de linguagem locais (SLMs) demonstrou que é possível trazer interfaces de conversação avançadas para dispositivos de borda sem depender de conexão com a nuvem, garantindo privacidade e funcionamento *offline*. A implementação da interface gráfica com tecnologias web modernas, auxiliada por ferramentas de geração de código como o Claude Agent e Copilot, resultou em um produto final com acabamento profissional e alta usabilidade.

Como trabalhos futuros, sugere-se a integração de uma câmera para análise visual da saúde da planta utilizando visão computacional, a implementação de um sistema de gerenciamento de energia para operação com baterias e a criação de uma placa de circuito impresso (PCB) dedicada para reduzir o tamanho físico e a complexidade das conexões. Além disso, o refinamento do modelo de linguagem (*fine-tuning*) com um conjunto de dados específico de botânica poderia melhorar ainda mais a precisão das respostas.

Também se propõe o desenvolvimento de um sistema de irrigação completo e automatizado. A versão atual utiliza apenas um servo motor para simulação do mecanismo de irrigação, sendo a rega realizada manualmente. Uma implementação real incluiria bomba, válvula solenóide ou microbomba peristáltica, reservatório e sensores adicionais, permitindo controle hídrico.

8 Referências

ROVAI, Marcelo José. **Edge AI Engineering**. Disponível em: https://mjrovai.github.io/EdgeML_Made_Ease_ebook/. Acesso em: 06/12/2025.

REDDI, Vijay Janapa. **Machine Learning Systems (MLSysBook). 2024**. Disponível em: https://mlsysbook.ai/. Acesso em: 06/12/2025.

ALMEIDA, Rodrigo Maximiano A.; GOMES, Otávio; MORAES, Carlos Henrique V.; SERAPHIM, Thatyana F. Piola. **Programação de Sistemas Embarcados. 2. ed.** São Paulo: Grupo GEN, 2024.