Sistema de Monitoramento e Reconhecimento Facial Distribuído em IoT: Uma Abordagem Híbrida com MQTT, ESP32 e Edge Computing

João André Simioni
PPGIa
PUCPR
Curitiba, Brazil
joao.asimioni@ppgia.pucpr.br

***Resumo*—O avanço da Internet das Coisas (IoT) possibilitou a criação de sistemas de monitoramento inteligentes e distribuídos. No entanto, o processamento de visão computacional em microcontroladores de borda (Edge) apresenta desafios significativos de desempenho e largura de banda. Este trabalho apresenta o desenvolvimento de uma solução completa de IoT para detecção e reconhecimento facial, integrando dispositivos ESP32-S3 e ESP32 Standard, protocolo MQTT e a plataforma Node-RED. A metodologia aborda desde a captura de imagem e tentativas de processamento local com TinyML (YOLO), até a implementação de uma arquitetura híbrida onde a detecção preliminar ocorre na borda e o reconhecimento robusto no servidor (Backend em Python). O sistema inclui mecanismos de otimização de tráfego de rede, enviando imagens apenas mediante detecção facial local, reduzindo o consumo de banda de 600kbps para 5kbps em estado de espera. Os resultados demonstram a viabilidade da integração entre hardware de baixo custo e processamento distribuído, visualizados através de um Dashboard Node-RED e displays físicos.**

***Index Terms*—IoT, MQTT, Face Recognition, ESP32, Node-RED, TinyML, Bandwidth Optimization.**

I. Introdução

A onipresença da computação e a miniaturização dos componentes eletrônicos impulsionaram a disciplina de Internet of Things (IoT) como um pilar fundamental na engenharia moderna. A capacidade de conectar objetos cotidianos à internet para coletar dados, tomar decisões e atuar no ambiente físico transformou setores como segurança, automação residencial e indústria 4.0.

Um dos desafios centrais no ensino e aplicação de IoT é a integração de múltiplos protocolos e arquiteturas para criar soluções robustas. O protocolo Message Queuing Telemetry Transport (MQTT) estabeleceu-se como o padrão de facto para comunicação leve e assíncrona. Contudo, quando se introduz dados complexos, como *streaming* de vídeo e reconhecimento facial, a arquitetura de IoT tradicional enfrenta gargalos de processamento e largura de banda.

Este trabalho foi motivado pela necessidade de explorar os fundamentos de IoT aplicados a um problema de alta complexidade: a visão computacional distribuída. O objetivo principal é desenvolver e validar uma solução completa (fim-a-fim) que utilize microcontroladores da família ESP32 (especificamente o ESP32-S3 para captura e o ESP32 padrão para atuação) comunicando-se via MQTT.

O sistema proposto realiza a leitura de imagens através de um módulo ESP32-S3 equipado com câmera. Diferente de abordagens triviais de streaming contínuo, este projeto investiga o processamento na borda (*Edge Computing*) para detecção facial preliminar, visando a otimização de recursos de rede. As imagens são transmitidas via MQTT para um backend em Python, que executa algoritmos de *Deep Learning* para identificar o indivíduo com base em uma lista pré-fornecida. O resultado do reconhecimento retorna ao ecossistema IoT, sendo exibido em um Dashboard Node-RED e fisicamente em um display de cristal líquido (LCD) conectado a um segundo microcontrolador.

Além da implementação funcional, este artigo discute as limitações encontradas ao tentar executar redes neurais complexas (como YOLOv5) diretamente no microcontrolador, justificando a adoção de uma arquitetura híbrida.

O trabalho inicia com a fundamentação teórica na Seção II, seguido da metodologia na Seção III. Na Seção IV são apresentados os resultados obtidos com a implementação. Por fim, a Seção V finaliza o trabalho.

II. Fundamentação Teórica

Para contextualizar o desenvolvimento do sistema, esta seção apresenta os conceitos fundamentais e as tecnologias empregadas.

A. Internet das Coisas (IoT)

A IoT refere-se à interconexão digital de objetos cotidianos com a internet. Segundo a definição clássica, trata-se de uma rede de objetos físicos incorporados a sensores, software e outras tecnologias com o objetivo de conectar e trocar dados com outros dispositivos e sistemas. A evolução dessa arquitetura permite não apenas a conectividade, mas a criação de serviços avançados através da interconexão de objetos físicos e virtuais, conforme detalhado no levantamento realizado por Al-Fuqaha et al.

[1]. Neste trabalho, a IoT é materializada pela rede de microcontroladores (nós sensores e atuadores) que operam de forma coordenada.

B. Protocolo MQTT

O MQTT é um protocolo de mensagens leve, baseado no modelo *publish/subscribe*, ideal para conexões com largura de banda limitada e dispositivos com recursos restritos. Diferente do HTTP (request/response), o MQTT permite que dispositivos publiquem dados em tópicos e outros assinem esses tópicos para receber informações, desacoplando o produtor do consumidor da informação. O servidor central que gerencia essas mensagens é denominado *Broker*. Estudos comparativos demonstram que o MQTT apresenta menor latência e consumo de banda em cenários de IoT quando comparado a protocolos baseados em requisição-resposta como o HTTP [2].

C. Microcontroladores ESP32 e ESP32-S3

O ESP32, desenvolvido pela Espressif Systems, é uma série de SoCs (*System on a Chip*) de baixo custo e baixo consumo de energia com Wi-Fi e Bluetooth integrados.

1) **ESP32 Standard:** Utiliza um microprocessador Tensilica Xtensa LX6 dual-core. Neste projeto, é utilizado para o controle do display LCD.
2) **ESP32-S3:** Uma evolução focada em IA e IoT, equipada com um processador Xtensa LX7 dual-core e instruções vetoriais para aceleração de redes neurais. Devido à sua maior capacidade de processamento e suporte nativo a interfaces de câmera, foi selecionado como o dispositivo de captura e processamento de imagem. [3]

A arquitetura dual-core e os periféricos integrados do ESP32 o tornam uma plataforma robusta para processamento na borda, superando limitações de microcontroladores de 8-bits em aplicações de tempo real [4].

D. Ambiente de Desenvolvimento Arduino IDE

A IDE do Arduino é um ambiente de desenvolvimento integrado multiplataforma que permite escrever e carregar código em microcontroladores compatíveis. Sua vasta biblioteca de código aberto e suporte da comunidade facilitam a prototipagem rápida, abstraindo complexidades de configuração de hardware de baixo nível. O uso desta plataforma em ambientes educacionais e de engenharia reduz significativamente o tempo de desenvolvimento de sistemas embarcados complexos, abstraindo a complexidade dos registradores de hardware [5].

E. Node-RED

Node-RED é uma ferramenta de programação baseada em fluxo (*flow-based programming*), desenvolvida originalmente pela IBM. Ela permite conectar dispositivos de hardware, APIs e serviços online de forma visual. Neste trabalho, o Node-RED atua como o integrador central, fornecendo o Dashboard para visualização de dados, controle

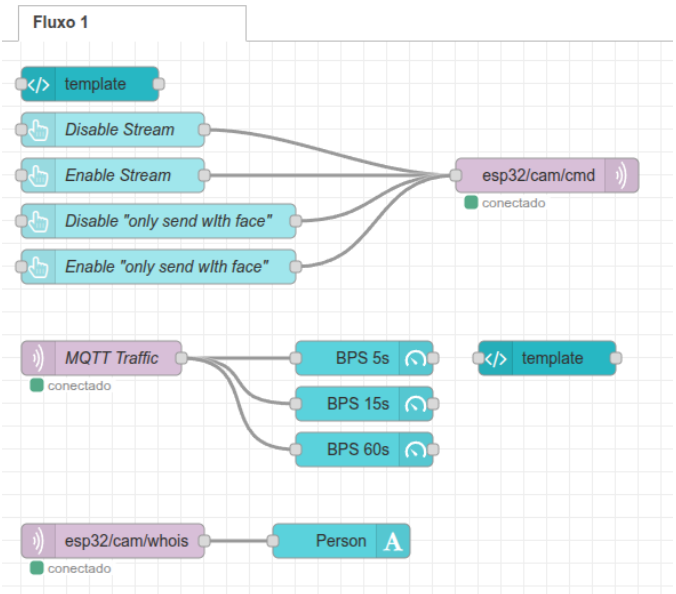

Figura 1: Node-RED Nodes utilizados nesse projeto

de fluxo e apresentação do stream de vídeo. A abordagem de programação visual do Node-RED facilita a orquestração de dispositivos heterogêneos e APIs, permitindo o desenvolvimento rápido de aplicações IoT baseadas em eventos [6]. A figura 1 mostra o exemplo desse trabalho de como são os fluxos no Node-RED.

F. YOLO (You Only Look Once)

YOLO é uma arquitetura de rede neural convolucional (CNN) projetada para detecção de objetos em tempo real. Diferente de abordagens que aplicam classificadores em várias partes da imagem, a YOLO aplica uma única rede neural em toda a imagem, dividindo-a em regiões e prevendo caixas delimitadoras e probabilidades para cada região. A evolução da família YOLO tem focado no equilíbrio entre precisão e eficiência computacional, sendo amplamente adotada para inferência em tempo real devido à sua arquitetura de estágio único [7].

G. TinyML

TinyML (*Tiny Machine Learning*) é uma área de estudo focada em implementar modelos de aprendizado de máquina em dispositivos de hardware com recursos extremamente limitados, como microcontroladores alimentados por bateria. O objetivo é realizar inferência na borda (*Edge*), reduzindo latência e dependência de conectividade. O paradigma TinyML permite a execução de análises de dados complexas diretamente no dispositivo final, reduzindo a latência e a dependência de conectividade contínua com a nuvem [8].

H. Bibliotecas de Detecção Facial

Frameworks modernos de percepção, como o MediaPipe e dlib, utilizam pipelines otimizados para acelerar a in-

ferência de modelos de visão computacional em CPUs convencionais, viabilizando o reconhecimento facial eficiente [9]. Nesse trabalho, as bibliotecas utilizadas foram:

1) **human_face_detect_msr01 (ESP32):** Uma biblioteca otimizada para detecção de faces em microcontroladores ESP32. Ela utiliza algoritmos de processamento de imagem clássicos ou redes neurais quantizadas extremamente leves para identificar a presença de rostos.
2) **face_detection (Python):** No backend, utilizam-se bibliotecas mais robustas baseadas em *dlib*, capazes não apenas de detectar a face, mas de extrair *embeddings* (características faciais) para reconhecimento e comparação com um banco de dados de faces conhecidas. [10]

III. Metodologia

A metodologia deste trabalho é dividida em quatro etapas principais: a implementação do firmware de captura e envio inteligente no ESP32-S3, o desenvolvimento do backend de processamento de imagem, a criação da interface de monitoramento e controle, e a integração com dispositivos de feedback físico. Todo o código desenvolvido está disponível publicamente [11].

A. Captura e Processamento na Borda (ESP32-S3)

A interface de desenvolvimento escolhida foi a Arduino IDE, utilizando a biblioteca `esp_camera.h` para gerenciamento do hardware de câmera. O objetivo inicial era implementar uma solução puramente *Edge AI* utilizando a arquitetura YOLO.

1) Tentativas de Implementação de YOLO: Inicialmente, buscou-se utilizar a implementação de YOLOv5n para ESP32-S3 baseada no trabalho de Shafi [12]. Embora funcional, essa implementação de YOLO genérica (treinada no dataset COCO) apresentou um tempo de inferência inviável para aplicações em tempo real, atingindo 30 segundos para processar um único frame de 320x240 pixels.

Optou-se então por migrar para a YOLOv5-face [13], especificamente o modelo yolov5n-0.5, que possui apenas 447 mil parâmetros, sendo teoricamente compatível com o hardware. O desafio, no entanto, residiu na conversão do modelo (originalmente em PyTorch) para código C compatível com o ecossistema Arduino/ESP32.

Ferramentas de conversão como synapedge [14] e onnx2c [15] falharam em converter a topologia específica da rede YOLOv5-face. Tentativas adicionais com keras2c [16] enfrentaram conflitos severos de dependências de versões no ambiente Python (conversão ONNX → H5).

A alternativa mais promissora seria o uso do TensorFlow Lite (TFLite) para Microcontroladores. Contudo, os exemplos funcionais exigiam o framework nativo ESP-IDF, o que desviaria do escopo de utilizar o Arduino IDE, adicionando complexidade excessiva à configuração do ambiente.

Solução Adotada:

Diante das limitações, a solução foi utilizar a biblioteca human_face_detect_msr01. Para garantir compatibilidade, foi necessário realizar o *downgrade* do Core do ESP32 na Arduino IDE para a versão 3.0.7.

O firmware desenvolvido opera da seguinte maneira:

1) **Captura:** O microcontrolador captura a imagem em formato JPEG.
2) **Pré-processamento:** Para a detecção local, a imagem JPEG é decodificada e convertida para o formato de cor RGB565 utilizando a função `jpg2rgb565` da biblioteca `img_converters.h`.
3) **Correção de Endianness:** O método de inferência da classe `HumanFaceDetectMSR01` espera a estrutura de bytes da imagem em *big-endian*, enquanto a saída do conversor é *little-endian*. Foi implementada uma rotina de conversão de bytes antes da inferência.
4) **Inferência:** Utiliza-se uma instância do detector (`detector(0.3F, 0.3F, 10, 0.3F)`) para verificar a existência de faces.

Gerenciamento de Transmissão via MQTT:

O sistema implementa lógica de controle de banda baseada em comandos MQTT recebidos no tópico esp32/cam/cmd. Os modos de operação são:

1) `disable`: Câmera desligada.
2) `enable`: Envio contínuo de stream (MJPEG sobre MQTT).
3) `enable_only_send_if_face_detected`: A imagem só é publicada no tópico `esp32/cam/image` se o detector local confirmar a presença de um rosto.
4) `disable_only_send_if_face_detected`: Retorna ao envio contínuo.

Esta lógica de "envio condicional"é crucial para sistemas IoT que operam em redes 4G/LoRa ou Wi-Fi congestionados, preservando largura de banda quando não há atividade de interesse.

O código está disponível em [17].

B. Processamento de Imagem no Backend (Python)

Um serviço em Python atua como assinante (*subscriber*) do tópico `esp32/cam/image`. O processamento ocorre em duas etapas sequenciais para cada frame recebido:

1) **Detecção Robusta:** Utiliza a biblioteca `yolov5-faces` para confirmar a detecção e delimitar a região de interesse (Bounding Box).
2) **Reconhecimento:** A região recortada é processada pela biblioteca `face_detection`, que compara as características faciais com um diretório de imagens conhecidas.

O script anota na imagem o nome da pessoa identificada (em verde) ou "Unknown"(em vermelho). O nome identificado é publicado no tópico MQTT `esp32/cam/whois`. Adicionalmente, o script disponibiliza um *endpoint* HTTP local que serve a imagem processada como um stream de

vídeo, permitindo a integração com navegadores web. O código está disponível em [18].

O sistema de reconhecimento facial não utiliza um banco de dados relacional tradicional (como SQL) para armazenar as identidades. Em vez disso, optou-se por uma abordagem baseada em sistema de arquivos, centralizada no diretório denominado `known_faces`. Este diretório atua como o *dataset* de referência biométrica do sistema.

O funcionamento deste componente ocorre em três etapas críticas durante a inicialização do *backend* em Python:

1) **Indexação Automática:** Ao iniciar, o script varre o diretório `known_faces` em busca de arquivos de imagem (formatos `.jpg`, `.jpeg`, `.png`).
2) **Extração de Características (*Embeddings*):** Para cada imagem encontrada, o algoritmo carrega o arquivo e executa a detecção facial. Uma vez localizada a face, a biblioteca `face_recognition` calcula um vetor de 128 dimensões (*embedding*) que representa matematicamente as características únicas daquele rosto.
3) **Associação de Identidade:** O sistema utiliza o nome do arquivo (excluindo a extensão) como o rótulo da identidade. Por exemplo, uma imagem salva como `João Simioni.jpg` criará um registro em memória vinculando o vetor calculado ao nome "João Simioni".

Esta lista de vetores pré-calculados é mantida em memória RAM para garantir alta velocidade durante o processo de inferência. Quando uma nova face é detectada no fluxo de vídeo em tempo real (provinda do ESP32), seu vetor é comparado via distância euclidiana contra todos os vetores carregados do diretório `known_faces`. Se a distância for menor que o limiar de tolerância (definido em 0.55), o sistema atribui o nome do arquivo correspondente à face detectada; caso contrário, a classifica como "Unknown".

Esta abordagem simplifica a gestão de usuários: para adicionar uma nova pessoa autorizada ao sistema, basta salvar uma foto do rosto dela neste diretório e reiniciar o serviço, sem necessidade de re-treinamento da rede neural ou complexas inserções em banco de dados.

C. Dashboard e Monitoramento (Node-RED)

O Node-RED foi configurado como a interface central de operação [19]. O fluxo desenvolvido contém:

- **Controle:** Quatro botões que publicam os comandos de configuração (`enable`, `disable`, etc.) para o tópico de comando do ESP32.
- **Visualização:** Um nó de *iframe* HTML exibe o stream de vídeo gerado pelo backend Python.
- **Informação:** Um campo de texto exibe em tempo real o último valor recebido no tópico `esp32/cam/whois`.
- **Monitoramento de Rede:** Três medidores (*gauges*) exibem o tráfego de dados na porta 1883 (MQTT) nos últimos 5, 15 e 60 segundos.

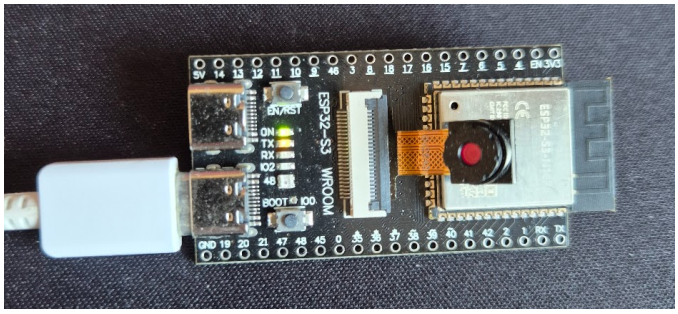

Figura 2: ESP32-S3 com o módulo de câmera acoplado

O monitoramento de tráfego é alimentado por um script auxiliar em Python que "escuta"(*sniffs*) o tráfego de rede na interface do Broker e publica as estatísticas no tópico `monitor/traffic/1883`.

D. Feedback Físico (ESP32 + LCD)

Para demonstrar a atuação em IoT, um módulo ESP32-WROVER foi conectado a um display de cristal líquido. Este dispositivo assina o tópico `esp32/cam/whois`. Ao receber uma mensagem, ele atualiza o display com o nome da pessoa reconhecida, simulando, por exemplo, um sistema de controle de acesso ou portaria inteligente [20].

IV. Resultados e Discussão

A integração dos componentes mostrou-se estável e funcional. O sistema foi capaz de realizar o ciclo completo: capturar a imagem, processar localmente para decisão de envio, processar remotamente para reconhecimento e atuar nos displays digitais e físicos.

A. Análise Visual

A figura 2 apresenta o ESP32-S3 em operação. A figura 3 demonstra o ESP32 com LCD exibindo o nome de um usuário reconhecido. A figura 4 exibe o Dashboard Node-RED, consolidando vídeo, controles e métricas.

O principal resultado quantitativo deste trabalho refere-se à economia de largura de banda proporcionada pela técnica de Edge Computing.

Ao observar o monitor de tráfego no Dashboard Node-RED:

- **Modo Stream Contínuo:** O tráfego oscila entre 400kbps e 600kbps, dependendo da complexidade visual da cena e da taxa de compressão JPEG variável.
- **Modo Detecção Ativa:** Quando configurado para `enable_only_send_if_face_detected`, o tráfego em repouso (sem faces na frente da câmera) cai para aproximadamente **5kbps**. Este tráfego residual refere-se apenas às mensagens de *keep-alive* do protocolo MQTT e telemetria básica.

A redução drástica de $\tilde{6}$00kbps para $\tilde{5}$kbps (uma redução de 99%) valida a importância de processar dados na borda antes da transmissão para a nuvem.

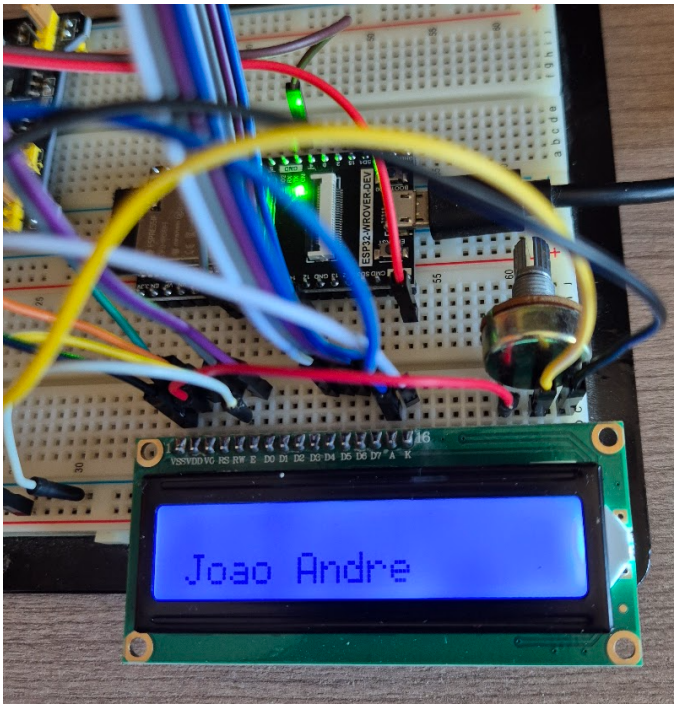

Figura 3: ESP32 com LCD utilizando um protoboard

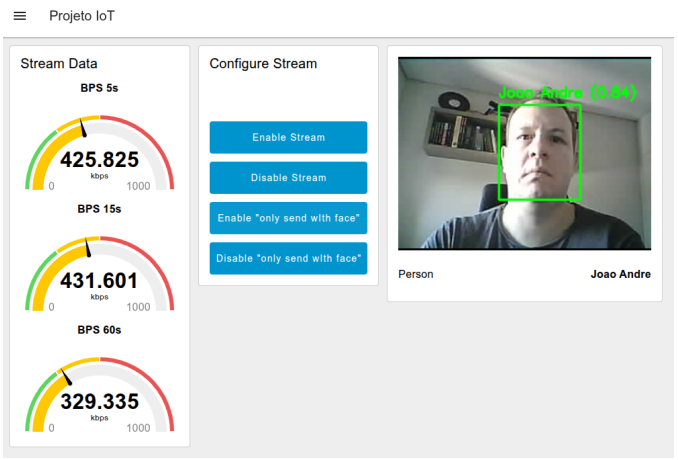

Figura 4: Dashboard utilizando Node-RED

B. Desempenho do Hardware

O ESP32-S3, operando com a biblioteca `human_face_detect_msr01`, conseguiu manter uma taxa de quadros aceitável para detecção de presença. No entanto, o processo de conversão de formato de imagem (JPEG → RGB565 → Endian Swap) consome ciclos de CPU significativos. A latência total do sistema (do movimento real até a atualização no LCD) ficou na casa de centenas de milissegundos, sendo adequada para controle de acesso, mas crítica para aplicações de alta velocidade.

C. Detalhamento dos Tópicos MQTT

A estrutura de comunicação do sistema é detalhada na Tabela I. Abaixo apresento alguns detalhes adicionais de cada um dos tópicos.

- **Fluxo de Vídeo** (`esp32/cam/image`):
  - É o tópico que consome mais banda. O ESP32-S3 captura o frame, converte para JPEG e publica aqui.
  - O script Python (`web-ui.py`) escuta este tópico, decodifica a imagem e executa a inferência YOLO/FaceRecognition.
- **Fluxo de Controle** (`esp32/cam/cmd`):
  - Permite a alteração dinâmica do comportamento do firmware sem reinicializar o dispositivo.
  - O Node-RED envia comandos simples de texto que alteram a máquina de estados interna do ESP32-S3.
- **Fluxo de Identificação** (`esp32/cam/whois`):
  - Este tópico atua como um gatilho de eventos.
  - Quando o backend Python identifica uma pessoa, ele publica o nome aqui.
  - Isso aciona instantaneamente a atualização do texto no Dashboard e a escrita no Display LCD do ESP32 remoto.
- **Fluxo de Telemetria** (`monitor/traffic/1883`):
  - Usado exclusivamente para observabilidade.
  - O *sniffer* de rede calcula a taxa de bits e publica um objeto JSON.
  - O Node-RED faz o *parse* deste JSON para atualizar os gráficos *Gauge*.

V. Conclusão

Este trabalho apresentou uma implementação prática e completa de um sistema IoT para reconhecimento facial, integrando hardware de baixo custo e protocolos padrão de indústria. A disciplina proporcionou a aplicação consolidada de conceitos de MQTT, visão computacional e desenvolvimento de dashboards.

A principal contribuição técnica foi a demonstração da arquitetura híbrida. Enquanto o processamento pesado (reconhecimento facial profundo via YOLOv5 e *embeddings*) é inviável no microcontrolador em tempo real, o uso do ESP32-S3 como um filtro inteligente (detectando apenas a presença de rostos) provou ser uma estratégia eficiente para economia de recursos de rede.

Como trabalhos futuros, sugere-se:

1) Aprofundar a implementação com **TensorFlow Lite (TFLite)** usando ESP-IDF para permitir o uso de modelos mais modernos de detecção diretamente no chip.
2) Implementar o recorte da face (*cropping*) diretamente no ESP32-S3. Atualmente, a imagem completa é enviada; enviar apenas o recorte do rosto

Tabela I: Resumo da Arquitetura de Tópicos MQTT

Tópico	Publicador	Assinante	Payload	Descrição Funcional
`esp32/cam/image`	ESP32-S3	Backend Python	Binário (JPEG)	Fluxo de imagens. No modo *Smart*, publica apenas se houver detecção facial local.
`esp32/cam/cmd`	Node-RED	ESP32-S3	Texto	Comandos de controle (ex: `enable`, `disable`) para alterar o modo de operação.
`esp32/cam/whois`	Backend Python	Node-RED, ESP32 (LCD)	Texto	Resultado do reconhecimento facial (Nome da pessoa ou "Unknown").
`monitor/traffic/1883`	Monitor Python	Node-RED	JSON	Estatísticas de tráfego de rede (bps) em tempo real para telemetria.

reduziria ainda mais o tráfego de dados, potencialmente para menos de 50kbps durante o reconhecimento.

3) Avaliar o consumo energético (bateria) comparando o modo de envio contínuo versus o modo de detecção local.

O código fonte completo e documentação estão disponíveis no repositório do projeto no GitHub, servindo como referência para futuras implementações acadêmicas e comerciais.

Referências

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[2] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A Survey on Application Layer Protocols for the Internet of Things," *IEEE Access*, vol. 3, pp. 11–17, 2015.

[3] Espressif Systems, *ESP32-S3 Technical Reference Manual*. Espressif Systems, 2023. Version 1.1.

[4] A. Maier, A. Sharp, and Y. Vagapov, "Comparative analysis and practical implementation of the ESP32 microcontroller module for the Internet of Things," in *2017 23rd International Conference on Automation and Computing (ICAC)*, pp. 1–6, 2017.

[5] Y. A. Badamasi, "The working principle of an Arduino," in *2014 IEEE Electronics, Computer and Artificial Intelligence (ECAI)*, pp. 1–4, 2014.

[6] I. U. ONWUEGBUZIE, "Node-red and iot analytics: A real-time data processing and visualization platform," 2024.

[7] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A Review of Yolo Algorithm Developments," *IEEE Access*, vol. 10, pp. 106617–106638, 2022.

[8] S. Heydari and Q. H. Mahmoud, "Tiny machine learning and on-device inference: A survey of applications, challenges, and future directions," *Sensors*, vol. 25, p. 3191, May 2025.

[9] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann, "Mediapipe: A framework for building perception pipelines," 2019.

[10] N. Boyko, O. Basystiuk, and N. Shakhovska, "Performance evaluation and comparison of software for face recognition, based on dlib and opencv library," in *2018 IEEE Second International Conference on Data Stream Mining amp; Processing (DSMP)*, p. 478–482, IEEE, Aug. 2018.

[11] J. A. Simioni, "IoT Face Detection Repository." https://github.com/jasimioni/iot/tree/main/face-detection, 2025. Acesso em: Dez. 2025.

[12] A. Shafi, "Running-YOLOv5n-on-an-ESP32-S3." . Acesso em: Dez. 2025.

[13] Deepcam-cn, "Yolov5-face." https://github.com/deepcam-cn/yolov5-face. Acesso em: Dez. 2025.

[14] A. Shafi, "SynapEdge: End-to-end framework for Edge AI." https://github.com/asad-shafi/synapedge. Acesso em: Dez. 2025.

[15] Kraiskil, "onnx2c: Portable ONNX to C compiler." https://github.com/kraiskil/onnx2c. Acesso em: Dez. 2025.

[16] PlasmaControl, "keras2c: A library for converting Keras models to C." https://github.com/PlasmaControl/keras2c. Acesso em: Dez. 2025.

[17] J. A. Simioni, "Camera to MQTT Firmware." https://github.com/jasimioni/iot/blob/main/face-detection/Image2MQTT/Image2MQTT.ino, 2025.

[18] J. A. Simioni, "Web UI and Python Processor." https://github.com/jasimioni/iot/blob/main/face-detection/web-ui.py, 2025.

[19] J. A. Simioni, "Node-RED Flows." https://github.com/jasimioni/iot/blob/main/face-detection/node-red/flows.json, 2025.

[20] J. A. Simioni, "MQTT Display Info Firmware." https://github.com/jasimioni/iot/blob/main/face-detection/MQTTDisplayInfo/MQTTDisplayInfo.ino, 2025.