Avaliação Multicritério de Detectores Leves de Pessoas para Edge Computing em SBCs: Desempenho, Latência e Eficiência Energética

Daniel Morais Cardozo, Sarita Mazzini Bruschi

[1]Instituto de Ciências Matemáticas e Computação (ICMC)
Universidade de São Paulo (USP)

danmoraisc@usp.br, sarita@icmc.usp.br

Abstract. *This work presents a comparative evaluation of lightweight person detectors for edge computing on the Jetson Nano and Raspberry Pi 4B platforms. The models YOLOv3-tiny, YOLOv5n, YOLOv7-tiny, YOLOv8n, and SSDLite320 MobileNetV3-Large were evaluated under a standardized protocol using ONNX Runtime on CPU. The analysis considered predictive performance, latency, FPS, memory usage, temperature, and energy consumption. The experimental results indicate that models from the YOLO family achieved better overall predictive performance, while YOLOv5n provided the best balance between detection quality, latency, and energy cost. In addition, the Raspberry Pi 4B showed higher energy efficiency across all evaluated models.*

Resumo. *Este trabalho apresenta uma avaliação comparativa de detectores leves de pessoas para computação de borda nas plataformas Jetson Nano e Raspberry Pi 4B. Foram avaliados os modelos YOLOv3-tiny, YOLOv5n, YOLOv7-tiny, YOLOv8n e SSDLite320 MobileNetV3-Large sob um protocolo padronizado, utilizando ONNX Runtime em CPU. A análise considerou desempenho preditivo, latência, FPS, uso de memória, temperatura e consumo energético. Os resultados dos experimentos indicam que os modelos da família YOLO obtiveram melhor desempenho preditivo geral, enquanto o YOLOv5n apresentou o melhor equilíbrio entre qualidade de detecção, latência e custo energético. Além disso, a Raspberry Pi 4B mostrou maior eficiência energética em todos os modelos avaliados.*

1. Introdução

A detecção de objetos é uma tarefa central da visão computacional, com aplicações em vigilância, robótica, mobilidade inteligente, cidades inteligentes e sistemas embarcados. Com a expansão da computação de borda (*edge computing*), esses modelos passaram a ser executados localmente, reduzindo latência, dependência da nuvem e tráfego de rede, além de favorecer requisitos de privacidade e autonomia operacional [Kolosov et al. 2022, Alqahtani et al. 2025]. Nesse contexto, a detecção de pessoas assume papel relevante em aplicações como monitoramento de ambientes, análise de fluxo, segurança e sistemas inteligentes distribuídos [Kim et al. 2019, Jain et al. 2024].

Entretanto, a implantação desses detectores em dispositivos de borda não é trivial. Plataformas compactas, como *single-board computers* (SBCs), operam sob restrições de

processamento, memória e energia, de modo que a escolha do modelo depende não apenas da qualidade preditiva, mas também de latência, uso de recursos e custo energético [Kolosov et al. 2022, Alqahtani et al. 2025]. Nesse cenário, arquiteturas de etapa única, como variantes leves da família YOLO e modelos baseados em *Single Shot Multibox Detector* (SSD), incluindo o SSDLite, surgem como candidatas adequadas por oferecerem melhor compromisso entre velocidade de inferência e qualidade de detecção do que detectores mais pesados [Cocks et al. 2023].

Apesar dos avanços recentes, comparações entre detectores em borda ainda são dificultadas por diferenças de plataforma, *runtime*, quantização, aceleradores, resolução de entrada e protocolo experimental. Assim, o mesmo modelo pode apresentar comportamentos distintos em diferentes dispositivos e pilhas de software, dificultando isolar o efeito da arquitetura sobre o desempenho final [Kolosov et al. 2022, Alqahtani et al. 2025]. Em trabalhos voltados à detecção de pessoas, essa heterogeneidade também limita comparações diretas e reprodutíveis [Kim et al. 2019, Jain et al. 2024].

Diante desse cenário, este trabalho apresenta uma avaliação multicritério de detectores leves de pessoas em duas plataformas amplamente utilizadas em computação de borda: Jetson Nano e Raspberry Pi 4B. São avaliados os modelos YOLOv3-tiny, YOLOv5n, YOLOv7-tiny, YOLOv8n e SSDLite320 MobileNetV3-Large sob um protocolo padronizado, com mesma resolução de entrada, pós-processamento uniforme, classe de interesse restrita a *person* e uso de um mesmo ambiente de execução baseado em *Open Neural Network Exchange* (ONNX) Runtime em *Central Processing Unit* (CPU). A análise considera conjuntamente métricas preditivas, computacionais e energéticas, permitindo discutir não apenas qual modelo detecta melhor, mas quais apresentam melhor compromisso prático entre qualidade, latência e eficiência energética em cenários reais de borda.

2. Trabalhos Relacionados

A literatura recente sobre visão computacional em *edge computing* converge na constatação de que a implantação de detectores em dispositivos com recursos limitados exige equilibrar, simultaneamente, desempenho preditivo, latência, uso de memória e eficiência energética. Nesse contexto, detectores de etapa única e variantes compactas, como modelos da família YOLO e arquiteturas baseadas em SSD/SSDLite com *backbones* leves, aparecem com frequência como alternativas mais adequadas à execução embarcada. Comparações mais recentes e conduzidas sob condições padronizadas mostram que famílias como YOLO, SSD e EfficientDet ocupam diferentes pontos de compromisso entre velocidade, precisão e custo computacional, enquanto estudos voltados a plataformas embarcadas reforçam que variantes leves dessas arquiteturas permanecem especialmente relevantes para cenários de borda [Yilmaz and Navruz 2025, Cocks et al. 2023].

Quando o foco recai especificamente sobre detecção de pessoas, a literatura também aponta tendências consistentes. Jain et al. compararam algoritmos de detecção de pessoas em SBCs, considerando métricas como FPS, uso de CPU, acurácia e latência, e mostraram que a escolha do detector em hardware restrito depende não apenas da qualidade preditiva, mas também da viabilidade de execução em tempo real [Jain et al. 2024]. Cocks et al., por sua vez, mostraram que o desempenho em detecção de pedestres

pode depender fortemente de quantização e do uso de aceleradores externos, evidenciando que resultados de FPS nem sempre são diretamente comparáveis entre estudos [Cocks et al. 2023].

No eixo de *benchmarks* em dispositivos de borda, diversos trabalhos ressaltam que a comparação entre modelos depende não apenas da arquitetura da rede, mas também do *runtime*, da quantização, do acelerador disponível, da resolução de entrada e do ecossistema de software. Kolosov et al. mostraram que a combinação entre modelo, dispositivo e *framework* pode alterar significativamente métricas como latência, eficiência energética e custo computacional relativo [Kolosov et al. 2022]. Em direção semelhante, Alqahtani et al. compararam YOLOv8, EfficientDet Lite e SSD em diferentes versões de Raspberry Pi e Jetson, concluindo que modelos com menor mAP (*mean Average Precision*) tendem a ser mais rápidos e energeticamente mais eficientes, enquanto variantes mais robustas do YOLO apresentam maior custo computacional e energético [Alqahtani et al. 2025]. Esses resultados reforçam que avaliações em inteligência artificial em borda (*edge AI*) precisam considerar conjuntamente qualidade preditiva, custo computacional e ambiente de execução. Em linha complementar, Azab et al. mostram que o desempenho em borda também pode depender fortemente de otimizações *hardware-aware*, como quantização, conversão para *runtimes* específicos e ajuste à plataforma-alvo, reforçando que a comparação entre detectores não pode ser dissociada do ambiente de implantação [Azab et al. 2026].

Apesar desses avanços, ainda há lacunas importantes para comparações mais controladas entre detectores leves em SBCs. Em muitos estudos, os experimentos combinam diferentes aceleradores, formatos de execução, estratégias de quantização, números de classes e condições de teste, o que dificulta isolar o efeito específico da arquitetura do modelo sobre o desempenho final [Kolosov et al. 2022, Alqahtani et al. 2025, Cocks et al. 2023]. Além disso, abordagens mais sistêmicas, como o ECORE, mostram que energia, latência e acurácia podem ser balanceadas dinamicamente entre pares modelo–dispositivo, reforçando a importância de distinguir, em benchmarks experimentais, o efeito da arquitetura do efeito da estratégia de implantação [Alqahtani et al. 2026]. Mesmo iniciativas mais sistemáticas, como o YOLOBench, apontam a necessidade de protocolos mais justos para analisar adequadamente o compromisso entre acurácia e latência em hardware embarcado [Lazarevich et al. 2023]. Nesse sentido, o presente trabalho se diferencia por adotar um protocolo padronizado, com resolução fixa, pós-processamento uniforme, avaliação restrita à classe *person*, uso de um mesmo ambiente de execução e análise conjunta de métricas preditivas, computacionais e energéticas em Jetson Nano e Raspberry Pi 4B.

3. Metodologia Experimental

Esta seção apresenta a metodologia experimental adotada neste estudo, descrevendo os modelos avaliados, as plataformas utilizadas, o protocolo de benchmark e os critérios empregados para garantir comparabilidade entre os experimentos.

3.1. Modelos avaliados

Esta subseção apresenta os modelos de detecção considerados neste estudo, destacando as principais características das arquiteturas selecionadas para a avaliação experimental em cenários de computação de borda.

(a) Jetson Nano. (b) Raspberry Pi 4B.

Figura 1. Plataformas utilizadas nos experimentos.

YOLO. A família YOLO (*You Only Look Once*) consolidou-se como referência em detecção de objetos em tempo real por formular a tarefa em uma única etapa, favorecendo alta velocidade de inferência em comparação com detectores de duas etapas [Redmon et al. 2016, Hemmatirad et al. 2022]. Ao longo de suas diferentes versões, a família evoluiu buscando equilibrar precisão, velocidade e custo computacional, tornando-se especialmente relevante para aplicações em *edge* [Terven et al. 2023, Hussain 2024, Ali and Zhang 2024]. Embora versões posteriores da família YOLO, como YOLOv9, YOLOv10, YOLO11 e YOLOV26, tenham sido propostas [Ultralytics 2024b, Wang et al. 2024, Ultralytics 2024a, Jocher and Qiu 2026], este estudo foi delimitado até o YOLOv8 por se tratar de uma geração consolidada e compatível com o ambiente experimental adotado.

SSDLite. O SSDLite é uma variante leve do *Single Shot Multibox Detector* (SSD), voltada a dispositivos móveis e embarcados. Ao empregar convoluções separáveis em profundidade, o modelo reduz custo computacional e número de parâmetros, preservando desempenho competitivo em detecção [Sandler et al. 2018]. Neste trabalho, foi adotado o SSDLite com *backbone* MobileNetV3-Large, por representar um ponto de operação equilibrado entre eficiência e qualidade preditiva. Assim, o modelo `ssdlite320_mobilenet_v3_large`, disponibilizado como implementação de referência no *torchvision*, foi incluído como representante de uma linha arquitetural distinta da família YOLO [Howard et al. 2019, Islam et al. 2023, Fan et al. 2018].

3.2. Plataformas e configuração experimental

A Figura 1 apresenta as plataformas utilizadas nos experimentos: Jetson Nano e Raspberry Pi 4B, ambas amplamente empregadas em aplicações de computação de borda.

Jetson Nano. A Jetson Nano utilizada, mostrada na Figura 1(a), possui CPU ARM Cortex-A57 quad-core de 1,43 GHz, processador gráfico (*Graphics Processing Unit* – GPU) NVIDIA Maxwell com 128 núcleos CUDA e 2 GB de memória LPDDR4. As principais especificações são apresentadas na Tabela 1. Embora a plataforma disponha de GPU dedicada, os experimentos comparativos foram conduzidos em CPU, com o *CPU-ExecutionProvider* do ONNX Runtime, para manter equivalência metodológica com a Raspberry Pi 4B. Durante os testes, a Jetson Nano operou no modo MAXN (10W), com dissipador e ventoinha.

Tabela 1. Especificações da Jetson Nano utilizadas no estudo.

CPU	Quad-core ARM Cortex-A57, 1,43 GHz
GPU	NVIDIA Maxwell com 128 núcleos CUDA
Memória RAM	2 GB LPDDR4 (64 bits)
Armazenamento	Cartão microSD
Sistema operacional	Ubuntu 20.04.6 LTS (GNOME Version 3.36.8)
Refrigeração no experimento	Dissipador + ventoinha

Raspberry Pi 4B. A Raspberry Pi 4B utilizada, mostrada na Figura 1(b), possui CPU Broadcom BCM2711 quad-core ARM Cortex-A72 de 1,5 GHz, 2 GB de memória LPDDR4 e armazenamento em cartão microSD, conforme a Tabela 2. Assim como na Jetson Nano, a inferência foi realizada com o *CPUExecutionProvider* do ONNX Runtime. A refrigeração adotada foi composta por dissipador passivo, sem ventoinha.

Tabela 2. Especificações da Raspberry Pi 4B utilizadas no estudo.

CPU	Quad-core ARM Cortex-A72, 1,5 GHz
GPU	Broadcom VideoCore VI
Memória RAM	2 GB LPDDR4
Armazenamento	Cartão microSD
Sistema operacional	Debian GNU/Linux 13 (trixie)
Refrigeração no experimento	Dissipador passivo

3.3. Protocolo de benchmark e métricas

O estudo relatado neste artigo compara modelos de detecção de pessoas, restritos à classe *person*, nas plataformas Jetson Nano e Raspberry Pi 4B. Para garantir comparabilidade, foi adotado um pipeline padronizado de entrada, pré-processamento, inferência e pós-processamento.

Foram avaliados os modelos YOLOv3-tiny, YOLOv5n, YOLOv7-tiny, YOLOv8n e SSDLite320 MobileNetV3-Large. Todos operaram com imagens de entrada em 320×320 pixels, utilizando *letterbox*. Também foram fixados os parâmetros de pós-processamento, com limiar de confiança de 0,25 e limiar de *Intersection over Union* (IoU) de 0,45 para a Supressão Não Máxima (*Non-Maximum Suppression* – NMS). Nos modelos YOLO, utilizou-se `class_id = 0`; no SSDLite, `class_id = 1`, já que a classe 0 corresponde ao *background*.

Os experimentos foram conduzidos com ONNX Runtime (ORT) em ambas as plataformas, adotando-se como base o *CPUExecutionProvider*. Antes dos *benchmarks*, todos os modelos foram exportados para ONNX e submetidos a *sanity check* para verificação estrutural e funcional.

Os modelos foram treinados a partir de um conjunto de dados público para detecção de pessoas, disponibilizado no repositório *Custom-Data-YOLOv8-Person-Detection*[1]. O conjunto está organizado no formato YOLO, com diretórios separados para

[1]Disponível em: `https://github.com/SMSajadi99/Custom-Data-YOLOv8-Person-Detection`. Acesso em: 20/04/2026.

imagens e anotações (`images` e `labels`) e partições independentes de treinamento, validação e teste (`train`, `valid` e `test`). Além disso, o arquivo `custom.yaml` define uma única classe de interesse, `Person`, compatível com o recorte adotado neste trabalho. No presente estudo, foram utilizadas 16.043 imagens de treinamento, 597 de validação e 4.774 de teste, totalizando 21.414 imagens. Para os modelos YOLO, o conjunto foi utilizado em seu formato original, conforme sua organização nativa no padrão YOLO. Para o SSDLite, as mesmas imagens e partições de treino e validação foram mantidas, enquanto as anotações em formato YOLO foram adaptadas dinamicamente no carregamento dos dados para o formato exigido pela implementação adotada, com conversão das caixas para coordenadas em pixel e remapeamento da classe *person* para o identificador compatível com o modelo, no qual a classe 0 é reservada ao *background*.

As métricas avaliadas foram divididas em dois grupos. Durante a inferência, foram coletadas latência fim a fim (E2E), latência de inferência pura (*infer-only*), FPS, número de detecções válidas, uso de CPU, uso de memória *Resident Set Size* (RSS) e temperatura/frequência da CPU. As métricas preditivas foram obtidas a partir dos resultados de treinamento e validação, considerando *Precision*, *Recall*, F1-score, mAP@0.5 e mAP@0.5:0.95 na melhor época segundo o maior mAP@0.5:0.95.

A medição de energia foi realizada de forma independente do laço principal de inferência, por meio de um sensor INA219 conectado via *Inter-Integrated Circuit* (I^2C) a uma Raspberry Pi 4 dedicada à aquisição. As leituras de tensão, corrente, potência e energia acumulada foram registradas em arquivos `CSV` por um script em Python e posteriormente associadas aos benchmarks por identificadores de plataforma, modelo e execução.

Para garantir reprodutibilidade, o benchmark utilizou uma fonte determinística de quadros proveniente de um conjunto fixo de imagens. Cada execução foi dividida em uma fase de *warm-up* com 10 quadros e uma fase de medição com 300 quadros. Para cada combinação de modelo e hardware, foram realizadas 7 repetições independentes. Ao final de cada repetição, foi gerado um arquivo CSV com uma linha por quadro e um resumo estatístico por execução, incluindo média e percentis p50, p90 e p99.

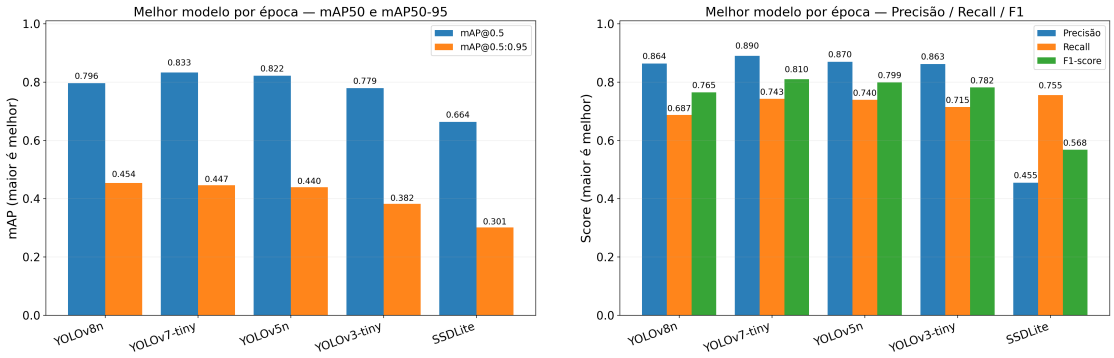
4. Resultados e Discussão

Esta seção apresenta os resultados preditivos, computacionais e energéticos obtidos nos experimentos, seguidos de uma análise multicritério que relaciona qualidade de detecção, latência, energia e memória.

4.1. Desempenho preditivo

As métricas de desempenho preditivo foram extraídas dos arquivos gerados ao final do treinamento, considerando, para cada arquitetura, a época de melhor desempenho segundo o maior mAP@0.5:0.95. A Figura 2 resume os resultados em termos de mAP, *Precision*, *Recall* e F1-score.

De modo geral, os modelos da família YOLO apresentaram desempenho superior e relativamente próximo entre si. O YOLOv7-tiny obteve o maior mAP@0.5 (0,833), a maior *Precision* (0,890) e o maior F1-score (0,810), enquanto o YOLOv8n alcançou o maior mAP@0.5:0.95 (0,454). O YOLOv5n também apresentou resultados consistentes, mantendo desempenho próximo aos melhores modelos da família.



(a) mAP@0.5 e mAP@0.5:0.95. (b) *Precision*, *Recall* e F1-score.

Figura 2. Desempenho preditivo dos modelos na melhor época de treinamento.

Esse resultado sugere que, no conjunto avaliado, a família YOLO reuniu melhor equilíbrio entre qualidade de localização e confiabilidade das detecções. Em particular, o destaque do YOLOv8n em mAP@0.5:0.95 indica maior robustez sob um critério mais rigoroso de sobreposição, enquanto o desempenho do YOLOv7-tiny em *Precision* e F1-score sugere melhor equilíbrio operacional no ponto de decisão considerado. Já o YOLOv5n, embora não lidere isoladamente as métricas preditivas, manteve desempenho próximo aos melhores modelos, o que se torna especialmente relevante quando essa qualidade é analisada em conjunto com latência e custo energético.

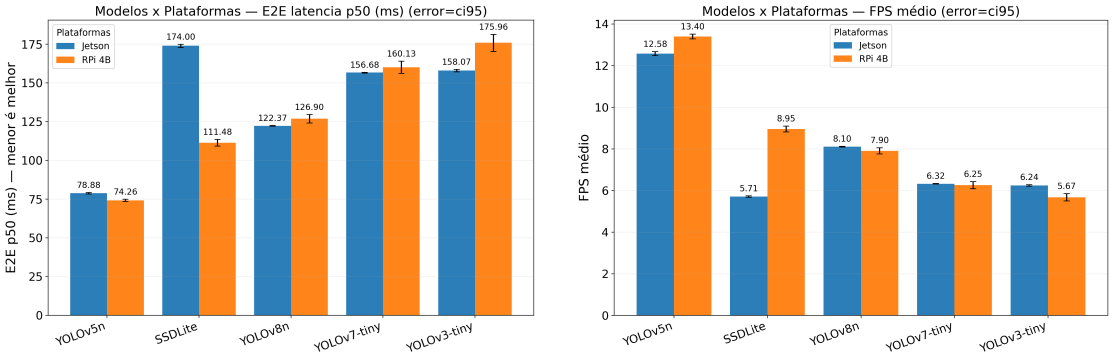
O SSDLite apresentou desempenho inferior aos modelos YOLO, com mAP@0.5 de 0,664, mAP@0.5:0.95 de 0,301, menor *Precision* (0,455) e menor F1-score (0,568), embora tenha obtido o maior *Recall* (0,755). Esse padrão sugere maior sensibilidade de detecção, porém com maior incidência de falsos positivos. Em termos práticos, isso indica que o modelo tendeu a ampliar a cobertura à custa de menor confiabilidade nas detecções, o que pode implicar maior necessidade de filtragem adicional ou maior tolerância a falsos positivos em aplicações reais. Como o objetivo do trabalho não foi otimizar individualmente cada arquitetura para máxima acurácia, essas métricas devem ser interpretadas como uma descrição comparativa do desempenho dos modelos no protocolo experimental adotado.

4.2. Desempenho computacional e energético

Em termos de latência, não houve superioridade uniforme de uma única plataforma para todos os modelos, como mostra a Figura 3(a). O Jetson Nano apresentou menores tempos medianos para YOLOv3-tiny e YOLOv8n, enquanto a Raspberry Pi 4B obteve melhores resultados para YOLOv5n e SSDLite. Para o YOLOv7-tiny, a diferença entre as plataformas foi pequena.

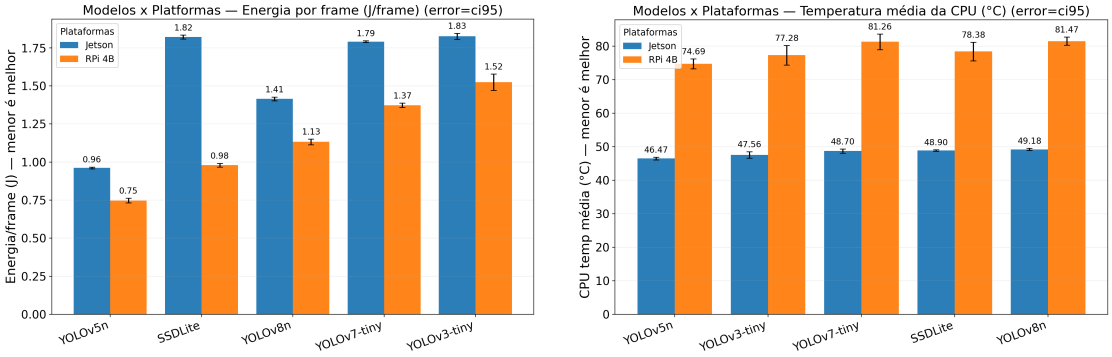
Em FPS, como observado na Figura 3(b), o YOLOv5n apresentou o melhor desempenho em ambas as plataformas. A Raspberry Pi 4B obteve vantagem clara para YOLOv5n e SSDLite, enquanto a Jetson Nano se saiu melhor com YOLOv3-tiny. Como esperado, o comportamento do FPS foi coerente com a latência de inferência.

Esse comportamento reforça que não houve uma superioridade uniforme de hardware em todos os modelos, mas sim uma interação entre arquitetura e plataforma. Ainda



(a) Latência E2E p50. (b) FPS médio.

Figura 3. Comparação de desempenho de inferência entre os modelos avaliados.



(a) Energia por quadro. (b) Temperatura média da CPU.

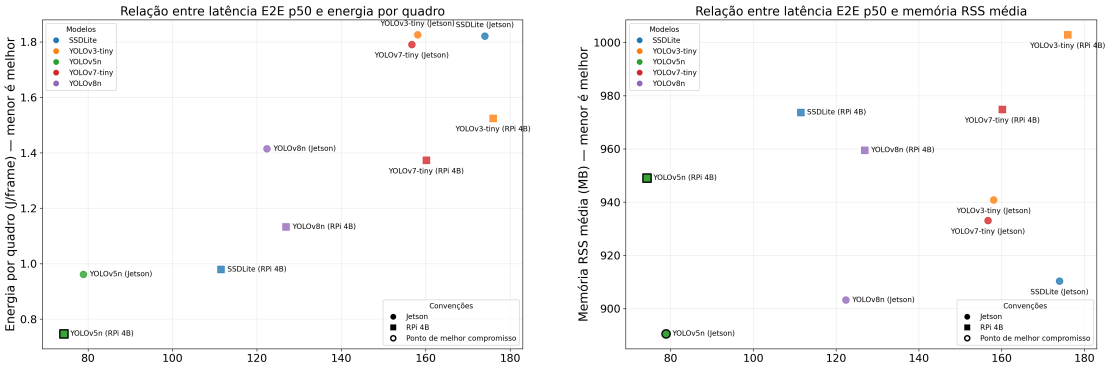
Figura 4. Comparação de custo energético e comportamento térmico entre os modelos e plataformas.

assim, o YOLOv5n destacou-se por combinar baixa latência e alto FPS nas duas SBCs, o que o coloca como a alternativa temporalmente mais estável do conjunto avaliado. Sob a perspectiva de implantação, esse resultado é relevante porque sugere maior previsibilidade de desempenho em cenários nos quais o requisito principal é resposta rápida com custo computacional moderado.

No que se refere ao consumo energético por quadro, a Raspberry Pi 4B apresentou menor custo energético que a Jetson Nano em todos os modelos avaliados, como mostra a Figura 4(a). O YOLOv5n foi o modelo mais eficiente energeticamente em ambas as plataformas. Além disso, o custo energético do SSDLite na Raspberry Pi 4B aproximou-se do custo do YOLOv5n na Jetson Nano, sugerindo que a escolha da plataforma pode compensar, em parte, diferenças entre arquiteturas.

A potência média medida na Jetson Nano foi superior à observada na Raspberry Pi 4B para todos os modelos, o que ajuda a explicar por que vantagens pontuais em latência não se traduziram em melhor eficiência energética.

Em outras palavras, pequenas vantagens temporais em alguns modelos não foram suficientes para compensar o maior consumo médio de potência da Jetson Nano. Isso



(a) E2E p50 vs energia por *frame*. (b) E2E p50 vs memória RSS.

Figura 5. Trade-offs de custo: latência em função de energia e memória.

ajuda a explicar por que a Raspberry Pi 4B se mostrou mais favorável do ponto de vista energético no conjunto avaliado, indicando que a escolha da plataforma não deve ser feita apenas com base em latência ou FPS, mas também no retorno obtido por watt consumido. Em aplicações com restrição de energia, dissipação térmica ou custo operacional contínuo, esse aspecto pode ser decisivo.

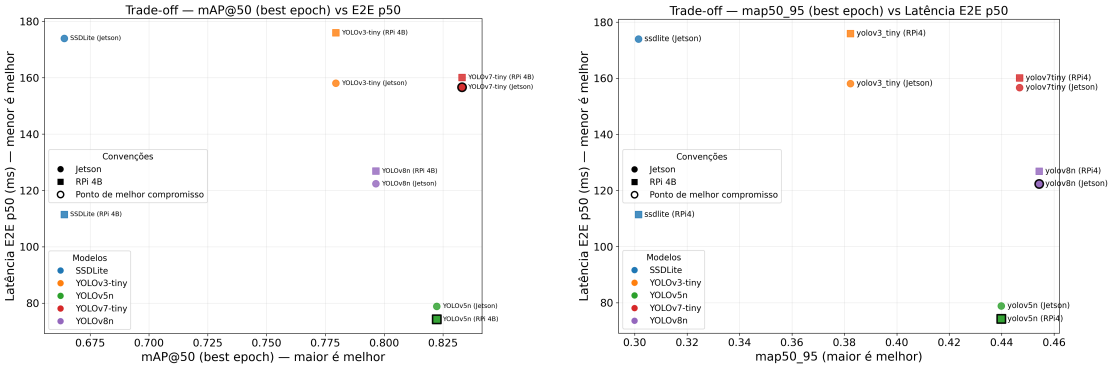
A temperatura média acompanhou, em parte, o custo computacional dos modelos, como mostra a como mostra a Figura 4(b): arquiteturas mais exigentes tenderam a elevar mais a temperatura média durante a execução. Entretanto, a interpretação dos valores absolutos deve ser feita com cautela, pois as plataformas não foram avaliadas sob condições térmicas estritamente equivalentes. A Jetson Nano operou com dissipação ativa, enquanto a Raspberry Pi 4B utilizou dissipação passiva, e não foi realizado um experimento especificamente voltado a isolar o impacto dessas diferenças, nem a controlar variáveis como temperatura ambiente. Desse modo, os resultados térmicos são úteis para descrever o comportamento observado no *setup* experimental adotado, mas não permitem atribuir causalmente as diferenças medidas a uma única condição física ou concluir, de forma geral, sobre o desempenho térmico relativo entre as plataformas.

4.3. Trade-offs multicritério

Nesta subseção, as métricas de custo (latência, energia e memória) são analisadas em conjunto com métricas de qualidade (mAP e F1), a fim de evidenciar o compromisso entre acurácia e viabilidade de execução em dispositivos de borda. Em todos os gráficos de *trade-off*, os pontos destacados representam configurações não dominadas na fronteira de Pareto.

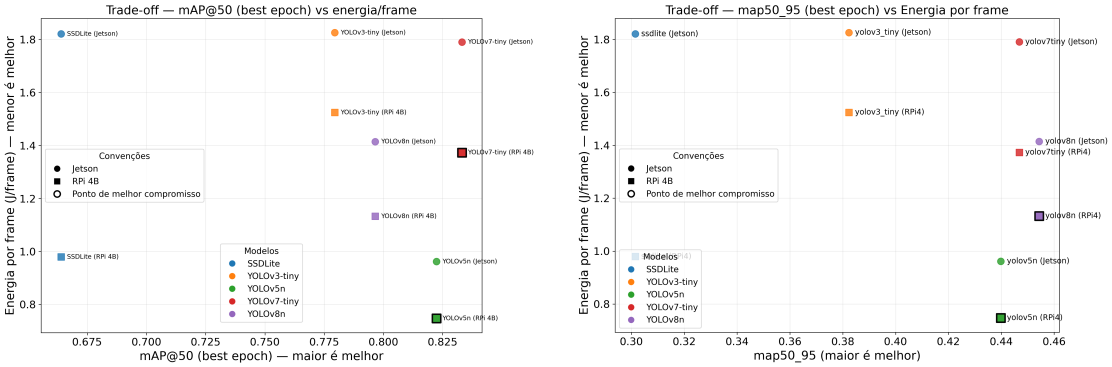
Latência × custo de execução. A relação entre latência E2E p50 e energia por *frame* (Figura 5(a)) define o espaço de viabilidade do sistema, enquanto a relação entre latência e memória RSS média (Figura 5(b)) evidencia que a escolha da plataforma pode depender do requisito predominante da aplicação.

Qualidade × latência. Os gráficos da Figura 6 mostram que ganhos de qualidade tendem a vir acompanhados de maior custo computacional. Nesse cenário, YOLOv7-tiny e YOLOv8n aparecem como referências, porém sob perspectivas distintas: o primeiro se mostra mais competitivo em mAP@0.5, enquanto o segundo se destaca em mAP@0.5:0.95,



(a) mAP@50 (*best epoch*) vs E2E p50. (b) mAP@50–95 (*best epoch*) vs E2E p50.

Figura 6. Trade-offs de qualidade em função da latência.



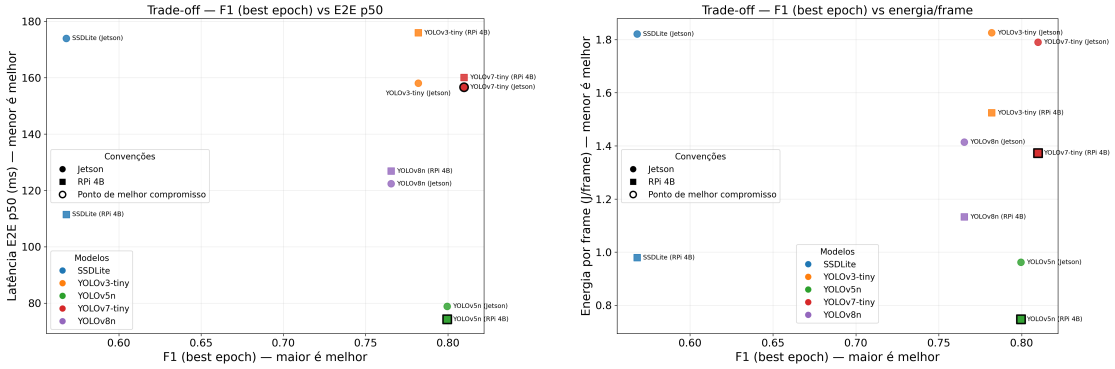
(a) mAP@50 (*best epoch*) vs energia/*frame*. (b) mAP@50–95 (*best epoch*) vs energia/*frame*.

Figura 7. Trade-offs de qualidade em função do custo energético.

isto é, sob um critério mais rigoroso de localização. Essa diferença é relevante porque indica que a noção de melhor qualidade depende da métrica adotada e, portanto, do requisito operacional da aplicação. Ao mesmo tempo, o YOLOv5n mantém desempenho preditivo próximo ao dos melhores modelos com menor latência, o que o torna particularmente atrativo em cenários nos quais o tempo de resposta é um requisito dominante.

Qualidade × energia. A Figura 7 sintetiza a discussão do ponto de vista energético, permitindo identificar configurações que oferecem maior qualidade a custo energético elevado e alternativas ligeiramente inferiores em qualidade, porém significativamente mais econômicas por inferência. Esse resultado é importante porque explicita o custo marginal associado a pequenos ganhos adicionais de qualidade preditiva. Em aplicações embarcadas de execução contínua, esse custo tende a se acumular ao longo do tempo e pode comprometer a viabilidade prática da solução. Nesse sentido, os gráficos mostram que a escolha do detector não deve ser guiada apenas pelo maior valor de mAP, mas também pelo custo energético necessário para sustentar esse ganho em operação.

F1 como métrica operacional. As Figuras 8(a) e 8(b) avaliam o F1 como métrica operacional, por refletir um ponto de operação mais próximo do uso real. Essa leitura é particularmente útil porque o F1 resume, em um único indicador, o compromisso entre



(a) F1 (*best epoch*) vs E2E p50. (b) F1 (*best epoch*) vs energia/*frame*.

Figura 8. Trade-offs usando F1 como métrica operacional.

precision e *recall* no limiar de decisão adotado, aproximando a análise das condições efetivas de uso do detector. Sob essa perspectiva, a análise multicritério reforça que modelos com F1 elevado, mas custo energético e latência muito altos, podem ser menos adequados a sistemas embarcados do que alternativas ligeiramente inferiores em qualidade, porém mais eficientes. Assim, o F1 ajuda a evidenciar que a melhor escolha prática nem sempre coincide com o modelo de maior qualidade em métricas mais estritas de localização.

Em conjunto, os resultados mostram que a decisão de implantação não deve ser formulada apenas como a escolha do detector mais preciso em abstrato, mas como a escolha da combinação modelo–plataforma mais adequada ao critério dominante da aplicação. Enquanto YOLOv7-tiny e YOLOv8n se destacam em métricas específicas de qualidade, o YOLOv5n preserva desempenho preditivo competitivo com menor custo temporal e energético, o que altera o seu posicionamento quando a análise deixa de ser unidimensional. De forma complementar, a Raspberry Pi 4B mostrou-se sistematicamente mais eficiente do ponto de vista energético, reforçando que o ponto de operação de um mesmo modelo depende também da plataforma em que ele é executado.

Síntese por cenários. A partir dos resultados obtidos, é possível resumir a escolha dos modelos conforme diferentes prioridades de implantação no conjunto avaliado.

- **Cenário A — prioridade em tempo de resposta:** neste estudo, o YOLOv5n apresentou o melhor compromisso temporal, com menor latência e maior FPS nas duas plataformas avaliadas.
- **Cenário B — prioridade em eficiência energética:** o YOLOv5n voltou a se destacar no conjunto analisado, enquanto a Raspberry Pi 4B apresentou menor custo energético por quadro em todos os modelos avaliados.
- **Cenário C — prioridade em qualidade de detecção:** o YOLOv8n e o YOLOv7-tiny apareceram como as principais referências, com destaque para o YOLOv8n em mAP@0.5:0.95 e para o YOLOv7-tiny em F1-score e mAP@0.5. Nesses casos, o ganho de qualidade deve ser ponderado frente ao aumento de latência e de custo energético em relação ao YOLOv5n. No conjunto avaliado, o SSDLite não se mostrou competitivo sob a ótica de qualidade preditiva; ainda assim, esse resultado deve ser interpretado no contexto da configuração experimental adotada, e não como uma limitação geral da família SSD/SSDLite.

Do ponto de vista de aplicação, esses cenários ajudam a interpretar os resultados em contextos reais de uso. Em sistemas embarcados voltados a monitoramento contínuo, contagem de pessoas, análise de fluxo ou processamento local com restrição de energia, modelos com melhor equilíbrio entre latência e custo energético, como o YOLOv5n no conjunto avaliado, tendem a ser mais adequados. Já em aplicações nas quais a prioridade recai sobre a qualidade da detecção, mesmo com maior custo de execução, variantes como YOLOv8n e YOLOv7-tiny podem ser mais interessantes. Esses resultados reforçam que, em Edge AI, a escolha do detector é dependente do cenário de implantação e do critério operacional dominante.

5. Conclusão

Este trabalho apresentou uma avaliação comparativa de detectores leves de pessoas em plataformas de computação de borda, considerando métricas preditivas, latência, consumo energético, uso de memória e comportamento térmico. Para isso, foi adotado um protocolo experimental padronizado, com resolução fixa, pós-processamento uniforme, classe de interesse restrita a *person* e uso de um mesmo ambiente de execução em CPU nas plataformas Jetson Nano e Raspberry Pi 4B.

Os resultados mostraram que os modelos da família YOLO apresentaram, em geral, melhor qualidade preditiva que o SSDLite320 MobileNetV3-Large. Entre eles, o YOLOv7-tiny se destacou em precisão e F1-score, enquanto o YOLOv8n obteve o maior mAP@0.5:0.95. Ainda assim, a análise multicritério indicou que o YOLOv5n apresentou o melhor equilíbrio global entre qualidade de detecção, latência e custo energético, configurando-se como a alternativa mais adequada para execução em borda no conjunto avaliado.

Em relação às plataformas, não houve superioridade absoluta de um único dispositivo em todos os cenários. O comportamento em latência variou conforme o modelo, mas a Raspberry Pi 4B apresentou melhor eficiência energética em todos os casos, mostrando-se particularmente atrativa para aplicações com maior restrição de consumo.

Os resultados, contudo, devem ser interpretados à luz de algumas limitações do estudo. A avaliação foi conduzida apenas em CPU, sob um ambiente homogêneo de execução, o que favoreceu a comparabilidade entre os modelos, mas não esgota o potencial de plataformas como a Jetson Nano sob aceleração por GPU e pilhas otimizadas, como TensorRT. Além disso, o estudo ficou restrito a duas SBCs, o que permite uma comparação controlada, mas ainda não contempla toda a diversidade de dispositivos embarcados e aceleradores disponíveis. Como continuidade, pretende-se expandir a análise para cenários com aceleração por GPU, investigar o impacto de quantização e compressão, padronizar mais rigorosamente as condições térmicas e incluir arquiteturas, dispositivos e conjuntos de dados adicionais, bem como comparações com outras classes de infraestrutura.

Agradecimentos

Os autores agradecem ao Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP) pelo suporte institucional. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Declaração sobre uso de Inteligência Artificial

As ferramentas de Inteligência Artificial generativas, ChatGPT e Manus AI foram utilizadas apenas como suporte à revisão de redação, incluindo sugestões de clareza, concisão e organização textual. Nenhuma ferramenta de IA foi utilizada para gerar dados, executar experimentos, interpretar resultados de forma autônoma ou definir conclusões científicas. Os autores revisaram integralmente o texto final e assumem total responsabilidade pelo conteúdo do manuscrito.

Referências

Ali, M. L. and Zhang, Z. (2024). The yolo framework: A comprehensive review of evolution, applications, and benchmarks in object detection. *Computers*, 13(12).

Alqahtani, D. K., Cheema, M. A., and Toosi, A. N. (2025). Benchmarking deep learning models for object detection on edge computing devices. In Gaaloul, W., Sheng, M., Yu, Q., and Yangui, S., editors, *Service-Oriented Computing*, pages 142–150, Singapore. Springer Nature Singapore.

Alqahtani, D. K., Rodriguez, M. A., Cheema, M. A., Rezatofighi, H., and Toosi, A. N. (2026). Ecore: Energy-conscious optimized routing for deep learning models at the edge.

Azab, E., Ehab, M., Shihata, L., and Mashaly, M. (2026). Optimizing computer vision for edge deployment in industry 4.0: A framework and experimental evaluation. *Technologies*, 14(2):126.

Cocks, G., Magbag, T., Hemmati, M., and Wang, K. I.-K. (2023). Real-time pedestrian detection using resource constrained embedded platforms – a review. In *2023 IEEE Smart World Congress (SWC)*, pages 1–8.

Fan, H., Liu, S., Ferianc, M., Ng, H.-C., Que, Z., Liu, S., Niu, X., and Luk, W. (2018). A real-time object detection accelerator with compressed ssdlite on fpga. In *2018 International Conference on Field-Programmable Technology (FPT)*, pages 14–21.

Hemmatirad, K., Babaie, M., Afshari, M., Maleki, D., Saiadi, M., and Tizhoosh, H. R. (2022). Quality control of whole slide images using the yolo concept. In *2022 IEEE 10th International Conference on Healthcare Informatics (ICHI)*, pages 282–287.

Howard, A., Sandler, M., Chen, B., Wang, W., Chen, L.-C., Tan, M., Chu, G., Vasudevan, V., Zhu, Y., Pang, R., Adam, H., and Le, Q. (2019). Searching for mobilenetv3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324.

Hussain, M. (2024). Yolov1 to v8: Unveiling each variant–a comprehensive review of yolo. *IEEE Access*, 12:42816–42833.

Islam, R. B., Akhter, S., Iqbal, F., Saif Ur Rahman, M., and Khan, R. (2023). Deep learning based object detection and surrounding environment description for visually impaired people. *Heliyon*, 9(6).

Jain, S., Kumar, A., Goswami, P., Khajuria, R., Dayal, P., and Banerjee, S. (2024). Efficient person detection on single board computers: A comparative analysis of algorithms. In *2024 10th International Conference on Advanced Computing and Communication Systems (ICACCS)*. IEEE.

Jocher, G. and Qiu, J. (2026). Ultralytics yolo26. `https://github.com/ultralytics/ultralytics`. Version 26.0.0. Software. Licença AGPL-3.0. Acesso em: 24/04/2026.

Kim, C. E., Oghaz, M. M. D., Fajtl, J., Argyriou, V., and Remagnino, P. (2019). A comparison of embedded deep learning methods for person detection. In *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2019)*, pages 459–465. SciTePress.

Kolosov, D., Kelefouras, V., Kourtessis, P., and Mporas, I. (2022). Anatomy of deep learning image classification and object detection on commercial edge devices: A case study on face mask detection. *IEEE Access*, 10:115313–115333.

Lazarevich, I., Grimaldi, M., Kumar, R., Mitra, S., Khan, S., and Sah, S. (2023). Yolobench: Benchmarking efficient object detectors on embedded systems. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pages 1169–1178.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520.

Terven, J., Córdova-Esparza, D.-M., and Romero-González, J.-A. (2023). A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. *Machine Learning and Knowledge Extraction*, 5(4):1680–1716.

Ultralytics (2024a). Ultralytics yolo11. `https://docs.ultralytics.com/models/yolo11/`. Acesso em: 2026-03-11.

Ultralytics (2024b). Yolov9: A leap forward in object detection technology. `https://docs.ultralytics.com/models/yolov9/`. Acesso em: 2026-03-11.

Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., Han, J., and Ding, G. (2024). Yolov10: Real-time end-to-end object detection. *arXiv preprint arXiv:2405.14458*.

Yilmaz, E. N. and Navruz, T. S. (2025). Real-time object detection: A comparative analysis of yolo, ssd, and efficientdet algorithms. In *2025 7th International Congress on Human-Computer Interaction, Optimization and Robotic Applications (ICHORA)*, pages 1–9.