







Avaliação de Inferência em Edge AI sob Restrições Embarcadas em um Sistema Robótico Simulado Baseado na Internet das Coisas Robóticas

Kelton M. Dias¹ , João Pedro S. Rodrigues¹ , Eric Rohmer² ,
Felipe Augusto O. Mota^{1,2} 

¹Instituto Federal do Norte de Minas Gerais - Campus Januária (IFNMG)
Januária – MG – Brazil

²Universidade Estadual de Campinas (UNICAMP)
Campinas – SP – Brazil

keltonm6@gmail.com, jpsr.jpsantos@gmail.com,

rohmer@unicamp.br, felipe.mota@ifnmg.edu.br

Abstract. *This work evaluates Edge AI visual inference in the Internet of Robotic Things (IoRT) under strict hardware constraints. We adapted and deployed three CNNs on an ESP32-S3 microcontroller, integrating them into a CoppeliaSim industrial environment with heterogeneous robots coordinated via MQTT. Results reveal a critical discrepancy between desktop memory estimates and actual ESP32 requirements (MobileNetV2 demanded 204% more arena). Furthermore, MobileNets produced systematic misclassifications on real hardware. Only a custom lightweight CNN operated successfully, requiring just 38.4 KB of arena and 53.87 ms per inference, achieving 100% success rate across 200 cycles. Code and data: https://github.com/Keltonmd/EdgeAI_SBRC.*

Resumo. *Este trabalho avalia inferência Edge AI na Internet das Coisas Robóticas (IoRT). Três CNNs foram embarcadas em um ESP32-S3 de forma interligada a um cenário industrial simulado via MQTT no CoppeliaSim. Constatou-se uma grave discrepância entre estimativas teóricas de memória vis-à-vis sua alocação real: a MobileNetV2 consumiu 204% mais arena que a conversão acusava em placa, falhando junto da pré-treinada V3 nos ensaios embarcados práticos. Apenas a CNN autoral operou com fluidez sistêmica efetiva e 100% de acerto sob 53,87 ms consumindo enxutos 38,4 KB estáticos, isolando predições fidedignas no chão de fábrica simulado. Código: https://github.com/Keltonmd/EdgeAI_SBRC.*

1. Introdução

A convergência entre conectividade, automação e inteligência embarcada tem ampliado o escopo de aplicações baseadas em sistemas ciberfísicos, sobretudo em ambientes industriais e logísticos [Ray 2016], [Simoens et al. 2018]. Robôs industriais, móveis autônomos e de serviço operam cada vez mais integrados a redes de comunicação, exigindo *hardware* e *software* compatíveis com atuação coordenada [Vermesan et al. 2020], [Kabir et al. 2023].

Em cenários assim, a percepção do ambiente, a tomada de decisão e a atuação precisam ocorrer com previsibilidade temporal rigorosa. Em aplicações genéricas de *mobile*

edge, atrasos ou erros de inferência limitam-se à degradação de um serviço digital; na Internet das Coisas Robóticas (IoRT), entretanto, os erros produzem consequências físicas imediatas, podendo causar colisões ou falhas severas na cadeia logística [Simoens et al. 2018], [Vermesan et al. 2020], [Kabir et al. 2023].

A Internet das Coisas (IoT) consolidou-se como base tecnológica para a interconexão de objetos físicos por meio de redes digitais padronizadas [Simoens et al. 2018]. Entretanto, quando aplicada à cenários que demandam atuação física direta e tomada de decisão autônoma, a IoT tradicional apresenta limitações, uma vez que seus dispositivos operam predominantemente como nós passivos ou com capacidades cognitivas restritas [Ray 2016]. A superação dessas limitações impulsionou o surgimento da IoRT, que integra robótica, conectividade e inteligência artificial em sistemas distribuídos, permitindo que robôs atuem como agentes capazes de perceber o ambiente, decidir localmente e executar ações físicas coordenadas [Ray 2016], [Afanasyev et al. 2019], [Vermesan et al. 2020].

Apesar dos avanços na integração entre robótica e conectividade, a dependência de arquiteturas centralizadas ainda impõe desafios na prática. O tráfego contínuo de informações para servidores externos introduz atrasos na rede e eleva o risco de falhas de conectividade, o que compromete a confiabilidade e rapidez de autonomia do sistema [Ray 2016], [Vermesan et al. 2020]. À vista disso, Afanasyev et al. (2019) demonstram que esses gargalos de comunicação degradam consideravelmente o comportamento de frotas robóticas distribuídas, visto que exigem muita sincronia em operações tempo-sensíveis.

Para contornar esses problemas, o uso do Edge AI atua de forma a reduzir o sobrecarregamento da nuvem. Executar os cálculos próximos à fonte de dados melhora o tempo de resposta e a previsibilidade [Ferreira et al. 2022]. O contraponto, no entanto, são os desafios práticos de implementação: transferir modelos de visão computacional para microcontroladores exige equilibrar limites de memória SRAM disponível, processamento de CPU e consumo de bateria [Cruz et al. 2018], [Zhang et al. 2018].

Como forma de verificar a aplicação de Edge AI, a literatura demonstra que a execução de redes neurais convolucionais (CNN) em plataformas embarcadas é viável quando arquiteturas leves e estratégias específicas de adaptação são adotadas [Cruz et al. 2018], [Zhang et al. 2018], [Frost et al. 2025]. Entretanto, esses estudos evidenciam compromissos inevitáveis entre complexidade do modelo, qualidade da classificação e estabilidade de execução, especialmente em dispositivos com recursos severamente limitados. Embora esses estudos avaliem os modelos em termos de acurácia e tamanho, o impacto sistêmico da inferência no ciclo de decisão robótica, em coexistência com pilhas de comunicação sob restrições reais de *hardware*, permanece menos explorado na literatura [Simoens et al. 2018], [Afanasyev et al. 2019]. Diferentemente de aplicações genéricas de *mobile edge*, nas quais erros de inferência resultam em degradação de serviço digital, em sistemas IoRT tais erros produzem consequências físicas imediatas: um manipulador posiciona objetos incorretamente e a cadeia de coordenação entre agentes é comprometida [Vermesan et al. 2020].

Faz-se necessário, portanto, investigar como modelos de visão computacional se comportam quando efetivamente embarcados em dispositivos restritos e inseridos em um ciclo de decisão robótica, considerando não apenas acurácia, mas também estabilidade

de memória e latência. Comunicação, colaboração multiagente e inteligência embarcada são tratadas, frequentemente, como dimensões independentes na literatura. Essa separação dificulta a compreensão de ecossistemas IoRT reais, nos quais a sobrecarga da pilha de rede (como Wi-Fi/MQTT) afeta diretamente os recursos disponíveis para a inferência e vice-versa [Afanasyev et al. 2019]. Avaliar modelos de *Machine Learning* (ML) isoladamente é insuficiente; o efeito concorrente deve ser observado no sistema completo [Vermesan et al. 2020]. Ambientes simulados viabilizam essa análise com reprodutibilidade, isolando gargalos sistêmicos antes do dispêndio físico em robôs reais [Bogaerts et al. 2020].

Nesse contexto, este trabalho tem como objetivo avaliar a aplicação de Edge AI em um microcontrolador ESP32-S3 inserido em um cenário IoRT simulado no CoppeliaSim. O sistema integra manipuladores estacionários e um robô móvel autônomo, coordenados de forma descentralizada via protocolo *Message Queuing Telemetry Transport* (MQTT) (*publish/subscribe*). A contribuição principal reside na avaliação empírica do funcionamento concorrente entre inferência e pilhas de comunicação no mesmo microcontrolador, revelando discrepâncias entre estimativas de memória obtidas em ambiente de desenvolvimento e os requisitos reais medidos no dispositivo embarcado. Essa abordagem evidencia restrições sistêmicas que inviabilizam na prática modelos aparentemente compatíveis.

2. Referencial Teórico

2.1. Percepção visual e decisão em sistemas robóticos IoRT

Em sistemas robóticos conectados no paradigma da IoRT, a percepção visual não atua como módulo periférico, mas como componente estrutural do ciclo percepção–decisão–atuação, sustentando decisões que se materializam em ações físicas no ambiente [Simoens et al. 2018], [Vermesan et al. 2020]. Diferentemente de aplicações informacionais, nas quais erros perceptivos podem ser corrigidos em etapas posteriores, em sistemas robóticos tais erros tendem a produzir consequências físicas imediatas, afetando segurança e continuidade operacional [Afanasyev et al. 2019], [Kabir et al. 2023]. Em cenários distribuídos, a interpretação visual influencia não apenas a ação individual de um agente, mas a coordenação coletiva de tarefas entre múltiplos robôs [Kabir et al. 2023], [Vermesan et al. 2020].

A literatura indica que, em ecossistemas IoRT, falhas na percepção visual propagam-se ao longo do sistema distribuído, comprometendo a coordenação e a estabilidade operacional do conjunto de agentes [Simoens et al. 2018], [Afanasyev et al. 2019]. Decisões incorretas, atrasos na inferência ou inconsistências na interpretação do ambiente impactam o comportamento global do sistema, uma vez que a percepção constitui o principal ponto de entrada de informação para os mecanismos de decisão e controle robótico [Vermesan et al. 2020], [Ray 2016]. Esse caráter sistêmico diferencia a percepção visual em IoRT de aplicações tradicionais de visão computacional, nas quais modelos são frequentemente avaliados de forma isolada, sem considerar sua inserção em ciclos fechados de controle e atuação física [Simoens et al. 2018].

Nesse cenário, a tomada de decisão local assume papel central, pois a dependência exclusiva de processamento remoto pode introduzir latências incompatíveis com a dinâmica de sistemas robóticos distribuídos [Ray 2016], [Vermesan et al. 2020]. A variabilidade

da rede e eventuais falhas de conectividade afetam diretamente a confiabilidade do sistema [Afanasyev et al. 2019]. Assim, a inteligência embarcada emerge não apenas como estratégia de otimização, mas como requisito operacional para garantir consistência e previsibilidade do comportamento robótico [Vermesan et al. 2020].

2.2. Edge AI sob restrições embarcadas

O paradigma de Edge AI consiste na execução de inferência diretamente nos dispositivos de borda, reduzindo a dependência de processamento remoto e favorecendo respostas com menor variabilidade temporal [Ferreira et al. 2022], [Hoffpauir et al. 2023]. Dispositivos embarcados operam com capacidade computacional limitada, ausência de GPU, memória volátil restrita e severas limitações energéticas [Ferreira et al. 2022], [Ray 2016]. A ausência de aceleradores dedicados torna o custo computacional do modelo determinante para a temporalidade de resposta [Zhang et al. 2018]. A memória disponível constitui gargalo crítico: além dos pesos do modelo, a inferência exige memória para *buffers* intermediários e estruturas auxiliares, o que pode inviabilizar a execução mesmo de modelos aparentemente compactos quando avaliados fora do ambiente embarcado [Cruz et al. 2018], [Zhang et al. 2018]. O *hardware* não pode ser tratado como detalhe experimental, mas como parte integrante do problema científico [Afanasyev et al. 2019].

Para contornar essas limitações, Musa et al. (2025) categorizam as principais estratégias de redução: *network pruning* (remoção de conexões redundantes), *knowledge distillation* (transferência de conhecimento de um modelo complexo para um modelo simplificado), *architecture design* (projeto de redes originalmente compactas) e quantização numérica [Musa et al. 2025]. O *pruning* induz esparsidade na matriz de pesos, mas sua aceleração efetiva em microcontroladores genéricos requer *hardware* especializado, o qual nem sempre está disponível em plataformas de prateleira. A destilação de conhecimento, por sua vez, exige o treinamento e a orquestração de um *pipeline* complexo com um modelo “professor” supervisionando o aprendizado de um modelo “aluno”.

Além das abordagens baseadas em CNNs, a literatura registra topologias alternativas como as Redes Neurais Sem Peso (*Weightless Neural Networks* — WNNs), que substituem operações de multiplicação-acumulação (MAC) por consultas a tabelas em memória RAM, alcançando inferência com latência reduzida e menor dissipação térmica [Frost et al. 2025]. Contudo, WNNs carecem de suporte maduro em *toolchains* comerciais e não dispõem de conversores equivalentes ao *pipeline* consolidado do TensorFlow Lite Micro, o que limita sua adoção em dispositivos genéricos sem FPGA dedicado.

Diante dessas considerações, a estratégia adotada neste trabalho baseia-se na quantização inteira estática (INT8), que restringe a representação numérica dos tensores de 32 *bits* em ponto flutuante para 8 *bits* inteiros. Essa abordagem aproveita as extensões vetoriais para instrução única e múltiplos dados (SIMD) disponíveis nativamente na arquitetura Xtensa LX7 do ESP32, permitindo ganhos de eficiência em memória e processamento com degradação tolerável de acurácia para a tarefa robótica. As demais estratégias (*pruning*, destilação e WNNs) permanecem como direções de investigação futura.

2.3. CNNs em dispositivos restritos: adaptação, embarque e avaliação integrada

Redes neurais convolucionais são referência em visão computacional robótica, mas seu desempenho está historicamente associado a arquiteturas concebidas para ambientes com

recursos abundantes [Cruz et al. 2018], [Zhang et al. 2018]. Em microcontroladores, nem toda CNN bem treinada é necessariamente embarcável: o modelo pode exceder limites de memória ou apresentar comportamento instável durante a execução [Cruz et al. 2018]. Estratégias de adaptação são necessárias, mas não garantem viabilidade, pois o limite final é imposto pelo *hardware* e pela necessidade de operação estável no sistema integrado [Vermesan et al. 2020].

Dispositivos da família ESP representam um caso limite relevante, combinando baixo custo, integração típica em sistemas IoT e restrições rígidas de memória [Ferreira et al. 2022]. Nessa classe de *hardware*, o embarque envolve critérios de armazenamento, uso de memória intermediária e estabilidade de execução que podem inviabilizar modelos treinados com sucesso, e a inviabilidade de embarque constitui um resultado científico relevante [Cruz et al. 2018]. Dado que os efeitos mais relevantes emergem da interação entre inferência embarcada e comportamento robótico, ambientes simulados viabilizam a avaliação com reprodutibilidade experimental, permitindo observar limitações antes da implantação em sistemas físicos [Bogaerts et al. 2020], [Takaya et al. 2016]. Diferentemente de trabalhos que avaliam modelos de forma isolada, este estudo insere a inferência num ciclo completo de percepção, decisão e atuação sob restrições reais de memória compartilhada com pilhas de comunicação Wi-Fi e MQTT no mesmo microcontrolador.

3. Metodologia

3.1. Desenho da pesquisa e abordagem metodológica

O trabalho segue abordagem experimental e quantitativa. A inferência é tratada como etapa indissociável do ciclo percepção–decisão–atuação, e o foco recai sobre a avaliação de modelos existentes sob restrições reais de *hardware*, não sobre a proposição de novas arquiteturas. Em sistemas IoRT, latências ou instabilidades na inferência impactam diretamente a atuação física dos agentes; por isso, além da acurácia, avalia-se a viabilidade operacional e a estabilidade ao longo de múltiplos ciclos. A investigação foi organizada em três fases: treinamento, adaptação para TinyML e integração com o ambiente simulado. Cada fase possui critérios objetivos de validação (compatibilidade com o dispositivo, estabilidade da execução), estabelecendo uma base consistente para comparar os compromissos entre desempenho perceptivo e eficiência embarcada.

3.2. Definição da tarefa e do cenário IoRT

A tarefa consistiu em classificar três cores (Azul, Vermelho e *Background*), integrando a percepção visual ao controle robótico IoRT. Todo cenário robótico mecânico (atuadores, cinemática, esteiras e visão base) executou em simulação no CoppeliaSim v4.10.0 contendo três robôs: dois estacionários (Franka Panda e UR10) e um móvel (KUKA youBot). Esses robôs virtuais se comunicaram via protocolo MQTT (*broker* Mosquitto) com um ESP32-S3 físico, onde a inferência Edge AI foi executada paralelamente à simulação. Essa separação é importante: robôs estacionários são, em geral, conectados à rede elétrica por cabos, o que os mantém fixos e permite operação com menor restrição energética, enquanto robôs móveis como o youBot impõem restrições de bateria, exigindo a delegação dos cálculos pesados para a borda (ESP32). A classe *Background* serviu de controle para os falsos positivos.

3.3. Treinamento dos modelos

Nesta etapa, foram treinadas três arquiteturas distintas de redes neurais convolucionais: MobileNetV2, MobileNetV3 Small e uma CNN autoral simplificada. As arquiteturas da família MobileNet foram selecionadas por representarem uma solução eficiente para o dilema entre precisão e custo computacional em dispositivos móveis [Souza 2025], fundamentando-se no uso de convoluções separáveis em profundidade (*depthwise separable convolutions*) [Oliveira and Bianchi 2019]. Essa técnica permite reduzir significativamente o número de parâmetros e operações matemáticas, viabilizando a execução em *hardware* com recursos limitados sem comprometer substancialmente a capacidade representacional [Souza 2025], [Oliveira and Bianchi 2019]. Em contrapartida, a CNN autoral foi estruturada com apenas quatro camadas e construída empiricamente sem arquiteturas prévias, para servir como modelo base (*baseline*). Essa comparação possibilita analisar se a complexidade arquitetural das redes pré-existentz traz ganhos significativos na tarefa de classificação definida em relação a um modelo treinado do zero, que aposta na especialização de uma topologia rasa em vez de alta capacidade de representação.

Cada arquitetura apresenta compromissos distintos entre expressividade, custo computacional e uso de memória. Como a viabilidade de embarque não pode ser assumida a priori, optou-se por treinar todas as arquiteturas sem restringir previamente àquelas sabidamente executáveis, obtendo modelos funcionais para a etapa posterior de adaptação.

O treinamento supervisionado utilizou um *dataset* de 1.656 imagens de 32×32 px RGB, particionado em conjuntos de treinamento (70%), validação (15%) e teste (15%), sendo conduzido integralmente de forma *offline* na nuvem. No dispositivo, o modelo executa apenas a inferência estática (*TinyML*), sem atualizações de pesos em tempo de execução. Essa escolha não é uma simplificação, mas um requisito do determinismo exigido pela IoRT [Afanasyev et al. 2019]: o treinamento dinâmico no dispositivo (*on-line learning*) fragmentaria gradualmente a SRAM disponível e introduziria latências variáveis de *backpropagation*, comprometendo a previsibilidade temporal necessária para a coordenação dos agentes robóticos.

Para as MobileNets, o *fine-tuning* ocorreu em duas fases: inicialmente com as camadas convolucionais congeladas para ajuste apenas da nova camada densa final, preservando os filtros pré-treinados no *ImageNet*; e em seguida com as últimas camadas convolucionais liberadas para refinamento com taxa de aprendizado reduzida ($2e-5$), mitigando saltos no gradiente descendente na topologia profunda. A CNN autoral foi treinada do zero com arquitetura minimalista: duas camadas convolucionais (32 e 64 filtros, ativação ReLU), cada uma seguida de *MaxPooling* (2×2), achatamento (*Flatten*) e duas camadas densas (64 e 3 neurônios). Registraram-se acurácia, precisão macro, revocação e F1-Score no conjunto de validação para cada arquitetura.

3.4. Conversão e adaptação para TinyML

Os modelos treinados foram convertidos do formato Keras para TensorFlow Lite, gerando variantes com diferentes estratégias de quantização (normal, dinâmica e inteira). Microcontroladores da família ESP não executam modelos Keras diretamente; a conversão reduz o tamanho e o custo computacional, tornando a inferência local possível.

A conversão utilizou o *pipeline* oficial do TensorFlow. Cada variante foi organizada conforme o fluxo do TensorFlow Lite Micro e incorporada ao *firmware* como *array C*. A

redução de tamanho, contudo, não garante viabilidade de execução: conforme demonstrado na Seção 4.3, estimativas de arena obtidas em *desktop* podem subestimar os requisitos reais no *hardware*.

Nesta etapa, as variantes quantizadas INT8 foram configuradas para embarque no ESP32-S3. A viabilidade transcende o tamanho bruto do arquivo na memória *Flash*: fatores como alocação sequencial da *Tensor Arena* (uma região de memória contígua alocada estaticamente para gerenciar todos os tensores de entrada, saída e resultados intermediários do modelo) e a convivência com as pilhas Wi-Fi/MQTT no microcontrolador físico ditam o sucesso da integração sistêmica. Modelos complexos exigem que a *Tensor Arena* e o próprio armazenamento do *firmware* sejam adequados via redefinição customizada das tabelas de partições e manipulação do *heap* para viabilizar os testes.

O processo de embarque foi realizado mediante a conversão dos modelos para o formato TensorFlow Lite e sua posterior incorporação ao firmware do ESP32-S3 (S3-WROOM-1-N16R8), utilizando o fluxo de desenvolvimento do TensorFlow Lite Micro. As especificações técnicas da plataforma, que fundamentam a viabilidade das inferências locais, estão detalhadas na Tabela 1. Cada modelo foi transposto para um array em linguagem C e integrado a um projeto em C++ no ambiente ESP-IDF. A escolha desta plataforma justifica-se pela CPU Xtensa® LX7, que possui extensões vetoriais para aceleração de IA, e pela disponibilidade de 512 KB de SRAM interna, permitindo que a arena de memória fosse alocada integralmente na RAM de alta velocidade para minimizar latências. Durante os testes práticos, mensuraram-se o sucesso na alocação da arena, a inicialização do interpretador livre de erros e a estrita estabilidade sistêmica sob múltiplos ciclos de percepção, objetivando validar a capacidade de o dispositivo operar sem travamentos ou reinicializações indesejadas.

Tabela 1. Especificações do *hardware* utilizado no sistema embarcado

Componente	Especificação	Capacidade
Microcontrolador	ESP32-S3-WROOM-1-N16R8	Dual-Core 240 MHz
CPU	Xtensa® 32-bit LX7	Aceleração vetorial de IA
SRAM Interna	Memória estática de alta velocidade	512 KB
Memória Flash	Armazenamento persistente (NVM)	16 MB
PSRAM	Memória externa (opcional)	8 MB
Conectividade	Wi-Fi 802.11 b/g/n	2,4 GHz

Cada variante embarcada foi submetida a uma etapa de validação empírica visando a sua classificação dicotômica (viável ou inviável), fundamentada em critérios exatos de execução. Os critérios de aceitação englobaram não apenas as restrições estáticas de *storage*, mas também a capacidade de alocação contígua na arena de tensores, a imunidade a pânicos de *kernel* (*kernel panics*) ou estouros de *watchdog* e, fundamentalmente, a fidelidade perceptiva sistêmica pós-embarque. O detalhamento do cumprimento destes critérios de aceitação por cada arquitetura e as restrições físicas limitantes no microcontrolador são discutidos na Seção 4.3.

3.5. Integração com o ambiente simulado e ciclo decisão e atuação

A simulação no CoppeliaSim [Bogaerts et al. 2020], [Takaya et al. 2016] foi conectada à inferência embarcada no ESP32, fechando o ciclo percepção–decisão–atuação. A coordenação entre os agentes é inteiramente orientada a eventos via MQTT (*publish/subscribe*) com *Quality of Service* (QoS) 0, o que elimina bloqueios síncronos: caso a conexão Wi-Fi do youBot apresente atraso, os demais agentes continuam operando de forma independente, pois cada um subscreve e publica tópicos distintos.

O fluxo operacional utiliza 8 tópicos MQTT: o sensor infravermelho detecta a caixa (*/sensor*); o Franka Panda coleta e deposita no youBot (*/franka*); este transporta até a zona do UR10 (*/entregador*); o UR10 posiciona a caixa diante da câmera. Os *bytes* da imagem são enviados ao ESP32 (tópico */esp/classificar*) e o resultado retornado (*/esp/resultado*) determina o lado da estante em que o UR10 deposita a caixa.

O objetivo não é apenas verificar se a inferência funciona isoladamente, mas observar seus efeitos no sistema completo: atrasos acumulados, dependência temporal entre módulos e impacto nas decisões do manipulador. O rótulo retornado define o posicionamento na estante: Azul à esquerda, Vermelho à direita e a classe *Background* impede movimentos quando não há caixa válida.

O objetivo primário da integração foi observar o fluxo sistêmico completo: a captura simulada gera e transmite a carga (*payload*), a inferência ocorre no dispositivo eletrônico isolado, e a decisão é refletida de volta no mundo simulado, evidenciando as dependências de roteamento e memória.

4. Resultados

4.1. Treinamento perceptivo dos modelos

Os resultados do treinamento supervisionado indicam que os modelos MobileNetV2, MobileNetV3 Small e a CNN autoral foram capazes de extrair padrões visuais discriminativos robustos para a classificação das classes Azul, Vermelho e *Background*. As matrizes de confusão na Figura 1 e as métricas apresentadas na Tabela 2 mostram que as arquiteturas MobileNet obtiveram maior consistência na separação entre as classes Azul e Vermelho, enquanto a CNN autoral apresentou maior incidência de confusão envolvendo a classe *Background*. Ainda assim, todas as arquiteturas atingiram valores próximos de acurácia e métricas macro equilibradas, evidenciando comportamento perceptivo estável.

Tabela 2. Comparação de métricas de desempenho entre os modelos.

Modelos	Acurácia	Macro Precisão	Macro Revocação	Macro F1-Score
MobileNet V2	0.9518	0.9509	0.9509	0.9509
MobileNet V3 Small	0.9357	0.9352	0.9353	0.9352
CNN autoral	0.9317	0.9315	0.9317	0.9316

4.2. Impacto da conversão e quantização nos modelos

A Figura 2 compara o tamanho dos modelos originais em formato .keras com as versões convertidas para TensorFlow Lite em três variantes de quantização, normal, dinâmica e

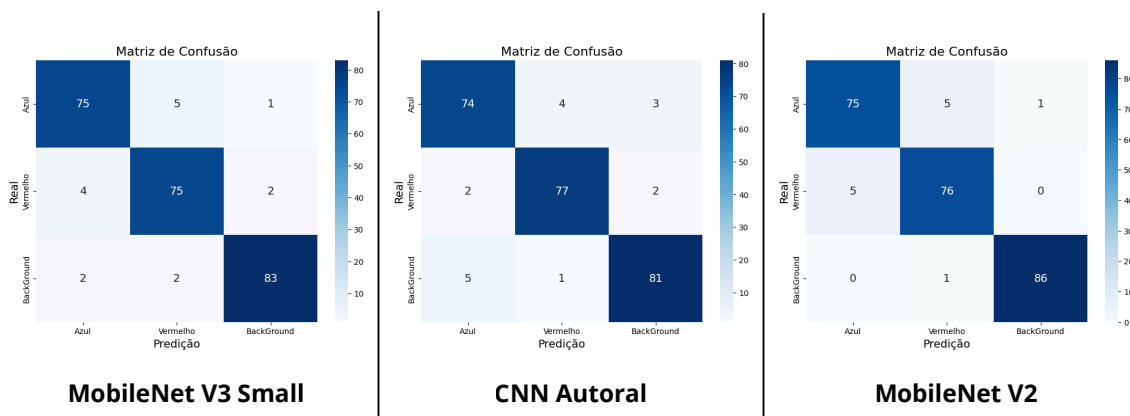


Figura 1. Matrizes de confusão dos modelos avaliados.

inteira, incluindo também a acurácia associada a cada configuração. Os arquivos .keras representam os modelos originais utilizados como base para gerar os arquivos .flite e são apresentados no gráfico como referência para evidenciar a redução obtida após a conversão e a quantização. Entre as variantes, a quantização inteira corresponde à estratégia de conversão mais rigorosa dos tipos de dados, a quantização normal representa a opção mais conservadora, e a quantização dinâmica atua como uma alternativa intermediária, buscando combinar características das duas abordagens. O conjunto de resultados evidencia a importância da quantização para reduzir o tamanho dos modelos e aproximá-los das restrições típicas de execução em microcontroladores, mantendo, ao mesmo tempo, a informação sobre acurácia para avaliar o impacto dessa redução na capacidade perceptiva.

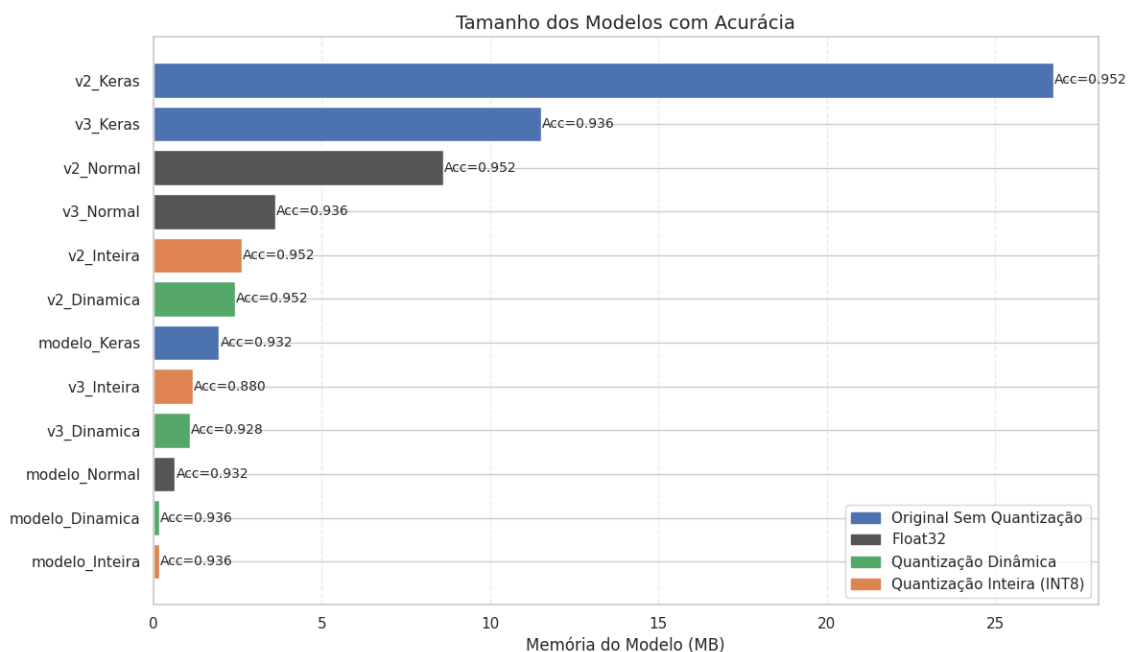


Figura 2. Tamanho e acurácia dos modelos nas variantes de quantização.

4.3. Viabilidade de embarque no ESP32

A avaliação experimental e a instrumentação do firmware, cujos dados de desempenho e compressão estão sintetizados na Tabela 3, evidenciaram que a quantização inteira (INT8)

constitui o principal pilar de viabilidade técnica para o Edge AI no ecossistema IoRT. As versões quantizadas alcançaram reduções de até 91,42 %, permitindo que a CNN autoral ocupasse apenas 0,16 MB de memória *Flash*. Contudo, a MobileNet V3 Small sofreu degradação acentuada para 87,95 %, sugerindo menor resiliência dessa arquitetura à compressão agressiva.

Conforme ilustrado na Figura 3, a análise da *Tensor Arena* em *desktop* sugeriu que todas as variantes operariam abaixo do limite da SRAM. No entanto, a Tabela 3 revela uma discrepância: no ESP32 real, a MobileNetV2 exigiu 545,4 KB de arena (204,5 % mais que o estimado), causaria o travamento imediato sem memória auxiliar. Essa divergência ocorre por diferença nativa nos interpretadores das plataformas: enquanto o cálculo em *desktop* baseou-se no repositório `tensorflow/tflite-micro` versão 2.6.5 (estável x86), a execução no microcontrolador utilizou a dependência otimizada e mantida pela fornecedora do *hardware* (`espressif/esp-tflite-micro`). Diferenças nos códigos *backend*, gerenciadores de memória interna e *kernels* otimizados geram *overhead* não refletido durante a estimativa teórica.

Como o ESP32 dispõe de ≈ 377 KB de SRAM interna disponível após a inicialização das pilhas Wi-Fi e MQTT, a arena de 545 KB da MobileNetV2 causaria o travamento imediato do *firmware*. Para viabilizar os ensaios, a RAM externa (PSRAM de 8 MB) precisou ser ativada no *hardware* a 80 MHz em modo *Octal*, redirecionando a alocação da arena criticamente por meio da instrução `C heap_caps_malloc` acionada com a *flag* `MALLOC_CAP_SPIRAM`.

Contudo, mesmo contornando o limite de armazenamento na PSRAM e rodando sem estouro de *heap*, ambas as MobileNet produziram classificações majoritariamente incorretas no embarcado. Três fatores justificam esse comportamento na placa: (i) as diferenças entre a biblioteca TFLite-Micro oficial empregada no *notebook* e a dependência modificada da Espressif no ESP32 originou inconsistências de matrizes de tensores nas arquiteturas elásticas; (ii) as MobileNets otimizadas para imagens com muitos pixels (ex: 224×224) perderam características cruciais de convolução utilizando entradas de 32×32 px; e (iii) a etapa de quantização para ponteiros inteiros de *8-bits* degradou os mapas de pesos de canais na arquitetura mais profunda. Isso demonstra de forma prática que a conversão estrutural do TFLite não engloba ou reflete imediatamente uma consistência dos mapas de características nos módulos móveis do ciclo real.

Apenas a CNN autoral operou corretamente na SRAM interna (38,4 KB), com folga de ≈ 339 KB para as pilhas de comunicação (Wi-Fi/MQTT), garantindo um ciclo percepção–decisão determinístico. Essa folga é crítica: a pilha Wi-Fi do ESP-IDF consome tipicamente entre 40 e 70 KB de SRAM, e a comunicação MQTT adiciona um *buffer* de 10 KB para cada conexão ativa. Em cenários que exigissem o protocolo *Transport Layer Security* (TLS) para segurança criptográfica do tráfego MQTT, o consumo adicional estimado seria de ≈ 40 KB, reduzindo a folga para ≈ 299 KB — ainda viável para a CNN autoral, mas provavelmente crítico para as MobileNets.

4.4. Resultados da integração com o sistema robótico simulado

Com a integração ao ambiente simulado, o sistema validou o ciclo completo utilizando a CNN autoral no ESP32. Realizaram-se testes em três infraestruturas para o *broker* MQTT: Local, Borda (Intel Edison) e Nuvem (Amazon EC2). Foram totalizadas 180 operações com

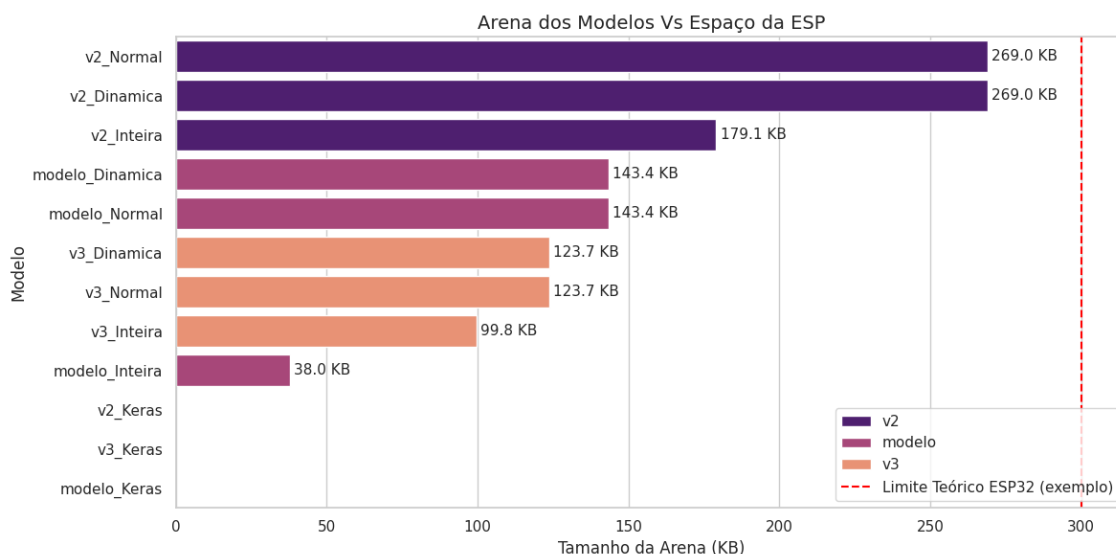


Figura 3. Consumo de *Tensor Arena* estimado em notebook vs. limite do ESP32-S3.

Tabela 3. Comparação entre modelos Keras e versões quantizadas INT8

Modelo	Tipo	Acur.	Δ Acur.	Flash (MB)	Red. (%)	Arena NB (KB)	Arena ESP (KB)
MobileNet V2	Keras	0,9518	–	26,72	–	–	–
	INT8	0,9518	0,00%	2,62	90,18	179,1	545,4
MobileNet V3 S.	Keras	0,9357	–	11,51	–	–	–
	INT8	0,8795	–5,62%	1,17	89,79	99,8	166,3
CNN autoral	Keras	0,9317	–	1,95	–	–	–
	INT8	0,9357	+0,40%	0,16	91,42	37,9	38,4

sucesso, das quais 60 operações (1 teste com 20 execuções para cada ambiente) tiveram seus *logs* salvos e tabulados para o cálculo oficial de métricas temporais. A Tabela 4 consolida os indicadores da rede extraídos dessas 60 ocorrências oficiais. Nesse contexto, a aferição do desempenho sistêmico concentrou-se no tópico `/esp/classificar`, que consolida o ciclo crítico: ele mensura o tempo transcorrido desde a inferência no microcontrolador até o respectivo resultado de decisão chegar e acionar a lógica de armazenamento no manipulador UR10.

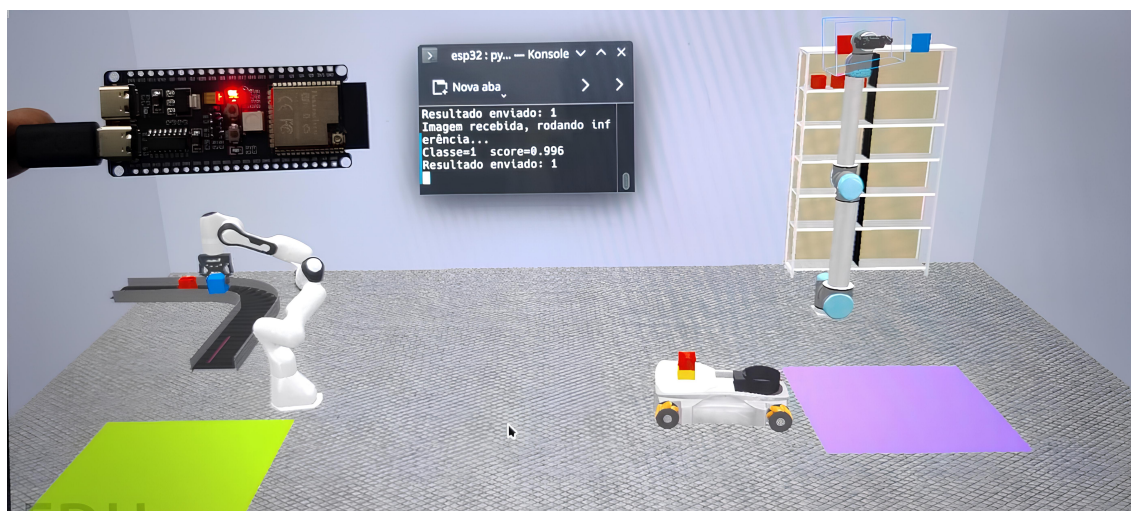
Conforme evidenciado na Figura 4, observa-se a operação simultânea do cenário (atuadores e cinemática geridos no simulador) e a rede real, atestando os tempos completos do ciclo (embora restassem latências médias globais indisponíveis [ND] nos extratos do Edison, seu desvio padrão temporal aferiu-se em ≈ 100 ms). O *jitter* com a inferência ativa estabilizou globalmente na faixa de 10 a 120 ms; surpreendentemente inferior aos picos aleatórios de até 480 ms aferidos no mesmo sistema operando unicamente como rede sem predição local. Esse efeito regulatório (*throttling*) ocorre porque o processamento contínuo fixo imposto pelo microcontrolador (da ordem de 53,87 ms por ciclo inferencial) amortece rajadas concorrentes de eventos provenientes da simulação, mitigando a imprevisibilidade advinda das plataformas remotas.

Tabela 4. Métricas oficiais e de integração (20 operações por ambiente, CNN Autoral)

Ambiente	Tópico	Acertos	Tempo M. Inf.	Lat. Mediana	Jitter
Local	/esp/classificar	20/20	53,80 ms	164,30 ms	30,14 ms
Edison	/esp/classificar	20/20	53,87 ms	258,88 ms	31,59 ms
AWS	/esp/classificar	20/20	53,78 ms	472,19 ms	71,55 ms

Os tempos de inferência computados isoladamente no ESP32 foram de 53,87 ms para a CNN autoral, 117,38 ms para a MobileNetV3 Small e 333,70 ms para a MobileNetV2. Em sistemas de automação com ciclos de controle na ordem de dezenas de milissegundos, latências superiores a 100 ms podem comprometer a sincronização entre módulos e a reatividade do sistema. Além do impacto temporal, há implicações energéticas relevantes: em modo ocioso (*Modem Sleep*), o ESP32 consome aproximadamente 20 mA, enquanto durante a inferência ativa o consumo pode atingir cerca de 240 mA.

Embora medições de amperagem não tenham sido diretas, estima-se que a MobileNetV2 reteria o processador ativado com alto consumo energético por atrasos de 333,70 ms/ciclo, equivalendo a cerca de $6,2\times$ maior o tempo ativo demandado pela CNN autoral (53,87 ms). Tratando-se de braços e robôs operados exclusivamente por baterias (ex: KUKA youBot), esse tempo retido maior causa menor autonomia na execução contínua, denotando na prática que a seleção arquitetônica baliza diretamente não apenas a rede como a durabilidade do turno no ecossistema robótico.

**Figura 4. Inferência no ESP32 integrada ao cenário simulado via MQTT.**

5. Conclusão

Este trabalho avaliou a aplicação de Edge AI em um sistema IoRT simulado, revelando resultados que questionam premissas comuns sobre a implantação de CNNs em microcontroladores. A principal contribuição empírica é a demonstração de uma discrepância crítica entre estimativas de arena de memória obtidas em ambiente de desenvolvimento e os requisitos reais medidos no ESP32: a MobileNetV2 exigiu 204,5% mais arena que

o estimado (179,1 KB \rightarrow 545,4 KB), enquanto a CNN autoral manteve-se estável com apenas 38,4 KB (+1,3%). Mais significativamente, ambas as arquiteturas MobileNet produziram classificações sistematicamente incorretas no *hardware* real, mesmo após resolução dos problemas de memória via PSRAM, demonstrando que modelos com maior acurácia em *benchmark* podem falhar no ambiente embarcado.

A CNN autoral, com acurácia de 93,17% e *footprint* mínimo, atingiu 100% de sucesso operacional em 200 ciclos, com 53,87 ms de inferência e 99,6% de confiança. Sua folga de memória (\approx 339 KB livres na SRAM) garantiu a coexistência segura com as pilhas Wi-Fi/MQTT. A inferência local não degradou o desempenho do sistema e, inclusive, reduziu o *jitter* temporal em relação à configuração sem Edge AI.

Os resultados evidenciam que a escolha de modelos para Edge AI em dispositivos severamente restritos não pode basear-se exclusivamente em métricas de acurácia isoladas ou estimativas de *profiling* em *desktop*. Em ecossistemas IoRT práticos, nos quais robôs móveis operam sob rígidas restrições de bateria, o tempo prolongado de inferência traduz-se diretamente em consumo energético exacerbado. Modelos computacionalmente densos não apenas elevam a latência e arriscam a fluidez de braços mecânicos, como também diminuem a autonomia de carga da frota móvel. Portanto, torna-se indispensável conduzir uma verificação empírica no *hardware* alvo, avaliando a coexistência transparente do modelo com as vitais pilhas de comunicação e aferindo a estabilidade determinística ao longo de múltiplas inferências contínuas. Essa prova de conceito certifica que a adoção da inteligência local atue efetivamente como viabilizador sistêmico operacional, em vez de se consolidar como um gargalo energético ou temporal no chão de fábrica.

Como limitações, o estudo avaliou apenas quantização como técnica de adaptação, operou em cenário simulado e não mensurou consumo energético. Como trabalhos futuros, indicam-se: (i) avaliação de *pruning* e destilação de conhecimento [Musa et al. 2025]; (ii) transição para robôs físicos; (iii) exploração de modelos de linguagem compactos para borda [Lamaakal et al. 2025]; e (iv) análise do impacto de TLS sobre os recursos do ESP32. O código-fonte, os modelos treinados e o *dataset* estão disponíveis em https://github.com/Keltonmd/EdgeAI_SBRC.

Agradecimentos

Os autores agradecem ao Instituto Federal de Educação, Ciência e Tecnologia do Norte de Minas Gerais (IFNMG) — Campus Januária pelo apoio e contribuição, em especial ao Programa Institucional de Bolsas de Iniciação em Desenvolvimento Tecnológico e Inovação (PIBITI).

Referências

- Afanasyev, I. et al. (2019). Towards the internet of robotic things: Analysis, architecture, components and challenges.
- Bogaerts, B. et al. (2020). Connecting the coppeliasim robotics simulator to virtual reality. *SoftwareX*, 11:100426.
- Cruz, N., Lobos-Tsunekawa, K., and Ruiz-del Solar, J. (2018). Using convolutional neural networks in robots with limited computational resources: Detecting nao robots while playing soccer. In *RoboCup 2017: Robot World Cup XXI*, pages 19–30. Springer.

- Ferreira, L. C. B. C. et al. (2022). Edge computing and microservices middleware for home energy management systems. *IEEE Access*, 10:109663–109676.
- Frost, A., Jean, G., and Ailyn, D. (2025). Lightweight CNN models for edge deployment in low-resource environments.
- Hoffpauir, K. et al. (2023). A survey on edge intelligence and lightweight machine learning support for future applications and services. *J. Sens. Actuator Netw.*
- Kabir, H., Tham, M.-L., and Chang, Y. C. (2023). Internet of robotic things for mobile robots: Concepts, technologies, challenges, applications, and future directions. *Digital Communications and Networks*, 9(6):1265–1290.
- Lamaakal, A. et al. (2025). Tiny language models for automation and control: Overview, potential applications, and future research directions.
- Musa, A., Kakudi, H. A., Hassan, M., Hamada, M., Umar, U., and Salisu, M. L. (2025). Lightweight deep learning models for edge devices—a survey. *Int. J. Comput. Inf. Syst. Ind. Manag. Appl.*, 17:18.
- Oliveira, J. H. R. d. and Bianchi, R. A. d. C. (2019). Estudo comparativo de arquiteturas de redes neurais em robôs humanoides. In *Anais do IX Simpósio de Iniciação Científica, Didática e de Ações Sociais da FEI*, São Bernardo do Campo, SP, Brasil. Centro Universitário FEI.
- Ray, P. P. (2016). Internet of robotic things: Concept, technologies, and challenges. *IEEE Access*, 4:9489–9500.
- Simoens, P., Dragone, M., and Saffiotti, A. (2018). The internet of robotic things: A review of concept, added value and applications. *International Journal of Advanced Robotic Systems*, 15(1):172988141875942.
- Souza, V. A. (2025). Classificação automática de imagens utilizando redes neurais convolucionais mobilenet: um estudo de caso com reconhecimento de felinos. In *Anais do IV Congresso Brasileiro de Ciência e Saberes Multidisciplinares*, Volta Redonda, RJ, Brasil. UniFOA. Encontro de Extensão Universitária do UniFOA.
- Takaya, K., Asai, T., Kroumov, V., and Smarandache, F. (2016). Simulation environment for mobile robots testing using ROS and gazebo. In *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*. IEEE.
- Vermesan, O. et al. (2020). Internet of robotic things intelligent connectivity and platforms. *Frontiers in Robotics and AI*, 7:104.
- Zhang, Y. et al. (2018). The implementation of cnn-based object detector on arm embedded platforms. In *IEEE 16th Intl Conference on Dependable, Autonomic and Secure Computing (DASC/PiCom/DataCom/CyberSciTech)*. IEEE.