FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Aprendizagem Computacional em Tempo Real para Processamento de Imagem: Métodos e Aplicações

Francisco Manuel Alves do Couto Ribeiro

Mestrado em Engenharia Eletrotécnica e de Computadores

Orientador: Prof. Aníbal Castilho de Coimbra Matos

28 de julho de 2026

Resumo

A deteção de objetos em tempo real em dispositivos embarcados de baixo consumo energético constitui um desafio crescente na área da visão computacional, sobretudo em cenários de *edge computing*. Esta dissertação avalia a viabilidade da implementação de modelos leves de deteção numa plataforma embarcada de baixo custo, o Raspberry Pi 5 equipado com o acelerador Hailo-8, analisando o compromisso entre precisão e eficiência computacional em dois domínios visuais distintos.

Foram estudados os modelos YOLO26n, YOLOv8n e SSDLite320-MobileNetV3-Large, avaliados nos *datasets* KITTI e VisDrone. O processo experimental incluiu treino em GPU, avaliação em PC, exportação para ONNX, compilação para o formato HEF com quantização INT8 e execução em plataforma embarcada através da NPU Hailo-8 e da CPU ARM via ONNXRuntime. Os resultados mostram que a NPU permite inferência em tempo real, enquanto a execução em CPU apresenta desempenho insuficiente para aplicações de vídeo em tempo real.

No KITTI, o YOLOv8n obteve o melhor desempenho em PC, enquanto no VisDrone os modelos YOLO apresentaram valores muito próximos, refletindo a maior dificuldade do domínio aéreo. O SSDLite revelou um desempenho inferior em ambos os casos. A análise também evidenciou uma degradação de precisão associada à quantização, mais acentuada no VisDrone, bem como diferenças na robustez dos modelos à compressão.

Adicionalmente, foi desenvolvido e validado um sistema de inferência em tempo real com *tracking* multi-objeto e contagem por linha virtual, que operou de forma estável em sessões independentes de validação. Em termos globais, os resultados confirmam que a utilização de aceleradores NPU dedicados constitui uma solução promissora para a execução eficiente de modelos de deteção em dispositivos embarcados.

Palavras-chave: deteção de objetos, visão computacional, *edge computing*, Raspberry Pi 5, Hailo-8, quantização INT8, inferência em tempo real

Abstract

Real-time object detection on low-power embedded devices is an increasingly relevant challenge in computer vision, particularly in *edge computing* scenarios. This dissertation evaluates the feasibility of deploying lightweight detection models on a low-cost embedded platform, the Raspberry Pi 5 equipped with the Hailo-8 accelerator, by analyzing the trade-off between accuracy and computational efficiency across two distinct visual domains.

The YOLO26n, YOLOv8n, and SSDLite320-MobileNetV3-Large models were studied and evaluated on the KITTI and VisDrone *datasets*. The experimental workflow included GPU training, PC evaluation, ONNX export, HEF compilation with INT8 quantization, and embedded execution using both the Hailo-8 NPU and the ARM CPU via ONNXRuntime. The results show that the NPU enables real-time inference, while CPU execution is insufficient for real-time video applications.

On KITTI, YOLOv8n achieved the best performance on PC, while on VisDrone the YOLO-based models produced very similar results, reflecting the greater difficulty of the aerial domain. SSDLite showed lower performance in both cases. The analysis also revealed an accuracy drop associated with quantization, more pronounced on VisDrone, as well as differences in model robustness to compression.

In addition, a real-time inference system with multi-object tracking and virtual line counting was developed and validated, operating stably across independent validation sessions. Overall, the results confirm that dedicated NPU accelerators are a promising solution for efficient execution of object detection models on embedded devices.

Keywords: object detection, computer vision, *edge computing*, Raspberry Pi 5, Hailo-8, INT8 quantization, real-time inference

Objectivos de Desenvolvimento Sustentável das Nações Unidas

Os Objectivos de Desenvolvimento Sustentável das Nações Unidas fornecem um enquadramento global para promover inovação, sustentabilidade e uso responsável de recursos. Esta dissertação está particularmente relacionada com o ODS 9, Indústria, Inovação e Infra-estruturas, o ODS 11, Cidades e Comunidades Sustentáveis, e o ODS 12, Produção e Consumo Sustentáveis, pelos motivos resumidos na tabela seguinte.

ODS	**Meta**	**Contribuição**	**Indicadores de Desempenho e Métricas**
9	9.1	Este trabalho contribui para a construção de infra-estruturas tecnológicas mais acessíveis e resilientes, ao desenvolver e avaliar modelos de detecção de objectos em hardware embarcado de baixo custo.	Latência de inferência, FPS, tamanho do modelo e viabilidade de implementação em dispositivos de baixo consumo.
	9.5	A análise de arquitecturas leves e de aceleradores embarcados promove inovação em visão computacional e reforça a ligação entre investigação académica e implementação prática.	Precisão de detecção, compatibilidade com hardware e compromisso entre precisão e eficiência.
11	11.2	O sistema de monitorização de tráfego e contagem de veículos pode apoiar soluções de mobilidade urbana mais seguras e eficientes, contribuindo para cidades mais inteligentes.	Precisão da contagem, capacidade de processamento em tempo real e robustez em cenários urbanos.
	11.6	A utilização de sistemas de percepção em *edge* pode reduzir a carga computacional centralizada e apoiar a monitorização de ambientes urbanos com maior eficiência.	*Throughput* em plataformas embarcadas e estabilidade da inferência em cenários reais.
12	12.2	A escolha de hardware de baixo consumo e a avaliação do desempenho por watt promovem um uso mais eficiente dos recursos computacionais.	Eficiência energética, consumo de energia e desempenho por watt.
	12.5	O estudo de técnicas de compressão de modelos, como quantização e *pruning*, incentiva estratégias de implementação mais eficientes em termos de recursos.	Retenção de precisão após compressão, dimensão do modelo e eficiência em hardware limitado.

Agradecimentos

A realização desta dissertação não teria sido possível sem o contributo e o apoio de várias pessoas, às quais desejo expressar o meu sincero agradecimento.

Ao Professor Aníbal Castilho de Coimbra Matos, orientador desta dissertação, agradeço a disponibilidade, o acompanhamento constante e a orientação prestada ao longo de todo o trabalho. O seu apoio e contributo foram fundamentais para a concretização deste projeto.

À minha família, em especial aos meus pais e à minha irmã, agradeço o apoio incondicional ao longo de todo o percurso académico. A vossa presença, incentivo e confiança foram determinantes em todos os momentos.

Por fim, aos meus amigos da FEUP, agradeço o apoio ao longo destes anos. Um agradecimento especial aos "Engenheiros Aspirantes", pela amizade e pelas memórias construídas ao longo deste percurso.

Francisco Ribeiro

Declaração de Honra

Declaro que a presente dissertação é de minha autoria e não foi previamente apresentada e avaliada nesta ou em outra instituição. As referências a outros autores (afirmações, ideias, pensamentos), assim como a utilização de trabalhos publicados respeitam escrupulosamente as regras da atribuição, e encontram-se devidamente indicadas no texto e nas referências bibliográficas, de acordo corn as normas de referenciação. Tenho consciência de que a prática de plágio ou de autoplágio constituem ilícitos académicos, conforme estabelecido no Código de Ética da U.Porto.
Declaro, ainda, que utilizei ferramentas de Inteligência Artificial para a realização de parte(s) da presente dissertação, e essa utilização e os seus resultados estão devidamente assinalados e foram objeto de cuidada verificação para deteção de erros, imprecisões, atribuições infundadas ou outras formas de enviesamento de conteúdos e argumentos, bem como de determinação de autoria.

Francisco Manuel Alves do Couto Ribeiro

Assinado por: Francisco Manuel Alves do Couto Ribeiro
Num. de Identificação: BI15376166
Data: 28-07-2026 14:13:10 +01:00

CHAVE MÓVEL

Índice

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

API	Interface de Programação de Aplicações
CNN	*Convolutional Neural Network* (Rede Neuronal Convolucional)
CPU	Unidade Central de Processamento
DL	*Deep Learning* (Aprendizagem Profunda)
FL	*Federated Learning* (Aprendizagem Federada)
FLOPs	Operações de Ponto Flutuante
FPGA	*Field-Programmable Gate Array*
FPS	*Frames* por Segundo
GPU	Unidade de Processamento Gráfico
IoT	*Internet of Things* (Internet das Coisas)
mAP	*mean Average Precision* (Precisão Média)
ML	*Machine Learning* (Aprendizagem Automática)
NAS	*Neural Architecture Search* (Pesquisa de Arquitetura Neuronal)
OpenCV	*Open Source Computer Vision Library*
ROS	*Robot Operating System* (Sistema Operativo de Robôs)
UAV	Veículo Aéreo Não Tripulado

Capítulo 1

Introdução

Este capítulo apresenta o enquadramento geral do trabalho, a formulação do problema abordado, os objetivos definidos e as principais contribuições da dissertação. Em primeiro lugar, discute-se o contexto e a motivação que fundamentam o estudo. Seguidamente, é apresentada a definição do problema, bem como os objetivos e contribuições do trabalho. Por fim, descreve-se a estrutura global da dissertação.

1.1 Contexto e Motivação

A visão computacional tem vindo a transformar de forma significativa a forma como os sistemas automáticos percepcionam e interpretam o mundo visual. Entre as suas tarefas centrais, a deteção de objetos assume particular relevância, uma vez que permite localizar e classificar, de forma simultânea, as instâncias presentes numa cena. Esta capacidade é essencial em áreas como a condução autónoma, a vigilância inteligente, a robótica e a gestão de tráfego urbano, onde a interpretação visual em tempo útil é determinante para o funcionamento dos sistemas.

Ao longo das últimas décadas, os avanços nas redes neuronais profundas têm permitido alcançar níveis de desempenho muito elevados em tarefas de deteção. No entanto, a maioria destes modelos foi concebida e avaliada em ambientes com recursos computacionais abundantes, o que contrasta com muitos cenários reais de utilização, onde o processamento tem de ocorrer em dispositivos com limitações severas de energia, memória e capacidade de computação.

Este contraste tem impulsionado o interesse em abordagens de *edge computing*, nas quais a inferência é realizada próximo da origem dos dados, reduzindo a dependência de infraestrutura remota. Em contextos como monitorização de tráfego, sistemas de segurança e aplicações de mobilidade inteligente, esta descentralização permite responder a exigências de latência, privacidade e fiabilidade que seriam difíceis de satisfazer através de processamento exclusivamente em nuvem.

Deste modo, surge a necessidade de estudar até que ponto modelos de deteção modernos podem ser executados de forma eficiente em plataformas embarcadas, mantendo um desempenho aceitável em condições próximas da utilização real. É neste enquadramento que se insere o presente trabalho.

1.2 Definição do Problema

Embora existam atualmente modelos de deteção de objetos com elevado grau de precisão, a sua execução em dispositivos embarcados continua a ser limitada por constrangimentos de hardware e por requisitos de eficiência energética. Em particular, a utilização destes modelos em cenários de inferência local exige o equilíbrio entre dois objetivos frequentemente conflitantes: maximizar a precisão das deteções e minimizar o custo computacional necessário para as produzir.

O problema abordado nesta dissertação consiste em avaliar a viabilidade da execução de modelos leves de deteção de objetos numa plataforma embarcada de baixo custo, procurando compreender em que medida é possível obter desempenho em tempo real sem comprometer excessivamente a qualidade das previsões. Para tal, torna-se necessário analisar o comportamento dos modelos em diferentes domínios visuais, bem como o impacto das transformações exigidas para a sua implementação em hardware dedicado.

Adicionalmente, a adaptação destes modelos a plataformas embarcadas obriga frequentemente à aplicação de técnicas de compressão e quantização, que podem introduzir perdas de precisão difíceis de antecipar. Assim, o problema central não é apenas medir o desempenho absoluto dos modelos, mas também caracterizar o compromisso entre precisão, eficiência e viabilidade operacional num cenário realista de deployment.

Neste contexto, esta dissertação procura responder à questão de saber até que ponto modelos leves de deteção de objetos podem ser executados com sucesso em hardware embarcado dedicado, e qual o impacto das restrições impostas pela plataforma na qualidade e utilidade das deteções obtidas.

1.3 Objetivos e Contribuições

O objetivo principal desta dissertação é avaliar a viabilidade da execução de modelos leves de deteção de objetos em tempo real numa plataforma embarcada de baixo custo, analisando o compromisso entre precisão e eficiência computacional.

Este objetivo desdobra-se nos seguintes objetivos específicos:

1. Avaliar modelos leves de deteção de objetos em dois *datasets* de referência, KITTI e VisDrone.
2. Comparar o desempenho dos modelos em diferentes plataformas de inferência.
3. Analisar o impacto de técnicas de compressão sobre a precisão e a eficiência dos modelos.
4. Desenvolver e validar um sistema de inferência em tempo real com *tracking* e contagem de tráfego.

As principais contribuições deste trabalho são:

- Uma caracterização empírica do compromisso entre precisão e eficiência em hardware embarcado com acelerador NPU.

- A avaliação de modelos leves de deteção em dois domínios visuais contrastantes.
- A análise do efeito de técnicas de compressão no desempenho dos modelos.
- A validação de um sistema funcional de inferência em tempo real em condições reais de operação.

1.4 Estrutura da Dissertação

Para além desta introdução, a dissertação está organizada em cinco capítulos principais.

No **Capítulo 2**, apresenta-se o estado da arte relevante para o enquadramento do trabalho, cobrindo a evolução dos modelos de deteção de objetos, as arquiteturas leves para *edge*, as técnicas de *tracking* e contagem, o hardware de inferência embarcada e as técnicas de compressão de modelos.

O **Capítulo 3** descreve a metodologia experimental adotada, incluindo os *datasets* utilizados, os modelos selecionados, o protocolo de treino e avaliação, e a preparação dos modelos para execução em hardware embarcado.

O **Capítulo 4** apresenta os resultados experimentais obtidos, incluindo a avaliação comparativa dos modelos em diferentes plataformas de inferência, a análise do impacto da quantização e compressão, e a validação do sistema de inferência em tempo real.

O **Capítulo 5** discute os resultados obtidos, analisa o compromisso entre precisão e eficiência, e identifica as limitações do trabalho.

Por fim, o **Capítulo 6** sintetiza as principais conclusões, responde à hipótese central e sugere linhas de trabalho futuro.

Capítulo 2

Estado da Arte: Deteção de Objetos em Tempo Real

2.1 Introdução

Este capítulo apresenta o estado da arte relevante para o desenvolvimento desta dissertação, com foco na deteção de objetos em tempo real e na sua implementação em ambientes de computação em *edge*.

Em primeiro lugar, é revista a evolução das arquiteturas de deteção, distinguindo-se detetores de dois estágios e de um estágio, bem como os principais requisitos de tempo real associados a diferentes domínios de aplicação.

De seguida, o capítulo centra-se nos modelos leves mais utilizados para inferência em dispositivos com recursos limitados, com especial destaque para a família MobileNet, para as variantes recentes da família YOLO e para a arquitetura SSD-MobileNet. Posteriormente, são abordadas as principais abordagens de *tracking* multi-objeto e de contagem de tráfego, bem como o enquadramento do paradigma de *edge computing* e das plataformas de inferência que o suportam.

Por fim, discutem-se técnicas de compressão e otimização de modelos, os *datasets* utilizados na avaliação experimental e as métricas mais relevantes para a análise de desempenho, terminando com a identificação das lacunas na literatura que motivam o presente estudo.

2.2 Arquiteturas de Deteção de Objetos

O desenvolvimento de modelos de deteção de objetos eficientes para deployment em hardware com recursos limitados constitui uma área de investigação activa, motivada pela crescente procura de sistemas de visão computacional em tempo real em dispositivos embarcados. Esta secção revê as principais arquitecturas leves desenvolvidas para este contexto, cobrindo os backbones de extracção de características da família MobileNet, a evolução da família YOLO desde as suas origens até às versões mais recentes, e o SSDLite como baseline arquitetural de geração anterior. A secção

termina com uma comparação entre modelos em termos de precisão e eficiência computacional, enquadrando as escolhas metodológicas deste estudo.

2.2.1 Evolução das Arquiteturas de Deteção

A deteção de objetos consiste na localização e classificação simultânea das instâncias de objetos presentes numa imagem. Antes da adoção generalizada de técnicas de *deep learning*, os métodos dominantes baseavam-se em características construídas manualmente, como o Histograma de Gradientes Orientados (*Histogram of Oriented Gradients*, HOG), combinadas com classificadores aplicados em janelas deslizantes [1]. Embora estas abordagens tenham desempenhado um papel importante no desenvolvimento inicial da visão computacional, acabaram por ser largamente superadas pelas técnicas de *deep learning* em termos de precisão e robustez [2]. No entanto, a elevada complexidade computacional destes novos modelos trouxe desafios adicionais de processamento e latência em cenários com requisitos de tempo real, o que motivou a adoção de paradigmas como o *edge computing* [2].

A introdução das redes neuronais convolucionais profundas transformou de forma decisiva este domínio, ao permitir a aprendizagem automática de representações hierárquicas diretamente a partir dos dados. Neste contexto, a família R-CNN estabeleceu o paradigma moderno da deteção baseada em duas etapas. Após o R-CNN inicial e a sua evolução para Fast R-CNN, Ren et al. propuseram o Faster R-CNN, que integrou uma *Region Proposal Network* (RPN) com a cabeça de classificação e regressão, partilhando características convolucionais entre os diferentes blocos da arquitetura [3]. Como demonstrado pelos autores, esta abordagem permitiu ganhos expressivos de precisão e tornou o Faster R-CNN uma arquitetura de referência em tarefas de deteção de elevada qualidade, com resultados de destaque em benchmarks como o PASCAL VOC e o MS COCO.

Apesar da elevada precisão alcançada, os detetores baseados em duas etapas apresentam uma limitação estrutural importante: a deteção depende de uma fase prévia de geração de propostas, seguida de classificação e refinamento [3]. Esta natureza sequencial introduz um custo computacional significativo, o que dificulta a sua adoção em cenários de inferência em tempo real, especialmente em plataformas com recursos limitados. Por essa razão, a evolução da área conduziu progressivamente ao desenvolvimento de arquiteturas mais eficientes, orientadas para um melhor equilíbrio entre precisão e latência [2, 4].

2.2.2 Detetores de Dois Estágios e de Um Estágio

Os detetores de dois estágios, como o Faster R-CNN [3], distinguem-se por separar explicitamente a geração de regiões candidatas da classificação final dos objetos. Em termos gerais, esta estratégia permite uma localização mais refinada e uma maior precisão, sobretudo em tarefas exigentes ou com objetos de pequenas dimensões. No entanto, essa vantagem é acompanhada por maior complexidade computacional e por tempos de inferência superiores, o que limita a sua utilização em aplicações com restrições severas de latência [2–4].

Em contraste, os detetores de um estágio reformulam a deteção como um problema de regressão densa direta, eliminando a etapa intermédia de propostas. Esta simplificação arquitetural permitiu reduzir substancialmente a latência e tornou estas abordagens particularmente adequadas para execução em tempo real. Entre os principais marcos desta linha destaca-se o *Single Shot MultiBox Detector* (SSD), proposto por Liu et al. [5], que realiza predições convolucionais em múltiplas escalas de mapas de características, permitindo detetar objetos de diferentes dimensões numa única passagem pela rede.

Paralelamente, Redmon et al. introduziram o YOLO (*You Only Look Once*), uma abordagem que reformula a deteção como uma única operação de inferência sobre a imagem completa, dividindo-a numa grelha espacial e prevendo diretamente caixas delimitadoras e probabilidades de classe [6]. Esta formulação constituiu um ponto de viragem importante, ao demonstrar que era possível atingir velocidades compatíveis com processamento em tempo real mantendo níveis de precisão competitivos. Em consequência, os detetores de um estágio estabeleceram-se como a principal base para o desenvolvimento de soluções orientadas para dispositivos *edge* [7].

A evolução subsequente da família YOLO reflete precisamente esta procura contínua de maior eficiência. O YOLOv3 introduziu o backbone Darknet-53 e predições multi-escala, melhorando a capacidade de deteção em diferentes resoluções [8]. Mais tarde, variantes recentes como o YOLOv8n adotaram cabeças de deteção *anchor-free* e desacopladas, bem como estratégias de atribuição mais eficazes durante o treino, visando melhorar o compromisso entre precisão e custo computacional [9]. De forma semelhante, o YOLOv10 propôs alterações orientadas para reduzir a dependência de pós-processamento, nomeadamente através da eliminação do *Non-Maximum Suppression* (NMS) em certos cenários de inferência [10].

No caso do YOLO26n, por se tratar de uma arquitetura recente, a literatura disponível é ainda limitada. Ainda assim, os documentos técnicos e avaliações preliminares sugerem que esta variante introduz modificações orientadas para reduzir a latência de inferência e simplificar o pós-processamento, preservando simultaneamente a adequação a cenários com restrições de energia e capacidade computacional [11, 12]. Esta trajetória confirma que a evolução dos detetores de um estágio tem sido marcada por um esforço contínuo de otimização arquitetural, com especial enfoque em aplicações onde a inferência local e em tempo real constitui um requisito central.

2.2.3 Domínios de Aplicação e Requisitos de Tempo Real

A relevância da deteção de objetos em tempo real varia em função do domínio de aplicação, mas em muitos contextos a latência do sistema tem implicações diretas na segurança, na eficiência operacional e na qualidade da decisão. Em ambientes dinâmicos, a capacidade de detetar e interpretar rapidamente entidades visuais pode determinar o sucesso ou a falha de uma determinada tarefa, sobretudo quando o sistema opera de forma contínua e com interação direta com o meio envolvente [2, 4].

Na condução autónoma, por exemplo, os veículos necessitam de analisar continuamente o espaço envolvente para identificar peões, veículos, sinais de trânsito e outros elementos relevantes. Neste contexto, atrasos de inferência podem traduzir-se em distâncias percorridas sem reação,

comprometendo a capacidade de resposta em situações críticas [7]. De forma semelhante, em sistemas de vigilância e monitorização, a deteção em tempo real permite identificar eventos relevantes, comportamentos anómalos ou alterações de fluxo com utilidade imediata para segurança e gestão operacional [13].

Na robótica e nos sistemas autónomos, o ciclo entre perceção, decisão e ação depende fortemente da rapidez do processamento visual. Qualquer atraso acumulado neste ciclo pode comprometer a navegação, o desvio de obstáculos ou a execução de tarefas de manipulação [14]. Em ambiente industrial, a deteção automática de defeitos ou irregularidades exige igualmente baixa latência e elevada consistência, especialmente em linhas de produção contínuas, onde decisões tardias podem afetar produtividade e controlo de qualidade. [4]

Em sistemas de vigilância, a deteção em tempo real é utilizada para identificar intrusões, comportamentos anómalos e alterações relevantes no fluxo de pessoas ou veículos, permitindo uma resposta rápida e aumentando a eficácia operacional dos sistemas de monitorização [13].

Apesar destes requisitos, os modelos de visão computacional não operam, muitas vezes, em infraestruturas com capacidade computacional abundante. Durante muito tempo, o processamento de dados visuais assentou em arquiteturas *cloud-centric*, nas quais os dados recolhidos localmente eram transmitidos para servidores remotos, onde ocorria a inferência. Embora este paradigma ofereça elevada capacidade de processamento, apresenta limitações importantes em cenários de tempo real, nomeadamente ao nível da latência, do consumo de largura de banda e da exposição de dados sensíveis durante a transmissão [2, 15].

Estas limitações tornam-se particularmente críticas quando se considera a crescente adoção de dispositivos distribuídos, sistemas embebidos e plataformas *edge*, frequentemente sujeitas a restrições de energia, memória e capacidade de processamento. Neste contexto, a deteção de objetos em tempo real deixou de depender exclusivamente da precisão dos modelos, passando a exigir também arquiteturas eficientes e compatíveis com execução local. Assim, a literatura recente tem vindo a concentrar-se não apenas na melhoria da qualidade da deteção, mas também na redução da latência e do custo computacional, de modo a viabilizar a implementação prática destes sistemas em ambientes reais [4].

2.2.4 Modelos Leves para Deteção na Edge

A execução de modelos de deteção de objetos em dispositivos *edge* exige arquiteturas capazes de equilibrar precisão, latência, consumo energético e ocupação de memória. Neste contexto, a literatura tem evoluído no sentido de desenvolver modelos leves que preservem desempenho de deteção aceitável, mantendo simultaneamente a viabilidade de execução em hardware com recursos limitados, como sistemas embebidos, plataformas móveis e aceleradores dedicados de baixo consumo. Entre as arquiteturas mais relevantes destacam-se os *backbones* leves, como a família MobileNet, e os detetores de estágio único, como SSD e YOLO, cuja evolução recente reflete uma preocupação crescente com a eficiência computacional e a adequação ao *deployment* em *edge* [2, 4, 15].

2.2.4.1 Backbones Leves: Família MobileNet

O principal bottleneck computacional nos detetores de objetos profundos reside no backbone responsável pela extração de características. As convoluções padrão em redes profundas implicam custos computacionais elevados, tornando a sua execução menos adequada para hardware com restrições energéticas, como dispositivos de baixa potência [15].

Howard et al. introduziram o MobileNet, que substitui as convoluções padrão por convoluções separáveis em profundidade, compostas por uma convolução *depthwise* por canal de entrada seguida de uma convolução *pointwise* 1×1 para mistura entre canais. Esta abordagem reduz significativamente a complexidade computacional face às convoluções convencionais [15]. O MobileNetV2 introduziu blocos residuais invertidos e camadas de *bottleneck* linear para melhorar o fluxo de gradiente, enquanto o MobileNetV3 refinou a arquitetura com ativações *hard-swish* e módulos *squeeze-and-excitation* [15].

Para além da família MobileNet, outras arquiteturas leves foram propostas com objetivos semelhantes. A família EfficientNet explora uma estratégia de *compound scaling* que equilibra profundidade, largura e resolução de entrada, alcançando um compromisso favorável entre precisão e custo computacional [16]. Por sua vez, o SqueezeNet adota uma estratégia agressiva de compressão paramétrica baseada em módulos *Fire*, demonstrando que é possível reduzir substancialmente o tamanho do modelo quando a memória constitui a principal restrição [17]. Ainda assim, a família MobileNet permanece como uma das bases arquiteturais mais influentes em cenários *edge*, pela sua adoção alargada em tarefas de classificação, deteção e segmentação.

2.2.4.2 A Família YOLO

Enquanto a família MobileNet atua sobretudo ao nível do extrator de características, a família YOLO procura otimizar o compromisso entre precisão e latência ao nível do detetor completo. A arquitetura YOLO passou por revisões contínuas desde a sua introdução, com cada geração contribuindo com inovações que avançam este compromisso em direção a cenários de deteção em tempo real e *deployment* prático em hardware restrito.

O YOLOv3 [8] substituiu o *backbone* Darknet-19 original pelo Darknet-53, introduzindo predições multi-escala em três resoluções de mapas de características com uma arquitetura ao estilo de *feature pyramid*. O YOLOv4 introduziu redes CSP (*Cross-Stage Partial*) e *data augmentation* em mosaico, inovações posteriormente adotadas pelo YOLOv5, que se tornou o *baseline* dominante para *deployment* prático no período 2020–2022 [10, 11].

O YOLOv8n [9] introduziu uma cabeça de deteção *anchor-free* e totalmente *decoupled*, com ramos separados de classificação e regressão, combinada com um *Dynamic Task-Aligned Assigner* para supervisão de treino melhorada. Com 3.2M de parâmetros e 8.7 GFLOPs, atinge 37.3 mAP@50:95 no COCO, sendo um alvo de projeto explícito para *deployment* em hardware com recursos limitados [10]. O YOLOv9 [18] contribuiu com o *Programmable Gradient Information* (PGI), que preserva informação completa através de um ramo auxiliar reversível durante o

treino, enquanto o YOLOv10 [10] eliminou o pós-processamento NMS através de uma estratégia de atribuição dupla consistente, reduzindo a latência ao desacoplar o pipeline de deteção do pós-processamento.

2.2.4.3 YOLO26n: Arquitetura e Evolução Recente

O YOLO26n [11, 12], a versão mais recente da família Ultralytics avaliada neste estudo, introduz melhorias arquiteturais relevantes face às gerações anteriores. A principal inovação consiste na eliminação da *Distribution Focal Loss* (DFL) e do NMS, simplificando a regressão de caixas delimitadoras e permitindo inferência *end-to-end* sem pós-processamento, o que reduz a latência em até 43% face ao YOLOv11 na variante nano em CPU [11]. A variante nano YOLO26n foi especificamente concebida para *deployment* em dispositivos com recursos limitados, com suporte nativo a quantização INT8 com degradação mínima de precisão, propriedade particularmente relevante para aceleradores como o Hailo-8 que operam exclusivamente neste modo.

Sendo uma arquitetura lançada em 2026, o YOLO26n dispõe de literatura publicada por terceiros ainda limitada, tornando a avaliação empírica conduzida neste estudo uma das primeiras caracterizações desta arquitetura em hardware NPU dedicado. Esta limitação da literatura disponível reforça a relevância de comparar o seu desempenho não apenas em termos de mAP, mas também em métricas operacionais concretas, como latência, FPS e sensibilidade à quantização.

2.2.4.4 SSDLite-MobileNet como Baseline Arquitetural

O SSDLite320-MobileNetV3-Large combina a cabeça de deteção multi-escala do SSD [5] com o *backbone* MobileNetV3-Large [19], representando uma abordagem consolidada anterior à dominância da família YOLO. A variante SSDLite substitui as convoluções padrão da cabeça SSD por convoluções separáveis em profundidade, reduzindo significativamente o número de parâmetros e operações. Esta combinação tornou-se uma linha de base recorrente em cenários de *edge* devido à sua simplicidade arquitetural, maturidade de implementação e ampla compatibilidade com *frameworks* de *deployment*.

A sua inclusão neste estudo serve como linha de base arquitetural que permite contextualizar os ganhos de desempenho introduzidos pelas inovações mais recentes em termos de precisão de deteção e eficiência computacional. Em particular, a comparação entre uma arquitetura *anchor-based* consolidada, como o SSDLite, e arquiteturas YOLO mais recentes permite observar de forma mais clara o impacto da evolução das cabeças de deteção, da atribuição de alvos e do desenho global da rede no compromisso entre precisão e latência.

2.2.4.5 Comparação entre Modelos: Precisão vs. Eficiência

A comparação entre modelos de deteção requer a consideração de múltiplas métricas para além do mAP bruto. Os FLOPs medem o custo computacional por passe de rede e constituem indicadores independentes do hardware, enquanto o número de parâmetros determina a pegada de memória do modelo. Contudo, a relação entre estas métricas e a latência real em hardware

específico é não trivial: a arquitetura do hardware, a hierarquia de memória, o suporte a operadores e a compatibilidade com quantização interagem para determinar o *throughput* efetivo [15, 20].

Deste modo, a seleção da arquitetura apropriada depende sempre dos requisitos específicos da aplicação, nomeadamente a precisão necessária, a latência máxima admissível, as restrições de memória e o orçamento energético disponível. Esta observação motiva a abordagem de *benchmarking* empírico em hardware específico adotada neste estudo, na qual os modelos são comparados não apenas pelas suas características teóricas, mas também pelo desempenho efetivamente observado nas plataformas de inferência consideradas.

Os valores apresentados na Tabela 2.1 correspondem a resultados de referência reportados na literatura para o conjunto de dados COCO. No caso do SSDLite-MobileNetV2, os números de parâmetros, FLOPs e mAP@50:95 foram extraídos da documentação do OpenVINO Model Zoo, que disponibiliza uma implementação otimizada deste modelo para inferência em edge.[21] Para o YOLOv8n e para o YOLO26n, os valores de parâmetros, FLOPs, mAP@50:95 e latência em CPU foram obtidos a partir da documentação oficial da Ultralytics e de relatórios técnicos de benchmarks recentes, onde ambos os modelos são avaliados de forma sistemática em COCO com pipelines de inferência baseados em ONNX Runtime.[22–25] Estes números não refletem o desempenho na plataforma experimental deste estudo, mas constituem um ponto de comparação neutro que permite observar o compromisso entre precisão, complexidade computacional e latência em condições de referência.

Tabela 2.1: Valores de referência de precisão e eficiência para modelos leves no COCO

Modelo	**Dataset**	**Parâmetros (M)**	**FLOPs (B)**	**mAP@50:95 (%)**	**Latência CPU (ms)**
SSDLite-MobileNetV2	COCO val	4.48	1.53	24.3	–
YOLOv8n	COCO val	3.20	8.70	37.3	80.4
YOLO26n	COCO val	2.40	5.40	40.9	38.9

2.3 Tracking e Contagem de Tráfego

O tracking multi-objeto estende a deteção de objetos a sequências temporais, atribuindo identidades consistentes às instâncias detetadas ao longo de frames de vídeo. Esta capacidade é essencial em aplicações de monitorização de tráfego, onde não basta detetar veículos e peões em cada frame isoladamente; é necessário seguir a trajetória de cada objeto para contabilizar cruzamentos, estimar velocidades e caracterizar fluxos de tráfego [7, 26].

2.3.1 Paradigmas de Tracking Multi-Objeto

O paradigma dominante em sistemas práticos é o *tracking-by-detection* (TBD), que trata a deteção e o tracking como estágios sequenciais: um detetor identifica objetos em cada frame de forma independente, e um algoritmo de associação faz a correspondência das deteções entre frames para manter as identidades [26]. A modularidade do paradigma TBD permite que os

componentes de deteção e tracking sejam otimizados independentemente, o que é vantajoso para deployment em hardware edge heterogéneo, onde o detetor pode correr num acelerador dedicado enquanto o tracking corre no CPU do host.

Abordagens alternativas de deteção e tracking conjuntos, como o FairMOT e o JDE [27], integram características de tracking diretamente na cabeça de deteção, eliminando o passo separado de ReID. Embora estas abordagens atinjam desempenho competitivo em benchmarks, a capacidade adicional de rede que requerem aumenta o tempo de inferência e o tamanho do modelo, tipicamente para níveis incompatíveis com deployment em tempo real na classe de dispositivos considerada neste trabalho.

2.3.2 Tracking-by-Detection: SORT e Hungarian Algorithm

O SORT (*Simple Online and Realtime Tracking*) [26] estabeleceu uma base importante para o tracking leve em sistemas em tempo real. O método combina um filtro de Kalman para prever o estado da caixa delimitadora com o algoritmo húngaro (*Hungarian Algorithm*) para fazer a associação *frame to frame*, usando IoU como métrica de custo. Por ser uma solução muito leve, consegue processar cerca de 260 Hz em CPU, o que o torna adequado para cenários com restrições computacionais. No entanto, como depende apenas da proximidade espacial para associar deteções, apresenta maior probabilidade de troca de identidade em situações de oclusão.

Para ultrapassar esta limitação, o DeepSORT [28] introduziu um descritor de aparência profundo, um *embedding* de 128 dimensões aprendido a partir de um *dataset* de reidentificação de pessoas. Este método combina a distância de Mahalanobis, associada à informação geométrica, com a distância cosseno no espaço de *embedding*, associada à aparência. Desta forma, reduz significativamente as trocas de identidade em oclusões prolongadas, mas introduz um custo adicional de inferência por deteção, o que pode limitar o desempenho em tempo real em cenas densas e em hardware com recursos limitados.

Já o ByteTrack [29] segue uma estratégia diferente. Este método procura resolver uma limitação comum dos *trackers* anteriores: o descarte de deteções de baixa confiança, que pode levar à perda de objetos e à fragmentação das trajetórias. Para isso, utiliza uma associação em dois estágios: primeiro associa deteções de alta confiança aos *tracks* existentes com base no IoU e, depois, tenta recuperar objetos não associados através de deteções de baixa confiança. Esta abordagem reduz a fragmentação sem recorrer a modelos de aparência. Ainda assim, como foi avaliada sobretudo em hardware GPU, a sua aplicação em plataformas *edge* com aceleradores dedicados requer uma caracterização empírica específica.

2.3.3 Contagem de Objetos por Linha Virtual

A contagem de tráfego a partir de streams de vídeo é uma aplicação de elevado valor da deteção e tracking em sistemas de transporte inteligentes. O paradigma de contagem por cruzamento de linha define linhas virtuais de contagem dentro do campo de visão da câmara e regista um evento de contagem sempre que a trajetória do centroide de um track cruza a linha numa direção

especificada. Esta abordagem requer apenas um teste geométrico de posição relativa por track ativo por frame, adicionando overhead computacional negligenciável ao pipeline de tracking. A contagem por classe é obtida mantendo contadores separados para cada classe detetada, permitindo a contagem simultânea de carros, vans, peões, ciclistas e outras classes [7].

2.4 Edge Computing e Hardware de Inferência

O processamento de imagem em tempo real em dispositivos embarcados exige uma compreensão clara das arquitecturas de computação disponíveis e das suas implicações para o *deployment* de modelos de *deep learning*. Esta secção apresenta o paradigma de *edge computing*, os seus benefícios face à abordagem *cloud-centric*, as limitações do hardware embarcado e os aceleradores NPU disponíveis, com destaque para o Hailo-8 utilizado neste trabalho.

O *edge computing* (EC) surge como um paradigma computacional descentralizado que altera a arquitectura tradicional da computação em *cloud*, ao aproximar os recursos computacionais e de armazenamento das fontes de dados, em vez de os manter centralizados em *data centers* remotos [4]. Formalmente, o EC é definido como um modelo que fornece serviços computacionais e infraestrutura de tecnologia da informação na periferia da rede, junto aos dispositivos terminais que geram e consomem dados [4]. Esta abordagem reduz a distância de transmissão e, consequentemente, a latência, ao contrário do paradigma *cloud-centric*, no qual a inteligência computacional se encontra concentrada em servidores remotos [2, 4].

A arquitectura geral de EC organiza-se em camadas hierárquicas, conforme ilustrado na Figura 2.1. Na camada terminal (*end layer*), encontram-se os dispositivos em contacto directo com o mundo físico, equipados com sensores que recolhem informação posteriormente processada na periferia da rede ou na *cloud*. A camada *edge* situa-se entre a *cloud* e os dispositivos terminais, integrando recursos de computação, armazenamento e comunicação, o que permite deslocar tarefas originalmente executadas na *cloud* para uma camada intermédia, mais próxima das fontes de dados [4]. Por sua vez, a camada *cloud* mantém os servidores centralizados com elevado poder computacional e grande capacidade de armazenamento, assegurando o controlo global da arquitectura [4].

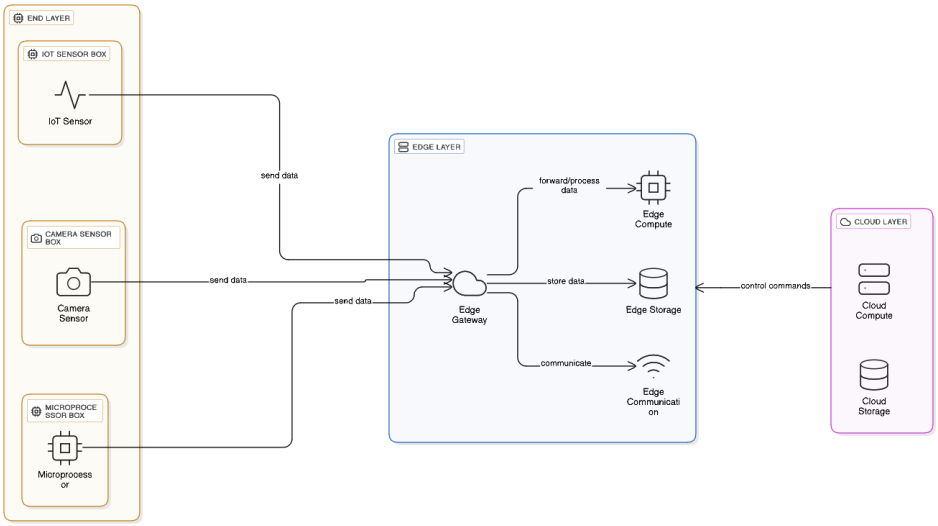


Figura 2.1: Arquitetura hierárquica do paradigma de *Edge Computing*.

Apesar de a arquitectura de EC não ter sido concebida para substituir a *cloud*, mas sim para a complementar [4], a distribuição de tarefas entre ambas traz vantagens claras. Os dados podem ser processados e filtrados na *edge*, reduzindo o volume de informação transmitida para a *cloud*, enquanto esta continua a fornecer capacidades computacionais de larga escala e armazenamento superior.

Apesar das vantagens do paradigma *edge*, a abordagem *cloud-centric* continua a apresentar limitações críticas em cenários que exigem resposta rápida [2, 4]. Em primeiro lugar, os sistemas *cloud* tradicionais caracterizam-se por latências elevadas, uma vez que os dados gerados nos dispositivos terminais têm de ser transmitidos para *data centers* remotos. Em contextos como a condução autónoma, esta latência pode representar um risco significativo [2, 4]. Adicionalmente, a transmissão contínua de fluxos de vídeo implica um consumo intensivo de largura de banda, sendo que estudos demonstram que apenas cerca de um terço dos dados recolhidos por sensores é efectivamente útil, o que conduz a desperdício de recursos de rede [4]. Por fim, a centralização de dados em servidores remotos expõe riscos ao nível da privacidade e da segurança da informação, além de introduzir um ponto único de falha na infraestrutura [4].

2.5 Compressão e Otimização de Modelos

A execução de modelos de *deep learning* em hardware embarcado com restrições computacionais requer técnicas de compressão que reduzam o custo de inferência sem degradação inaceitável de precisão. Esta secção revê as principais técnicas utilizadas, nomeadamente quantização, *pruning* e exportação para formatos intermédios, com especial atenção ao pipeline de *deployment* no Hailo-8 adoptado neste trabalho.

2.5.1 Quantização: FP32 para INT8

A quantização consiste em reduzir a precisão numérica dos pesos e das ativações de um modelo, passando de FP32 para representações de menor precisão, como INT8 ou FP16. Esta técnica traz três vantagens principais: diminui o uso de memória, pode reduzir significativamente a latência de inferência em hardware que suporta operações inteiras de forma nativa, e tende também a baixar o consumo energético [30]. Contudo, a redução de precisão numérica não é isenta de custos: estudos recentes em modelos YOLO quantizados para INT8 via PTQ documentam quedas de precisão entre 3 e 7 pontos percentuais de mAP50-95 em dados limpos, com ganhos de velocidade de 1.5 a 3.3$\times$ face ao FP32 [31]. A Tabela 4.15 apresenta, no contexto deste trabalho, a degradação de precisão observada na transição PC float32 $\rightarrow$ Hailo INT8 para os modelos avaliados, confirmando este padrão.

Os fundamentos da quantização INT8 com treino consciente da quantização (*quantization-aware training*, QAT) foram estabelecidos por Jacob et al. [30]. Nesse trabalho, os autores mostraram que é possível realizar inferência apenas com aritmética inteira em CPUs ARM mantendo uma precisão próxima da obtida em FP32, desde que os fatores de escala por camada sejam bem estimados.

Atualmente, existem duas estratégias principais de quantização. A quantização pós-treino (*post-training quantization*, PTQ) é aplicada a um modelo já treinado em FP32, recorrendo a um pequeno conjunto de dados de calibração representativo. Não exige re-treino, o que permite uma implementação rápida. Já o QAT simula o efeito da quantização durante a fase de *forward* no treino, permitindo ao modelo adaptar-se ao ruído introduzido e, em muitos casos, alcançar melhor precisão do que o PTQ, sobretudo quando se usam níveis de quantização mais agressivos [30].

Segundo Francy e Singh [32], a quantização INT8 oferece um dos melhores compromissos entre precisão e latência entre várias técnicas de compressão aplicadas a modelos CNN em cenários de edge AI. No caso do Hailo-8, o funcionamento é nativamente em INT8. Por isso, todos os modelos implementados nesta plataforma têm de ser quantizados para INT8 através do passo de PTQ no Hailo Dataflow Compiler. Este processo utiliza um conjunto de dados de calibração fornecido pelo utilizador para estimar as estatísticas de ativação de cada camada, conforme descrito na Secção 3.4.3.

Tabela 2.2: Baseline performance no dataset COCO val2017

Model Scale	Precision	Calibration	mAP50-95	mAP50
N	FP32 (TRT)	N/A	0.4047	0.5686
N	FP16 (TRT)	N/A	0.4044	0.5685
N	Dynamic UINT8 (ONNX)	N/A	0.4047	0.5686
N	Static INT8 (TRT)	Clean	0.3325	0.4787
N	Static INT8 (TRT)	Mixed	0.3327	0.4789
S	FP32 (TRT)	N/A	0.4763	0.6462
S	FP16 (TRT)	N/A	0.4763	0.6464
S	Dynamic UINT8 (ONNX)	N/A	0.4763	0.6462
S	Static INT8 (TRT)	Clean	0.4114	0.5670
S	Static INT8 (TRT)	Mixed	0.4114	0.5670
M	FP32 (TRT)	N/A	0.5226	0.6943
M	FP16 (TRT)	N/A	0.5222	0.6939
M	Dynamic UINT8 (ONNX)	N/A	0.5226	0.6943
M	Static INT8 (TRT)	Clean	0.4853	0.6517
M	Static INT8 (TRT)	Mixed	0.4853	0.6517
L	FP32 (TRT)	N/A	0.5347	0.7049
L	FP16 (TRT)	N/A	0.5345	0.7050
L	Dynamic UINT8 (ONNX)	N/A	0.5347	0.7049
L	Static INT8 (TRT)	Clean	0.5079	0.6802
L	Static INT8 (TRT)	Mixed	0.5079	0.6803
X	FP32 (TRT)	N/A	0.5514	0.7194
X	FP16 (TRT)	N/A	0.5513	0.7193
X	Dynamic UINT8 (ONNX)	N/A	0.5514	0.7194
X	Static INT8 (TRT)	Clean	0.5202	0.6881
X	Static INT8 (TRT)	Mixed	0.5032	0.6659

2.5.2 Pruning por Magnitude

O *pruning* de redes neuronais remove pesos redundantes, reduzindo o número de parâmetros e, no caso do *pruning* estruturado, os FLOPs. Distinguem-se duas abordagens principais: o *pruning* não estruturado, que remove pesos individuais de menor magnitude e impõe esparsidade sem alterar a topologia da rede, e o *pruning* estruturado, que remove filtros ou canais completos, produzindo modelos densos compatíveis com *runtimes* de inferência standard [20]. Bhalgaonkar et al. [20] identificam o *pruning* estruturado como a abordagem preferível para *deployment* em *edge*, dado que o *pruning* não estruturado requer suporte de hardware para computação esparsa, que a maioria dos aceleradores *edge*, incluindo o Hailo-8, não dispõe.

Na prática, o *pruning* implica tipicamente uma fase de *fine-tuning* após a remoção de pesos, de modo a recuperar parte da precisão perdida, uma vez que a imposição de esparsidade degrada o desempenho do modelo antes da readaptação [20]. O compromisso entre taxa de compressão

e degradação de precisão depende da arquitectura específica e da distribuição dos pesos, sendo avaliado empiricamente na Secção 4.5.3 deste trabalho.

2.5.3 Impacto na Precisão e Latência

A quantização e o *pruning* introduzem inevitavelmente alguma degradação de precisão face ao modelo de referência em FP32. A magnitude desta degradação depende da arquitectura do modelo, da estratégia de compressão adoptada e das características do domínio de aplicação. Em geral, a transição FP32 para FP16 introduz degradação residual inferior a 1 ponto percentual de mAP [31], enquanto a quantização INT8 pode introduzir degradação mais expressiva, em particular em modelos treinados exclusivamente em precisão de vírgula flutuante sem QAT [30, 32].

Em deteção de objetos, esta degradação não é uniforme: modelos diferentes podem responder de forma distinta à mesma estratégia de quantização, e a perda observada em conjuntos de validação genéricos nem sempre coincide com o comportamento em dados do mundo real. Por isso, a avaliação deve ser feita sempre no domínio de aplicação específico, de modo a medir o compromisso real entre eficiência e precisão.

O *pruning* por magnitude não destrutivo, que zera pesos sem alterar a estrutura do modelo, permite avaliar a tolerância estrutural de cada arquitectura à remoção de pesos sem redução efectiva de FLOPs ou tamanho em disco [20, 32]. O impacto combinado destas técnicas sobre a precisão e latência dos modelos avaliados neste trabalho é quantificado empiricamente na Secção 4.5.

2.6 Datasets e Métricas de Avaliação

A avaliação de modelos de deteção de objectos requer datasets com anotações de referência que permitam comparação reproduzível entre abordagens. Esta secção apresenta dois benchmarks amplamente utilizados na literatura, KITTI e VisDrone, cuja diversidade de cenários permite analisar o comportamento dos modelos em contextos visuais significativamente distintos.

2.6.1 KITTI

O *KITTI Vision Benchmark Suite* [33] é um dos benchmarks mais influentes na investigação em condução autónoma e deteção de objectos em cenários rodoviários ao nível do solo. Recolhido em Karlsruhe, Alemanha, com um veículo de investigação equipado com câmaras estéreo sincronizadas, LiDAR de 64 feixes e sensores GPS/IMU, o conjunto de dados foi concebido para suportar múltiplas tarefas de percepção, incluindo deteção 2D e 3D, estimativa de profundidade e *tracking*. A componente de deteção de objectos inclui oito classes: *car*, *van*, *truck*, *pedestrian*, *Person_sitting*, *cyclist*, *tram* e *misc*.

O KITTI tornou-se uma referência importante para avaliar modelos destinados a aplicações automóveis, devido à presença de objectos em diferentes escalas, distâncias e graus de oclusão, frequentemente num ambiente urbano e suburbano visualmente complexo. Além disso, o dataset define níveis de dificuldade amplamente usados na literatura, o que facilitou a comparabilidade

entre métodos e contribuiu para a sua adoção como benchmark padrão em deteção rodoviária [7, 33]. Por estas razões, o KITTI continua a ser um dataset relevante para estudar o compromisso entre precisão, robustez e eficiência computacional em cenários de condução autónoma.

2.6.2 VisDrone

O *VisDrone2019-DET* [34] é um benchmark de deteção de objectos em imagens aéreas captadas por drone, amplamente utilizado para avaliar modelos em cenários com elevada densidade de instâncias e objectos de pequena dimensão. O conjunto cobre várias cidades da China e inclui condições variadas de altitude, iluminação e contexto urbano, o que o torna particularmente desafiante do ponto de vista da generalização. As classes consideradas são *pedestrian*, *people*, *bicycle*, *car*, *van*, *truck*, *tricycle*, *awning-tricycle*, *bus* e *motor*.

Em comparação com benchmarks ao nível do solo, o VisDrone introduz dificuldades adicionais devido à escala reduzida dos objectos, à sobreposição frequente entre instâncias e ao forte desequilíbrio entre classes. Estes fatores tornam-no um dataset especialmente exigente para modelos de deteção [34]. A sua utilização é particularmente útil em estudos sobre deteção em ambientes aéreos, vigilância urbana e análise de tráfego por drones, onde a presença de objectos pequenos e densamente agrupados compromete a eficácia de arquiteturas menos robustas.

2.6.3 Métricas de Avaliação

Na literatura de deteção de objectos, a precisão é normalmente avaliada através de métricas baseadas no *mean Average Precision* (mAP), que combinam a qualidade da classificação com a qualidade da localização das caixas delimitadoras. Estas métricas dependem do *Intersection over Union* (IoU), usado para medir o grau de sobreposição entre a previsão do modelo e a anotação de referência. A definição formal das métricas utilizadas neste trabalho, bem como o respetivo protocolo de cálculo, é apresentada na Secção 3.5 da metodologia.

Em tarefas de deteção em tempo real, a eficiência computacional é igualmente relevante. Por esse motivo, trabalhos recentes consideram não só a precisão, mas também métricas como FPS e latência, que permitem avaliar o compromisso entre desempenho e custo computacional em cenários de execução na *edge*.

2.6.4 Comparação Entre Datasets

Os dois datasets são intencionalmente contrastantes nas suas características visuais e nos desafios que colocam aos modelos. O KITTI representa cenários ao nível do solo com objectos de dimensões variadas, normalmente mais favoráveis à deteção convencional, enquanto o VisDrone representa cenários aéreos com objectos de dimensões muito reduzidas, elevada densidade e maior probabilidade de oclusão. Esta complementaridade permite avaliar a robustez dos modelos em dois regimes visuais distintos e analisar como a escala, a densidade e a perspetiva influenciam o desempenho.

Do ponto de vista experimental, a combinação destes datasets é particularmente útil porque obriga os modelos a lidar com desafios diferentes de localização, classificação e generalização. Assim, a comparação entre resultados no KITTI e no VisDrone fornece uma leitura mais completa do comportamento das arquiteturas estudadas, tanto em precisão como em eficiência.

2.7 Lacunas da Literatura e Motivação do Estudo

A revisão da literatura apresentada nas secções anteriores permite identificar um conjunto de lacunas que motivam o trabalho experimental desenvolvido nesta dissertação.

Em primeiro lugar, a maioria dos estudos sobre *deployment* de modelos de deteção em hardware *edge* avalia plataformas com GPU embarcada, como a família NVIDIA Jetson, ou aceleradores amplamente documentados como o Google Coral Edge TPU [15]. O Hailo-8, apesar do seu rácio TOPS/Watt superior, permanece pouco estudado na literatura académica independente, em particular no que respeita ao impacto da sua pipeline de compilação obrigatória para INT8 sobre a precisão de modelos treinados em *float32*. A ausência de uma caracterização empírica sistemática do Hailo-8 com modelos de deteção modernos em datasets de referência constitui uma lacuna diretamente abordada neste trabalho.

Em segundo lugar, o YOLO26n, lançado em 2026, dispõe ainda de literatura publicada por terceiros muito limitada [11]. As afirmações dos autores sobre a sua robustez à quantização INT8 e a redução de latência face ao YOLOv11 carecem de validação empírica independente, em particular em hardware NPU dedicado. Este trabalho contribui com uma das primeiras avaliações do YOLO26n em condições de *deployment* real, comparando-o diretamente com o YOLOv8n, um modelo de referência amplamente documentado na literatura, nas mesmas condições experimentais.

Em terceiro lugar, os estudos sobre compressão de modelos para *edge* tendem a analisar quantização e *pruning* de forma isolada e em plataformas genéricas [20, 32]. A análise conjunta do impacto da quantização imposta pelo pipeline Hailo e do *pruning* por magnitude sobre arquiteturas com paradigmas distintos, nomeadamente *anchor-free* com DFL no YOLOv8n e *anchor-free* sem DFL e sem NMS no YOLO26n, não está documentada na literatura disponível.

Em quarto lugar, a validação de sistemas de deteção e *tracking* em tempo real em hardware embarcado é frequentemente reportada apenas em condições sintéticas de *throughput*, sem validação em cenários reais de captura com câmara [7]. Este trabalho inclui uma validação em condições reais com o sistema completo de inferência, *tracking* e contagem por linha virtual, demonstrando a viabilidade operacional do *pipeline* num cenário de utilização concreto.

Em conjunto, estas lacunas definem o espaço de contribuição desta dissertação: a caracterização empírica do compromisso entre precisão e eficiência no *deployment* de modelos de deteção leves no Raspberry Pi 5 com acelerador Hailo-8, em dois domínios visuais contrastantes, com análise do impacto das técnicas de compressão aplicadas e validação em condições reais de operação.

Capítulo 3

Metodologia

Este capítulo descreve a abordagem metodológica adotada para avaliar a viabilidade da deteção de objetos em tempo real em dispositivos de computação embarcada de baixo consumo energético. São apresentados o problema experimental, os critérios de seleção dos modelos e dos conjuntos de dados, o protocolo de treino e avaliação, as métricas utilizadas e o ambiente experimental completo.

3.1 Definição do Problema Experimental

Os métodos convencionais de visão computacional mostram-se, frequentemente, insuficientes quando se exige um processamento rápido, robusto e adaptável em ambientes dinâmicos [2, 4]. A sua aplicação em ambientes com recursos limitados, nomeadamente: sistemas embebidos, dispositivos móveis ou plataformas de computação na *edge*, apresenta desafios significativos ao nível da latência, capacidade de processamento e eficiência energética [2].

O problema central desta dissertação consiste em avaliar a viabilidade da deteção de objetos em tempo real em dispositivos de computação embarcada de baixo consumo energético, quantificando o compromisso entre precisão e eficiência computacional quando se transitam modelos de *deep learning* de um ambiente de treino convencional para hardware de inferência na *edge*.

A deteção de objetos em tempo real impõe requisitos simultâneos e frequentemente contraditórios: por um lado, exige precisão suficiente para que as deteções sejam fiáveis em cenários de aplicação prática; por outro, exige uma cadência de processamento que acompanhe o fluxo de imagens em tempo real, tipicamente a partir de 25 *frames* por segundo (FPS), valor de referência para câmaras de videovigilância de tráfego [7]. Em plataformas de alto desempenho, com acesso a GPUs dedicadas, estes requisitos são satisfeitos sem dificuldade [2]. O desafio surge quando se pretende deslocar a inferência para a *edge*, ou seja, executar o modelo diretamente no dispositivo de captura, sem recurso a infraestrutura de computação remota, em condições de memória, potência e largura de banda severamente limitadas [4].

Neste trabalho, o problema é operacionalizado através de uma abordagem experimental comparativa. São selecionados modelos de deteção leves, treinados e avaliados em dois conjuntos de

dados com características distintas: o KITTI, representativo de cenários de condução autónoma ao nível do solo, e o VisDrone, representativo de cenas aéreas com objetos de pequenas dimensões e elevada densidade. O desempenho dos modelos é medido em três plataformas de inferência: um processador central (CPU) convencional, uma unidade de processamento gráfico (GPU) e uma unidade de processamento neural (NPU) embarcada, o acelerador Hailo-8 integrado no Raspberry Pi AI HAT+.

Esta estrutura experimental permite isolar o efeito da plataforma de inferência sobre as métricas de precisão e eficiência, e avaliar em que medida as técnicas de compressão de modelos aplicadas afetam a qualidade das deteções face ao modelo de referência em precisão de vírgula flutuante [4].

A hipótese central deste trabalho é que a inferência na NPU Hailo-8 permite alcançar cadências de processamento adequadas a aplicações de tempo real. Assume-se que a eventual degradação de precisão resulta sobretudo da sensibilidade de cada arquitetura aos processos de compressão dos respetivos modelos, e não de uma limitação inerente à computação na *edge*.

3.2 Escolha dos Modelos

A seleção dos modelos foi orientada por três critérios principais: adequação ao hardware embarcado, diversidade arquitetural para comparação significativa entre paradigmas de deteção, e compatibilidade com o *pipeline* de exportação para o formato HEF do Hailo-8. A seleção final recaiu sobre três modelos que cobrem paradigmas arquiteturais distintos: um detector moderno especificamente otimizado para a *edge* (YOLO26n), um detector *anchor-free* consolidado como referência na literatura (YOLOv8n), e um detector *anchor-based* de geração anterior (SSDLite). Modelos de maior capacidade, como o YOLOv8s ou o RT-DETR, foram excluídos por excederem os requisitos de leveza para *deployment* em hardware embarcado. Modelos baseados em *transformers*, como o DETR, foram igualmente excluídos por apresentarem compatibilidade limitada com o *pipeline* de compilação Hailo na versão do DFC utilizada.

O YOLO26n [11, 12] constitui o modelo central deste estudo. Lançado pela Ultralytics em janeiro de 2026, foi concebido especificamente para inferência em dispositivos de baixo consumo. A sua arquitetura elimina a *Distribution Focal Loss* (DFL), simplificando a regressão de caixas delimitadoras e melhorando a compatibilidade com aceleradores de hardware. A inferência é realizada *end-to-end* sem NMS, o que reduz a latência em até 43% face ao YOLOv11 na variante nano em CPU [11]. O suporte nativo a quantização INT8 com degradação mínima de precisão é particularmente relevante dado que o Hailo-8 opera exclusivamente neste modo.

O YOLOv8n [9] é a variante nano da família YOLOv8, adotada como modelo de referência. Trata-se do detector *anchor-free* mais amplamente utilizado na literatura e na indústria, com suporte maduro para exportação ONNX e *pipelines* Ultralytics. A sua arquitetura CNN com cabeça *one2many* e pós-processamento NMS contrasta deliberadamente com o YOLO26n, permitindo isolar o impacto das inovações arquiteturais mais recentes sobre as métricas de desempenho, tanto em precisão como em comportamento face à quantização INT8.

O SSDLite320-MobileNetV3-Large combina a arquitetura *Single Shot MultiBox Detector* [5] com o *backbone* MobileNetV3-Large [19], representando uma abordagem de deteção leve consolidada, anterior à dominância da família YOLO. O SSD, proposto em 2016, introduziu a deteção em múltiplas escalas num único passe de rede, eliminando a necessidade de uma fase separada de proposta de regiões. A variante SSDLite substitui as convoluções padrão por convoluções separáveis em profundidade, reduzindo significativamente o número de parâmetros e operações. A sua inclusão neste estudo é metodologicamente motivada: sendo uma arquitetura de geração anterior à família YOLO moderna, com paradigma de deteção anchor-based e backbone MobileNetV3, serve como linha de base arquitetural que permite contextualizar os ganhos de desempenho introduzidos pelas inovações mais recentes em termos de precisão de deteção e eficiência computacional.

3.3 Preparação dos Datasets

Foram utilizados dois datasets públicos com características intencionalmente contrastantes: o KITTI, representativo de cenários de condução autónoma ao nível do solo, e o VisDrone, representativo de cenas aéreas com objetos de pequenas dimensões e elevada densidade. Esta escolha visa avaliar a generalização dos modelos a domínios visuais distintos.

3.3.1 KITTI

O KITTI [33] é um dataset de condução autónoma ao nível do solo, contendo imagens captadas a partir de um veículo equipado com câmara estéreo em ambiente urbano e suburbano. Integra 8 classes de objetos: *car*, *van*, *truck*, *pedestrian*, *Person_sitting*, *cyclist*, *tram* e *misc*. As anotações foram convertidas para o formato YOLO pelo *pipeline* da Ultralytics, sem alterações às classes ou filtragem de instâncias.

O KITTI não disponibiliza anotações públicas para o seu conjunto de teste oficial, o que inviabiliza a utilização da partição original para avaliação rigorosa. Foi por isso adotada uma partição própria com proporção 70/15/15 (treino/validação/teste), gerada de forma estratificada por classe dominante por imagem, utilizando a biblioteca `scikit-learn` com `seed=42` para garantir reprodutibilidade. Esta estratégia assegura que a distribuição de classes se mantém aproximadamente uniforme entre os três subconjuntos, evitando enviesamentos de avaliação. O conjunto de treino contém 5 236 imagens, o de validação 1 122 imagens e o de teste 1 123 imagens.

3.3.2 VisDrone

O VisDrone2019-DET [34] é um dataset de deteção de objetos em imagens aéreas captadas por drone. Integra 10 classes de objetos: *pedestrian*, *people*, *bicycle*, *car*, *van*, *truck*, *tricycle*, *awning-tricycle*, *bus* e *motor*, caracterizado por cenas de elevada densidade e objetos de pequenas dimensões, com uma densidade média de objetos por imagem substancialmente superior à do KITTI, o que aumenta consideravelmente a dificuldade da tarefa de deteção. As anotações originais foram convertidas para o formato YOLO, com filtragem prévia das regiões marcadas

como ignoradas pelos autores do dataset, ou seja, instâncias ambíguas ou de avaliação excluída, identificadas pela *flag* de exclusão presente em cada anotação.

Foi utilizada a partição oficial VisDrone2019-DET, o que garante comparabilidade direta com resultados publicados na literatura para este dataset. O conjunto de treino contém 6 471 imagens, o de validação 548 imagens e o de teste 1 610 imagens, num total de 8 629 imagens.

3.4 Protocolo Experimental

3.4.1 Protocolo de Treino

Os três modelos foram inicializados a partir de pesos pré-treinados no dataset COCO, seguindo uma estratégia de *fine-tuning*. O treino foi realizado numa GPU NVIDIA RTX 2060, com um máximo de 100 épocas e critério de paragem antecipada (*early stopping*) com paciência de 20 épocas, interrompendo o treino caso o mAP@50 na validação não melhorasse durante 20 épocas consecutivas. A *seed* foi fixada em 42 para garantir reprodutibilidade dos resultados.

Os modelos YOLO26n e YOLOv8n foram treinados com o *framework* Ultralytics, com o otimizador SGD, *learning rate* inicial de 0.01, *momentum* de 0.937 e *weight decay* de 0.0005. A resolução de entrada foi de 640×640 píxeis com *batch size* de 4. O parâmetro `amp=False` foi definido explicitamente para garantir estabilidade numérica durante o treino. A augmentação de dados foi mantida com as configurações predefinidas da Ultralytics.

O SSDLite320-MobileNetV3-Large foi treinado com PyTorch, com o mesmo otimizador SGD e hiperparâmetros equivalentes sempre que aplicável, utilizando um *scheduler* CosineAnnealingLR e critério de paragem antecipada manual baseado no mAP@50 na validação. O mesmo esquema de épocas máximas e *seed* foi aplicado para garantir comparabilidade entre os três modelos.

Todos os modelos foram treinados de forma independente em cada um dos dois datasets, resultando em seis experiências de treino no total. Os hiperparâmetros utilizados encontram-se resumidos na Tabela 3.1.

3.4.2 Avaliação em PC

Após o treino, cada modelo foi avaliado sobre o conjunto de teste do respetivo dataset, separadamente em GPU e em CPU, de forma a caracterizar o desempenho em ambos os contextos de execução disponíveis na máquina de treino. A avaliação em GPU foi realizada com aceleração CUDA na NVIDIA RTX 2060 enquanto a avaliação em CPU foi realizada no processador AMD Ryzen 7 4800H, sem recurso a aceleração por hardware dedicado. As métricas recolhidas em cada contexto encontram-se descritas na Secção 3.5. Os resultados obtidos são apresentados no Capítulo 4.

Tabela 3.1: Hiperparâmetros de treino utilizados em todos os modelos.

Hiperparâmetro	**YOLO26n / YOLOv8n**	**SSDLite**
Épocas máximas	100	100
Early stopping patience	20	20
Resolução de entrada	640×640	320×320
Batch size	4	4
Otimizador	SGD	SGD
Learning rate inicial	0.01	0.01
Momentum	0.937	0.937
Weight decay	0.0005	0.0005
Scheduler	CosineLR decay com warmup	CosineAnnealingLR
AMP	False	False
Seed	42	42
Pesos iniciais	COCO	COCO

3.4.3 Exportação e Compilação para o Hailo-8

Para a avaliação no hardware embarcado, os modelos YOLO26n e YOLOv8n foram exportados para o formato ONNX com *opset* 11, requisito de compatibilidade com o *parser* do Hailo Dataflow Compiler (DFC). O SSDLite não foi incluído nesta fase por incompatibilidade estrutural com o pipeline de compilação Hailo: a arquitetura SSDLite320-MobileNetV3-Large, treinada com os pesos COCO v1 do torchvision, exporta operações de pós-processamento (decode de anchors e NMS) fundidas no grafo ONNX, que o parser do DFC não consegue traduzir corretamente para o formato HAR. Adicionalmente, o desempenho obtido pelo SSDLite nos datasets utilizados revelou-se substancialmente inferior ao dos modelos YOLO, em particular no VisDrone, onde a resolução de entrada fixa de 320×320 píxeis se mostrou inadequada para a deteção de objetos de pequenas dimensões característicos deste dataset. Por estas razões, a avaliação do SSDLite é reportada exclusivamente para as plataformas PC, servindo como linha de base arquitetural para comparação com os modelos YOLO modernos.

A compilação para o formato HEF (*Hailo Executable Format*) compreendeu três etapas sequenciais. Na primeira, o modelo ONNX foi traduzido para o formato HAR (*Hailo Archive*) através do *parser* do DFC. Na segunda, o modelo HAR foi quantizado para INT8 com pesquisa exaustiva de particionamento para maximizar o desempenho em inferência, recorrendo a *Quantization-Aware Training* (QAT) com 4 épocas de *fine tuning* com *learning rate* de 10^{-4}, utilizando 1024 imagens de calibração extraídas do conjunto de treino de cada dataset. Na terceira, o modelo HAR quantizado foi compilado para o formato HEF com a arquitetura alvo correspondente ao hardware físico disponível, o Hailo-8. Todo o processo foi executado na máquina de treino, em ambiente WSL2, utilizando o Hailo DFC v3.33.1.

3.4.4 Avaliação no Hardware Embarcado

A avaliação no Raspberry Pi 5 foi realizada em dois modos de inferência distintos: através da NPU Hailo-8, utilizando os modelos no formato HEF, e através do CPU ARM Cortex-A76, utilizando os modelos no formato ONNX via ONNXRuntime 1.26.0 com quatro *threads* de execução. Em ambos os modos, a avaliação foi conduzida sobre o conjunto de teste completo de cada dataset.

O protocolo de medição incluiu um período de aquecimento prévio, durante o qual as primeiras imagens processadas foram descartadas para eliminar efeitos de inicialização do sistema. A latência por *frame* foi medida de forma a incluir o pré-processamento da imagem, a inferência propriamente dita e o pós-processamento das deteções, permitindo uma caracterização realista do tempo de resposta total do sistema. As métricas de precisão foram calculadas com interpolação de 101 pontos sobre o conjunto de teste completo, com limiar de confiança mínimo de 0.001 para maximizar a cobertura das deteções na curva de precisão-revocação. As métricas recolhidas encontram-se descritas na Secção 3.5 e os resultados obtidos são apresentados no Capítulo 4.

3.4.5 Sistema de Inferência em Tempo Real

Para complementar a avaliação em dataset com uma validação em condições reais, foi desenvolvido um sistema de inferência em tempo real para o Raspberry Pi 5, operando com os modelos YOLO26n compilados para o Hailo-8. O sistema é descrito nas subsecções seguintes, cobrindo o pipeline de inferência, o algoritmo de tracking e re-identificação, e o mecanismo de contagem de tráfego. Os parâmetros do sistema de tracking encontram-se resumidos na Tabela 3.2.

3.4.5.1 Pipeline de Inferência

O pipeline de inferência em tempo real foi implementado em Python, operando diretamente no Raspberry Pi 5 com acesso à NPU Hailo-8 através da API HailoRT. O sistema estrutura-se em duas fases sequenciais, separadas por *design* para maximizar o desempenho durante a gravação.

Na primeira fase, é apresentada ao utilizador uma janela de configuração interativa com pré-visualização ao vivo da câmara, implementada em OpenCV. O utilizador pode selecionar o modelo a utilizar (treinado no KITTI ou no VisDrone), ajustar o brilho da imagem, definir a posição da linha virtual de contagem e especificar a duração da gravação. Esta fase inclui um modo de teste com inferência ao vivo que permite verificar o desempenho do sistema antes de iniciar a gravação formal. Como é possível verificar na Figura 3.1, estes parâmetros são configurados de forma interativa na interface gráfica antes do início da gravação. A janela é encerrada ao iniciar a gravação, libertando os recursos de renderização.

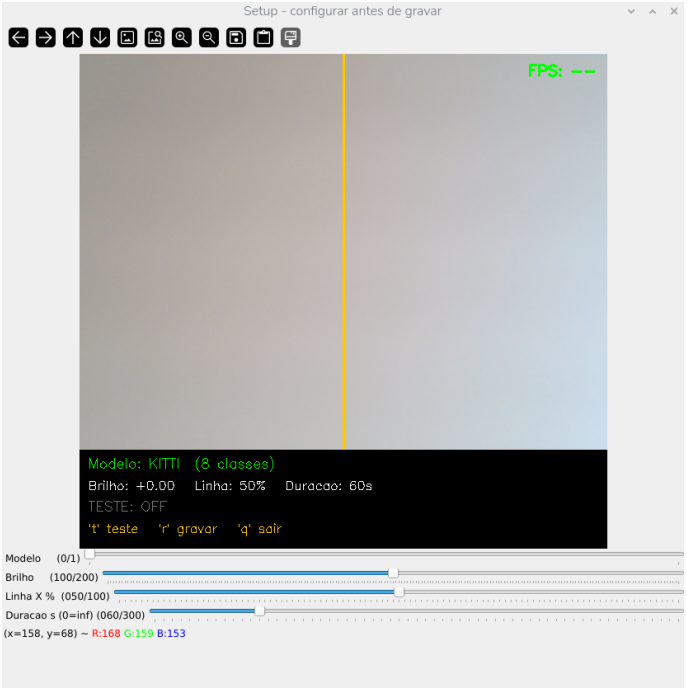


Figura 3.1: Interface gráfica do sistema de inferência em tempo real no Raspberry Pi 5.

Na segunda fase, o pipeline executa em ciclo contínuo as seguintes etapas: captura de *frame* via `picamera2`, pré-processamento da imagem, inferência na NPU Hailo-8 via HailoRT, pós-processamento das deteções com filtragem por limiar de confiança e área mínima, *tracking* de objetos, atualização das contagens, e gravação do *frame* processado em ficheiro de vídeo MP4. No final de cada sessão, as estatísticas são exportadas para ficheiro JSON, incluindo FPS médio, latências percentílicas (p50, p95), contagens de cruzamento por classe e direção, e pico de objetos simultâneos por classe.

3.4.5.2 Algoritmo de Tracking e Re-identificação

O *tracking* de objetos entre *frames* consecutivas é formulado como um problema de associação entre deteções e *tracks* ativos. Para cada par de *frames*, é construída uma matriz de distâncias euclidianas entre os centroides dos *tracks* existentes e os centroides das novas deteções. A associação ótima é obtida pelo *Hungarian algorithm*, que minimiza a soma total das distâncias de associação garantindo uma atribuição globalmente ótima, em contraste com heurísticas de associação *greedy*. Esta formulação segue a abordagem do algoritmo SORT (*Simple Online and Realtime Tracking*) [26], que estabeleceu o uso do *Hungarian algorithm* combinado com distância euclidiana como solução eficiente para *multi-object tracking* em tempo real em visão computacional. Esta abordagem é preferível ao *nearest-neighbour* simples em cenários com múltiplos objetos próximos, onde associações conflituosas são frequentes. Pares com distância superior a 90 píxeis são rejeitados, evitando associações entre objetos distantes.

Para lidar com oclusões temporárias, foi implementado um mecanismo de tolerância a *frames* sem deteção: um *track* só é eliminado após 20 *frames* consecutivas sem associação, correspondendo a aproximadamente 0.4 segundos a uma cadência típica de 50 FPS. Este valor foi definido empiricamente para equilibrar a robustez a oclusões curtas com a eliminação atempada de *tracks* de objetos que abandonaram a cena.

A classificação de cada objeto é estabilizada por um mecanismo de votação temporal, motivado pela observação de *flickering* de classe nos testes iniciais, onde o mesmo objeto oscilava entre duas classes em *frames* consecutivas. Para cada *track*, é mantido um contador de votos por classe ao longo da sua vida. A classe é considerada estável quando a classe maioritária acumula pelo menos 3 votos e representa no mínimo 70% do total de votos. Esta estratégia de filtro temporal é consistente com abordagens de estabilização de classificação em *pipelines* de *tracking* em tempo real.

Tabela 3.2: Parâmetros do sistema de tracking do pipeline em tempo real.

Parâmetro	**Descrição**	**Valor**
Distância máxima de associação	Limiar de matching entre centroides	90 px
Frames de tolerância	Máximo de *frames* sem deteção	20
Área mínima de caixa	Filtro de deteções demasiado pequenas	32 px²
Votos mínimos para estabilização	Confirmação de classe por votação	3
Rácio mínimo de votos	Consistência mínima de classe	0.7
Idade mínima de track	*Frames* antes de track ser contabilizado	5
Duração do cemitério	Período de re-identificação	175 *frames*
Raio de re-identificação	Distância máxima para ressuscitar track	50 px

3.4.5.3 Contagem por Linha Virtual

A contagem de objetos é realizada por uma linha virtual vertical, configurável pelo utilizador através de *trackbar* na fase de *setup*, expressa como percentagem da largura da imagem. O mecanismo de deteção de cruzamento baseia-se na transição do centroide de cada *track* relativamente à posição da linha entre *frames* consecutivas, abordagem estabelecida em sistemas de contagem de tráfego baseados em *tracking* [26]. Um cruzamento da esquerda para a direita (LR) é registado quando o centroide se encontrava à esquerda da linha no *frame* anterior e passa para a direita no *frame* atual. O cruzamento inverso (RL) é registado na condição simétrica.

Para evitar contagens incorretas de objetos com classificação instável ou recém-criados, o cruzamento só é registado para *tracks* que satisfaçam simultaneamente três condições: a classe deve estar estabilizada pelo mecanismo de votação descrito na secção anterior, o *track* deve ter pelo menos 5 *frames* de vida, e cada *track* é contabilizado no máximo uma vez. Esta última condição evita duplas contagens em situações em que o centroide oscila em torno da linha virtual por múltiplos *frames* consecutivos.

O sistema regista adicionalmente o pico de objetos simultâneos por classe ao longo da sessão, calculado apenas sobre *tracks* visíveis, estáveis e com idade mínima suficiente, fornecendo uma

métrica complementar à contagem de cruzamentos para caracterizar a densidade de tráfego na cena observada.

3.4.6 Análise de Compressão de Modelos

Com o objetivo de explorar o potencial de compressão dos modelos estudados e o seu impacto no desempenho de deteção, foram conduzidas duas análises complementares fora do pipeline principal de *deployment*: uma análise de quantização em diferentes precisões numéricas e uma análise de pruning por magnitude. Estas experiências foram realizadas exclusivamente sobre os modelos YOLO26n e YOLOv8n treinados no conjunto de dados KITTI, sendo apresentadas no Capítulo 4 como uma exploração complementar do espaço de otimização.

A quantização foi selecionada como primeira técnica de análise por ser o método de compressão mais diretamente relevante para o contexto deste trabalho. No caso da plataforma Hailo-8, a execução do modelo implica a sua conversão para uma representação quantizada em INT8, pelo que esta etapa não constitui uma opção de otimização adicional, mas sim um requisito inerente ao pipeline de deployment. A análise em diferentes níveis de precisão, nomeadamente FP32, FP16 e INT8, permite assim avaliar em que medida cada etapa de conversão afeta a precisão do modelo e o seu comportamento computacional. O pruning foi selecionado como técnica complementar por fornecer uma perspetiva distinta sobre o espaço de compressão disponível. Enquanto a quantização reduz a precisão numérica dos pesos e ativações existentes, o pruning promove esparsidade através da remoção de parâmetros considerados menos relevantes, permitindo analisar a tolerância estrutural dos modelos à compressão.

3.4.6.1 Quantização

Na análise de quantização, os modelos YOLO26n e YOLOv8n foram convertidos para representações ONNX em diferentes precisões numéricas, nomeadamente FP32, FP16 e INT8. As versões em FP32 e FP16 foram utilizadas para avaliar o impacto da redução de precisão na preservação do desempenho de deteção e no custo computacional do modelo.

A versão INT8 foi obtida por quantização estática pós-treino, recorrendo a um conjunto de calibração constituído por 200 imagens representativas do conjunto de validação do KITTI. Na quantização estática, os dados de calibração são utilizados para estimar os intervalos de ativação do modelo e calcular os parâmetros necessários à representação em INT8.

Para a quantização INT8, foi adotado o formato QDQ (*Quantize-DeQuantize*), com quantização de pesos e ativações em `QInt8`. A avaliação das versões FP32 e FP16 incidiu sobre as métricas *mAP*@50 e *mAP*@50-95, bem como sobre o tamanho em disco do modelo, o FPS e a latência. As medições de desempenho temporal foram realizadas separadamente em CPU e GPU, com um protocolo de aquecimento e medição controlado, permitindo isolar o efeito da precisão numérica nas características computacionais do modelo.

3.4.6.2 Pruning

A análise de pruning foi realizada através de pruning não estruturado por magnitude L1 aplicado aos pesos das camadas convolucionais (`Conv2d`) dos modelos YOLO26n e YOLOv8n. Em concreto, para cada camada convolucional, foi aplicada uma estratégia de remoção dos pesos de menor magnitude segundo a norma L1, impondo esparsidade sem alterar a topologia global da rede.

Foram avaliados cinco níveis de pruning, correspondentes a 10%, 20%, 30%, 40% e 50% dos pesos em cada camada convolucional. Após cada operação de pruning, o modelo foi sujeito a uma fase de *fine-tuning* com 30 épocas, com o objetivo de recuperar parte do desempenho perdido devido à imposição de esparsidade. Durante esta fase, foi utilizado um *learning rate* inicial de 0.005 e uma paciência de *early stopping* de 10 épocas, mantendo-se, sempre que aplicável, o restante protocolo de treino definido anteriormente.

Em cada nível de pruning, os modelos resultantes foram avaliados no conjunto de teste do KITTI com base nas métricas $mAP@50$ e $mAP@50\text{-}95$, sendo igualmente calculada a esparsidade real obtida após a aplicação das máscaras de pruning. Dado tratar-se de pruning não estruturado e não destrutivo, a estrutura do modelo e o tamanho em disco permanecem, em termos práticos, inalterados, pelo que esta análise deve ser interpretada sobretudo como uma avaliação exploratória da tolerância dos modelos à esparsidade, e não como uma compressão estrutural efetiva com redução direta de FLOPs ou parâmetros armazenados.

Importa ainda clarificar que os modelos efetivamente implementados nas plataformas embarcadas, descritos nas secções de resultados experimentais, não incorporam pruning. Esta opção deve-se ao facto de o pipeline de compilação para Hailo já impor quantização em INT8, introduzindo por si só uma degradação de precisão que precisava de ser cuidadosamente controlada. A aplicação adicional de pruning sobre modelos já sujeitos a quantização poderia agravar essa degradação sem garantia de benefício prático no hardware alvo. Assim, as análises de quantização e pruning devem ser entendidas como estudos exploratórios e independentes, destinados a caracterizar o comportamento dos modelos sob diferentes formas de compressão, e não como etapas cumulativas do pipeline final de deployment.

3.5 Métricas de Avaliação

A avaliação comparativa dos modelos assenta em duas dimensões complementares: a precisão das deteções produzidas e a eficiência computacional durante a inferência. A escolha das métricas seguiu os critérios estabelecidos na literatura para tarefas de deteção de objetos em tempo real, privilegiando métricas que permitam comparação direta entre modelos e entre plataformas de execução.

3.5.1 Precisão de Deteção

A métrica central é o *mean Average Precision* (mAP), que agrega o desempenho de deteção ao longo de múltiplos limiares de confiança e de sobreposição espacial. O cálculo baseia-se no conceito de *Intersection over Union* (IoU), que quantifica a sobreposição entre a caixa delimitadora prevista pelo modelo e a anotação de referência, definido como a razão entre a área de interseção e a área de união das duas caixas, conforme a Equação 3.1.

$$\text{IoU} = \frac{\text{Área de Interseção}}{\text{Área de União}} \tag{3.1}$$

O mAP@50 considera uma deteção como verdadeiro positivo quando o IoU é superior a 0,50. Embora seja o limiar mais utilizado na literatura e permita comparação com a maioria dos trabalhos publicados, tende a ser permissivo na avaliação da qualidade de localização. O mAP@50-95 representa a média do AP calculado para limiares de IoU entre 0,50 e 0,95 com passo de 0,05, constituindo uma avaliação significativamente mais exigente, penalizando localizações imprecisas mesmo quando a classe é corretamente identificada. Esta métrica é adotada como referência principal nas competições COCO [35] e é considerada mais representativa do desempenho real em aplicações práticas.

A *Precision* mede a fração de deteções produzidas pelo modelo que são efetivamente corretas, sendo relevante em cenários onde falsos positivos têm custo elevado. O *Recall* mede a fração de objetos presentes nas imagens que o modelo consegue detetar, sendo crítico em aplicações onde a omissão de objetos é inaceitável. O F1-*score* corresponde à média harmónica entre *Precision* e *Recall*, conforme a Equação 3.2.

$$F1 = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.2}$$

Todas estas métricas são calculadas globalmente sobre o conjunto de teste de cada dataset, para cada combinação modelo/dataset.

3.5.2 Eficiência Computacional

As métricas de eficiência são avaliadas separadamente em CPU e GPU, de forma a caracterizar o comportamento de cada modelo nos diferentes contextos de inferência considerados. O FPS (*frames per second*) traduz diretamente a capacidade de processamento em tempo real, sendo a métrica mais intuitiva para comparar modelos face aos requisitos de resposta. A latência média por *frame*, expressa em milissegundos, complementa esta análise ao quantificar o tempo de processamento por imagem, sendo reportada juntamente com o desvio padrão para dar conta da variabilidade observada entre *frames*. O tamanho do modelo em disco, expresso em MB, e o número de parâmetros, expresso em milhões, caracterizam a pegada de memória de cada arquitetura, com implicações diretas para a viabilidade de *deployment* em hardware embarcado.

A medição de FPS e latência em PC foi realizada com imagens sintéticas geradas aleatoriamente, com as mesmas dimensões de entrada de cada modelo, constituindo um *benchmark* de *throughput* puro independente do conteúdo visual. Foi utilizado um período de aquecimento prévio de 50 iterações e a latência foi calculada como média de 200 iterações. No Raspberry Pi 5, a latência foi medida sobre o conjunto de teste completo de cada *dataset*, incluindo pré-processamento e pós-processamento das deteções.

3.6 Ambiente Experimental

O ambiente experimental divide-se em duas componentes distintas: o ambiente de treino e avaliação em PC, e o ambiente de inferência embarcada no Raspberry Pi 5.

3.6.1 Ambiente de Treino (PC)

O treino e a avaliação em PC foram realizados numa máquina equipada com uma GPU NVIDIA RTX 2060 com suporte a CUDA 12.8, e um processador AMD Ryzen 7 4800H com 8 núcleos a 2.90 GHz. O sistema operativo utilizado foi o Windows 11 com WSL2 (Ubuntu), onde foram executados todos os scripts de treino e compilação. O *framework* principal foi o Ultralytics para os modelos YOLO26n e YOLOv8n, e o PyTorch com `torchvision` para o SSDLite320-MobileNetV3-Large. A compilação dos modelos para o formato HEF do Hailo foi igualmente realizada nesta máquina, utilizando o Hailo Dataflow Compiler (DFC) v3.33.1 em ambiente WSL2.

3.6.2 Ambiente de Inferência Embarcada (Raspberry Pi 5)

A avaliação em hardware embarcado foi realizada num Raspberry Pi 5 com 8GB de memória LPDDR4X, equipado com o Raspberry Pi AI HAT+ que integra o acelerador NPU Hailo-8 com 26 TOPS de desempenho nominal em operações INT8. A captura de imagem foi realizada com o Camera Module 3, equipado com sensor Sony IMX708 de 12MP com *autofocus* e suporte a HDR. O sistema operativo instalado foi o Raspberry Pi OS 64-bit (Bookworm), com HailoRT 4.23.0 e Python 3.13. A inferência foi realizada em dois modos: NPU Hailo-8 com modelos no formato HEF, e CPU ARM Cortex-A76 com modelos ONNX via ONNXRuntime.

A Tabela 3.3 resume as especificações de ambos os ambientes.

Tabela 3.3: Especificações dos ambientes experimentais.

Componente	**PC (Treino)**	**Raspberry Pi 5 (Inferência)**
Processador	AMD Ryzen 7 4800H, 8 *cores*, 2.90 GHz	ARM Cortex-A76, 4 *cores*, 2.4 GHz
Memória RAM	16 GB	8 GB LPDDR4X
Acelerador	NVIDIA RTX 2060 (CUDA 12.8)	Hailo-8 NPU (26 TOPS INT8)
Câmara	—	Sony IMX708 12MP (*Camera Module 3*)
Sistema Operativo	Windows 11 + WSL2 (Ubuntu)	Raspberry Pi OS 64-bit (Bookworm)
Framework ML	Ultralytics, PyTorch, `torchvision`	HailoRT 4.23.0, ONNXRuntime
Python	3.12.3	3.13
Compilador Hailo	DFC v3.33.1	—

3.7 Considerações Éticas

A condução deste trabalho levanta várias questões éticas relacionadas com o uso de dados visuais, a privacidade dos intervenientes e as potenciais implicações de sistemas de visão computacional em contextos de mobilidade e vigilância. A literatura recente em ética da visão computacional e da curadoria de dados sublinha riscos associados à recolha massiva de imagens de pessoas sem consentimento informado, à presença de enviesamentos nos *datasets* e a usos secundários não previstos, sobretudo quando os dados são obtidos por *web scraping* ou em espaços públicos [36, 37]. Estes trabalhos apontam para a necessidade de integrar, desde a fase de conceção, princípios de minimização de dados, respeito pelos direitos fundamentais e transparência quanto às finalidades de tratamento dos dados visuais [38, 39].

Todas as gravações de vídeo e testes de inferência dos modelos descritos neste capítulo foram realizados em espaços privados, não livremente acessíveis ao público, e em cenários controlados. Os intervenientes potencialmente identificáveis nas gravações (por exemplo, condutores ou peões presentes nas proximidades do sistema) foram previamente informados sobre os objetivos do estudo, o tipo de dados recolhidos e a forma como estes seriam utilizados, tendo dado o seu consentimento explícito para participação e utilização das imagens exclusivamente para fins de investigação académica. Não foram captados nem tratados identificadores biométricos (como reconhecimento facial ou reidentificação de identidade), e as gravações não foram reutilizadas para qualquer finalidade comercial ou de vigilância.

Para o treino e avaliação dos modelos foram utilizados *datasets* públicos amplamente estabelecidos na comunidade científica, nomeadamente o KITTI e o VisDrone, disponibilizados pelos respetivos autores para fins de investigação sob licenças que autorizam o seu uso académico. A recolha e anotação destas bases de dados foi realizada por terceiros, de acordo com os termos e

condições publicados, não tendo sido captadas imagens adicionais de pessoas ou matrículas no âmbito deste projeto. Sempre que são apresentados exemplos visuais na dissertação, as imagens correspondem a amostras destes *datasets* públicos ou a gravações próprias em ambiente privado, nas quais não é possível identificar de forma fiável indivíduos específicos, evitando assim a exposição de informação pessoal sensível [37].

No que respeita à funcionalidade dos modelos, o sistema desenvolvido limita-se à deteção de classes genéricas de objetos (*car*, *pedestrian*, *bus*, etc.), não incluindo qualquer módulo de reconhecimento facial, reidentificação de pessoas ou extração de características biométricas. A infraestrutura de validação em tempo real não integra mecanismos de armazenamento prolongado, partilha sistemática ou cruzamento das imagens com outras bases de dados, adotando um paradigma de processamento local e de minimização dos dados tratados sempre que possível. Esta abordagem está alinhada com recomendações internacionais que enfatizam a importância da limitação de finalidade, da minimização de dados e do desenho *privacy by design* em sistemas de visão computacional aplicados em contextos sensíveis [38, 39].

Em termos de enquadramento jurídico e institucional, a videovigilância em espaço público e em espaços privados de acesso ao público é objeto de regulamentação específica em Portugal, sendo o seu uso fortemente condicionado por razões de proporcionalidade, necessidade e respeito pelos direitos fundamentais. A Comissão Nacional de Proteção de Dados (CNPD) tem reiterado, em diversos pareceres, que a utilização de sistemas de videovigilância para vigilância generalizada da via pública acarreta riscos significativos de vigilância em massa e compressão desrazoável do direito à privacidade, devendo ser reservada a situações excecionais e, em regra, às forças e serviços de segurança ao abrigo de bases legais próprias [40, 41]. Neste trabalho, não foi instalado qualquer sistema permanente de videovigilância nem foram realizadas gravações contínuas da via pública para fins de segurança ou fiscalização, evitando assim a configuração de um cenário de vigilância massiva.

Em consonância com o referido enquadramento, todas as experiências de validação em tempo real foram conduzidas em cenário controlado e em espaço privado, com duração limitada ao estritamente necessário para recolher as métricas de desempenho apresentadas nos capítulos seguintes. Nos casos em que o campo de visão das câmaras pudesse, de forma residual, abranger porções de via pública, foram adotadas medidas técnicas e organizativas para minimizar a área captada e evitar a identificação de terceiros não participantes, em linha com o princípio da minimização e com as recomendações doutrinárias sobre videovigilância e compressão da privacidade [41].

Importa ainda sublinhar que o trabalho aqui apresentado tem natureza estritamente experimental e não envolve decisões automáticas com impacto direto em direitos, liberdades ou oportunidades de pessoas concretas. Os resultados obtidos são utilizados exclusivamente para comparar arquiteturas e *pipelines* de *deployment* em contexto de investigação académica, não existindo qualquer integração do sistema em plataformas de monitorização ou controlo em produção. A literatura em ética da inteligência artificial e em sistemas de videovigilância inteligentes destaca, porém, que soluções deste tipo, quando transpostas para contextos reais de segurança rodoviária ou vigilância, devem ser sujeitas a avaliações de impacto, supervisão humana efetiva e mecanismos

de responsabilização claros, incorporando princípios de beneficência, não maleficência, justiça e explicabilidade [36, 38, 39].

Por fim, este trabalho não endossa nem incentiva o uso de modelos de visão computacional para finalidades de vigilância intrusiva ou discriminação de grupos específicos. O enquadramento adotado assume a visão computacional como ferramenta para aumentar a segurança rodoviária, apoiar a gestão de tráfego e contribuir para o avanço do estado da arte em investigação académica, e não como instrumento de monitorização individualizada. Qualquer esforço futuro que pretenda transpor abordagens semelhantes para ambientes operacionais deverá incorporar explicitamente mecanismos adicionais de proteção de dados, transparência e participação dos *stakeholders*, em consonância com as recomendações internacionais para sistemas de IA confiáveis e respeitadores dos direitos fundamentais [39].

Capítulo 4

Resultados Experimentais

Este capítulo apresenta os resultados experimentais obtidos seguindo o protocolo descrito no Capítulo 3. Para cada plataforma e *dataset* são reportadas as métricas de precisão e eficiência definidas na Secção 3.5.

4.1 Pipeline de Implementação

O desenvolvimento experimental seguiu um *pipeline* estruturado em cinco fases sequenciais, que transformam os modelos treinados em artefactos prontos para inferência nas plataformas alvo. A Figura 4.1 ilustra este fluxo de forma esquemática, identificando para cada bloco a plataforma de execução e os artefactos produzidos.

Figura 4.1: Pipeline completo de implementação, desde o treino até à inferência nas plataformas alvo.

O Bloco 1 corresponde ao treino dos modelos em PC com GPU, utilizando o *framework* Ultralytics para o YOLO26n e YOLOv8n e o PyTorch com `torchvision` para o SSDLite, produzindo os pesos `best.pt` ou `best.pth` para cada combinação modelo/*dataset* (Secção 3.4.1). O Bloco 2 corresponde à avaliação de inferência em PC, tanto em GPU via Ultralytics como em CPU via ONNXRuntime, sobre os conjuntos de teste completos de cada *dataset*. Os resultados obtidos determinaram quais os modelos a avançar para as fases seguintes (Secção 4.4). O Bloco 3 corresponde à exportação dos modelos YOLO26n e YOLOv8n para o formato ONNX com *opset* 11, após validação dos resultados em PC. O SSDLite não foi exportado pelas razões descritas na Secção 3.4.3. O Bloco 4 corresponde à compilação para o formato HEF através do Hailo Dataflow Compiler v3.33.1, com quantização INT8 e *fine-tuning* pós-quantização (QAT), exclusivamente

para os modelos YOLO26n e YOLOv8n (Secção 3.4.3). O Bloco 5 corresponde à inferência embarcada no Raspberry Pi 5, em dois modos: NPU Hailo-8 com os modelos HEF, e CPU ARM Cortex-A76 com os modelos ONNX via ONNXRuntime (Secção 4.6).

A Tabela 4.1 resume os artefactos produzidos na fase de treino para cada combinação modelo/*dataset*, incluindo o número de épocas efectuadas e o tempo total de execução.

Tabela 4.1: Artefactos produzidos na fase de treino.

Modelo	**Dataset**	**Épocas**	**Tempo de Treino**	**Observação**
YOLO26n	KITTI	100	6h 41m	Convergência normal
YOLO26n	VisDrone	100	9h 08m	Convergência normal
YOLOv8n	KITTI	100	5h 05m	Convergência normal
YOLOv8n	VisDrone	100	6h 27m	Convergência normal
SSDLite	KITTI	100	4h 43m	Convergência normal
SSDLite	VisDrone	35	2h 08m	*Early stopping*

O SSDLite não foi compilado para o formato HEF nem avaliado no Raspberry Pi 5, pelas razões descritas em detalhe na Secção 3.4.3. Por este motivo, está ausente das Secções 4.5 e 4.6.

4.2 Resultados de Treino

Esta secção apresenta os resultados obtidos durante o treino e a avaliação offline dos modelos nos conjuntos de teste do KITTI e do VisDrone. A análise centra-se na evolução das métricas ao longo das épocas, na comparação entre arquiteturas e na interpretação do comportamento por classe, de forma a caracterizar o desempenho dos modelos antes da sua exportação e execução em plataformas embarcadas.

4.2.1 KITTI

Nesta subsecção analisa-se a evolução do desempenho dos modelos no KITTI ao longo do treino, com base nas métricas de convergência e no comportamento observado durante o processo de otimização. Tal análise complementa o protocolo de treino descrito no Capítulo 3.4.1, no qual se definiram as condições experimentais, os hiperparâmetros utilizados e o critério de paragem antecipada. As Figuras 4.2 e 4.3 apresentam a evolução do mAP@50 e do mAP@50-95 ao longo das épocas, permitindo avaliar a rapidez de convergência e a estabilidade de cada arquitetura.

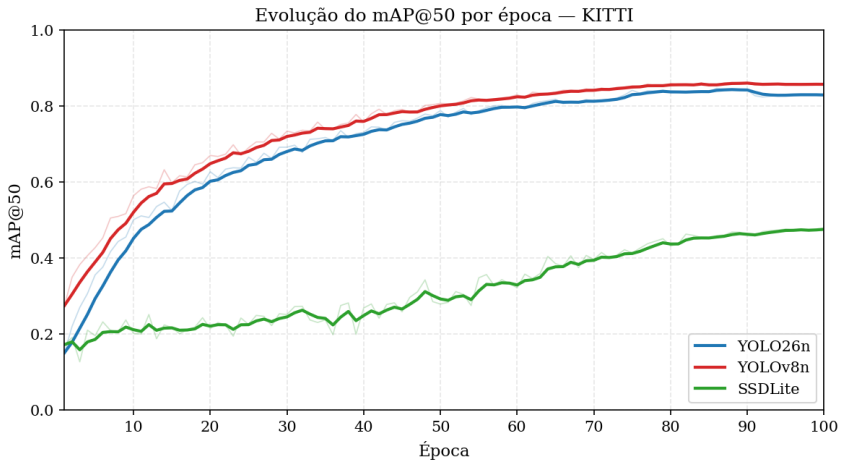


Figura 4.2: Evolução do mAP@50 durante o treino no KITTI.

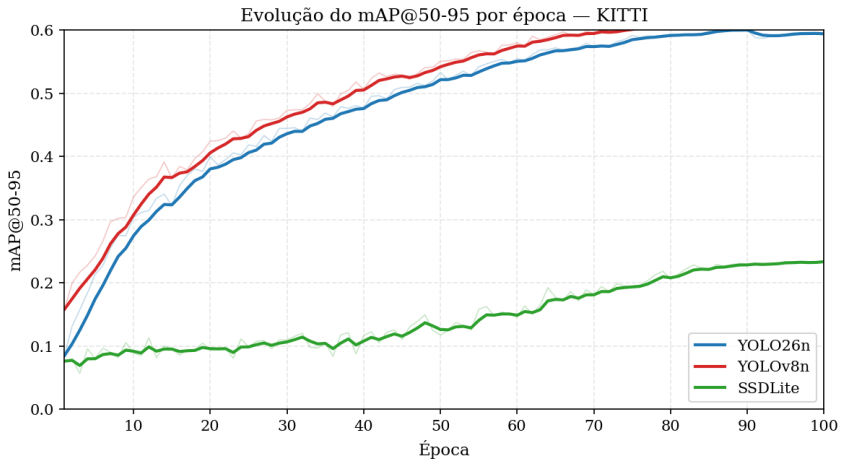


Figura 4.3: Evolução do mAP@50-95 durante o treino no KITTI.

Como se pode observar na Figura 4.2, o YOLO26n e o YOLOv8n convergem de forma estável ao longo das 100 épocas, sem sinais de *overfitting*, com as duas curvas a evoluir em paralelo desde as primeiras épocas. O YOLOv8n mantém uma vantagem consistente ao longo de todo o treino, atingindo mAP@50 de 0.858 no final contra 0.837 do YOLO26n. A Figura 4.3 confirma o mesmo padrão para o mAP@50-95, com o YOLOv8n a atingir 0.616 e o YOLO26n 0.594. O SSDLite apresenta uma progressão significativamente mais lenta em ambas as métricas, não ultrapassando mAP@50 de 0.45 nem mAP@50-95 de 0.23 ao fim das 100 épocas, e sem evidenciar uma assíntota clara, o que sugere que o modelo não atingiu convergência plena neste domínio.

A Figura 4.4 apresenta a evolução da *validation loss* ao longo do treino para os três modelos.

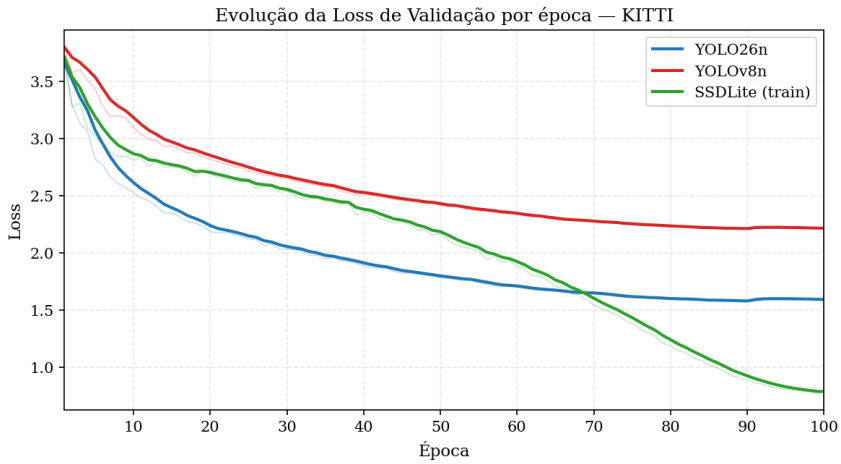


Figura 4.4: *Validation loss* por época durante o treino no KITTI.

A *validation loss* do YOLO26n converge para valores inferiores aos do YOLOv8n (1.59 vs. 2.21 na época 100), o que não implica melhor desempenho no conjunto de teste. Esta diferença reflete a estrutura distinta das funções de custo dos dois modelos: o YOLO26n, sem DFL, opera num espaço de representação mais compacto que produz valores de *loss* numa escala diferente. A comparação da *loss* entre arquiteturas distintas não é significativa, pelo que os resultados no conjunto de teste constituem a métrica de referência. A curva apresentada para o SSDLite corresponde à *training loss*, uma vez que este modelo não regista *validation loss* por época no formato compatível com os modelos YOLO, não sendo portanto diretamente comparável.

A Figura 4.5 apresenta a evolução de Precision e a Figura 4.6 a evolução de Recall ao longo das épocas de treino para os três modelos.

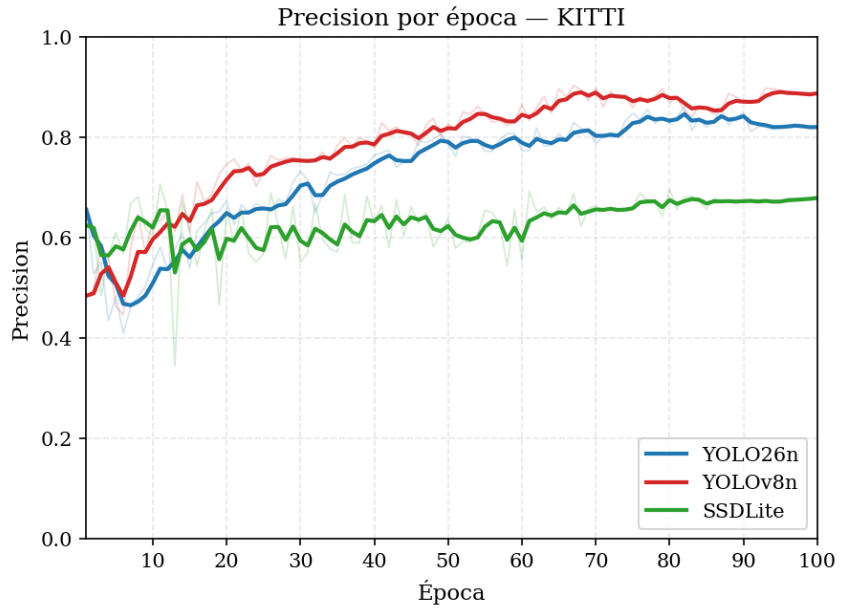


Figura 4.5: Evolução de Precision durante o treino no KITTI.

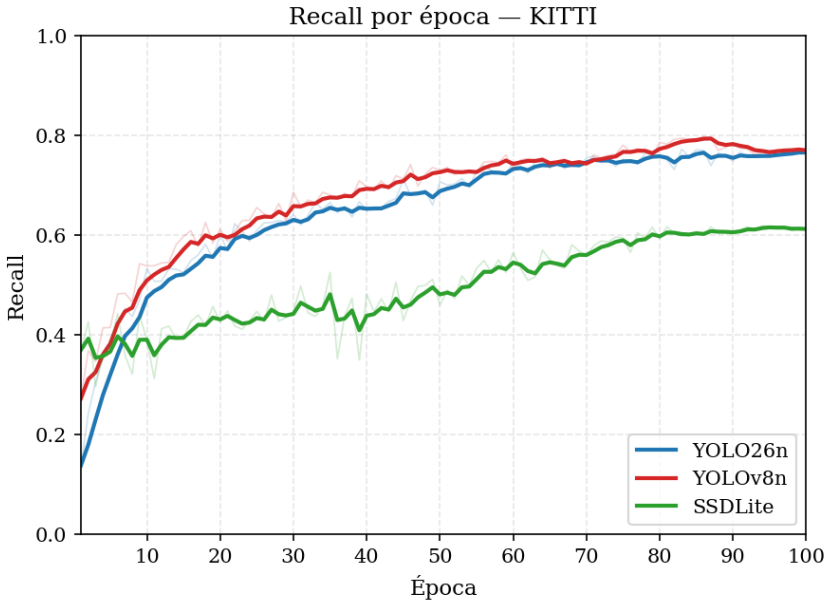


Figura 4.6: Evolução de Recall durante o treino no KITTI.

As curvas de Precision e Recall confirmam o padrão observado nas métricas globais. O YOLOv8n mantém uma Precision consistentemente superior ao longo de todo o treino, atingindo valores próximos de 0.89 nas épocas finais, contra 0.82 do YOLO26n. O Recall converge para valores semelhantes em ambos os modelos YOLO (0.77), o que indica que a vantagem do YOLOv8n reside na qualidade das deteções produzidas e não na quantidade de objetos detetados. O SSDLite estabiliza com Precision de 0.67 e Recall de 0.61, valores que refletem as limitações da resolução de entrada e a dificuldade de adaptação dos pesos COCO ao domínio KITTI.

A avaliação final do desempenho dos modelos no conjunto de teste do KITTI é apresentada mais adiante na Subsecção 4.3.1.

4.2.2 VisDrone

Nesta subsecção apresenta-se a evolução do desempenho dos modelos no VisDrone ao longo do treino, com base nas métricas de convergência e no comportamento observado durante o processo de otimização. Dado tratar-se de um *dataset* mais exigente do que o KITTI, com elevada densidade de objetos, classes desbalanceadas e instâncias de pequenas dimensões, a análise foca-se na estabilidade das curvas e na rapidez de convergência de cada arquitetura. As Figuras 4.7 e 4.8 mostram a evolução do mAP@50 e do mAP@50-95 ao longo das épocas.

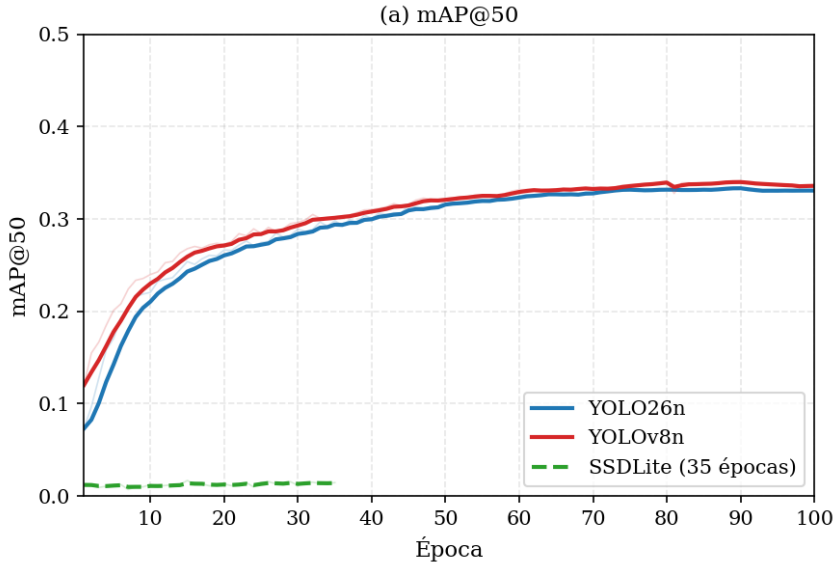


Figura 4.7: Evolução do mAP@50 durante o treino no VisDrone.

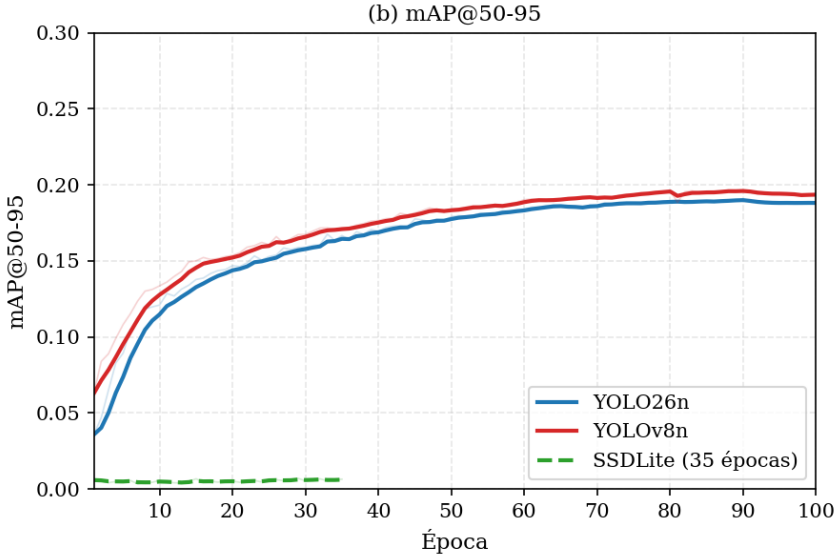


Figura 4.8: Evolução do mAP@50-95 durante o treino no VisDrone.

As curvas de mAP ao longo do treino, apresentadas nas Figuras 4.7 e 4.8, mostram que os dois modelos YOLO convergem de forma paralela e estável, sem sinais de *overfitting*, atingindo contudo um patamar significativamente inferior ao KITTI, reflexo direto da dificuldade intrínseca do domínio aéreo. O SSDLite, representado com linha tracejada até à época 35 onde ocorreu *early*

stopping, não evidencia qualquer progressão significativa, com mAP@50 máximo de 0.014, valor praticamente nulo que confirma a falha de adaptação ao domínio.

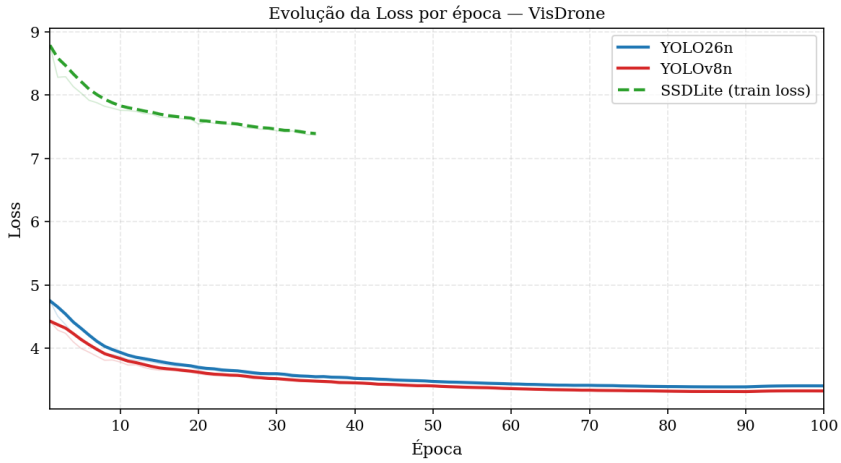


Figura 4.9: *Validation loss* por época durante o treino no VisDrone.

A *validation loss*, apresentada na Figura 4.9, mostra descida estável nos dois modelos YOLO ao longo das 100 épocas, sem divergência. A curva apresentada para o SSDLite corresponde à *training loss*, que estagna nas primeiras épocas, não sendo diretamente comparável com os modelos YOLO pelas razões já descritas na Secção 4.2.1.

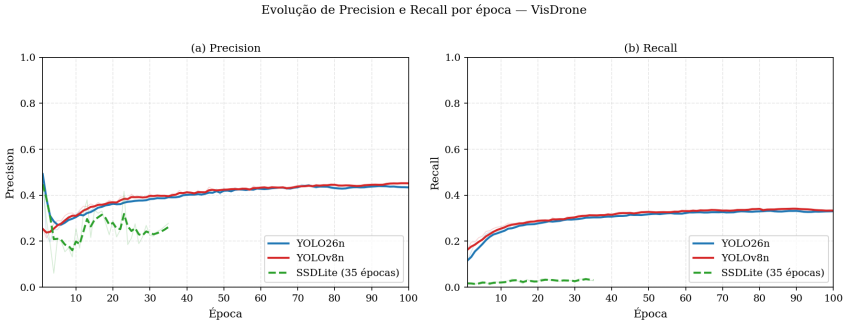


Figura 4.10: Evolução de Precision e Recall durante o treino no VisDrone.

As curvas de Precision e Recall confirmam que o YOLO26n e o YOLOv8n apresentam desempenhos muito próximos ao longo de todo o treino, convergindo para Precision próxima de 0.44 e Recall de 0.33 nas curvas de validação por época, valores superiores aos obtidos no conjunto de teste final (Tabela 4.4), diferença consistente com a maior dificuldade da partição de teste oficial face ao conjunto de validação. Note-se que o YOLO26n apresenta Recall marginalmente superior no conjunto de teste (0.304 vs. 0.302), sendo o único indicador global em que supera o YOLOv8n. O SSDLite apresenta um comportamento característico nas épocas iniciais: a Precision parte de valores elevados com Recall próximo de zero, colapsando abruptamente até à época 15. Este padrão resulta do comportamento conservador do modelo nas primeiras épocas de *fine-tuning* num

domínio muito distante do COCO: os pesos pré-treinados produzem poucas deteções de alta confiança, inflacionando artificialmente a Precision inicial; à medida que o treino avança, o número de falsos positivos cresce mais rapidamente do que os verdadeiros positivos, colapsando a Precision para valores próximos de 0.25, com Recall próximo de zero até ao *early stopping* à época 35.

4.3 Avaliação no Conjunto de Teste

Nesta secção apresentam-se os resultados obtidos na avaliação final dos modelos nos conjuntos de teste do KITTI e do VisDrone. A análise centra-se nas métricas globais de desempenho, na avaliação por classe e na interpretação das matrizes de confusão normalizadas, permitindo comparar o comportamento dos diferentes modelos em condições de teste independentes.

4.3.1 KITTI

O conjunto de teste do KITTI é composto por 1 123 imagens distribuídas por 8 classes, obtidas a partir do split estratificado descrito na Secção 3.3.1. A Tabela 4.2 resume as métricas globais obtidas para os três modelos avaliados, incluindo mAP@50, mAP@50-95, Precision, Recall e F1.

Tabela 4.2: Métricas globais no conjunto de teste KITTI.

Modelo	**mAP@50**	**mAP@50-95**	**Precision**	**Recall**	**F1**
YOLO26n	0.837	0.594	0.840	0.765	0.801
YOLOv8n	**0.858**	**0.616**	**0.875**	**0.774**	**0.821**
SSDLite	0.454	0.216	0.672	0.607	0.638

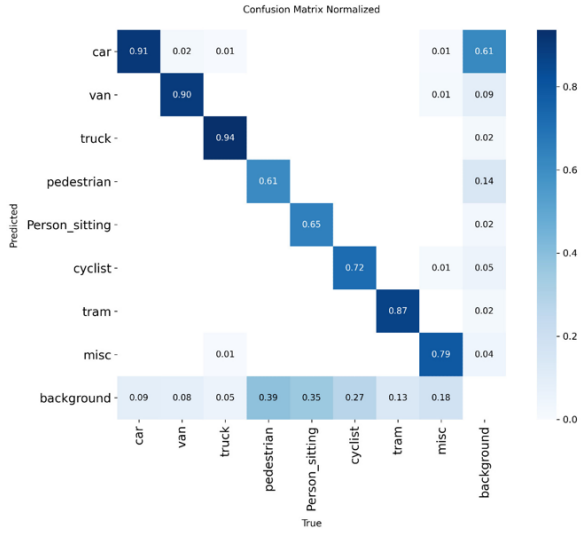
O YOLOv8n obtém o melhor desempenho global no KITTI, com mAP@50 de 0.858 e mAP@50-95 de 0.616, superando o YOLO26n em 2.1 e 2.2 pontos percentuais, respetivamente. A diferença é mais pronunciada na Precision (0.875 vs. 0.840), indicando que o YOLOv8n produz uma proporção menor de falsos positivos. O Recall é próximo entre os dois modelos YOLO (0.774 vs. 0.765), sugerindo capacidade de deteção semelhante. O SSDLite fica claramente abaixo em todas as métricas, com mAP@50 de 0.454, correspondendo a pouco mais de metade do desempenho do YOLOv8n, o que é consistente com as limitações arquiteturais discutidas na Secção 3.2.

Tabela 4.3: AP@50 por classe no conjunto de teste KITTI.

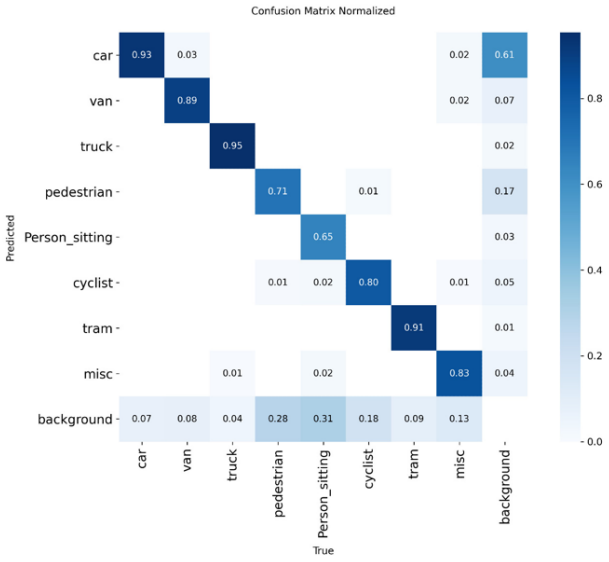
Classe	**YOLO26n**	**YOLOv8n**	**SSDLite**
car	0.950	**0.959**	0.399
van	0.925	**0.928**	0.278
truck	0.975	**0.980**	0.311
pedestrian	0.705	**0.769**	0.112
Person_sitting	0.580	**0.584**	0.085
cyclist	0.791	**0.821**	0.131
tram	0.919	**0.947**	0.242
misc	0.855	**0.875**	0.165

O YOLOv8n supera o YOLO26n em todas as 8 classes, embora a margem seja reduzida nas classes de veículos (car, van, truck, tram), onde ambos os modelos atingem AP@50 acima de 0.91. As classes com menor desempenho são *Person_sitting* (0.580 e 0.584) e *pedestrian* (0.705 e 0.769), refletindo a maior dificuldade de deteção de objetos de pequena dimensão e com sobreposição frequente no domínio de condução. O SSDLite apresenta um colapso generalizado em todas as classes, com AP@50 inferior a 0.40 em todas elas e valores abaixo de 0.15 nas classes pedonais. Esta limitação é de natureza estrutural, decorrente da resolução fixa de 320×320 imposta pela arquitetura, que é insuficiente para a diversidade de escalas presentes no KITTI e não corrigível com treino adicional. Por este motivo, o SSDLite não prossegue para as fases de exportação ONNX, compilação para Hailo-8 e avaliação em plataforma embarcada.

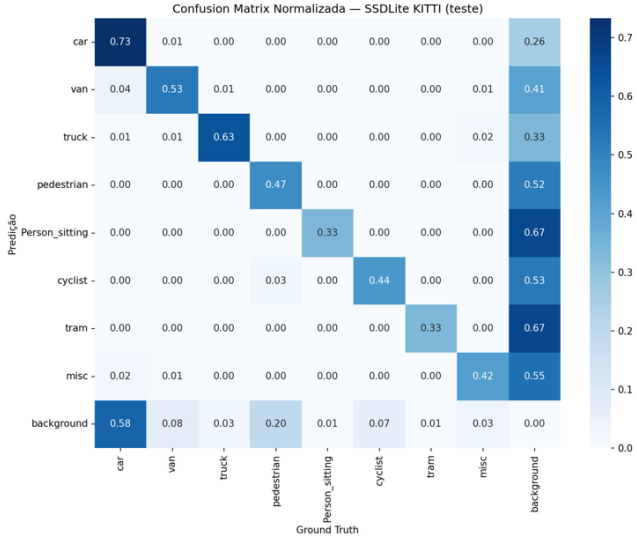
A Figura 4.11 apresenta as matrizes de confusão normalizadas dos três modelos no conjunto de teste do KITTI, complementando a análise quantitativa da Tabela 4.3. Note-se que os valores diagonais da matriz de confusão não correspondem diretamente ao AP@50 reportado na Tabela 4.3: o AP@50 integra a curva precision-recall completa ao longo de múltiplos limiares de confiança, enquanto a matriz de confusão reflete as deteções a um limiar fixo de 0.25.



(a) YOLO26n



(b) YOLOv8n



(c) SSDLite

Figura 4.11: Matrizes de confusão normalizadas no conjunto de teste do KITTI (da esquerda para a direita): YOLO26n, YOLOv8n e SSDLite.

A análise das matrizes de confusão normalizadas confirma e complementa os resultados quantitativos. Nos modelos YOLO, a diagonal é dominante em todas as classes, evidenciando uma classificação maioritariamente correta das deteções produzidas. As classes de veículos de grande dimensão (truck, tram) atingem valores diagonais acima de 0.87 em ambos os modelos, enquanto as classes pedonais (*pedestrian*, *Person_sitting*) apresentam os valores mais baixos, consistentes com os AP@50 reportados na Tabela 4.3. O YOLOv8n supera o YOLO26n na maioria das classes, com exceção do *Person_sitting*, onde o YOLO26n obtém 0.65 face a 0.51 do YOLOv8n, constituindo a única classe em que o YOLO26n apresenta vantagem no KITTI. No SSDLite, o padrão é substancialmente diferente: a coluna *background* compete com a diagonal em todas as classes, indicando que o modelo tende a não detetar objetos presentes na cena em vez de os classificar incorretamente. Este comportamento é particularmente acentuado nas classes *Person_sitting* e tram, onde apenas 33% das instâncias são corretamente detetadas, com os restantes 67% a não serem reconhecidos pelo modelo.

4.3.2 VisDrone

O conjunto de teste do VisDrone é composto por 1 610 imagens distribuídas por 10 classes, correspondendo à partição oficial *VisDrone2019-DET-test-dev*. A Tabela 4.4 apresenta as métricas globais obtidas no conjunto de teste para os três modelos avaliados, em termos de mAP@50, mAP@50-95, Precision, Recall e F1.

Tabela 4.4: Métricas globais no conjunto de teste VisDrone.

Modelo	**mAP@50**	**mAP@50-95**	**Precision**	**Recall**	**F1**
YOLO26n	0.277	0.154	0.391	0.304	0.342
YOLOv8n	**0.279**	**0.157**	**0.394**	0.302	**0.342**
SSDLite	0.010	0.004	0.200	0.017	0.031

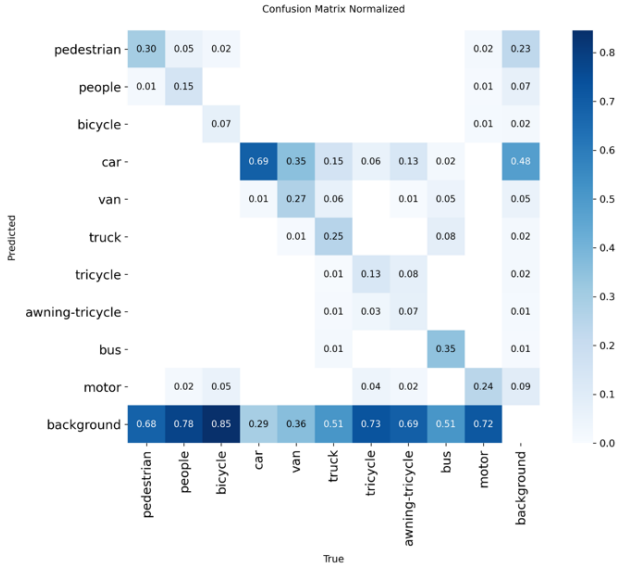
Os resultados no VisDrone são substancialmente inferiores aos obtidos no KITTI em todos os modelos, o que era esperado dado o carácter intrinsecamente mais difícil do *dataset*: cenas aéreas com densidade elevada, objetos de dimensões muito reduzidas e 10 classes com distribuição desequilibrada. O YOLOv8n obtém o melhor desempenho global com mAP@50 de 0.279, marginalmente superior ao YOLO26n (0.277), uma diferença de apenas 0.2 pontos percentuais. Ao contrário do KITTI, onde o YOLOv8n apresentava uma vantagem clara em Precision, no VisDrone os dois modelos YOLO convergem para valores praticamente idênticos em todas as métricas, sugerindo que as diferenças arquiteturais entre os dois modelos têm menor impacto neste domínio. O SSDLite apresenta desempenho residual no VisDrone, com mAP@50 de 0.010 e *early stopping* às 35 épocas, confirmando a incompatibilidade fundamental da sua resolução de entrada com a escala dos objetos presentes neste *dataset*.

Tabela 4.5: AP@50 por classe no conjunto de teste VisDrone.

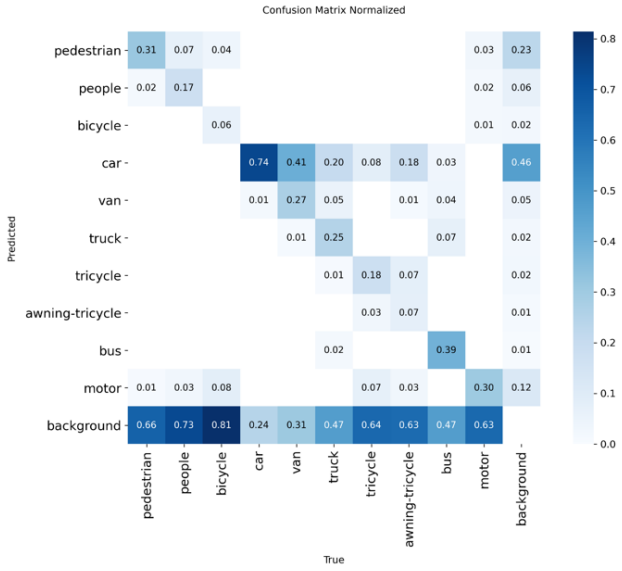
Classe	**YOLO26n**	**YOLOv8n**	**SSDLite**
pedestrian	**0.246**	0.238	0.000
people	**0.143**	0.132	0.000
bicycle	**0.073**	0.057	0.000
car	**0.680**	0.678	0.015
van	**0.308**	0.300	0.004
truck	0.307	**0.346**	0.008
tricycle	0.142	**0.143**	0.000
awning-tricycle	0.132	**0.150**	0.000
bus	0.491	**0.512**	0.013
motor	**0.252**	0.237	0.000

A análise por classe revela padrões distintos dos observados no KITTI. A classe car continua a ser a mais fácil de detetar (AP@50 próximo de 0.68 em ambos os modelos YOLO), beneficiando do seu tamanho relativo superior e da sua representação abundante no conjunto de treino. As classes pedonais (*pedestrian*, *people*) e de pequenas dimensões (*bicycle*, *tricycle*, *awning-tricycle*, *motor*) apresentam AP@50 consistentemente abaixo de 0.25, refletindo a dificuldade em resolver objetos a partir de imagens aéreas. Ao contrário do KITTI, onde o YOLOv8n superava o YOLO26n em todas as classes, no VisDrone a vantagem alterna: o YOLO26n é superior em *pedestrian* (0.246 vs. 0.238), *people* (0.143 vs. 0.132), *bicycle* (0.073 vs. 0.057) e *motor* (0.252 vs. 0.237), enquanto o YOLOv8n supera em *truck* (0.346 vs. 0.307), *bus* (0.512 vs. 0.491) e *awning-tricycle* (0.150 vs. 0.132). O SSDLite apresenta AP@50 nulo em 6 das 10 classes, com valores residuais apenas nas classes de veículos de maior dimensão.

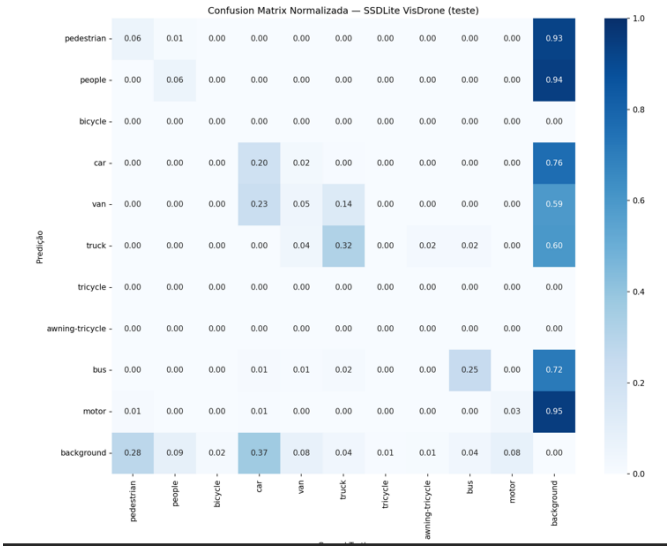
A Figura 4.12 apresenta as matrizes de confusão normalizadas dos três modelos no conjunto de teste do VisDrone. Note-se que os valores diagonais não correspondem diretamente ao AP@50 reportado na Tabela 4.5: o AP@50 integra a curva precision-recall completa ao longo de múltiplos limiares de confiança, enquanto a matriz de confusão reflete as deteções a um limiar fixo de 0.25.



(a) YOLO26n



(b) YOLOv8n



(c) SSDLite

Figura 4.12: Matrizes de confusão normalizadas no conjunto de teste do VisDrone (da esquerda para a direita): YOLO26n, YOLOv8n e SSDLite.

As matrizes de confusão do VisDrone revelam um padrão substancialmente diferente do KITTI. Nos modelos YOLO, a coluna *background* é dominante em quase todas as classes, com valores entre 0.63 e 0.85, indicando que a principal fonte de erro não é a classificação incorreta mas sim a falha de deteção. A classe car é a mais bem detetada em ambos os modelos (0.69 e 0.74 na diagonal), com confusão frequente com van e truck, o que é esperado dado que estas classes partilham características visuais semelhantes em perspetiva aérea. As classes pedonais e de pequenas dimensões apresentam valores diagonais próximos de zero, confirmando os AP@50 reportados na Tabela 4.5. No SSDLite, o colapso é total: *bicycle*, *tricycle* e *awning-tricycle* têm diagonal nula, e as restantes classes têm entre 93% e 95% das instâncias a ir para *background*, confirmando a incompatibilidade estrutural da arquitetura com o domínio aéreo.

Os dois modelos YOLO são avaliados nas secções seguintes nas plataformas de inferência embarcada; o SSDLite não prossegue para estas fases pelas razões já descritas.

4.4 Inferência em PC

Esta secção avalia a eficiência computacional dos modelos nos dois *datasets*, nas modalidades GPU e CPU. O SSDLite é incluído apenas para referência comparativa, dado que não prossegue para as plataformas embarcadas pelas razões descritas na Secção 4.2.

A Tabela 4.6 resume as características estáticas dos três modelos, independentes do *dataset* e da plataforma de inferência.

Tabela 4.6: Características estáticas dos modelos avaliados.

Modelo	**Parâmetros (M)**	**Tamanho (MB)**	**FLOPs (G)**
YOLO26n	2.51	5.1[a]	5.8[b]
YOLOv8n	3.01	6.0[a]	8.2[b]
SSDLite	2.34	9.5[a]	0.85[b]

[a] Tamanho do ficheiro de pesos no disco após treino.

[b] FLOPs calculados para a resolução de entrada de cada modelo: 640×640 para YOLO26n e YOLOv8n, 320×320 para SSDLite (valor medido via `thop`).

O SSDLite apresenta o menor número de parâmetros (2.34 M) e os menores FLOPs declarados (0.85 G), resultado da arquitectura MobileNetV3-Large com convoluções *depthwise separable* e da resolução de entrada reduzida. Contudo, esta eficiência arquitectural não se traduz em melhor desempenho de deteção, como demonstrado nas Secções 4.2.1 e 4.2.2. O YOLOv8n, apesar de apresentar o maior número de parâmetros (3.01 M) e FLOPs (8.2 G), mantém um tamanho em disco reduzido (6.0 MB).

A Tabela 4.7 apresenta os resultados de *benchmark* de inferência em PC para os três modelos nos dois *datasets*.

Tabela 4.7: *Benchmark* de inferência em PC (GPU e CPU). Valores de latência apresentados como média ± desvio padrão.

Modelo	**Dataset**	**FPS$_{GPU}$**	**Lat.$_{GPU}$ (ms)**	**FPS$_{CPU}$**	**Lat.$_{CPU}$ (ms)**
YOLO26n	KITTI	70.6	14.17 ± 2.02	11.1	89.79 ± 4.86
YOLOv8n	KITTI	**110.7**	$\mathbf{9.03 \pm 0.90}$	**11.3**	$\mathbf{88.39 \pm 8.28}$
YOLO26n	VisDrone	73.4	13.61 ± 0.90	11.1	90.47 ± 10.08
YOLOv8n	VisDrone	**109.0**	$\mathbf{9.17 \pm 0.62}$	11.1	89.89 ± 8.80
SSDLite	—[c]	51.6	19.36 ± 1.21	21.1	47.32 ± 11.24

Os valores de FPS e latência do SSDLite são idênticos para KITTI e VisDrone porque foram medidos com tensor sintético de dimensão fixa (320×320), independente do *dataset*. Os valores YOLO foram também medidos com tensor sintético mas em sessões separadas, o que explica as diferenças marginais entre *datasets*.

Em GPU, o YOLOv8n é claramente o modelo mais rápido, atingindo 110.7 FPS no KITTI e 109.0 FPS no VisDrone, com latência média de aproximadamente 9 ms, cerca de 1.5× mais rápido que o YOLO26n (70.6 e 73.4 FPS) e mais de 2× mais rápido que o SSDLite (51.6 FPS em ambos os *datasets*). Esta vantagem do YOLOv8n em GPU é consistente com diferenças arquitecturais que favorecem o paralelismo das suas operações de convolução, nomeadamente convoluções mais largas que são vectorizadas de forma mais eficiente em GPU. O SSDLite, paradoxalmente, é o mais lento em GPU apesar dos menores FLOPs declarados, o que se explica pela sua resolução de entrada inferior (320×320) que reduz o grau de paralelismo e pela menor eficiência das convoluções *depthwise* em GPU face a convoluções densas.

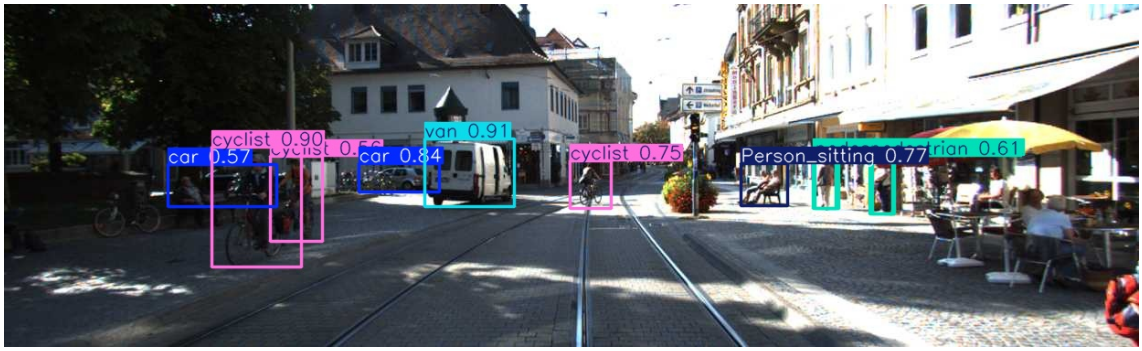
Em CPU, o panorama inverte-se parcialmente. O SSDLite torna-se o modelo mais rápido com 21.1 FPS e latência de 47.32 ± 11.24 ms, quase o dobro dos dois modelos YOLO (11.1–11.3 FPS, ≈89–90 ms), beneficiando directamente da resolução reduzida e das convoluções *depthwise separable* que são mais eficientes em CPU. O YOLO26n e o YOLOv8n apresentam desempenho praticamente idêntico em CPU (≈11.1 FPS), com o YOLOv8n a manter uma vantagem marginal no KITTI (11.3 vs. 11.1 FPS). O desvio padrão elevado da latência CPU do SSDLite (±11.24 ms) reflete maior variabilidade de execução em CPU, consistente com a natureza das operações *depthwise separable* nesta plataforma.

Os valores de FPS e latência reportados foram obtidos com *tensores sintéticos* (imagens aleatórias), constituindo um *benchmark* de *throughput* puro. Latências reais sobre o conjunto de teste podem diferir ligeiramente devido ao *overhead* de carregamento e pré-processamento de imagens reais.

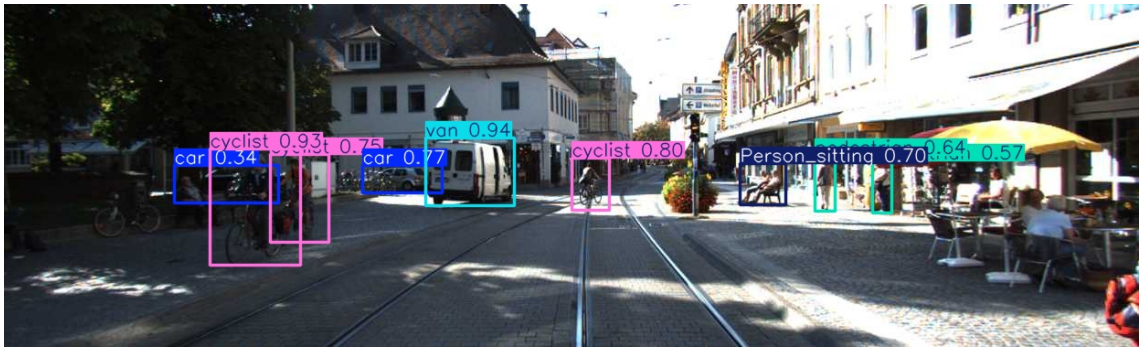
É importante contextualizar estes resultados: o PC não é a plataforma alvo deste trabalho, servindo apenas como referência de desempenho máximo para comparação com as plataformas embarcadas avaliadas nas secções seguintes. Os valores de FPS obtidos em GPU são substancialmente superiores aos requisitos de tempo-real (tipicamente 25 FPS), pelo que ambos os modelos YOLO são adequados para inferência em tempo-real em plataformas GPU dedicadas.

4.4.1 Comparação Visual entre Modelos: KITTI

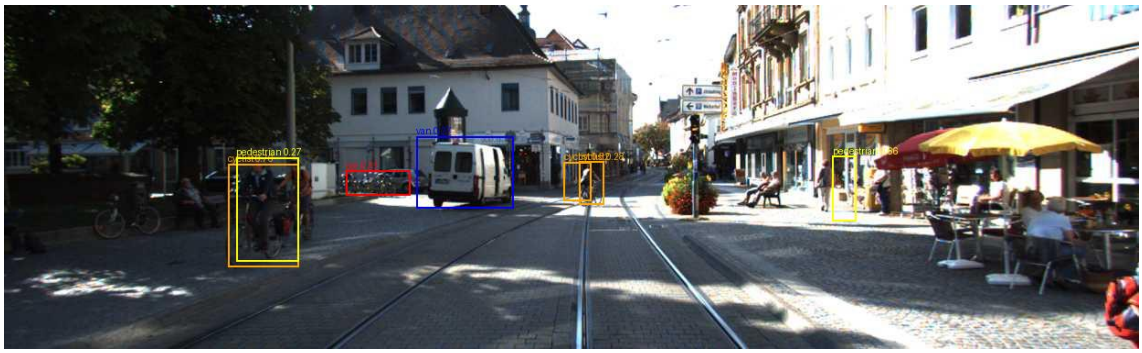
A Figura 4.13 apresenta a comparação visual dos três modelos numa cena representativa do conjunto de teste KITTI, seleccionada por conter simultaneamente cinco classes distintas, ciclistas, veículos, peões e uma pessoa sentada, características de um ambiente urbano de condução.



(a) YOLO26n



(b) YOLOv8n



(c) SSDLite

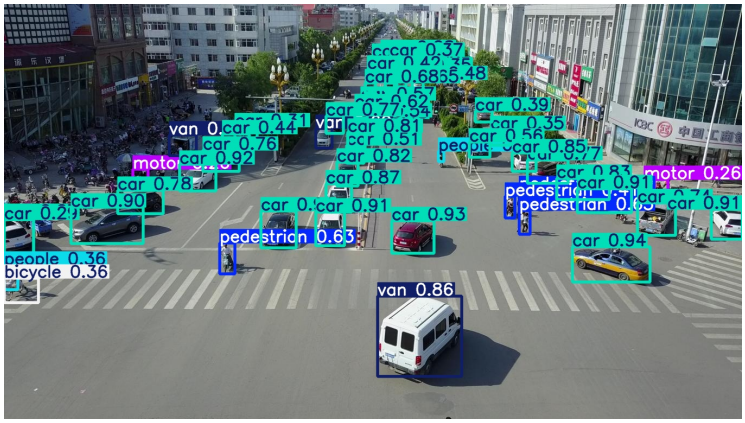
Figura 4.13: Comparação Visual entre Modelos: KITTI.

A análise visual confirma e complementa os resultados quantitativos apresentados na Tabela 4.2. O YOLO26n e o YOLOv8n detetam um conjunto semelhante de objectos, identificando ciclistas, a van, o carro e a pessoa sentada. O YOLOv8n apresenta confiança ligeiramente superior nas deteções de van (0.94 vs. 0.91) e ciclista (0.93 vs. 0.90), consistente com a sua Precision superior reportada na mesma tabela. Ambos os modelos detetam correctamente a classe *Person_sitting*

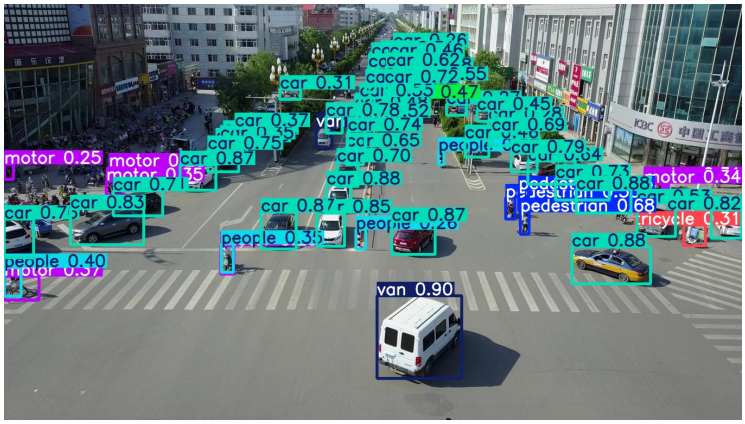
no lado direito da imagem, uma das classes mais difíceis do KITTI. Contudo, a instância presente no lado esquerdo da cena não é detetada por nenhum dos dois modelos, sugerindo sensibilidade à posição e ao contexto visual envolvente. O SSDLite, em contraste, produz um número significativamente menor de deteções, limitando-se a identificar um peão, a van e um ciclista, falhando as restantes instâncias presentes na cena. As caixas delimitadoras são também menos precisas em termos de localização, reflexo do impacto da resolução de entrada reduzida (320×320) na qualidade de localização.

4.4.2 Comparação Visual entre Modelos: VisDrone

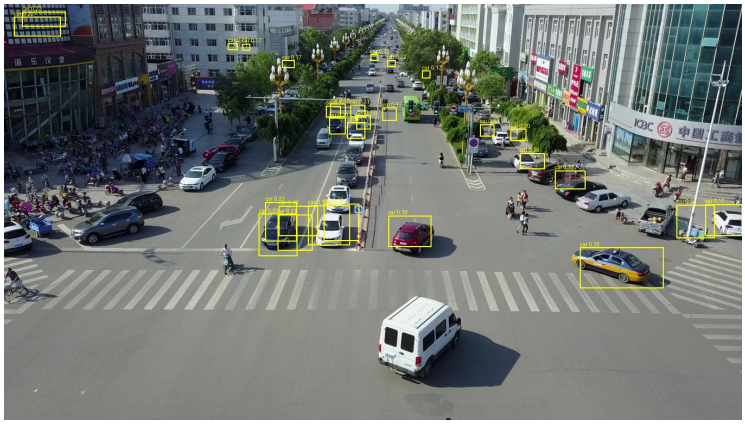
A Figura 4.14 ilustra o comportamento dos modelos numa cena aérea representativa do VisDrone, caracterizada por elevada densidade de objectos de pequenas dimensões observados em perspectiva aérea oblíqua.



(a) YOLO26n



(b) YOLOv8n



(c) SSDLite

Figura 4.14: Comparação Visual entre Modelos: VisDrone.

A análise visual ilustra de forma clara os desafios intrínsecos do domínio aéreo e as diferenças de comportamento entre os modelos. O YOLO26n e o YOLOv8n produzem um número elevado de deteções na zona superior da imagem, onde a densidade de objectos é maior, identificando correctamente carros, motos, peões e vans com confianças entre 0.60 e 0.94. A van no centro da imagem é detetada com alta confiança por ambos os modelos YOLO (0.86 e 0.90, respectivamente), sendo o objecto de maior dimensão da cena e o mais facilmente identificável. Ambos os modelos apresentam dificuldade com objectos de dimensões reduzidas e sobrepostos na zona superior da imagem, onde se observam múltiplas caixas com confiança baixa e sobreposição entre classes, consistente com os AP@50 reduzidos reportados para as classes *pedestrian*, *people* e *motor* na Tabela 4.5.

O SSDLite, mesmo com limiar de confiança reduzido para 0.10 de forma a maximizar o número de deteções visíveis, produz apenas um pequeno número de deteções, limitadas a objectos de maior dimensão na zona central e inferior da imagem, falhando completamente os objectos de dimensões reduzidas na zona de maior densidade. Este comportamento é consistente com o colapso observado na matriz de confusão do VisDrone, onde a grande maioria das instâncias é classificada como *background*, e confirma visualmente que a resolução de entrada reduzida é estruturalmente insuficiente para resolver objectos à escala aérea do VisDrone.

4.5 Otimização e Compressão de Modelos

A otimização dos modelos constituiu uma etapa central deste trabalho, uma vez que o objetivo final não se limitava à obtenção de bons resultados de deteção em ambiente de treino, mas também à sua execução eficiente em hardware *edge* com restrições computacionais significativas. Neste contexto, tornou-se necessário avaliar até que ponto a conversão dos modelos para diferentes formatos intermédios e a redução da precisão numérica afetavam o desempenho do sistema.

Para esse efeito, foi analisado o comportamento dos modelos em diferentes representações, desde o formato original em PyTorch até às versões exportadas para ONNX em FP32 e FP16. Esta comparação permitiu verificar se o processo de exportação preservava o comportamento da rede e identificar eventuais perdas de precisão associadas às transformações aplicadas ao modelo antes da compilação para hardware embarcado.

4.5.1 Impacto da exportação na precisão

A Tabela 4.8 apresenta os resultados obtidos para os modelos YOLO26n e YOLOv8n em diferentes formatos de representação. A comparação entre o formato nativo em PyTorch e as versões exportadas para ONNX em FP32 e FP16 permite avaliar se a conversão introduz degradação relevante na precisão de deteção.

Modelo	**Formato**	*mAP*@50	*mAP*@50-95	**Observação**
YOLO26n	PT-FP32	0.8375	0.5944	Baseline
YOLO26n	ONNX-FP32	0.8327	0.5948	Paridade quase total
YOLO26n	ONNX-FP16	0.8329	0.5957	Perda mínima
YOLOv8n	PT-FP32	0.8581	0.6163	Baseline
YOLOv8n	ONNX-FP32	0.8570	0.6125	Paridade quase total
YOLOv8n	ONNX-FP16	0.8561	0.6115	Perda mínima

Tabela 4.8: Resultados de *mAP* para os modelos YOLO26n e YOLOv8n em diferentes formatos de exportação.

Os resultados mostram que a exportação para ONNX preserva, de forma muito próxima, o desempenho observado no formato original em PyTorch. No caso do YOLO26n, a diferença entre PT-FP32 e ONNX-FP32 é residual, verificando-se igualmente que a variante FP16 mantém valores praticamente idênticos aos do modelo base. Este comportamento sugere que a simples alteração de formato e a redução para meia precisão não introduzem degradação significativa na capacidade de deteção.

O mesmo padrão é observado no modelo YOLOv8n, em que as versões ONNX-FP32 e ONNX-FP16 mantêm uma paridade quase total com o desempenho medido no formato original. Estes resultados confirmam que o processo de exportação para ONNX é, em ambos os casos, suficientemente estável para preservar o comportamento dos modelos, o que constitui uma condição necessária para as etapas subsequentes de otimização e compilação.

De forma geral, esta análise confirma que a exportação dos modelos para ONNX não constitui uma fonte relevante de perda de precisão, ao passo que a compressão numérica para FP16 tem impacto reduzido sobre o desempenho de deteção. Assim, a conversão para formatos intermédios deve ser entendida como uma etapa de compatibilidade e preparação para *deployment*, sem degradação prática significativa na qualidade dos resultados.

4.5.2 Impacto da compressão na eficiência

A Tabela 4.9 apresenta os resultados de eficiência observados para os modelos em diferentes precisões e dispositivos, considerando apenas medições reais de inferência. Estes valores permitem avaliar o impacto da compressão numérica no comportamento temporal do sistema, através da comparação entre FPS e latência média.

Modelo	**Precisão**	**Disp.**	**FPS real**	**Lat. média (ms)**
YOLO26n	FP32	GPU	27.01	37.02
YOLO26n	FP32	CPU	24.87	40.21
YOLO26n	FP16	GPU	15.91	62.86
YOLO26n	FP16	CPU	15.99	62.53
YOLOv8n	FP32	GPU	23.28	42.96
YOLOv8n	FP32	CPU	22.92	43.63
YOLOv8n	FP16	GPU	13.87	72.08
YOLOv8n	FP16	CPU	14.74	67.84

Tabela 4.9: Resultados de eficiência e latência dos modelos em diferentes precisões e dispositivos.

Os resultados mostram que a execução em GPU e CPU apresenta diferenças moderadas, embora a tendência geral dependa mais da precisão do modelo do que do dispositivo de inferência em si. No caso do YOLO26n, observa-se uma maior taxa de processamento em FP32 do que em FP16, mas esta diferença não se traduz num ganho de desempenho temporal para a variante de menor precisão. O mesmo comportamento é observado no YOLOv8n, sugerindo que a conversão para meia precisão não trouxe vantagem direta em termos de FPS no pipeline avaliado.

Os resultados mostram que a execução em FP16 apresenta latências superiores às registadas em FP32, em aparente contradição com a expectativa de ganho de desempenho associado à redução de precisão. Este comportamento explica-se pela ausência de suporte nativo eficiente a operações FP16 no ONNXRuntime em CPU x86 e pela capacidade limitada de aceleração FP16 na GPU utilizada. Neste contexto, o *runtime* converte internamente parte das operações para FP32, introduzindo *overhead* adicional que anula o potencial ganho de velocidade. A vantagem do FP16 manifesta-se sobretudo na redução da precisão numérica, sem tradução direta em ganho computacional nesta configuração experimental.

De forma geral, esta análise confirma que a exportação dos modelos para formatos intermédios não introduz alterações relevantes na estrutura do estudo, mas também evidencia que a compressão numérica deve ser interpretada com cautela. Em vez de assumir ganhos lineares de desempenho, os resultados sugerem um compromisso entre precisão, compatibilidade e eficiência operacional, especialmente em cenários de *deployment* em hardware *edge*.

4.5.3 Análise Exploratória de Compressão - Pruning

Esta secção apresenta os resultados da análise exploratória de *pruning* por magnitude aplicada aos modelos YOLO26n e YOLOv8n treinados no KITTI, seguindo o protocolo descrito na Secção 3.4.6. O objetivo é avaliar a tolerância de cada arquitetura à remoção progressiva de pesos, complementando a análise de quantização da secção anterior com uma perspetiva distinta sobre o espaço de compressão disponível.

As Tabelas 4.10 e 4.11 apresentam os resultados obtidos para o YOLO26n e o YOLOv8n, respetivamente, para cinco níveis de *pruning* entre 10% e 50%, após *fine-tuning* de 30 épocas com *learning rate* de 0.005.

Tabela 4.10: Resultados da análise de *pruning* por magnitude para o YOLO26n no conjunto KITTI.

Taxa de *pruning*	*mAP*@50	*mAP*@50-95	Δ*mAP*@50
0% (baseline)	0.8375	0.5944	—
10%	0.7409	0.4906	-11.5%
20%	0.7153	0.4702	-14.6%
30%	0.6610	0.4250	-21.1%
40%	0.5839	0.3674	-30.3%
50%	0.4344	0.2658	-48.2%

Tabela 4.11: Resultados da análise de *pruning* por magnitude para o YOLOv8n no conjunto KITTI.

Taxa de *pruning*	*mAP*@50	*mAP*@50-95	Δ*mAP*@50
0% (baseline)	0.8581	0.6163	—
10%	0.8438	0.5972	-1.7%
20%	0.8289	0.5890	-3.4%
30%	0.8138	0.5685	-5.2%
40%	0.7953	0.5477	-7.3%
50%	0.7664	0.5143	-10.7%

Os resultados revelam um contraste pronunciado entre os dois modelos. O YOLO26n apresenta uma degradação progressiva e acentuada com o aumento da taxa de *pruning*, perdendo 11.5% de *mAP*@50 com apenas 10% de pesos removidos e atingindo 48.2% de degradação com 50% de *pruning*. O YOLOv8n, em contraste, mostra uma robustez claramente superior: com 50% de *pruning*, perde apenas 10.7% de *mAP*@50, e mesmo com 30% de *pruning* mantém um desempenho de 0.814 em *mAP*@50, ainda elevado face ao baseline.

Este comportamento diferenciado sugere que as duas arquiteturas apresentam tolerâncias estruturais distintas à remoção de pesos. O YOLOv8n parece distribuir a informação de forma mais redundante ao longo da rede, o que o torna mais resiliente à imposição de esparsidade. O YOLO26n, por sua vez, parece depender de forma mais crítica de cada parâmetro, tornando-se

mais sensível ao *pruning*. Ainda assim, esta interpretação deve ser vista como uma leitura experimental dos resultados, e não como uma propriedade absoluta das arquiteturas.

É importante contextualizar estes resultados face às limitações do método utilizado. O *pruning* não destrutivo por magnitude não reduz efetivamente os FLOPs nem o tamanho em disco do modelo, uma vez que os pesos zerados continuam a ocupar memória e a ser processados pelo hardware. O ganho prático deste tipo de *pruning* só se materializaria com técnicas adicionais de compressão de esparsidade ou com hardware especializado para inferência esparsa. Neste trabalho, o *pruning* é portanto interpretado como uma análise da capacidade intrínseca de cada arquitetura para tolerar redundância, e não como uma técnica de otimização aplicada ao modelo de *deployment*.

No contexto mais amplo do trabalho, o YOLOv8n revela-se também mais robusto do que o YOLO26n face a diferentes formas de compressão, incluindo a quantização INT8 que será analisada na Secção 4.6.4. Em conjunto, os resultados sugerem que a sua arquitetura é estruturalmente mais favorável à compressão do que a do YOLO26n, embora essa maior robustez venha acompanhada de um comportamento diferente no pipeline de *deployment* embarcado.

4.6 Resultados no Raspberry Pi 5

Esta secção avalia os modelos YOLO26n e YOLOv8n no hardware embarcado alvo, o Raspberry Pi 5, nos dois modos de inferência disponíveis: NPU Hailo-8, com modelos no formato HEF e quantização INT8, e CPU ARM Cortex-A76, com modelos ONNX em precisão float32 via ONNXRuntime. O protocolo de avaliação inclui 50 imagens de *warm-up* seguidas da inferência sobre o conjunto de teste completo de cada *dataset*, com limiar de confiança `conf=0.001` e cálculo de mAP por interpolação de 101 pontos (COCO style).

4.6.1 NPU Hailo-8

A Tabela 4.12 apresenta os resultados de *benchmark* de inferência na NPU Hailo-8 para os dois modelos nos dois *datasets*.

Tabela 4.12: *Benchmark* de inferência na NPU Hailo-8. Latência apresentada como média ± desvio padrão.

Modelo	Dataset	FPS	Latência (ms)	p95 (ms)	mAP@50	mAP@50-95
YOLO26n	KITTI	53.42	18.72 ± 0.77	20.56	0.544	0.384
YOLOv8n	KITTI	**55.81**	$\mathbf{17.92 \pm 0.42}$	**18.45**	0.537	**0.405**
YOLO26n	VisDrone	**49.65**	$\mathbf{20.14 \pm 0.82}$	**21.56**	0.150	0.110
YOLOv8n	VisDrone	45.01	22.22 ± 2.06	24.08	**0.161**	**0.117**

Em termos de velocidade, ambos os modelos atingem *throughput* claramente acima do limiar de tempo real (25 FPS) na NPU Hailo-8. No KITTI, o YOLOv8n é marginalmente mais

rápido (55.81 vs. 53.42 FPS), com latência média de 17.92 ± 0.42 ms e p95 de 18.45 ms, evidenciando também menor variabilidade face ao YOLO26n (18.72 ± 0.77 ms). No VisDrone, a relação inverte-se: o YOLO26n é mais rápido (49.65 vs. 45.01 FPS), com o YOLOv8n a apresentar variabilidade de latência superior (22.22 ± 2.06 ms), reflexo da maior densidade de objectos por imagem no VisDrone. A diferença de velocidade entre os dois modelos é reduzida, o que sugere que a NPU opera abaixo do seu limite de saturação para modelos de escala *nano*, diluindo a vantagem arquitectural do YOLO26n face ao YOLOv8n neste contexto.

Em termos de precisão, o KITTI apresenta uma inversão face aos resultados em PC: o YOLO26n obtém mAP@50 de 0.544, superando o YOLOv8n (0.537), ao contrário do verificado em PC onde o YOLOv8n era superior. No mAP@50-95, o YOLOv8n mantém vantagem (0.405 vs. 0.384), sugerindo melhor localização apesar da menor mAP@50. No VisDrone, o YOLOv8n obtém mAP@50 ligeiramente superior (0.161 vs. 0.150), com valores globalmente muito reduzidos em ambos os modelos, consistentes com a dificuldade intrínseca do domínio aéreo já documentada na Secção 4.2.2.

4.6.2 CPU ARM

A Tabela 4.13 apresenta os resultados de *benchmark* de inferência em CPU ARM Cortex-A76 via ONNXRuntime, em precisão float32.

Tabela 4.13: *Benchmark* de inferência em CPU ARM Cortex-A76 (ONNXRuntime, float32). Latência apresentada como média $\pm$ desvio padrão.

Modelo	Dataset	FPS (img/s)	Latência (ms)	p95 (ms)	mAP@50a
YOLO26n	KITTI	**7.66**	$\mathbf{130.49 \pm 3.06}$	**137.44**	**0.797**
YOLOv8n	KITTI	6.64	150.61 ± 3.19	157.66	0.543
YOLO26n	VisDrone	**7.51**	$\mathbf{133.11 \pm 3.29}$	**140.39**	**0.288**
YOLOv8n	VisDrone	6.53	153.25 ± 3.43	160.48	0.165

mAP@50-95 não disponível para CPU ARM.

Em CPU ARM, ambos os modelos ficam significativamente abaixo do limiar de tempo real, com 7.66 FPS (YOLO26n) e 6.64 FPS (YOLOv8n) no KITTI, e valores similares no VisDrone. O YOLO26n é consistentemente mais rápido em CPU ARM em ambos os *datasets*, com latência média aproximadamente 20 ms inferior ao YOLOv8n (130 ms vs. 150 ms), beneficiando da ausência de DFL na cabeça de deteção, que simplifica o grafo ONNX e reduz o número de operações sequenciais em CPU.

O resultado mais relevante desta secção é a anomalia observada no YOLOv8n no KITTI: mAP@50 de 0.543, substancialmente inferior ao YOLO26n (0.797) e ao próprio YOLOv8n em PC (0.858). Esta discrepância não se observa no VisDrone, onde ambos os modelos apresentam degradação esperada face ao PC. A hipótese mais provável é uma incompatibilidade entre a exportação ONNX do YOLOv8n e o ONNXRuntime em ARM: o YOLOv8n utiliza DFL (*Distribution*

Focal Loss) na cabeça de deteção, cujo grafo ONNX pode ser processado de forma incorrecta pelo ONNXRuntime em arquitectura ARM, ao contrário do YOLO26n que, sem DFL, exporta um grafo mais simples e portável. Este resultado requer investigação adicional para confirmação da causa. A degradação de precisão face ao PC é analisada em detalhe na Secção 4.6.4.

4.6.3 Comparação NPU vs. CPU ARM

A Tabela 4.14 resume o *speedup* de FPS e o *trade-off* de precisão entre a NPU Hailo-8 e o CPU ARM. Em todos os cenários testados, a NPU Hailo-8 atinge inferência em tempo real (>25 FPS) e o CPU ARM não.

Tabela 4.14: *Speedup* de FPS da NPU Hailo-8 face ao CPU ARM e *trade-off* de precisão (ΔmAP@50 = $\text{mAP}_{\text{CPU}} - \text{mAP}_{\text{NPU}}$).

Modelo	**Dataset**	**FPS_{NPU}**	**FPS_{CPU}**	**Speedup**	**ΔmAP@50**
YOLO26n	KITTI	53.42	7.66	6.97×	+0.253
YOLOv8n	KITTI	55.81	6.64	8.41×	+0.006
YOLO26n	VisDrone	49.65	7.51	6.61×	+0.138
YOLOv8n	VisDrone	45.01	6.53	6.89×	+0.004

A NPU Hailo-8 é entre 6.6× e 8.4× mais rápida que o CPU ARM. Em termos de precisão, o CPU ARM opera em float32 e mantém resultados mais próximos dos obtidos em PC; o ΔmAP@50 é particularmente pronunciado no YOLO26n KITTI (+0.253), em larga medida influenciado pela anomalia do YOLOv8n já discutida, enquanto no YOLOv8n a diferença é residual (+0.006), indicando que a quantização INT8 da Hailo penaliza pouco a precisão deste modelo. A NPU Hailo-8 é assim a única opção viável para inferência em tempo real no Raspberry Pi 5, com o *trade-off* de uma redução de precisão que será quantificada na Secção 4.6.4. A discussão das implicações da escolha arquitectural face aos diferentes paradigmas de aceleração é retomada no Capítulo 5.

4.6.4 Impacto da Quantização INT8

Esta secção analisa a degradação de precisão introduzida pelo pipeline de compilação Hailo, comparando os resultados obtidos em float32 no PC com os resultados INT8 na NPU Hailo-8. Ao contrário da transição FP32 → FP16 analisada na Secção 4.5.1, que introduziu degradação residual inferior a 0.5 pontos percentuais, a quantização INT8 imposta pelo pipeline Hailo produz uma degradação expressiva e consistente em ambos os modelos e datasets.

A Tabela 4.15 apresenta a comparação direta entre os valores de *mAP*@50 obtidos em PC (float32) e na NPU Hailo-8 (INT8).

Tabela 4.15: Degradação de *mAP*@50 introduzida pela quantização INT8 do pipeline Hailo (PC float32 → Hailo INT8).

Modelo	**Dataset**	*mAP*@50 **PC**	*mAP*@50 **Hailo**	Δ*mAP*@50	**Degradação**
YOLO26n	KITTI	0.8375	0.5443	-0.2932	-35.0%
YOLOv8n	KITTI	0.8581	0.5367	-0.3214	-37.5%
YOLO26n	VisDrone	0.2770	0.1501	-0.1269	-45.8%
YOLOv8n	VisDrone	0.2790	0.1607	-0.1183	-42.4%

A degradação introduzida pelo pipeline Hailo é expressiva e contrasta fortemente com a paridade quase total verificada entre FP32 e FP16 via ONNXRuntime. No KITTI, ambos os modelos perdem entre 35.0% e 37.5% de *mAP*@50 face ao baseline em float32. No VisDrone, a degradação é ainda mais acentuada, atingindo 45.8% no YOLO26n e 42.4% no YOLOv8n. Esta diferença entre datasets sugere que a degradação não é uniforme e depende das características do domínio: o VisDrone, com objetos de dimensões muito reduzidas e elevada densidade, é particularmente sensível à perda de resolução numérica introduzida pela quantização INT8.

A magnitude desta degradação aponta para fatores específicos do pipeline de compilação Hailo. O processo de quantização com fine-tuning pós-quantização e calibração pode não ser suficiente para compensar completamente a distorção introduzida pela conversão para INT8 em modelos treinados exclusivamente em float32. Adicionalmente, a distribuição de ativações de cada arquitetura pode ser mais ou menos favorável à quantização INT8, o que poderá explicar as diferenças observadas entre modelos no mesmo dataset. Estas hipóteses são consistentes com os dados disponíveis, embora a sua confirmação requeira análise adicional do processo de calibração.

É importante notar que a degradação afeta ambos os modelos de forma semelhante, o que sugere que o pipeline de compilação Hailo é o fator dominante e não uma limitação específica de uma arquitetura. Apesar desta degradação, os resultados obtidos com a NPU Hailo-8 permitem inferência em tempo real a cadências superiores a 45 FPS, viabilizando a aplicação em contextos onde a velocidade é prioritária face à precisão máxima.

4.6.5 Validação em Tempo Real

O sistema de inferência em tempo real descrito na Secção 3.4.5 foi validado em condições reais de utilização, com o Raspberry Pi 5 equipado com o Camera Module 3, utilizando exclusivamente o modelo treinado no KITTI. A validação com o modelo VisDrone não foi realizada dado que este dataset é composto por imagens aéreas captadas por drone, condições que não foi possível replicar no contexto desta dissertação. A validação foi conduzida em múltiplas execuções independentes, utilizando o modelo YOLO26n compilado para a NPU Hailo-8 com os pesos treinados no KITTI, por ser o modelo com melhor desempenho global neste domínio na plataforma embarcada. As métricas apresentadas nesta secção correspondem à agregação dos resultados dessas execuções, enquanto as figuras mostradas mais adiante ilustram exemplos representativos de uma delas.

A Tabela 4.16 resume o comportamento temporal observado nas execuções consideradas. O FPS médio situou-se em 52.45, com desvio padrão de 0.36, variando entre 51.62 e 52.96. A latência média foi de 19.06 ms, com desvio padrão de 0.13 ms, enquanto o percentil 95 da latência se manteve em 20.41 ms. Estes resultados evidenciam baixa variabilidade entre execuções e uma cadência de processamento estável ao longo do tempo.

Tabela 4.16: Resumo agregado das execuções de validação em tempo real no Raspberry Pi 5.

Métrica	**Média**	**Desvio padrão**	**Mínimo**	**Máximo**
FPS	52.45	0.36	51.62	52.96
Latência média (ms)	19.06	0.13	18.88	19.37
Latência p95 (ms)	20.41	0.41	19.66	20.90
Duração da execução (s)	74.72	40.98	26.10	145.80
Frames processados	2993.92	1638.91	1048	5829

Os resultados mostram que o sistema manteve um comportamento temporal consistente ao longo de execuções com durações e conteúdos visuais distintos. A variação observada no FPS é reduzida, mantendo-se sempre acima de 51.6 FPS, o que confirma a capacidade do sistema para operar em regime de tempo real com margem confortável. De forma semelhante, a latência média permaneceu praticamente constante entre execuções, sem indícios de degradação progressiva de desempenho.

Do ponto de vista funcional, o sistema demonstrou capacidade para detetar, rastrear e contabilizar objetos em movimento em contexto real. Em diferentes execuções, foram observadas pistas estáveis de veículos e peões, com o mecanismo de estabilização por votação temporal descrito na Secção 3.4.5.2 a reduzir eficazmente a flutuação de classes ao longo dos *tracks*. No entanto, métricas como o número absoluto de objetos detetados, o total de cruzamentos da linha virtual ou o pico de objetos simultâneos não são aqui agregadas, uma vez que dependem fortemente da cena observada, da duração da execução e do fluxo de tráfego presente em cada vídeo, não sendo diretamente comparáveis entre ocorrências distintas.

As Figuras 4.15 e 4.16 ilustram *frames* representativos de uma das execuções de validação, mostrando a deteção de um veículo e de um peão, respetivamente, com a linha virtual de contagem e o *overlay* de telemetria. Estas imagens devem ser interpretadas como exemplos qualitativos do funcionamento do sistema, e não como síntese estatística do conjunto completo das execuções analisadas.

Figura 4.15: Frame representativo da validação em tempo real: deteção de veículo com linha virtual de contagem e *overlay* de telemetria. YOLO26n na NPU Hailo-8, Camera Module 3.

Figura 4.16: Frame representativo da validação em tempo real com deteção de peão e contagem ativa. YOLO26n na NPU Hailo-8, Camera Module 3.

Em termos globais, esta validação confirma a viabilidade operacional do sistema em cenário real. A NPU Hailo-8 manteve cadência superior a 50 FPS em *stream* contínuo de câmara, com latência média inferior a 20 ms e baixa variabilidade entre execuções. Em conjunto com o funcionamento estável do *tracking* e da contagem por linha virtual, estes resultados confirmam a adequação do modelo YOLO26n compilado para a plataforma embarcada ao caso de utilização proposto.

Capítulo 5

Discussão

Este capítulo discute os resultados experimentais obtidos nos capítulos anteriores, com foco na interpretação do desempenho dos modelos em diferentes domínios visuais, arquiteturas e plataformas de execução. A análise começa pela avaliação em PC, prossegue para o comportamento em hardware embarcado e para o impacto das técnicas de compressão, e termina com a identificação das principais limitações do estudo.

5.1 Desempenho em PC

A análise em PC fornece a base de comparação para interpretar o comportamento dos modelos nos restantes contextos experimentais. Nesta secção discute-se o impacto do domínio visual nos resultados obtidos nos dois datasets e compara-se o desempenho das diferentes arquiteturas avaliadas.

5.1.1 Impacto do Domínio Visual: KITTI vs VisDrone

Os resultados dos dois datasets mostram um contraste claro, explicado pelas diferenças entre os domínios visuais. No KITTI, os modelos YOLO26n e YOLOv8n atingiram mAP@50 de 0.837 e 0.858, respetivamente, valores que demonstram uma capacidade de deteção robusta em cenários de condução ao nível do solo. No VisDrone, os mesmos modelos atingiram mAP@50 de 0.277 e 0.279, o que corresponde a uma redução de cerca de 67 pontos percentuais face ao KITTI, reflexo directo da maior dificuldade intrínseca do domínio aéreo.

Esta diferença não é surpreendente face às características dos dois datasets. O KITTI apresenta objetos de dimensões relativamente grandes em perspectiva ao nível do solo, com escalas variadas mas geralmente superiores aos limiares mínimos de deteção dos modelos avaliados. O VisDrone, por sua vez, caracteriza-se por objetos de dimensões muito reduzidas em perspectiva aérea, frequentemente inferiores a 32×32 píxeis, com elevada densidade por imagem e sobreposição frequente entre instâncias da mesma classe. A análise por classe confirma este padrão: no VisDrone, apenas a classe *car* atinge AP@50 próximo de 0.68 em ambos os modelos YOLO,

enquanto as classes pedonais e de pequenas dimensões permanecem consistentemente abaixo de 0.25.

O comportamento do SSDLite ilustra de forma mais extrema o impacto do domínio visual: com mAP@50 de 0.454 no KITTI e de apenas 0.010 no VisDrone, o modelo demonstra uma degradação muito acentuada no domínio aéreo. Este resultado confirma que a resolução de entrada fixa de 320×320 píxeis, insuficiente para resolver objetos à escala aérea, constitui uma limitação estrutural que o treino não consegue compensar.

Um aspecto relevante é a inversão do padrão de vantagem entre os dois modelos YOLO nos dois domínios. No KITTI, o YOLOv8n supera o YOLO26n em todas as 8 classes. No VisDrone, a vantagem alterna entre modelos consoante a classe, com o YOLO26n a superar o YOLOv8n em classes pedonais e de pequenas dimensões como *pedestrian*, *people*, *bicycle* e *motor*. Esta observação sugere uma possível maior adequação do YOLO26n a certas instâncias de pequena escala, embora as diferenças sejam marginais e não permitam conclusões definitivas sem investigação adicional.

No seu conjunto, estes resultados mostram que a robustez de um modelo em deteção de objetos depende fortemente do domínio visual em que é avaliado, sendo a escala, a densidade e a perspectiva factores determinantes para o desempenho final.

5.1.2 Comparação entre Arquiteturas: YOLO26n, YOLOv8n e SSDLite

Os três modelos avaliados representam paradigmas arquitecturais distintos e gerações diferentes de detetores leves, o que permite avaliar o impacto das inovações arquitecturais recentes sobre o compromisso entre precisão e eficiência.

Em termos de precisão, o YOLOv8n estabelece-se como o modelo mais capaz no KITTI, superando o YOLO26n nas métricas globais e na maioria das classes. A margem de 2.1 pontos percentuais em mAP@50 e de 2.2 pontos percentuais em mAP@50-95 é consistente ao longo do treino e não decorre apenas de diferenças na capacidade paramétrica, dado que o YOLOv8n tem 3.01M de parâmetros face aos 2.51M do YOLO26n. Em vez disso, esta diferença parece estar associada à maturidade do paradigma one2many com DFL, que introduz um esquema de supervisão mais rico durante o treino. No VisDrone, essa vantagem reduz-se praticamente a uma equivalência estatística, com uma diferença de apenas 0.2 pontos percentuais em mAP@50, sugerindo que, em domínios de elevada dificuldade, as características do dataset condicionam mais fortemente o desempenho do que as diferenças arquitecturais entre os dois modelos.

O SSDLite apresenta um perfil de desempenho claramente inferior aos dois modelos YOLO em ambos os datasets. Com mAP@50 de 0.454 no KITTI, o modelo alcança pouco mais de metade do desempenho do YOLOv8n e degrada-se drasticamente no VisDrone, onde atinge apenas mAP@50 de 0.010 e early stopping às 35 épocas. Este resultado confirma que a resolução de entrada fixa de 320×320 píxeis e a arquitectura anchor-based de geração anterior constituem limitações estruturais que dificultam a adaptação a domínios visuais exigentes, independentemente do protocolo de treino adoptado. A sua inclusão neste estudo serve, assim, como linha de base arquitectural para contextualizar os ganhos introduzidos pelas inovações da família YOLO moderna.

Em termos de eficiência estática, o YOLO26n apresenta menor número de parâmetros (2.51M vs. 3.01M) e menores FLOPs (5.8G vs. 8.2G) do que o YOLOv8n, o que seria expectável dada a sua concepção orientada para dispositivos de baixo consumo. O SSDLite, paradoxalmente, apresenta o maior tamanho em disco (9.5MB), apesar do menor número de parâmetros (2.34M), reflexo do overhead de armazenamento do formato de pesos da implementação utilizada. A relação entre FLOPs declarados e latência real em hardware, contudo, não se traduz de forma linear em desempenho temporal, como se observa nos benchmarks de inferência em PC e no Raspberry Pi 5, sendo essa relação discutida em detalhe na Secção 5.2.

5.2 Desempenho em Hardware Embarcado

A avaliação em hardware embarcado permite analisar o comportamento dos modelos em condições mais próximas da utilização prática, onde os recursos computacionais são significativamente mais limitados do que em PC. Nesta secção discute-se o desempenho obtido no Raspberry Pi 5, comparando a inferência realizada na NPU Hailo-8 com a execução em CPU ARM, e analisando o compromisso entre precisão, latência e viabilidade de *deployment* na *edge*.

5.2.1 Inferência em PC: GPU vs CPU

Os benchmarks em PC mostram padrões distintos consoante o dispositivo de execução, com implicações relevantes para a compreensão do comportamento dos modelos em diferentes contextos computacionais.

Em GPU, o YOLOv8n é claramente o modelo mais rápido, atingindo 110.7 FPS no KITTI e 109.0 FPS no VisDrone, face a 70.6 e 73.4 FPS do YOLO26n. Esta vantagem, próxima de 1.5 vezes, é consistente entre datasets e sugere que a arquitectura do YOLOv8n, com convoluções mais largas e maior número de FLOPs, beneficia de forma mais expressiva do paralelismo massivo disponível em GPU. O SSDLite apresenta o desempenho mais baixo em GPU, com 51.6 FPS, apesar dos menores FLOPs declarados. Este resultado ilustra a não linearidade entre FLOPs e latência real, mediada pela eficiência das operações *depthwise separable* em GPU, que tendem a ser menos vetorizáveis do que convoluções densas.

Em CPU, o panorama inverte-se parcialmente. O SSDLite torna-se o modelo mais rápido, com 21.1 FPS, beneficiando da resolução de entrada reduzida e das convoluções *depthwise separable*, que são mais eficientes em CPU do que em GPU. O YOLO26n e o YOLOv8n apresentam desempenho praticamente idêntico em CPU, com cerca de 11.1 FPS, muito abaixo do limiar de tempo real. Este resultado é relevante porque mostra que a vantagem arquitectural do YOLO26n face ao YOLOv8n, nomeadamente o menor número de FLOPs e a ausência de DFL, não se traduz em ganho mensurável de throughput em CPU x86. Neste contexto, o desempenho parece ser condicionado sobretudo pela largura de banda de memória e pela eficiência do *pipeline* de execução.

Em síntese, o PC com GPU satisfaz confortavelmente os requisitos de tempo real para ambos os modelos YOLO, enquanto o PC em modo CPU não o consegue. Este resultado contextualiza a necessidade de aceleração dedicada na plataforma *edge*, discutida na secção seguinte.

5.2.2 Hailo-8 e CPU ARM no Raspberry Pi 5

A avaliação no Raspberry Pi 5 em dois modos de inferência, NPU Hailo-8 e CPU ARM via ONNXRuntime, revela uma diferença de throughput de 6.6 a 8.4 vezes entre as duas plataformas, sendo a NPU a única opção que satisfaz os requisitos de tempo real.

Na NPU Hailo-8, ambos os modelos atingem throughput claramente acima do limiar de 25 FPS em ambos os datasets: 53.42 e 55.81 FPS no KITTI, e 49.65 e 45.01 FPS no VisDrone para YOLO26n e YOLOv8n, respetivamente. A latência média situa-se entre 18 e 22 ms, com variabilidade reduzida, confirmando a estabilidade do pipeline de inferência embarcada. No VisDrone, o YOLO26n é mais rápido do que o YOLOv8n, com 49.65 contra 45.01 FPS, numa inversão face ao KITTI que pode reflectir diferenças na distribuição de activações entre datasets e o seu impacto sobre o scheduling interno do Hailo-8.

Em CPU ARM, ambos os modelos ficam muito abaixo do limiar de tempo real, com 7.66 e 6.64 FPS no KITTI para YOLO26n e YOLOv8n, respetivamente. O YOLO26n é consistentemente mais rápido em CPU ARM, com latência média aproximadamente 20 ms inferior à do YOLOv8n, beneficiando da ausência de DFL, que simplifica o grafo ONNX e reduz as operações sequenciais em CPU. Este resultado contrasta com o comportamento em CPU x86, onde os dois modelos apresentavam throughput idêntico, sugerindo que a arquitectura ARM é mais sensível à complexidade do grafo de inferência do que a x86.

A discrepância observada no YOLOv8n em CPU ARM no KITTI, com mAP@50 de 0.543 face a 0.858 em PC, merece discussão. A hipótese mais provável é uma incompatibilidade entre o grafo ONNX do YOLOv8n, que utiliza DFL na cabeça de deteção, e o ONNXRuntime em arquitectura ARM. Esta incompatibilidade não se verifica no YOLO26n, que, sem DFL, exporta um grafo mais simples, nem no YOLOv8n no VisDrone, onde os valores de mAP são baixos em ambas as plataformas. Este resultado reforça a relevância da escolha do *pipeline* de exportação e do *runtime* para *deployment* em arquitecturas ARM, e constitui uma limitação documentada na Secção 5.4.

5.2.3 Compromisso entre Precisão e Eficiência na Edge

A comparação entre NPU e CPU ARM permite caracterizar o compromisso fundamental entre velocidade e precisão na plataforma *edge*. A NPU Hailo-8 é a única opção que satisfaz os requisitos de tempo real, mas ao custo de uma degradação de precisão de 35 a 46% em mAP@50 face ao baseline float32 em PC, introduzida pela quantização INT8 obrigatória do *pipeline* de compilação Hailo. O CPU ARM mantém a precisão float32, mas opera a 6 a 8 FPS, o que é insuficiente para processamento de vídeo em tempo real.

Este resultado confirma que, na classe de hardware considerada, não existe uma opção que satisfaça simultaneamente os requisitos de tempo real e a precisão máxima do modelo. É necessário escolher entre throughput e fidelidade de deteção. A NPU Hailo-8, apesar da degradação de precisão, mantém mAP@50 de 0.544 no KITTI, valor que pode ser considerado aceitável para aplicações de monitorização de tráfego onde a velocidade de resposta é prioritária face à precisão máxima. O impacto da quantização INT8 é analisado em detalhe na Secção 5.3.1.

5.3 Efeito da Compressão de Modelos

A compressão de modelos foi analisada para perceber até que ponto as arquitecturas estudadas toleram reduções de precisão numérica e remoção de parâmetros sem perda excessiva de desempenho. Nesta secção discute-se, por um lado, o impacto da quantização INT8 imposta pelo pipeline de deployment e, por outro, a robustez dos modelos ao pruning por magnitude, distinguindo sempre entre efeitos reais de compressão e alterações apenas exploratórias na estrutura do modelo.

5.3.1 Impacto da Quantização INT8

A quantização INT8 imposta pelo pipeline Hailo introduz uma degradação de precisão expressiva em todos os cenários avaliados. No KITTI, o YOLO26n perde 35.0% de mAP@50 e o YOLOv8n 37.5% face ao baseline float32 em PC. No VisDrone, a degradação é ainda maior: 45.8% e 42.4%, respetivamente. Para comparação, a transição de FP32 para FP16 via ONNX-Runtime introduziu menos de 0.5 pontos percentuais de degradação em ambos os modelos, o que mostra uma diferença muito marcada entre as duas formas de compressão numérica.

A maior degradação no VisDrone é consistente com o facto de os valores de partida de mAP serem baixos, o que torna qualquer perda absoluta mais visível em termos percentuais. Ainda assim, também existe um efeito de domínio. Os objetos de pequenas dimensões dependem de activações de maior resolução, que são mais sensíveis ao erro de quantização INT8 do que activações associadas a objetos de maior dimensão.

Ambos os modelos são afectados de forma semelhante, o que aponta para o pipeline de compilação Hailo como factor dominante, e não para limitações específicas de cada arquitectura. O QAT de 4 épocas com 1024 imagens de calibração usado neste trabalho pode não ter sido suficiente para compensar totalmente a distorção introduzida pela conversão para INT8, uma hipótese que a Secção 6.3 retoma como direcção de trabalho futuro.

Apesar da degradação, os modelos mantêm mAP@50 de 0.544 e 0.537 no KITTI, valores que ainda permitem deteção funcional em cenários de monitorização de tráfego. O trade-off é claro: a NPU permite inferência em tempo real a mais de 50 FPS, mas com menor precisão; o CPU ARM mantém a precisão float32, mas fica pelos 6 a 7 FPS, o que é insuficiente para processamento de vídeo em tempo real.

5.3.2 Robustez ao Pruning por Magnitude

Os resultados do pruning mostram uma diferença marcada entre os dois modelos. O YOLO26n perde 11.5% de mAP@50 com apenas 10% de pesos removidos e chega a 48.2% de degradação com 50% de pruning. O YOLOv8n revela-se significativamente mais robusto: com 50% de pruning perde apenas 10.7%, e mesmo com 30% mantém mAP@50 de 0.814, um valor ainda muito próximo do baseline do YOLO26n sem compressão.

Esta diferença sugere que o YOLOv8n distribui a informação de forma mais redundante ao longo da rede, tornando-se mais resiliente à remoção de pesos. O YOLO26n, optimizado para eficiência com uma cabeça mais compacta e sem DFL, parece depender de forma mais crítica de cada parâmetro. Esta leitura é consistente com os resultados observados, mas continua a ser uma interpretação experimental e não uma propriedade provada das arquitecturas.

É importante notar que o pruning utilizado é não destrutivo. Os pesos zerados continuam a ocupar memória e a ser processados pelo hardware, pelo que não há redução real de FLOPs nem de tamanho em disco. O ganho prático só se materializaria com pruning estruturado ou com hardware com suporte a inferência esparsa, nenhum dos quais foi avaliado neste trabalho. Assim, a análise deve ser lida como uma avaliação da tolerância estrutural de cada arquitectura à esparsidade, e não como uma técnica de compressão aplicada ao modelo de deployment.

Face a estes resultados, e tendo em conta que o pipeline de compilação Hailo já introduz uma degradação de 35 a 46% em mAP@50 devido à quantização INT8, optou-se por não aplicar pruning nos modelos efectivamente colocados em funcionamento na plataforma embarcada. A aplicação cumulativa de pruning sobre modelos já sujeitos a quantização agravaria a degradação de precisão sem garantia de benefício prático num hardware como o Hailo-8, onde o principal constrangimento não é o número de parâmetros, mas sim a largura de banda e o throughput do acelerador. Assim, as análises de quantização e pruning devem ser entendidas como estudos independentes e exploratórios, e não como etapas cumulativas do pipeline de deployment.

5.4 Limitações do Estudo

O trabalho desenvolvido apresenta um conjunto de limitações que importa reconhecer para uma interpretação adequada dos resultados obtidos.

A quantização INT8 é imposta pelo processo de compilação da plataforma Hailo, não sendo possível controlar livremente o nível de precisão alvo nem configurar de forma detalhada o procedimento de calibração. Assim, a degradação de precisão observada deve ser entendida não apenas como consequência das propriedades intrínsecas dos modelos avaliados, mas também como resultado das características específicas deste fluxo de implementação. Uma calibração mais extensa ou estratégias de *quantization-aware training* mais elaboradas poderiam mitigar parte desta degradação, mas ficaram fora do âmbito deste trabalho.

O *dataset* KITTI foi recolhido em condições diurnas e sem condições meteorológicas adversas, o que limita a generalização dos modelos treinados a cenários com iluminação noturna, sombras

acentuadas, chuva ou névoa. O desempenho em condições de iluminação difícil não foi avaliado, constituindo uma limitação relevante para aplicações de monitorização de tráfego em ambiente real.

A validação do sistema de inferência em tempo real foi realizada num contexto experimental controlado, ainda que representativo de utilização prática. Além disso, a validação com o modelo treinado no VisDrone não pôde ser realizada por ausência de equipamento de captura aérea, pelo que os resultados de validação em tempo real refletem apenas o cenário de condução ao nível do solo com o modelo treinado no KITTI.

De forma mais geral, não foi possível validar o sistema em cenários reais mais diversos, como condições noturnas ou ambientes urbanos mais complexos. Esta limitação reduz a abrangência da avaliação experimental e deixa em aberto a análise do comportamento da solução em contextos operacionais mais exigentes.

Capítulo 6

Conclusões e Trabalho Futuro

6.1 Resposta à Hipótese Central

A hipótese central desta dissertação consistia em verificar se modelos leves de deteção de objetos poderiam ser executados com desempenho em tempo real em hardware de *edge*, e em particular numa plataforma embarcada de baixo custo, demonstrando a viabilidade prática desta abordagem em cenários reais. Para além da simples demonstração de inferência em tempo real, esta hipótese implicava ainda avaliar de que forma o domínio visual, a arquitetura do modelo e o processo de implementação influenciam o equilíbrio entre precisão e eficiência.

Os resultados obtidos confirmam esta hipótese. A avaliação experimental mostrou que a NPU Hailo-8 permite executar modelos leves de deteção com cadências compatíveis com aplicações em tempo real, mantendo uma latência reduzida e um *throughput* estável em todos os cenários avaliados. A validação em tempo real confirmou este comportamento em condições práticas de operação, evidenciando que a solução proposta não se limita a um resultado laboratorial, mas pode efetivamente ser utilizada em aplicações de monitorização visual com requisitos de resposta rápida.

Os resultados mostram também que a viabilidade da solução depende fortemente da combinação entre modelo, domínio visual e plataforma de execução. O desempenho observado foi substancialmente mais elevado em cenários rodoviários do que em cenários aéreos, refletindo a influência das características intrínsecas dos *datasets*. Da mesma forma, as diferenças entre arquiteturas e o impacto das técnicas de compressão evidenciam que a execução em *edge* não é apenas uma questão de reduzir o tamanho do modelo, mas de encontrar uma configuração equilibrada entre capacidade de generalização, tolerância à quantização e adequação ao hardware disponível.

Em síntese, os resultados suportam a hipótese de que a aprendizagem computacional pode ser aplicada com sucesso em sistemas de *edge* para processamento de imagem em tempo real, desde que a solução seja cuidadosamente adaptada às restrições da plataforma e ao domínio da aplicação. O presente trabalho confirma assim a viabilidade técnica desta abordagem e demonstra o seu potencial de utilização prática em cenários reais.

6.2 Síntese dos Contributos Principais

Este trabalho produziu contributos empíricos e práticos para o domínio da inferência de deteção de objetos em hardware embarcado.

Em primeiro lugar, foi realizada uma caracterização sistemática do compromisso entre precisão e eficiência na execução de modelos leves de deteção no Raspberry Pi 5 com acelerador Hailo-8, em dois domínios visuais contrastantes. Os resultados mostram que a NPU Hailo-8 permite inferência em tempo real com *throughput* elevado, ao custo de uma degradação de precisão que deve ser considerada na conceção de aplicações práticas.

Em segundo lugar, este trabalho apresenta uma das primeiras avaliações do YOLO26n em plataforma Hailo-8. Os resultados mostram que, apesar das expectativas associadas à sua conceção orientada para eficiência, o YOLO26n apresenta uma degradação de precisão semelhante à do YOLOv8n no *pipeline* Hailo. Em contrapartida, revelou-se mais sensível ao *pruning* por magnitude, o que reforça a importância de avaliar a robustez dos modelos para além das métricas de desempenho absoluto.

Em terceiro lugar, foi desenvolvido e validado um sistema funcional de inferência em tempo real com *tracking* multi-objeto e contagem por linha virtual, demonstrando a viabilidade operacional da solução em condições reais de utilização. Esta componente confirma que a plataforma estudada não se limita a servir de ambiente experimental, mas pode suportar uma aplicação completa de monitorização de tráfego.

6.3 Trabalho Futuro

Apesar dos resultados obtidos, existem várias linhas de trabalho futuro que podem aprofundar e expandir o estudo desenvolvido nesta dissertação.

O *Federated Learning*, um paradigma de aprendizagem distribuída que permite treinar modelos sem centralizar os dados, constitui uma direção relevante para extensões futuras deste trabalho, em particular em cenários de monitorização urbana onde a privacidade dos dados é uma preocupação central. Esta abordagem poderá permitir a atualização colaborativa de modelos entre múltiplos dispositivos, preservando simultaneamente a confidencialidade das imagens adquiridas.

Outra linha de trabalho consiste na realização de testes em contexto real com maior densidade de objetos e maior variabilidade de condições, incluindo diferentes níveis de iluminação, oclusões mais frequentes e cenários de tráfego mais exigentes. A concretização deste tipo de avaliação deverá, no entanto, respeitar os requisitos éticos, legais e institucionais aplicáveis, incluindo os procedimentos de autorização adequados sempre que necessário.

Em terceiro lugar, seria relevante realizar um estudo de densidade de tráfego em múltiplos pontos, de forma a caracterizar fluxos em diferentes locais e horários. Este tipo de análise permitiria não apenas validar o sistema em mais contextos, mas também explorar o seu potencial como ferramenta de apoio à monitorização e análise de mobilidade urbana.

Outra direção importante passa pela validação da solução em diferentes contextos urbanos, como cruzamentos, rotundas, vias rápidas, zonas escolares, parques e interfaces multimodais. A avaliação nestes ambientes permitiria compreender melhor a generalização do sistema e a sua adequação a cenários com características visuais e operacionais distintas.

Adicionalmente, a integração com múltiplas câmaras constitui uma evolução natural do sistema, especialmente em interseções urbanas e em soluções de monitorização distribuída. Esta abordagem poderia aumentar a cobertura espacial da cena observada e permitir uma análise mais completa do comportamento do tráfego.

Por fim, seria pertinente estudar a robustez do sistema em condições adversas, incluindo cenários noturnos, chuva, nevoeiro, reflexos e movimentos de câmara. A avaliação nestas condições permitiria identificar com maior precisão os limites operacionais da solução e orientar futuras melhorias ao nível da aquisição, pré-processamento e inferência.

Referências

[1] Navneet Dalal e Bill Triggs. «Histograms of Oriented Gradients for Human Detection». Em: *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 1. 2005, pp. 886–893. DOI: 10.1109/CVPR.2005.177. URL: https://doi.org/10.1109/CVPR.2005.177.

[2] Jiasi Chen e Xukan Ran. «Deep Learning With Edge Computing: A Review». Em: *Proceedings of the IEEE* 107.8 (2019), pp. 1655–1674. DOI: 10.1109/JPROC.2019.2921977.

[3] Shaoqing Ren et al. «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks». Em: *Advances in Neural Information Processing Systems*. Vol. 28. 2015. URL: https://arxiv.org/abs/1506.01497.

[4] Haochen Hua et al. «Edge Computing with Artificial Intelligence: A Machine Learning Perspective». Em: *ACM Computing Surveys* 55.9 (2023), pp. 1–35. DOI: 10.1145/3555802.

[5] Wei Liu et al. «SSD: Single Shot MultiBox Detector». Em: *European Conference on Computer Vision (ECCV)*. 2016, pp. 21–37.

[6] Joseph Redmon et al. «You Only Look Once: Unified, Real-Time Object Detection». Em: *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[7] Md Nahid Sadik, Tahmim Hossain e Faisal Sayeed. *Real-Time Detection and Analysis of Vehicles and Pedestrians using Deep Learning*. 2024. DOI: 10.48550/arXiv.2404.08081.

[8] Joseph Redmon e Ali Farhadi. «YOLOv3: An Incremental Improvement». Em: *arXiv preprint arXiv:1804.02767* (2018). URL: https://arxiv.org/abs/1804.02767.

[9] Glenn Jocher, Ayush Chaurasia e Jing Qiu. *Ultralytics YOLOv8*. Versão 8.0.0. 2023. URL: https://github.com/ultralytics/ultralytics.

[10] Ao Wang et al. «YOLOv10: Real-Time End-to-End Object Detection». Em: *arXiv preprint arXiv:2405.14458* (2024). URL: https://arxiv.org/abs/2405.14458.

[11] Ranjan Sapkota et al. «YOLO26: Key Architectural Enhancements and Performance Benchmarking for Real-Time Object Detection». Em: *arXiv preprint arXiv:2509.25164* (2026). URL: https://arxiv.org/abs/2509.25164.

[12] Glenn Jocher e Jing Qiu. *Ultralytics YOLO26*. Versão 26.0.0. 2026. URL: https://github.com/ultralytics/ultralytics.

[13] Miao Hu et al. *Edge-Based Video Analytics: A Survey*. 2023. DOI: 10.48550/arXiv.2303.14329.

[14] Aditya Vardhan Reddy Katkuri et al. «Autonomous UAV navigation using deep learning-based computer vision frameworks: A systematic literature review». Em: *Array* 23 (2024), p. 100361. DOI: 10.1016/j.array.2024.100361.

[15] Kailai Sun et al. «A review of AI edge devices and lightweight CNN and LLM deployment». Em: *Neurocomputing* 614 (2025), p. 128791. DOI: 10.1016/j.neucom.2024.128791.

[16] Mingxing Tan e Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].

[17] Forrest N. Iandola et al. «SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size». Em: *arXiv preprint arXiv:1602.07360* (2016).

[18] Chien-Yao Wang, I-Hau Yeh e Hong-Yuan Mark Liao. «YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information». Em: *arXiv preprint arXiv:2402.13616* (2024).

[19] Andrew Howard et al. «Searching for MobileNetV3». Em: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 1314–1324.

[20] Saurabh Bhalgaonkar, Madhura Munot e Abhijit Anuse. «Model compression of deep neural network architectures for visual pattern recognition: Current status and future directions». Em: *Computers & Electrical Engineering* 116 (2024), p. 109180. DOI: 10.1016/j.compeleceng.2024.109180.

[21] *ssdlite_mobilenet_v2 - OpenVINO Model Zoo*. https://docs.openvino.ai/2023.3/omz_models_model_ssdlite_mobilenet_v2.html. Acedido em 2026-06-21.

[22] Ultralytics. *Explore Ultralytics YOLOv8*. https://docs.ultralytics.com/models/yolov8. Acedido em 2026-06-21.

[23] Ultralytics. *Ultralytics YOLO26*. https://docs.ultralytics.com/models/yolo26. Acedido em 2026-06-21.

[24] Ultralytics. *YOLO26 vs YOLO11: A Generational Leap in Vision AI*. https://docs.ultralytics.com/compare/yolo26-vs-yolo11. Acedido em 2026-06-21.

[25] UnitLab AI. *YOLO-26 Release: Architecture and Performance Benchmarks*. https://blog.unitlab.ai/yolo-26-release-architecture-performance-benchmarks-and-real-world-use-cases-2026-guide/. Acedido em 2026-06-21.

[26] Alex Bewley et al. «Simple Online and Realtime Tracking». Em: *Proceedings of the IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003.

[27] Yifu Zhang et al. «FairMOT: On the Fairness of Detection and Re-Identification in Multiple Object Tracking». Em: *International Journal of Computer Vision* 129 (2021), pp. 3069–3087. DOI: 10.1007/s11263-021-01513-4.

[28] Nicolai Wojke, Alex Bewley e Dietrich Paulus. «Simple Online and Realtime Tracking with a Deep Association Metric». Em: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 3645–3649. DOI: 10.1109/ICIP.2017.8296962.

[29] Yifu Zhang et al. «ByteTrack: Multi-Object Tracking by Associating Every Detection Box». Em: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2022.

[30] Benoit Jacob et al. «Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference». Em: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 2704–2713. DOI: 10.1109/CVPR.2018.00285. URL: https://openaccess.thecvf.com/content_cvpr_2018/html/Jacob_Quantization_and_Training_CVPR_2018_paper.html.

[31] Toghrul Karimov, Hassan Imani e Allan Kazakov. «Quantization Robustness to Input Degradations for Object Detection». Em: *arXiv preprint arXiv:2508.19600* (2025).

[32] Samer Francy e Raghubir Singh. *Edge AI: Evaluation of Model Compression Techniques for Convolutional Neural Networks*. 2024. DOI: 10.48550/arXiv.2409.02134.

[33] Andreas Geiger, Philip Lenz e Raquel Urtasun. «Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite». Em: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012, pp. 3354–3361.

[34] Pengfei Zhu et al. «Detection and Tracking Meet Drones Challenge». Em: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. DOI: 10.1109/TPAMI.2021.3119563.

[35] Tsung-Yi Lin et al. «Microsoft COCO: Common Objects in Context». Em: Springer. 2014, pp. 740–755.

[36] Ghalib Ahmed Tahir. «Ethical Challenges in Computer Vision: Ensuring Privacy and Mitigating Bias in Publicly Available Datasets». Em: *arXiv preprint* arXiv:2409.10533 (2024). URL: https://arxiv.org/abs/2409.10533.

[37] Jerone T. A. Andrews et al. «Ethical Considerations for Responsible Data Curation». Em: *Proceedings of the 37th Conference on Neural Information Processing Systems, Datasets and Benchmarks Track*. 2023. URL: https://openreview.net/forum?id=Qf8uzIT1OK.

[38] Babak Rahimi Ardabili et al. «Understanding Ethics, Privacy, and Regulations in Smart Video Surveillance for Public Safety». Em: *arXiv preprint* arXiv:2212.12936 (2022). URL: https://arxiv.org/abs/2212.12936.

[39] UNESCO. *Recommendation on the Ethics of Artificial Intelligence*. UNESCO Recommendation adopted by the General Conference. 2021. URL: https://www.unesco.org/en/articles/recommendation-ethics-artificial-intelligence.

[40] Comissão Nacional de Proteção de Dados. *Parecer 143/2021 sobre a Proposta de Lei relativa à utilização de videovigilância pelas forças e serviços de segurança*. Rel. téc. Comissão Nacional de Proteção de Dados, 2021. URL: https://www.gpasa.pt/xms/files/NL_Parecer_CNPD_-_Videovigilancia_forcas_e_servicos_de_seguranca_-1-.pdf.

[41] Lurdes Dias Alves. *A videovigilância e a compressão da privacidade*. https://protecaodedadosue.cedis.fd.unl.pt/wp-content/uploads/2022/10/6.-Lurdes-Dias-Alves.pdf. 2022.