UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LUIZ GUILHERME MARTINHO SAMPAIO ITO

# Monitoramento de Segurança em Canteiros de Obra utilizando Inferência Multiestágio em Inteligência Artificial na Borda

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Orientador: Prof. Dr. José Rodrigo Azambuja

Porto Alegre
2026

**CIP — CATALOGAÇÃO NA PUBLICAÇÃO**

Ito, Luiz Guilherme Martinho Sampaio

Monitoramento de Segurança em Canteiros de Obra utilizando Inferência Multiestágio em Inteligência Artificial na Borda / Luiz Guilherme Martinho Sampaio Ito. – Porto Alegre: PPGC da UFRGS, 2026.

92 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2026. Orientador: José Rodrigo Azambuja.

1. Visão Computacional. 2. Inteligência Artificial de Borda. 3. Inferência em Cascata. 4. Segurança na Construção Civil. 5. NVIDIA DeepStream. I. Azambuja, José Rodrigo. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Prof[a]. Marcia Barbosa
Vice-Reitor: Prof. Pedro Costa
Pró-Reitora de Pós-Graduação: Prof[a]. Claudia Wasserman
Diretor do Instituto de Informática: Prof. Luciano Paschoal Gaspary
Coordenador do PPGC: Prof. Gabriel Luca Nazar
Bibliotecário-chefe do Instituto de Informática: Alexsander Borges Ribeiro

*Dedico este trabalho à minha esposa, familiares e amigos,*
*pelo apoio durante toda esta jornada acadêmica.*

*“Segurança é projeto: medir, detectar e agir antes do acidente.”*

— Heinrich, H. W.

## AGRADECIMENTOS

Ao Prof. Dr. José Rodrigo Azambuja, pela orientação, disponibilidade e apoio técnico ao longo do desenvolvimento desta dissertação.

À Universidade Federal do Rio Grande do Sul e ao Programa de Pós-Graduação em Computação, pela infraestrutura e pelo ambiente acadêmico que viabilizaram a realização desta pesquisa.

À minha esposa, pelo apoio constante, pela paciência e pelo incentivo durante toda a trajetória.

À minha família, pelo suporte e pela compreensão nos períodos de maior dedicação aos estudos.

Aos amigos, pelo incentivo e pela motivação constante para a busca por inovação e atualização contínua.

## RESUMO

A indústria da construção civil apresenta índices elevados de acidentes de trabalho, o que evidencia as limitações dos métodos tradicionais de monitoramento. A visão computacional, impulsionada por técnicas de inteligência artificial, apresenta potencial para automatizar a vigilância de segurança. Contudo, soluções baseadas em processamento na nuvem enfrentam desafios de latência e dependência de conectividade. Trabalhos anteriores demonstraram a viabilidade de detecção com modelo único em dispositivos de borda, porém nenhum emprega inferência em cascata com modelos especializados por domínio. Esta dissertação apresenta o projeto e a validação de um sistema de análise de vídeo inteligente baseado em computação de borda. O sistema utiliza uma arquitetura de inferência em cascata composta por um modelo primário, denominado PGIE (*Primary GPU Inference Engine*), que detecta cinco classes (pessoa, veículo, capacete, colete e ferramenta) e três modelos secundários especializados, denominados SGIEs (*Secondary GPU Inference Engines*), que operam sobre as regiões recortadas: SGIE-EPI para equipamentos de proteção individual, SGIE-Maquinário para identificação de máquinas pesadas e SGIE-Ferramentas para detecção de ferramentas e materiais. O sistema opera em um dispositivo NVIDIA Jetson AGX Xavier e processa fluxos de vídeo em tempo real. A metodologia emprega o *framework* NVIDIA DeepStream para orquestrar o *pipeline* de inferência acelerado por *hardware*. Os modelos foram treinados com *datasets* públicos (SODA, MOCS, Roboflow) e otimizados com TensorRT. Como resultado, o sistema alcançou vazão de 20 quadros por segundo (FPS – *Frames Per Second*) com latência de 58 milissegundos no *pipeline* completo. O PGIE obteve mAP@0,5 (*mean Average Precision*) de 0,83, enquanto os SGIEs alcançaram mAP@0,5 entre 0,71 e 0,84. Esta dissertação demonstra a viabilidade técnica de um sistema de monitoramento autônomo com inferência em cascata na borda.



**Palavras-chave:** Visão Computacional. Inteligência Artificial de Borda. Inferência em Cascata. Segurança na Construção Civil. NVIDIA DeepStream.

**Safety Monitoring on Construction Sites Using Video Analytics and Multistage Inference in Edge AI**

## ABSTRACT

The construction industry consistently exhibits high rates of occupational accidents, highlighting the limitations of traditional monitoring methods. Computer vision, powered by artificial intelligence techniques, shows potential to automate safety surveillance. However, cloud-based solutions face latency and connectivity challenges. Previous works have demonstrated the feasibility of single-model detection on edge devices, but none employs cascade inference with domain-specialized models. This dissertation presents the design and validation of an intelligent video analytics system based on edge computing. The system employs a cascade inference architecture composed of a Primary GPU Inference Engine (PGIE) that detects five classes (person, vehicle, hardhat, safety vest, and tool) and three specialized Secondary GPU Inference Engines (SGIEs) operating on cropped regions: SGIE-PPE for personal protective equipment, SGIE-Machinery for heavy machine identification, and SGIE-Tools for tool and material detection. The system operates on an NVIDIA Jetson AGX Xavier device and processes real-time video streams. The methodology employs the NVIDIA DeepStream SDK to orchestrate the hardware-accelerated inference pipeline. The models were trained using public datasets (SODA, MOCS, Roboflow) and optimized with TensorRT. As a result, the system achieved 20 FPS throughput with 58 milliseconds latency for the complete pipeline. The PGIE achieved mAP@0.5 of 0.83, while the SGIEs reached mAP@0.5 between 0.71 and 0.84. This dissertation demonstrates the technical feasibility of an autonomous monitoring system with cascade inference at the edge.

**Keywords:** Computer Vision. Edge AI. Cascade Inference. Construction Site Safety. NVIDIA DeepStream.

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| AP | Average Precision |
| C2f | CSP Bottleneck with 2 Convolutions – Faster |
| CIoU | Complete Intersection over Union |
| CNN | Convolutional Neural Network |
| COCO | Common Objects in Context |
| CSP | Cross Stage Partial |
| CUDA | Compute Unified Device Architecture |
| cuDNN | CUDA Deep Neural Network Library |
| DLA | Deep Learning Accelerator |
| EPI | Equipamento de Proteção Individual |
| FP16 | Floating Point 16 bits |
| FP32 | Floating Point 32 bits |
| FPN | Feature Pyramid Network |
| FPS | Frames Per Second |
| GDPR | General Data Protection Regulation |
| GIE | GPU Inference Engine |
| GPU | Graphics Processing Unit |
| INT8 | Integer 8 bits |
| IoU | Intersection over Union |
| LGPD | Lei Geral de Proteção de Dados |
| mAP | mean Average Precision |
| MOCS | Multi-Object Construction Site Dataset |
| NMS | Non-Maximum Suppression |
| NR-18 | Norma Regulamentadora 18 |

| | |
|---|---|
| NVDEC | NVIDIA Video Decoder |
| ONNX | Open Neural Network Exchange |
| OSD | On-Screen Display |
| PANet | Path Aggregation Network |
| PGIE | Primary GPU Inference Engine |
| PIB | Produto Interno Bruto |
| PVA | Programmable Vision Accelerator |
| PyDS | Python DeepStream Bindings |
| R-CNN | Region-based Convolutional Neural Network |
| ReLU | Rectified Linear Unit |
| ROI | Region of Interest |
| RPN | Region Proposal Network |
| RT-DETR | Real-Time Detection Transformer |
| RTSP | Real Time Streaming Protocol |
| SAHI | Slicing Aided Hyper Inference |
| SDK | Software Development Kit |
| SGD | Stochastic Gradient Descent |
| SGIE | Secondary GPU Inference Engine |
| SODA | Safety-related Object Detection in Construction |
| SPPF | Spatial Pyramid Pooling – Fast |
| SSD | Single Shot MultiBox Detector |
| TensorRT | NVIDIA Tensor Runtime |
| TOPS | Tera Operations Per Second |
| YOLO | You Only Look Once |

## LISTA DE FIGURAS

Figura 2.1 Processo de *Transfer Learning*. O conhecimento adquirido durante o pré-treinamento em *datasets* genéricos (ImageNet/COCO) é transferido para o domínio específico de detecção de EPIs. Camadas iniciais que detectam características genéricas são congeladas, enquanto camadas finais são ajustadas para o novo domínio. ....24
Figura 2.2 Comparação entre arquiteturas de detectores de objetos. À esquerda, detectores de dois estágios (Faster R-CNN) com geração de propostas de região seguida de classificação. À direita, detectores de estágio único (YOLO) com predição direta sobre uma grade de *features*. ....26
Figura 2.3 Arquitetura do modelo YOLOv8s. O *backbone* CSPDarknet utiliza blocos C2f para extração de características em três escalas (P3, P4, P5). O *neck* combina FPN (*Feature Pyramid Network*) e PANet com blocos C2f para fusão de características. As cabeças de detecção desacopladas (*anchor-free*) operam com ramos separados para classificação e regressão. ....28
Figura 2.4 Funcionamento do algoritmo *Non-Maximum Suppression* (NMS). Antes do NMS, múltiplas caixas delimitadoras sobrepostas com diferentes *scores* de confiança detectam o mesmo objeto. O algoritmo seleciona a caixa com maior confiança e suprime as demais com IoU acima do limiar. ....28
Figura 2.5 Arquitetura de *hardware* do NVIDIA Jetson AGX Xavier. A memória unificada LPDDR4x é compartilhada entre CPU e GPU. Os aceleradores dedicados (DLA e PVA) permitem execução otimizada de modelos de *deep learning* com menor consumo energético. ....32
Figura 2.6 Arquitetura de inferência em cascata do DeepStream. O PGIE processa o *frame* completo e detecta objetos de interesse. Os SGIEs operam sobre as regiões recortadas (*crops*) das detecções do PGIE, permitindo classificação especializada. Cada SGIE pode ser configurado para processar apenas classes específicas do PGIE. ....34

Figura 4.1 Visão geral do sistema. Câmeras IP transmitem vídeo via RTSP para o Jetson AGX Xavier, que executa o *pipeline* de inferência em cascata na borda. As saídas incluem alertas de não conformidade e vídeo anotado com as detecções.42
Figura 4.2 Arquitetura de inferência em cascata proposta. O PGIE detecta cinco classes no *frame* completo. Três dessas classes são encaminhadas a SGIEs especializados: Person ao SGIE-EPI, Vehicle ao SGIE-Maquinário e Tool ao SGIE-Ferramentas. As classes Hardhat e Safety-vest são utilizadas diretamente pelo sistema de análise. ....43
Figura 4.3 Distribuição estatística do *dataset* de treinamento do PGIE com 5 classes. O gráfico mostra a contagem de instâncias por classe (person, vehicle, hardhat, safety-vest, tool) e a distribuição espacial das anotações. ....49
Figura 4.4 Distribuição do *dataset* SGIE-Ferramentas com 23 classes. O gráfico evidencia o desbalanceamento entre as classes, com variação significativa na quantidade de instâncias por categoria. ....50
Figura 4.5 Distribuição do *dataset* SGIE-Maquinário com 9 classes de maquinário pesado. A distribuição entre as classes é mais balanceada em comparação ao *dataset* de ferramentas. ....51
Figura 4.6 Amostra de imagens do *batch* de treinamento com anotações de *ground truth*, demonstrando a diversidade do *dataset*. ....52

Figura 5.1 Arquitetura de alto nível do sistema. O fluxo de processamento inicia com a configuração do ambiente, seguido pela leitura do vídeo, detecção de objetos (PGIE + SGIEs), rastreamento, análise de conformidade de EPIs e geração de alertas. ..... 57
Figura 5.2 *Pipeline* GStreamer/DeepStream implementado. O fluxo inicia com a captura de vídeo (Source), decodificação por *hardware* (Decoder), *batching* (Streammux), inferência primária (PGIE), *tracking* (Tracker), inferências secundárias paralelas (SGIEs), renderização de anotações (OSD) e saída para *display* ou arquivo (Sink). ..... 58
Figura 5.3 *Pipeline* detalhado do sistema. O diagrama apresenta o fluxo de processamento desde a configuração do ambiente até a geração de alertas, incluindo a detecção primária (PGIE), os três SGIEs especializados (EPI, Maquinário e Ferramentas), o rastreamento de objetos e a análise de conformidade. ..... 59

Figura 6.1 Curvas de treinamento do modelo PGIE, apresentando a evolução das funções de perda e métricas mAP ao longo das épocas. ..... 65
Figura 6.2 Matriz de confusão normalizada do modelo PGIE para as 5 classes. ..... 66
Figura 6.3 Curvas *Precision-Recall* do modelo PGIE, com mAP@0,5 indicado na legenda. ..... 67
Figura 6.4 Desempenho por classe do modelo SGIE-EPI (AP@0,5) para as 14 classes de EPIs e situações de risco. As barras são ordenadas do maior para o menor valor de AP@0,5. ..... 68
Figura 6.5 Curvas de treinamento do modelo SGIE-EPI, apresentando a evolução das funções de perda e métricas mAP ao longo das épocas. ..... 69
Figura 6.6 Matriz de confusão normalizada do modelo SGIE-EPI para as 14 classes de EPIs e situações de risco. ..... 70
Figura 6.7 Curvas *Precision-Recall* do modelo SGIE-EPI, com mAP@0,5 indicado na legenda. ..... 71
Figura 6.8 Curvas *F1-Score* do modelo SGIE-EPI em função do limiar de confiança. ..... 72
Figura 6.9 Curvas de treinamento do modelo SGIE-Maquinário, apresentando a evolução das funções de perda e métricas mAP ao longo das épocas. ..... 73
Figura 6.10 Matriz de confusão normalizada do modelo SGIE-Maquinário para as 9 classes de maquinário. ..... 74
Figura 6.11 Curvas *Precision-Recall* do modelo SGIE-Maquinário, com mAP@0,5 indicado na legenda. ..... 75
Figura 6.12 Exemplos de predições do SGIE-Maquinário no conjunto de validação, demonstrando a capacidade de identificação de diferentes tipos de maquinário. ..... 76
Figura 6.13 Curvas de treinamento do modelo SGIE-Ferramentas, apresentando a evolução das funções de perda e métricas mAP ao longo das épocas. ..... 77
Figura 6.14 Matriz de confusão normalizada do modelo SGIE-Ferramentas para as 23 classes de equipamentos e ferramentas. ..... 78
Figura 6.15 Curvas *Precision-Recall* do modelo SGIE-Ferramentas, com mAP@0,5 indicado na legenda. ..... 79
Figura 6.16 Exemplos de predições do SGIE-Ferramentas no conjunto de validação, demonstrando a detecção de diferentes ferramentas e equipamentos. ..... 80
Figura 6.17 Diagrama de propagação de erros no *pipeline* em cascata. O desempenho do PGIE limita a entrada dos SGIEs, e a cobertura efetiva é estimada como o produto $AP_{PGIE} \times mAP_{SGIE}$ ..... 81

## LISTA DE TABELAS

Tabela 3.1 Comparação dos trabalhos relacionados. ..... 40

Tabela 4.1 Hiperparâmetros de treinamento. ..... 52

Tabela 5.1 Estrutura de diretórios do projeto. ..... 60
Tabela 5.2 Parâmetros principais do arquivo de configuração do PGIE. ..... 60
Tabela 5.3 Parâmetros principais do arquivo de configuração do SGIE-EPI. ..... 61
Tabela 5.4 Parâmetros principais do arquivo de configuração do *tracker*. ..... 62

Tabela 6.1 Métricas do modelo PGIE. ..... 65
Tabela 6.2 Métricas do modelo SGIE-EPI (14 classes). ..... 66
Tabela 6.3 Métricas do modelo SGIE-Maquinário. ..... 69
Tabela 6.4 Métricas do modelo SGIE-Ferramentas (23 classes). ..... 71
Tabela 6.5 *Throughput* do *pipeline* no Jetson AGX Xavier. ..... 73
Tabela 6.6 Latência por componente do *pipeline* (FP16). ..... 74
Tabela 6.7 Utilização de recursos do Jetson AGX Xavier. ..... 75
Tabela 6.8 Cobertura efetiva estimada por cadeia de detecção no *pipeline* em cascata. ..... 77
Tabela 6.9 Comparação com trabalhos da literatura. ..... 81

**SUMÁRIO**

**1 INTRODUÇÃO ..... 16**
**1.1 Motivação e Hipótese ..... 18**
**1.2 Objetivos ..... 19**
1.2.1 Objetivo Geral ..... 19
1.2.2 Objetivos Específicos ..... 19
**1.3 Contribuições ..... 19**
**1.4 Organização do Documento ..... 20**
**2 FUNDAMENTAÇÃO TEÓRICA ..... 21**
**2.1 Redes Neurais Convolucionais ..... 21**
2.1.1 Componentes Arquiteturais ..... 21
2.1.2 Evolução das Arquiteturas ..... 22
2.1.3 *Transfer Learning* ..... 23
**2.2 Arquiteturas de Detecção de Objetos ..... 23**
2.2.1 Detectores de Dois Estágios ..... 24
2.2.2 Detectores de Estágio Único ..... 25
2.2.3 Família YOLO ..... 25
2.2.4 *Non-Maximum Suppression* ..... 27
2.2.5 Métricas de Avaliação ..... 29
**2.3 Computação de Borda ..... 29**
2.3.1 Motivações para *Edge AI* ..... 29
2.3.2 Desafios do *Edge AI* ..... 30
2.3.3 Plataforma NVIDIA Jetson ..... 31
**2.4 NVIDIA DeepStream SDK ..... 31**
2.4.1 Arquitetura do DeepStream ..... 32
2.4.2 Inferência Primária e Secundária ..... 33
2.4.3 Sistema de Metadados ..... 33
**2.5 TensorRT e Otimização de Modelos ..... 34**
2.5.1 Otimizações Aplicadas ..... 34
2.5.2 Quantização ..... 35
**3 TRABALHOS RELACIONADOS ..... 36**
**3.1 Visão Computacional na Construção Civil ..... 36**
**3.2 Sistemas de Detecção de EPIs ..... 37**
**3.3 *Edge AI* para Monitoramento em Tempo Real ..... 38**
**3.4 Arquiteturas de Inferência em Cascata ..... 38**
**3.5 Análise Comparativa ..... 39**
**4 METODOLOGIA ..... 42**
**4.1 Arquitetura do Sistema ..... 42**
4.1.1 Modelo Primário (PGIE) ..... 43
4.1.2 Modelos Secundários (SGIEs) ..... 44
4.1.3 Fluxo de Processamento ..... 45
**4.2 *Datasets* ..... 46**
4.2.1 Fontes de Dados ..... 46
4.2.2 Processo de Curadoria ..... 47
4.2.3 *Data Augmentation* ..... 47
4.2.4 Caracterização do *Dataset* ..... 48
**4.3 Treinamento dos Modelos ..... 49**
4.3.1 Configuração do Ambiente ..... 50
4.3.2 Hiperparâmetros ..... 51

4.3.3 Função de Perda ..... 53
**4.4 Conversão e Otimização ..... 53**
4.4.1 Exportação para ONNX ..... 53
4.4.2 Otimização com TensorRT ..... 54
**4.5 Integração com DeepStream ..... 54**
4.5.1 Configuração do *Pipeline* ..... 54
4.5.2 Configuração dos SGIEs ..... 55
4.5.3 *Parser* Customizado ..... 55
**5 IMPLEMENTAÇÃO ..... 57**
**5.1 Ambiente de Desenvolvimento ..... 58**
5.1.1 *Hardware* ..... 58
5.1.2 *Software* ..... 58
**5.2 Estrutura do Projeto ..... 59**
**5.3 Configuração do *Pipeline* ..... 60**
5.3.1 Configuração do PGIE ..... 60
5.3.2 Configuração dos SGIEs ..... 61
5.3.3 Configuração do *Tracker* ..... 61
**5.4 Implementação do *Parser* ..... 61**
**5.5 Aplicação Principal ..... 62**
**5.6 Sistema de Análise ..... 63**
**5.7 Otimizações de Performance ..... 63**
**6 RESULTADOS E DISCUSSÃO ..... 64**
**6.1 Avaliação dos Modelos ..... 64**
6.1.1 Métricas de Avaliação ..... 64
6.1.2 Resultados do PGIE ..... 64
6.1.3 Resultados do SGIE-EPI ..... 66
6.1.4 Resultados do SGIE-Maquinário ..... 68
6.1.5 Resultados do SGIE-Ferramentas ..... 70
**6.2 *Performance* do *Pipeline* ..... 73**
6.2.1 *Throughput* ..... 73
6.2.2 Latência por Componente ..... 74
6.2.3 Utilização de Recursos ..... 75
**6.3 Análise *End-to-End* do *Pipeline* em Cascata ..... 76**
6.3.1 Propagação de Erros ..... 76
6.3.2 Fluxo de Propagação ..... 77
6.3.3 Impacto dos Falsos Negativos do PGIE ..... 78
6.3.4 Estratégias de Mitigação ..... 79
6.3.5 Vantagens Apesar da Limitação ..... 80
**6.4 Comparação com a Literatura ..... 81**
**6.5 Discussão ..... 82**
6.5.1 Viabilidade Técnica ..... 82
6.5.2 Vantagens da Arquitetura em Cascata ..... 82
6.5.3 Limitações Identificadas ..... 82
6.5.4 Considerações para Produção ..... 83
**7 CONCLUSÃO ..... 84**
**7.1 Síntese das Contribuições ..... 84**
**7.2 Obtenção dos Objetivos Específicos ..... 84**
**7.3 Reflexão sobre os Resultados ..... 85**
**7.4 Limitações do Trabalho ..... 85**
**7.5 Trabalhos Futuros ..... 86**
**7.6 Considerações Finais ..... 86**

**REFERÊNCIAS.....87**

# 1 INTRODUÇÃO

A indústria da construção civil representa um pilar fundamental da economia global. O setor responde por aproximadamente 13% do produto interno bruto mundial e emprega cerca de 7% da força de trabalho global (Oesterreich; Teuteberg, 2016). Estimativas da Organização Internacional do Trabalho indicam que acidentes e doenças ocupacionais causam aproximadamente 2,78 milhões de mortes por ano no mundo, das quais uma parcela significativa ocorre no setor da construção (International Labour Organization, 2024; Hämäläinen et al., 2024). A construção civil figura consistentemente entre as atividades econômicas com maiores índices de fatalidades ocupacionais em países desenvolvidos e em desenvolvimento (Guo; Yiu; González, 2020).

No Brasil, a construção civil representa aproximadamente 6,2% do PIB nacional e emprega mais de 2,5 milhões de trabalhadores formais (Câmara Brasileira da Indústria da Construção, 2024). Contudo, o setor também lidera as estatísticas de acidentes graves e fatais no país (Guo; Yiu; González, 2020).

Segundo dados do Observatório de Segurança e Saúde no Trabalho (Ministério Público do Trabalho; Organização Internacional do Trabalho, 2024), o Brasil registrou mais de 600 mil acidentes de trabalho em 2022, dos quais aproximadamente 12% ocorreram na construção civil. Estudos globais sobre mortalidade ocupacional indicam que o setor mantém taxas de fatalidade consistentemente superiores à média das demais atividades econômicas (Hämäläinen et al., 2024). As quedas de altura representam a principal causa de óbitos, seguidas por soterramento, choque elétrico e impacto de objetos em queda (Fang et al., 2018). Além do custo humano, acidentes de trabalho geram impactos econômicos significativos: custos diretos com tratamento médico e indenizações, e custos indiretos como paralisação de obras, investigações e danos à reputação das empresas (Zhou; Goh; Li, 2015).

A Norma Regulamentadora NR-18 (Ministério do Trabalho e Emprego, 2024) estabelece as condições de trabalho na indústria da construção e determina o uso obrigatório de Equipamentos de Proteção Individual (EPIs). O cumprimento dessas normas depende fundamentalmente de fiscalização efetiva, um desafio em canteiros de obras de grande porte.

A gestão de segurança tradicional em canteiros de obras depende de inspeções manuais realizadas por profissionais de segurança do trabalho. Esta abordagem apresenta limitações que comprometem sua eficácia: as inspeções são intermitentes, estão sujeitas a

erros humanos e fadiga, e são fundamentalmente reativas, identificando violações apenas após sua ocorrência (Seo et al., 2015). A complexidade espacial dos canteiros de obras modernos, que podem abranger áreas extensas com centenas de trabalhadores em atividades simultâneas, torna impossível a vigilância contínua de todas as áreas de risco (Fang et al., 2018). Estudos sobre reconhecimento de riscos em canteiros de obras identificaram que trabalhadores detectam em média apenas 38% dos riscos presentes no ambiente (Albert; Hallowell; Kleiner, 2014). Pesquisas complementares indicam que a taxa de reconhecimento varia conforme a gravidade do risco: 57% para riscos fatais e apenas 18% para riscos não fatais (Albert; Pandit; Patil, 2020).

A convergência de tecnologias emergentes como Inteligência Artificial, Internet das Coisas e computação de borda apresenta oportunidades para a transformação das práticas de monitoramento de segurança ocupacional (Oesterreich; Teuteberg, 2016). A visão computacional, impulsionada pelos avanços em aprendizado profundo (*deep learning*), emerge como uma ferramenta promissora para a automação da vigilância de segurança (Fang et al., 2020). Sistemas baseados em câmeras e algoritmos de detecção de objetos podem realizar análise contínua de ambientes de trabalho para identificação automática de situações de risco (Nath; Behzadan; Paal, 2020). Revisões sistemáticas da literatura indicam crescimento de publicações na área de visão computacional aplicada à construção civil, com foco em detecção de EPIs, reconhecimento de atividades perigosas e monitoramento de zonas de risco (Konstantinov et al., 2024).

A implementação de sistemas de análise de vídeo inteligente apresenta um dilema arquitetural fundamental: processamento centralizado na nuvem versus processamento distribuído na borda da rede (Chen; Ran, 2019). A abordagem baseada em computação em nuvem apresenta desvantagens para aplicações em canteiros de obras. A transmissão de vídeo demanda largura de banda substancial: uma câmera *Full HD* operando a 30 quadros por segundo gera fluxos de dados de 3 a 10 Mbps. A latência de rede introduz atrasos significativos entre a captura do evento e a geração do alerta, comprometendo a capacidade de resposta em situações críticas (Xu; Razavi; Zheng, 2023). A dependência de conectividade com a internet representa um ponto de falha, pois interrupções de rede incapacitam o sistema de monitoramento. Estudos sobre análise de vídeo na borda identificam a latência como fator determinante para aplicações de segurança, onde segundos podem determinar a diferença entre prevenção e acidente (Murshed et al., 2022).

A computação de borda propõe uma mudança de paradigma na qual a análise dos dados ocorre próximo de sua fonte de geração (Shi et al., 2016). Ao executar os mo-

delos de inteligência artificial em *hardware* especializado instalado no próprio canteiro de obras, elimina-se a dependência de conectividade externa para a detecção principal, reduz-se a latência de resposta para menos de 100 milissegundos, minimizam-se os custos de transmissão e preserva-se a privacidade dos trabalhadores (Mendez et al., 2022). Plataformas como a família NVIDIA Jetson oferecem capacidade de processamento de redes neurais profundas em dispositivos de baixo consumo energético, viabilizando a implantação de sistemas de visão computacional em ambientes industriais (Mittal, 2019).

## 1.1 Motivação e Hipótese

A hipótese deste trabalho é que uma arquitetura de inferência em cascata, composta por um modelo primário generalista e modelos secundários especializados por domínio, é capaz de operar em tempo real em dispositivos de borda com precisão adequada para monitoramento de segurança em canteiros de obras.

Esta dissertação é motivada pela necessidade de desenvolver soluções de monitoramento de segurança que sejam robustas, de baixa latência e adequadas para implantação em ambientes de construção civil. O trabalho propõe uma arquitetura de inferência em cascata que supera as limitações de sistemas com detector único, aproveitando os avanços recentes em detectores de objetos baseados em aprendizado profundo (Zhao et al., 2019).

Sistemas tradicionais de detecção de EPIs utilizam um único modelo para detectar todas as classes de objetos. Esta abordagem apresenta problemas quando o número de classes aumenta: o modelo precisa aprender características muito distintas, desde pessoas até pequenos objetos como luvas, o que compromete a precisão (Nath; Behzadan; Paal, 2020). Além disso, processar o *frame* inteiro em alta resolução para detectar objetos pequenos demanda recursos computacionais excessivos, limitando a aplicação em dispositivos de borda (Mittal, 2024).

A arquitetura proposta utiliza inferência em cascata com um modelo primário (PGIE – *Primary GPU Inference Engine*) e três modelos secundários (SGIEs – *Secondary GPU Inference Engines*). O PGIE detecta cinco classes (pessoa, veículo, capacete, colete, ferramenta) no *frame* completo. Três dessas classes são encaminhadas a SGIEs especializados: detecções de pessoa são processadas pelo SGIE-EPI, detecções de veículo pelo SGIE-Maquinário e detecções de ferramenta pelo SGIE-Ferramentas. As detecções de capacete e colete são utilizadas diretamente pelo sistema de análise de conformidade, sem necessidade de classificação secundária. Cada SGIE opera apenas sobre as regiões recor-

tadas das detecções do PGIE. Esta abordagem oferece vantagens: modelos menores e mais precisos para cada domínio, processamento otimizado apenas nas regiões de interesse, e escalabilidade para adicionar novos SGIEs sem modificar o sistema existente. A literatura recente demonstra que arquiteturas hierárquicas podem melhorar tanto a precisão quanto a eficiência computacional em tarefas de detecção complexas (Cai; Vasconcelos, 2021).

## 1.2 Objetivos

### 1.2.1 Objetivo Geral

Projetar, implementar e validar uma arquitetura de *software* para um sistema de análise de vídeo inteligente baseado em computação de borda, com inferência em cascata, capaz de realizar monitoramento de segurança em tempo real em canteiros de obras.

### 1.2.2 Objetivos Específicos

Os objetivos específicos desta dissertação são:

1. Desenvolver e treinar um modelo primário (PGIE) para detecção de cinco classes em canteiros de obras;
2. Desenvolver e treinar três modelos secundários (SGIEs) especializados em EPIs, maquinário e ferramentas;
3. Implementar um *pipeline* de inferência em cascata utilizando o NVIDIA DeepStream SDK (*Software Development Kit*) em dispositivo Jetson AGX Xavier;
4. Desenvolver funções de *parsing* customizadas para integração de modelos YOLO (*You Only Look Once*) com o DeepStream;
5. Avaliar a *performance* do sistema em termos de acurácia (mAP – *mean Average Precision*) e velocidade de processamento (FPS – *Frames Per Second*).

## 1.3 Contribuições

As principais contribuições desta dissertação são:

- Uma arquitetura de inferência em cascata (PGIE + 3 SGIEs) validada para sistemas

de monitoramento de segurança em borda;

- Quatro modelos YOLO treinados e otimizados: um genérico e três especializados;
- Documentação do processo de integração de múltiplos modelos YOLO em cascata com o NVIDIA DeepStream;
- Análise empírica de *performance* da arquitetura em cascata em dispositivo de borda.

## 1.4 Organização do Documento

Esta dissertação está organizada em sete capítulos. O Capítulo 2 apresenta a fundamentação teórica necessária para compreensão do trabalho. O Capítulo 3 revisa os trabalhos relacionados na literatura. O Capítulo 4 descreve a metodologia empregada no desenvolvimento do sistema. O Capítulo 5 detalha os aspectos de implementação. O Capítulo 6 apresenta e discute os resultados experimentais obtidos. O Capítulo 7 apresenta as conclusões e direções para trabalhos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo estabelece a base teórica necessária para a compreensão do trabalho desenvolvido. São apresentados os conceitos de redes neurais convolucionais, as arquiteturas de detecção de objetos com ênfase na família YOLO, os princípios da computação de borda, e os *frameworks* utilizados no desenvolvimento do sistema.

## 2.1 Redes Neurais Convolucionais

As Redes Neurais Convolucionais (CNNs) constituem uma classe de arquiteturas de aprendizado profundo (*deep learning*) adequadas para o processamento de dados com estrutura de grade, sendo imagens o caso mais representativo (LeCun et al., 1998). O sucesso das CNNs em tarefas de visão computacional deve-se a três propriedades arquiteturais fundamentais: conectividade local esparsa, compartilhamento de parâmetros e equivariância translacional. Estas propriedades tornam as CNNs a base de virtualmente todos os sistemas modernos de detecção de objetos (Zhao et al., 2019).

A conectividade local implica que cada neurônio em uma camada convolucional está conectado apenas a uma região espacialmente restrita da camada anterior, denominada campo receptivo. Esta propriedade reduz o número de parâmetros da rede e introduz um viés indutivo apropriado para processamento de imagens, onde pixels vizinhos tendem a apresentar correlação estatística.

O compartilhamento de parâmetros determina que o mesmo conjunto de pesos (filtro ou *kernel*) é aplicado em todas as posições espaciais da entrada. Esta característica reduz adicionalmente o número de parâmetros e confere à rede a capacidade de detectar o mesmo padrão independentemente de sua localização na imagem.

### 2.1.1 Componentes Arquiteturais

A arquitetura das CNNs é composta por uma sequência de blocos funcionais que transformam progressivamente a representação da entrada, desde pixels até vetores de características de alto nível semântico. Os principais componentes são camadas convolucionais, funções de ativação, camadas de normalização, camadas de *pooling* e camadas totalmente conectadas. A combinação hierárquica destes componentes permite que a rede

aprenda representações cada vez mais abstratas dos dados de entrada (Krizhevsky; Sutskever; Hinton, 2012).

A operação de convolução consiste na aplicação de filtros sobre a entrada para extração de características. Para uma entrada $\mathbf{X}$ e um filtro $\mathbf{W}$ de dimensões $k \times k$, a saída $\mathbf{Y}$ em cada posição $(i, j)$ é computada como:

$$Y_{i,j} = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} W_{m,n} \cdot X_{i+m,j+n} + b \tag{2.1}$$

onde $b$ representa o termo de viés. O resultado desta operação é um mapa de características que codifica a presença de padrões específicos em diferentes posições espaciais.

A função ReLU (*Rectified Linear Unit*), definida como $f(x) = \max(0, x)$, tornou-se o padrão em arquiteturas de *deep learning* devido à sua simplicidade computacional e eficácia no combate ao problema de desvanecimento do gradiente. Variantes como Leaky ReLU e GELU (*Gaussian Error Linear Unit*) foram propostas para mitigar o problema de neurônios mortos.

A normalização de lote (*Batch Normalization*) normaliza as ativações de cada camada para ter média zero e variância unitária (Ioffe; Szegedy, 2015). Os benefícios incluem aceleração do treinamento, regularização implícita e possibilidade de uso de taxas de aprendizado maiores. A técnica *Dropout* complementa a regularização ao desativar aleatoriamente neurônios durante o treinamento, reduzindo o *overfitting* (Srivastava et al., 2014).

### 2.1.2 Evolução das Arquiteturas

O marco inaugural das CNNs modernas foi estabelecido pela arquitetura AlexNet em 2012, que venceu o desafio ImageNet com margem significativa sobre métodos tradicionais (Krizhevsky; Sutskever; Hinton, 2012). A arquitetura VGGNet demonstrou que redes mais profundas com camadas convolucionais de *kernel* pequeno ($3 \times 3$) poderiam alcançar resultados superiores, estabelecendo o princípio de profundidade sobre largura (Simonyan; Zisserman, 2015).

A arquitetura ResNet propôs conexões residuais como solução para o problema de degradação do gradiente em redes profundas (He et al., 2016). As conexões de atalho permitem que o gradiente flua diretamente através de caminhos alternativos durante o *backpropagation*, viabilizando o treinamento de redes com centenas de camadas. A Ef-

ficientNet introduziu o conceito de escalonamento composto, balanceando profundidade, largura e resolução de forma sistemática (Tan; Le, 2019).

No contexto de dispositivos com recursos limitados, arquiteturas leves como MobileNet foram desenvolvidas para aplicações móveis e de borda (Howard et al., 2017). Estas arquiteturas empregam convoluções separáveis em profundidade para reduzir o número de operações em até 8-9 vezes com perda mínima de acurácia. A evolução para MobileNetV2 introduziu blocos residuais invertidos com gargalos lineares (Sandler et al., 2018). *Surveys* recentes sobre modelos leves para dispositivos de borda consolidam as técnicas de otimização disponíveis (Liu et al., 2024).

### 2.1.3 *Transfer Learning*

O aprendizado por transferência (*transfer learning*) é uma técnica que permite alavancar conhecimento adquirido durante o treinamento em uma tarefa fonte para acelerar o aprendizado em uma tarefa alvo relacionada (Pan; Yang, 2010). No contexto de visão computacional, esta técnica manifesta-se através da utilização de pesos pré-treinados em grandes *datasets*, como ImageNet com 1,2 milhões de imagens ou COCO com 330 mil imagens (Lin et al., 2014), como inicialização para treinamento em domínios específicos.

A eficácia do *transfer learning* fundamenta-se na observação de que as primeiras camadas de CNNs aprendem detectores de características genéricas, como bordas e texturas, que são aplicáveis a diversas tarefas visuais. Camadas intermediárias capturam padrões de complexidade crescente, enquanto camadas finais especializam-se em características do domínio de treinamento. Esta hierarquia de representações permite que modelos pré-treinados sejam adaptados para novos domínios com conjuntos de dados significativamente menores.

A Figura 2.1 ilustra o processo de *transfer learning*, desde o pré-treinamento em um domínio fonte até o *fine-tuning* no domínio alvo.

## 2.2 Arquiteturas de Detecção de Objetos

A detecção de objetos consiste em localizar e classificar múltiplas instâncias de objetos em uma imagem. Dada uma imagem de entrada, o objetivo é produzir um conjunto de detecções onde cada detecção contém as coordenadas da caixa delimitadora, a classe

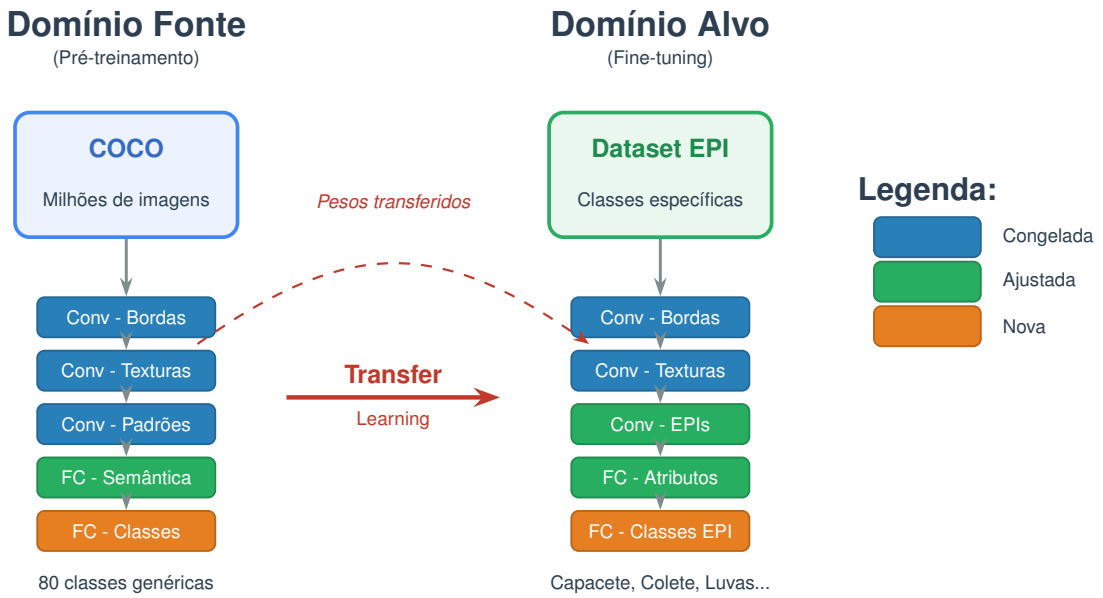



Figura 2.1 – Processo de *Transfer Learning*. O conhecimento adquirido durante o pré-treinamento em *datasets* genéricos (ImageNet/COCO) é transferido para o domínio específico de detecção de EPIs. Camadas iniciais que detectam características genéricas são congeladas, enquanto camadas finais são ajustadas para o novo domínio.
Fonte: Elaborado pelo autor.

do objeto e a confiança da predição (Zhao et al., 2019). Este problema é mais complexo que a classificação de imagens, pois requer simultaneamente localização espacial precisa e classificação semântica de múltiplos objetos com escalas e proporções variadas.

As arquiteturas de *deep learning* para detecção de objetos são categorizadas em dois paradigmas principais: detectores de dois estágios, que separam a geração de propostas de região da classificação, e detectores de estágio único, que realizam localização e classificação em uma única passagem pela rede. A escolha entre os paradigmas envolve um *trade-off* entre acurácia e velocidade de processamento, fator crítico para aplicações em tempo real (Liu et al., 2016).

### 2.2.1 Detectores de Dois Estágios

Os detectores de dois estágios, exemplificados pela família R-CNN (*Region-based Convolutional Neural Network*), operam em duas fases sequenciais (Girshick et al., 2014). Na primeira fase, um mecanismo de propostas de região gera um conjunto de regiões candidatas que potencialmente contêm objetos. Na segunda fase, cada região proposta é classificada e sua localização é refinada através de regressão de *bounding box*.

A arquitetura Faster R-CNN introduziu a *Region Proposal Network* (RPN), uma

rede neural que compartilha camadas convolucionais com a rede de detecção e gera propostas de região de forma eficiente (Ren et al., 2015). Variações como o Cascade R-CNN utilizam múltiplos estágios de refinamento em cascata para melhorar progressivamente a precisão de localização, aplicando limiares de IoU crescentes em cada estágio (Cai; Vasconcelos, 2021). O Cascade R-CNN demonstrou que a inferência em cascata pode superar detectores de estágio único em termos de precisão, princípio que fundamenta a arquitetura proposta nesta dissertação. Apesar da alta acurácia, a latência de processamento dos detectores de dois estágios permanece elevada para aplicações de tempo real em dispositivos de borda.

### 2.2.2 Detectores de Estágio Único

Os detectores de estágio único tratam a detecção de objetos como um problema de regressão direta sobre uma grade espacial. A rede prediz simultaneamente as coordenadas das caixas delimitadoras, as probabilidades de classe e os *scores* de confiança em uma única passagem pela rede (Liu et al., 2016). Esta formulação elimina a necessidade de propostas de região explícitas, simplificando significativamente o *pipeline* de detecção.

A principal vantagem dos detectores de estágio único é a velocidade de processamento significativamente superior, frequentemente alcançando dezenas ou centenas de *frames* por segundo em *hardware* apropriado. A arquitetura SSD (*Single Shot MultiBox Detector*) introduziu predições em múltiplas escalas para lidar com objetos de tamanhos variados (Liu et al., 2016). Estudos comparativos demonstram que detectores de estágio único oferecem o melhor *trade-off* entre velocidade e acurácia para aplicações embarcadas (Ganesh et al., 2022).

A Figura 2.2 ilustra as diferenças arquiteturais entre detectores de dois estágios e detectores de estágio único.

### 2.2.3 Família YOLO

A família YOLO (*You Only Look Once*) representa uma das linhas de pesquisa mais influentes em detecção de objetos de estágio único (Redmon et al., 2016). O paradigma YOLO formula a detecção como um problema de regressão unificado, processando a imagem completa em uma única avaliação da rede neural. Uma revisão abrangente das

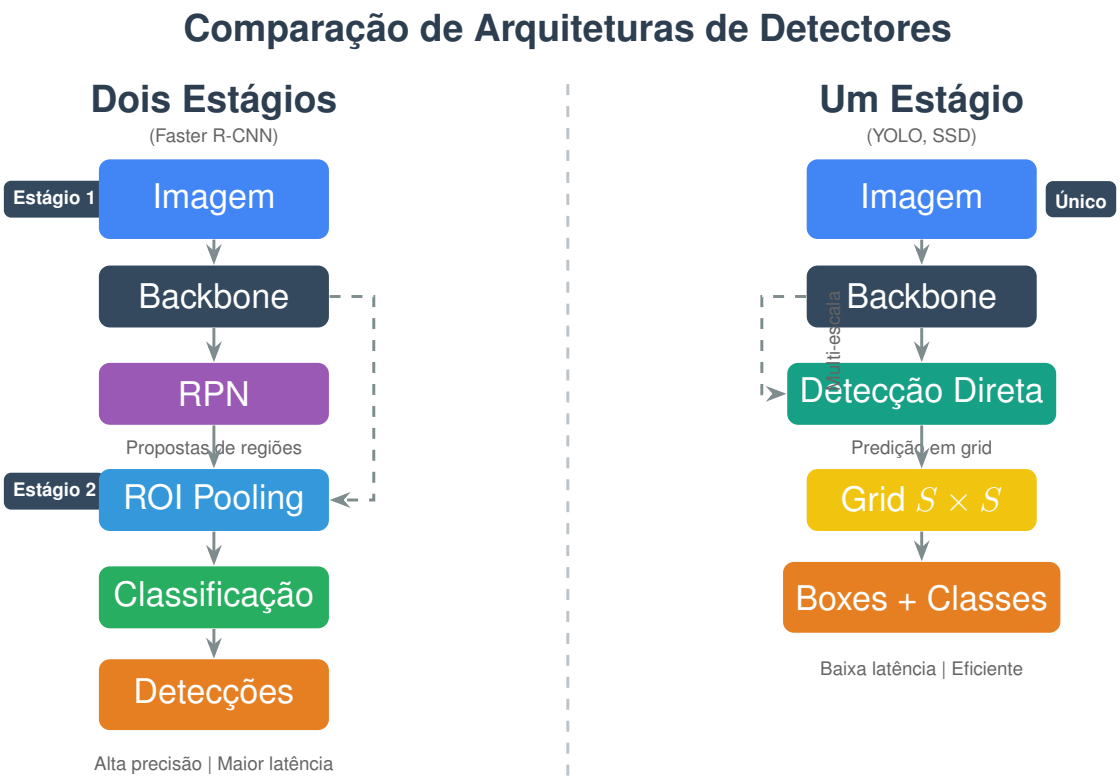



Figura 2.2 – Comparação entre arquiteturas de detectores de objetos. À esquerda, detectores de dois estágios (Faster R-CNN) com geração de propostas de região seguida de classificação. À direita, detectores de estágio único (YOLO) com predição direta sobre uma grade de *features*. Fonte: Elaborado pelo autor.

arquiteturas YOLO desde a versão original até YOLOv8 é apresentada em Terven and Cordova-Esparza (2023).

O YOLOv1 original dividiu a imagem de entrada em uma grade $S \times S$ e fez cada célula responsável por predizer objetos cujo centro caísse dentro daquela célula. O YOLOv2 introduziu *Batch Normalization*, uso de âncoras pré-definidas e treinamento em alta resolução (Redmon; Farhadi, 2017). O YOLOv3 expandiu a arquitetura com predições em múltiplas escalas, melhorando significativamente a detecção de objetos pequenos (Redmon; Farhadi, 2018).

O YOLOv4 consolidou diversas técnicas do estado da arte em uma arquitetura otimizada (Bochkovskiy; Wang; Liao, 2020). As contribuições incluem o *backbone* CSP-Darknet53 (*Cross Stage Partial Darknet*), *neck* SPP (*Spatial Pyramid Pooling*) e PANet (*Path Aggregation Network*), além de técnicas de treinamento como *Mosaic augmentation* e *Self-Adversarial Training*.

O YOLOv8, desenvolvido pela Ultralytics, representa a evolução mais recente da família YOLO com código aberto, oferecendo melhorias significativas em arquitetura e desempenho (Jocher; Chaurasia; Qiu, 2023). A arquitetura do YOLOv8 é modular e

disponível em múltiplas variantes de escala (nano, small, medium, large, extra-large), permitindo adequação a diferentes requisitos de acurácia e velocidade.

A arquitetura do YOLOv8 é composta por três componentes principais:

- ***Backbone* (CSPDarknet + C2f)**: Responsável pela extração hierárquica de características em múltiplas escalas. O YOLOv8 utiliza blocos C2f (*CSP Bottleneck with 2 convolutions – faster*) ao invés dos blocos C3 do YOLOv5, proporcionando maior eficiência computacional com melhor fluxo de gradientes através da concatenação de saídas de múltiplos *bottlenecks* (Wang et al., 2020);
- ***Neck* (PANet + SPPF)**: Implementa o *Path Aggregation Network* para agregação de características de diferentes níveis da hierarquia, combinando informações semânticas de alto nível com detalhes espaciais de baixo nível (Liu et al., 2018). O módulo SPPF (*Spatial Pyramid Pooling – Fast*) aumenta o campo receptivo de forma eficiente;
- ***Head* (*Decoupled Anchor-free*)**: O YOLOv8 utiliza cabeças desacopladas com ramos separados para classificação e regressão, além de detecção *anchor-free* que elimina a dependência de âncoras pré-definidas. Diferentemente de versões anteriores, o YOLOv8 elimina o ramo de objetividade (*objectness*), delegando a confiança da detecção inteiramente ao ramo de classificação. A função de perda é composta por $\mathcal{L}_{box}$ (CIoU), $\mathcal{L}_{dfl}$ (*Distribution Focal Loss*) e $\mathcal{L}_{cls}$ (*Binary Cross-Entropy*).

Estudos comparativos demonstram que o YOLOv8 oferece desempenho superior às versões anteriores tanto em acurácia quanto em velocidade de inferência (Terven; Cordova-Esparza, 2023).

A Figura 2.3 ilustra a arquitetura do modelo YOLOv8s, destacando os três componentes principais e o fluxo de dados através da rede.

### 2.2.4 *Non-Maximum Suppression*

O algoritmo de *Non-Maximum Suppression* (NMS) é responsável por eliminar detecções redundantes e reter apenas a predição de maior confiança para cada objeto. Detectores densos como YOLO geram múltiplas predições sobrepostas para o mesmo objeto, tornando o NMS uma etapa essencial do *pipeline*. O processo seleciona iterativamente a detecção com maior *score* de confiança, remove todas as detecções com *overlap* (IoU) acima de um limiar, e repete até processar todas as detecções.

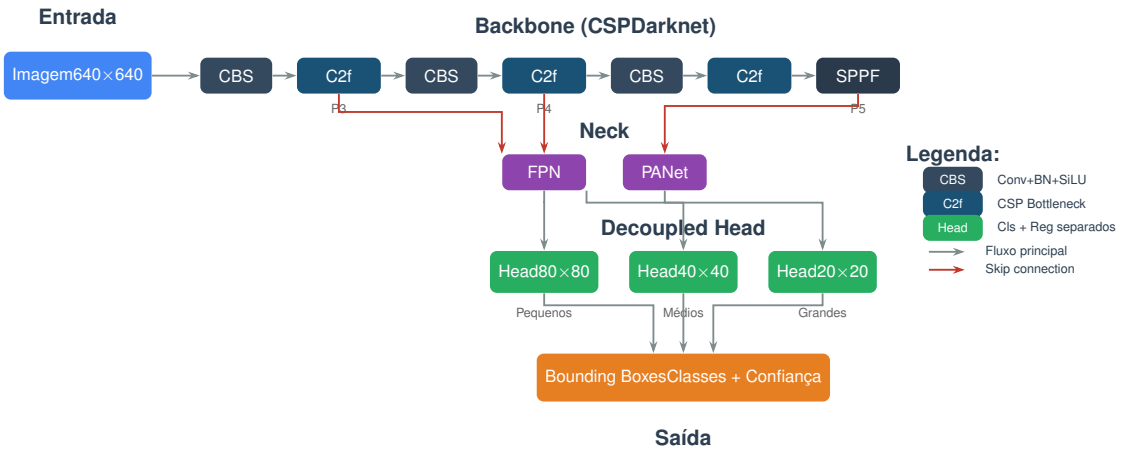



Figura 2.3 – Arquitetura do modelo YOLOv8s. O *backbone* CSPDarknet utiliza blocos C2f para extração de características em três escalas (P3, P4, P5). O *neck* combina FPN (*Feature Pyramid Network*) e PANet com blocos C2f para fusão de características. As cabeças de detecção desacopladas (*anchor-free*) operam com ramos separados para classificação e regressão.
Fonte: Elaborado pelo autor.

A Figura 2.4 ilustra o funcionamento do algoritmo NMS, demonstrando como múltiplas detecções sobrepostas são reduzidas a uma única detecção.

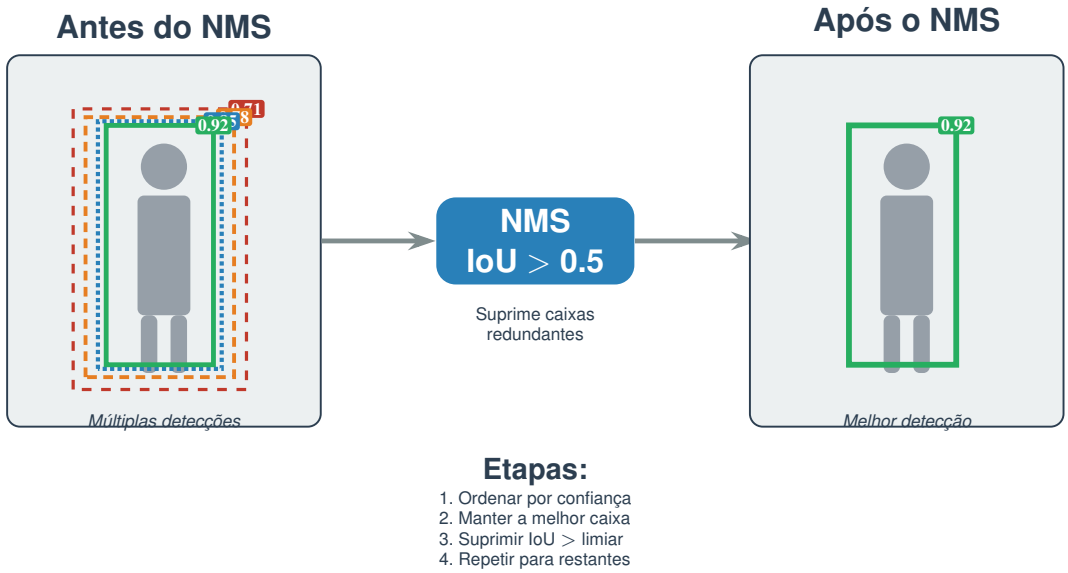



Figura 2.4 – Funcionamento do algoritmo *Non-Maximum Suppression* (NMS). Antes do NMS, múltiplas caixas delimitadoras sobrepostas com diferentes *scores* de confiança detectam o mesmo objeto. O algoritmo seleciona a caixa com maior confiança e suprime as demais com IoU acima do limiar.
Fonte: Elaborado pelo autor.

### 2.2.5 Métricas de Avaliação

A avaliação de detectores de objetos emprega métricas que consideram simultaneamente a qualidade da localização espacial e da classificação semântica (Padilla; Netto; Silva, 2020):

- **Intersection over Union (IoU)**: Mede a sobreposição entre a *bounding box* predita e a *ground truth*, definida como a razão entre a área de interseção e a área de união das caixas;
- **Precisão e Revocação**: Precisão é a fração de detecções corretas entre todas as detecções; revocação é a fração de objetos *ground truth* corretamente detectados;
- **Average Precision (AP)**: Área sob a curva precisão-revocação para uma classe específica, considerando um limiar de IoU (tipicamente 0,5 ou 0,5:0,95);
- **mean Average Precision (mAP)**: Média da AP sobre todas as classes, sendo a métrica padrão para comparação de detectores no *benchmark* COCO (Lin et al., 2014).

## 2.3 Computação de Borda

A computação de borda representa um paradigma arquitetural que propõe a execução de processamento de dados próximo de sua fonte de geração, em contraste com o modelo centralizado de computação em nuvem (Shi et al., 2016). Este paradigma emergiu como resposta às limitações da arquitetura centralizada para aplicações que requerem baixa latência, operação autônoma ou processamento de grandes volumes de dados sensoriais (Zhou et al., 2019).

### 2.3.1 Motivações para *Edge AI*

A aplicação de inteligência artificial na borda da rede é motivada por fatores técnicos, econômicos e operacionais (Chen; Ran, 2019):

- **Latência ultra-baixa**: Aplicações de tempo real requerem respostas em milissegundos. A latência para *data centers* varia tipicamente de 50 a 200 milissegundos, valor incompatível com requisitos de tempo real crítico. O processamento na borda

reduz esta latência para menos de 50 milissegundos, viabilizando aplicações como monitoramento de segurança (Xu; Razavi; Zheng, 2023);

- **Conservação de largura de banda**: A transmissão contínua de *streams* de vídeo consome largura de banda substancial. Uma câmera 4K pode gerar até 25 Mbps de dados. O processamento na borda transmite apenas metadados estruturados, reduzindo o tráfego de rede em ordens de magnitude;
- **Privacidade**: Regulamentações como a LGPD (Lei Geral de Proteção de Dados) e GDPR (*General Data Protection Regulation*) impõem restrições sobre a transmissão e armazenamento de dados pessoais. O processamento local mantém os dados brutos no dispositivo de origem, transmitindo apenas informações agregadas ou anonimizadas;
- **Confiabilidade**: Sistemas de borda operam de forma autônoma durante interrupções de conectividade, característica crítica para ambientes industriais (Murshed et al., 2022).

### 2.3.2 Desafios do *Edge AI*

A implementação de sistemas de inteligência artificial na borda apresenta desafios específicos que demandam técnicas de otimização (Mendez et al., 2022):

- **Restrições computacionais**: Dispositivos de borda possuem capacidade computacional limitada em comparação com servidores GPU. Modelos devem ser otimizados através de técnicas como quantização, *pruning* e destilação de conhecimento (Liang et al., 2021);
- **Eficiência energética**: Muitos dispositivos operam com orçamento de potência limitado, especialmente em aplicações alimentadas por bateria ou com dissipação térmica restrita;
- **Heterogeneidade de *hardware***: Diferentes dispositivos empregam arquiteturas distintas (CPU, GPU, NPU, FPGA), exigindo otimizações específicas para cada plataforma (Ghaffar et al., 2021).

### 2.3.3 Plataforma NVIDIA Jetson

A família NVIDIA Jetson compreende sistemas embarcados projetados para aplicações de inteligência artificial na borda, oferecendo capacidade computacional de GPU em formatos compactos e energeticamente eficientes (NVIDIA Corporation, 2024d). O NVIDIA Jetson AGX Xavier, utilizado nesta dissertação, oferece equilíbrio entre desempenho computacional e maturidade do ecossistema de *software*, sendo amplamente adotado em aplicações de visão computacional embarcada (Prashanthi; Kesanapalli; Simmhan, 2022).

As especificações técnicas do Jetson AGX Xavier incluem:

- **GPU**: NVIDIA Volta com 512 CUDA cores e 64 Tensor Cores para aceleração de operações de inferência;
- **CPU**: Processador ARM Carmel de 8 núcleos com frequência de até 2,26 GHz;
- **Memória**: 32 GB de memória LPDDR4x unificada com 136,5 GB/s de largura de banda, compartilhada entre CPU e GPU;
- **Desempenho de IA**: Até 32 TOPS em operações INT8 (GPU + 2×DLA combinados), viabilizando inferência de múltiplos modelos simultaneamente;
- **Consumo de energia**: Perfis configuráveis entre 10 W e 30 W, permitindo *trade-off* entre desempenho e eficiência energética.

A Figura 2.5 apresenta a arquitetura de *hardware* do Jetson AGX Xavier, evidenciando os subsistemas de processamento e interfaces de entrada e saída.

Uma revisão das técnicas de otimização para a plataforma Jetson é apresentada por Mittal (2019).

## 2.4 NVIDIA DeepStream SDK

O NVIDIA DeepStream SDK é um *framework* para desenvolvimento de aplicações de análise de vídeo inteligente, otimizado para execução em *hardware* NVIDIA (NVIDIA Corporation, 2024c). O DeepStream é construído sobre o GStreamer, um *framework* maduro para processamento multimídia baseado no paradigma de *pipeline*. Esta escolha permite aproveitar a ampla base de *plugins* do GStreamer para entrada/saída de vídeo enquanto adiciona capacidades de inferência acelerada por GPU.

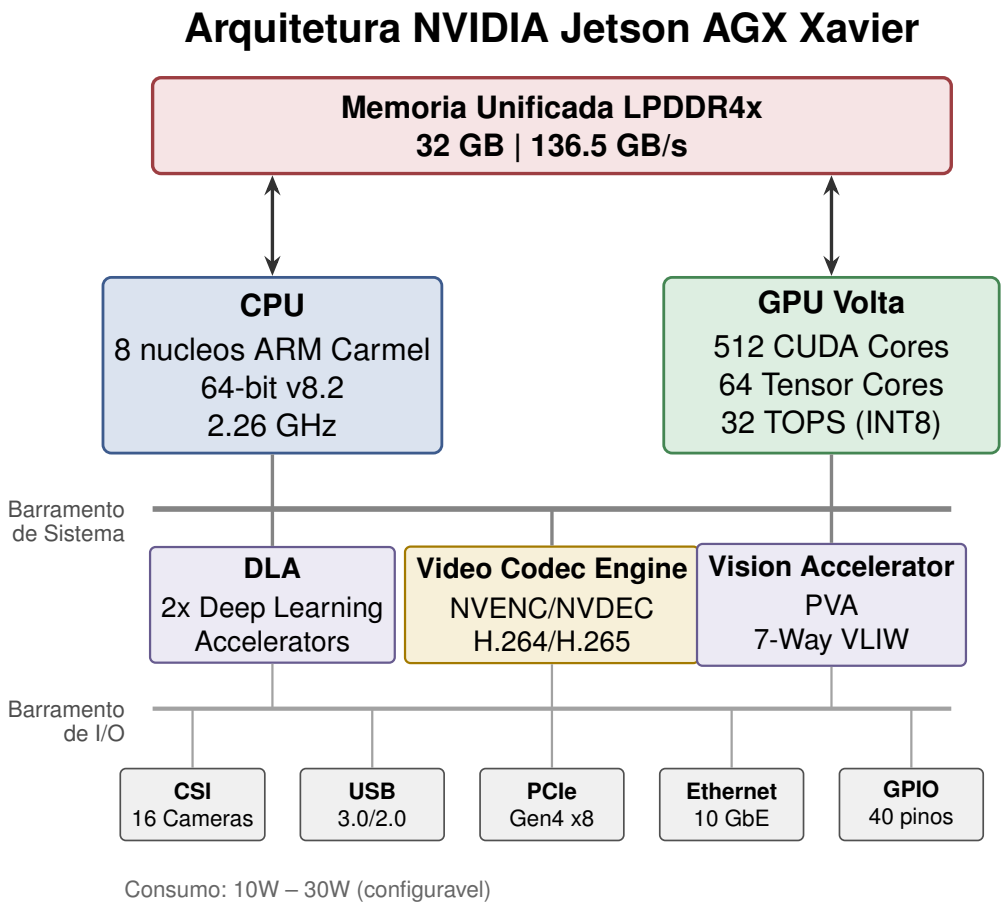



Figura 2.5 – Arquitetura de *hardware* do NVIDIA Jetson AGX Xavier. A memória unificada LPDDR4x é compartilhada entre CPU e GPU. Os aceleradores dedicados (DLA e PVA) permitem execução otimizada de modelos de *deep learning* com menor consumo energético.
Fonte: Elaborado pelo autor.

### 2.4.1 Arquitetura do DeepStream

No modelo GStreamer, o processamento é organizado como uma sequência de elementos conectados, onde cada elemento realiza uma transformação sobre os dados que fluem através do *pipeline*. O DeepStream estende o GStreamer com elementos customizados otimizados para GPU que implementam as operações de análise de vídeo com mínima cópia de memória entre CPU e GPU.

Os elementos fundamentais utilizados em aplicações de detecção incluem:

- **`nvstreammux`**: Multiplexador que combina múltiplos fluxos de vídeo em *batches* para processamento eficiente, maximizando a utilização da GPU;
- **`nvinfer`**: Elemento de inferência que executa modelos de redes neurais utilizando o TensorRT. Pode ser configurado como PGIE (*Primary GPU Inference Engine*) ou SGIE (*Secondary GPU Inference Engine*) para arquiteturas em cascata;

- **`nvtracker`**: Implementa algoritmos de rastreamento de objetos entre *frames* consecutivos, incluindo DeepSORT e NvDCF (NVIDIA Corporation, 2023);
- **`nvdsosd`**: Renderiza informações visuais (*bounding boxes*, *labels*, trilhas) sobre os *frames* de vídeo.

### 2.4.2 Inferência Primária e Secundária

O DeepStream suporta nativamente arquiteturas de inferência em cascata através dos conceitos de PGIE e SGIE (NVIDIA Corporation, 2024a):

- **PGIE (Primary GPU Inference Engine)**: Opera sobre o *frame* completo e detecta objetos de interesse. As detecções do PGIE definem as regiões de interesse (ROIs) que serão processadas pelos SGIEs subsequentes;
- **SGIE (Secondary GPU Inference Engine)**: Opera sobre as regiões recortadas (*crops*) das detecções do PGIE. Cada SGIE pode ser configurado para processar apenas classes específicas do PGIE, permitindo especialização por domínio.

Esta arquitetura em cascata permite que modelos menores e mais especializados operem sobre regiões de interesse compactas, ao invés de processar o *frame* completo repetidamente. O benefício é duplo: redução do custo computacional total e possibilidade de usar modelos otimizados para cada tipo de objeto, resultando em maior acurácia.

A Figura 2.6 ilustra a arquitetura de inferência em cascata, demonstrando o fluxo de processamento desde a detecção primária até a classificação especializada pelos modelos secundários.

### 2.4.3 Sistema de Metadados

O DeepStream implementa um sistema hierárquico de metadados que acompanha cada *frame* através do *pipeline*, permitindo que elementos *downstream* acessem informações geradas por elementos *upstream*:

- **`NvDsBatchMeta`**: *Container* raiz para metadados de um *batch* de *frames* processados simultaneamente;
- **`NvDsFrameMeta`**: Metadados de cada *frame* individual, incluindo índice da fonte, *timestamp* e dimensões;

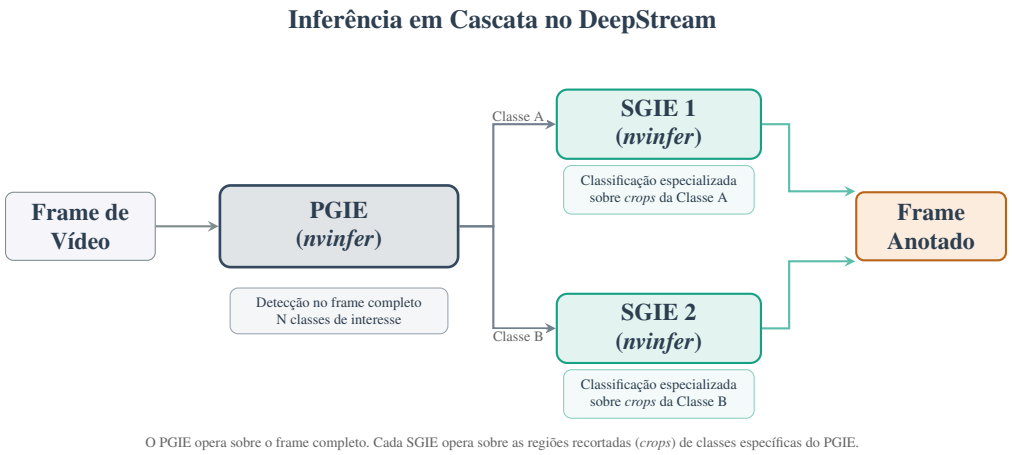



Figura 2.6 – Arquitetura de inferência em cascata do DeepStream. O PGIE processa o *frame* completo e detecta objetos de interesse. Os SGIEs operam sobre as regiões recortadas (*crops*) das detecções do PGIE, permitindo classificação especializada. Cada SGIE pode ser configurado para processar apenas classes específicas do PGIE.
Fonte: Adaptado de NVIDIA Corporation (2024a).

- **`NvDsObjectMeta`**: Metadados de cada objeto detectado, incluindo *bounding box*, classe, confiança e ID de rastreamento atribuído pelo *tracker*.

Esta estrutura hierárquica permite que aplicações acessem e manipulem os resultados de inferência de forma eficiente, sem necessidade de *parsing* de tensores brutos.

## 2.5 TensorRT e Otimização de Modelos

O NVIDIA TensorRT é uma biblioteca de otimização de inferência que maximiza o desempenho de redes neurais em *hardware* NVIDIA (NVIDIA Corporation, 2024e). O TensorRT recebe modelos treinados em *frameworks* como PyTorch ou TensorFlow, exportados para formato ONNX, e aplica otimizações específicas para o *hardware* alvo. Estudos demonstram que o TensorRT pode acelerar a inferência em 2-6x comparado com *frameworks* de treinamento (Jeong; Kim; Ha, 2022).

### 2.5.1 Otimizações Aplicadas

O processo de otimização do TensorRT inclui:

- **Fusão de camadas**: Combina sequências de operações (Conv-BatchNorm-ReLU) em *kernels* únicos, reduzindo *overhead* de lançamento de *kernels* e acesso à memória;

- **Seleção automática de *kernel***: Para cada operação, o TensorRT perfila múltiplas implementações de *kernel* e seleciona a mais rápida para o *hardware* específico;
- **Otimização de memória**: Reutiliza *buffers* de memória entre camadas quando possível, reduzindo o *footprint* de memória;
- **Calibração de precisão**: Converte pesos e ativações para precisões reduzidas (FP16 ou INT8) com calibração automática para minimizar perda de acurácia.

### 2.5.2 Quantização

A quantização de modelos para precisões reduzidas é uma das técnicas mais eficazes para aceleração de inferência em dispositivos de borda (Gholami et al., 2022). Os Tensor Cores das GPUs NVIDIA Volta e posteriores oferecem *throughput* significativamente maior para operações em precisão reduzida:

- **FP16**: Representa valores em 16 bits de ponto flutuante. Tipicamente dobra o *throughput* de inferência com impacto mínimo na acurácia ($< 1\%$ de degradação para a maioria dos modelos);
- **INT8**: Representa valores como inteiros de 8 bits. Oferece ganhos de 4-8x sobre FP32, mas requer calibração cuidadosa com *dataset* representativo para minimizar a degradação de acurácia.

A escolha entre FP16 e INT8 envolve *trade-off* entre *speedup* e complexidade de calibração. Para modelos YOLOv8 em aplicações de detecção de objetos, a quantização FP16 oferece excelente equilíbrio, sendo adotada nesta dissertação (Mittal, 2019).

Com os fundamentos teóricos estabelecidos, o próximo capítulo apresenta uma revisão dos trabalhos relacionados, analisando as abordagens existentes para detecção de objetos em canteiros de obras e identificando as lacunas que motivam a arquitetura proposta nesta dissertação.

# 3 TRABALHOS RELACIONADOS

Este capítulo apresenta uma revisão dos trabalhos relacionados nas áreas de visão computacional aplicada à segurança na construção civil, sistemas de detecção de EPIs e aplicações de inteligência artificial em dispositivos de borda.

## 3.1 Visão Computacional na Construção Civil

A aplicação de técnicas de visão computacional para monitoramento de segurança em canteiros de obras tem sido objeto de crescente interesse acadêmico ao longo da última década (Fang et al., 2020). A disponibilidade de câmeras de vigilância em canteiros e os avanços em aprendizado profundo viabilizaram sistemas de monitoramento automatizado.

Seo et al. (2015) apresentaram uma das primeiras revisões das técnicas de visão computacional para monitoramento de segurança na construção civil. O trabalho categorizou as aplicações em quatro áreas: detecção de EPIs, reconhecimento de atividades de risco, identificação de zonas perigosas e monitoramento de conformidade. Esta taxonomia permanece relevante para classificar trabalhos subsequentes.

Fang et al. (2018) propuseram um sistema baseado em *deep learning* para detecção de não uso de capacete a partir de vídeos de vigilância. O trabalho utilizou uma arquitetura Faster R-CNN modificada e demonstrou a viabilidade de detectar violações mesmo em vídeos de baixa resolução. Os resultados indicaram precisão de 95% em condições controladas.

Luo et al. (2018) desenvolveram uma abordagem para reconhecimento de atividades de construção através da detecção e análise de relações espaciais entre objetos. A metodologia propôs que a presença e configuração de equipamentos podem ser utilizadas para inferir a atividade sendo executada.

Teizer (2015) identificou desafios específicos do domínio de construção civil para sistemas de visão: variabilidade de iluminação, poeira, oclusões frequentes e movimentação constante de equipamentos. O autor argumentou que estes fatores demandam soluções robustas e adaptáveis.

## 3.2 Sistemas de Detecção de EPIs

A detecção automática de Equipamentos de Proteção Individual representa a aplicação mais estudada de visão computacional para segurança ocupacional na construção civil. Uma revisão sistemática recente identificou crescimento exponencial de publicações na área entre 2018 e 2023 (Konstantinov et al., 2024).

Nath, Behzadan and Paal (2020) propuseram um sistema para detecção de múltiplas categorias de EPIs em tempo real, incluindo capacetes, coletes, luvas e óculos. O trabalho utilizou uma arquitetura Faster R-CNN com *backbone* ResNet-101. Os resultados indicaram mAP de 72%, com variação entre classes: capacetes e coletes apresentaram melhor desempenho (AP > 80%), enquanto itens menores como luvas foram mais desafiadores (AP < 60%).

Wu et al. (2019) disponibilizaram o Safety Helmet Wearing *Dataset* (SHWD) contendo mais de 7.500 imagens com anotações para capacetes. Os autores apresentaram uma análise comparativa de diferentes arquiteturas, concluindo que o YOLOv3 oferecia o melhor *trade-off* entre acurácia (mAP de 85%) e velocidade.

Mneymneh, Abbas and Khoury (2019) propuseram um *framework* para monitoramento de capacetes que integra detecção com rastreamento de trabalhadores. A combinação de detector e *tracker* permitiu análise temporal de conformidade, identificando padrões de uso inadequado ao longo do dia de trabalho.

Wang et al. (2021) realizaram uma comparação entre as arquiteturas YOLOv3, YOLOv4 e YOLOv5 para detecção de EPIs. Barlybayev et al. (2024) estenderam esta análise para incluir o YOLOv8, demonstrando que a arquitetura *anchor-free* e os blocos C2f proporcionam mAP@0,5 superior a 90% com maior eficiência computacional.

Delhi, Sankarlal and Thomas (2020) propuseram um sistema de verificação de conformidade de EPIs utilizando SSD. O trabalho enfatizou os desafios do domínio, incluindo variações de iluminação, oclusões parciais e variabilidade de ângulos de câmera. Os autores reportaram mAP de 78%.

Lee et al. (2023) desenvolveram um *framework* para monitoramento de EPIs em tempo real com foco na eficiência computacional. O trabalho demonstrou que técnicas de quantização podem reduzir a latência em 50% com perda de menos de 2% em acurácia.

## 3.3 *Edge AI* para Monitoramento em Tempo Real

A implantação de sistemas de visão computacional em dispositivos de borda representa um desafio técnico devido às restrições de recursos computacionais. A literatura recente demonstra crescente interesse em soluções que equilibrem acurácia e eficiência (Murshed et al., 2022).

Mittal (2019) apresentaram uma revisão das técnicas de otimização para execução de modelos na plataforma NVIDIA Jetson. Os autores identificaram que a quantização para INT8 pode oferecer *speedups* de 3-4x com degradação de acurácia inferior a 1%.

Chen and Ran (2019) discutiram as oportunidades e desafios da convergência entre *deep learning* e computação de borda. Os autores argumentaram que a inteligência na borda pode ser distribuída em múltiplos níveis, desde *edge devices* até *edge servers*, conforme os requisitos de latência e capacidade computacional.

Xu, Razavi and Zheng (2023) apresentaram uma revisão sobre *video analytics* na borda. O trabalho identificou que a latência é o principal desafio para aplicações de segurança, onde segundos podem determinar a diferença entre prevenção e acidente.

Badidi, Moumane and Ghazi (2023) analisaram aplicações de *Edge AI* para *video analytics* em cidades inteligentes. Os autores identificaram monitoramento de segurança ocupacional como caso de uso prioritário devido aos requisitos de baixa latência e operação autônoma.

Shin and Kim (2022) avaliaram a *performance* de diferentes versões do YOLO na plataforma Jetson. O YOLOv8s, com sua arquitetura otimizada, alcança desempenho superior ao YOLOv5s no Jetson AGX Xavier com precisão FP16, beneficiando-se dos blocos C2f mais eficientes.

Ganesh et al. (2022) propuseram otimizações específicas para execução de YOLO em GPUs de borda, demonstrando que ajustes na arquitetura podem melhorar o *throughput* sem comprometer a acurácia. Os resultados validaram a viabilidade de detecção em tempo real em dispositivos com restrições de recursos.

## 3.4 Arquiteturas de Inferência em Cascata

A inferência em cascata, onde múltiplos modelos operam em sequência, tem sido explorada para melhorar a precisão e eficiência de sistemas de detecção. O princípio fundamental é decompor uma tarefa complexa em subtarefas especializadas.

Cai and Vasconcelos (2021) propuseram o Cascade R-CNN, que utiliza múltiplos estágios de detecção com *thresholds* crescentes de IoU para melhorar progressivamente a qualidade das detecções. Esta abordagem demonstrou que a inferência em cascata pode superar detectores de estágio único em termos de precisão de localização, estabelecendo um paradigma adotado em trabalhos subsequentes.

Singh and Davis (2018) demonstraram que objetos de diferentes escalas requerem tratamento especializado. A detecção multi-escala em cascata permite que cada estágio foque em uma faixa de tamanhos, melhorando a acurácia geral do sistema.

O NVIDIA DeepStream SDK suporta nativamente arquiteturas de inferência em cascata através dos conceitos de PGIE e SGIE (NVIDIA Corporation, 2024a). O PGIE processa o *frame* completo para detecção inicial, enquanto SGIEs operam sobre regiões recortadas para classificação especializada. Esta capacidade permite que sistemas de produção utilizem modelos especializados para diferentes tarefas com orquestração automática pelo *framework*.

A combinação de inferência em cascata com especialização de domínio permanece pouco explorada para aplicações de segurança em construção civil. Os trabalhos existentes utilizam predominantemente modelos únicos que detectam todas as classes simultaneamente, limitando a precisão para categorias diversas como EPIs, maquinário e ferramentas.

## 3.5 Análise Comparativa

A Tabela 3.1 apresenta uma comparação dos principais trabalhos relacionados, considerando arquitetura utilizada, plataforma de execução, métricas de acurácia e limitações identificadas.

A análise comparativa permite identificar limitações recorrentes nos trabalhos existentes. O trabalho de Nath, Behzadan and Paal (2020), apesar de alcançar boa acurácia com Faster R-CNN, apresenta latência elevada que inviabiliza aplicações em tempo real. Wu et al. (2019) e Barlybayev et al. (2024) obtiveram resultados expressivos com YOLOv3 e YOLOv8, respectivamente, porém ambos foram validados apenas em servidores GPU, sem avaliação em dispositivos de borda. Os trabalhos de Wang et al. (2021) e Shin and Kim (2022) utilizam modelo único para todas as classes, o que limita a precisão quando o número de categorias aumenta ou quando os objetos possuem escalas muito distintas. Delhi, Sankarlal and Thomas (2020) reportou resultados com SSD, porém o *da-*

Tabela 3.1 – Comparação dos trabalhos relacionados.

| Trabalho | Arquitetura | Plataforma | Classes | mAP@0,5 | Limitações |
|---|---|---|---|---|---|
| Nath et al. (2020) | Faster R-CNN | GPU Server | 5 | 0,72 | Latência elevada |
| Wu et al. (2019) | YOLOv3 | GPU Server | 2 | 0,85 | Sem validação em borda |
| Wang et al. (2021) | YOLOv5 | GPU Server | 5 | 0,90 | Modelo único |
| Delhi et al. (2020) | SSD | GPU Server | 4 | 0,78 | *Dataset* limitado |
| Shin et al. (2022) | YOLOv5 | Jetson Xavier | 5 | 0,82 | Modelo único |
| Barlybayev et al. (2024) | YOLOv8 | GPU Server | 7 | 0,90 | Sem validação em borda |
| **Este trabalho** | **YOLOv8s (cascata)** | **Jetson Xavier** | **46**† | **0,71–0,84*** | - |

* Faixa de mAP@0,5 dos SGIEs. O PGIE alcançou mAP@0,5 de 0,83.
† 46 categorias únicas distribuídas em 4 modelos (PGIE + 3 SGIEs).
Fonte: Elaborado pelo autor com base nos trabalhos citados.

*taset* utilizado era limitado em diversidade e quantidade de instâncias. O único trabalho com validação em dispositivo de borda, Shin and Kim (2022), opera com modelo único de 5 classes no Jetson Xavier, sem especialização por domínio.

A análise da literatura revela três lacunas principais que motivam as contribuições desta dissertação:

1. **Predominância de modelo único**: A maioria dos trabalhos utiliza um único modelo para todas as classes, o que limita a precisão para categorias heterogêneas como EPIs (objetos pequenos), maquinário (objetos grandes) e ferramentas (alta variabilidade);
2. **Foco em servidor GPU**: Poucos trabalhos validam *performance* em dispositivos de borda, onde restrições de memória e processamento impactam diretamente a viabilidade de implantação;
3. **Ausência de inferência em cascata para construção**: Nenhum trabalho identificado utiliza arquitetura PGIE + SGIEs especializados para o domínio de segurança em construção civil.

Esta dissertação diferencia-se ao apresentar uma solução com inferência em cascata (PGIE + 3 SGIEs) validada em dispositivo de borda Jetson AGX Xavier, combinando especialização de domínio com execução em *hardware* de custo acessível.

No capítulo seguinte, descreve-se a metodologia empregada no desenvolvimento do sistema, detalhando a arquitetura proposta, os *datasets* utilizados, o processo de treinamento dos modelos e a integração com o *framework* DeepStream.

# 4 METODOLOGIA

Este capítulo descreve a metodologia empregada no desenvolvimento do sistema de monitoramento de segurança, endereçando as três lacunas identificadas na análise da literatura (Capítulo 3): predominância de modelo único, foco em servidor GPU e ausência de inferência em cascata para o domínio de construção civil. São apresentados a arquitetura geral do sistema, a composição dos modelos PGIE e SGIEs, o processo de preparação dos *datasets*, as configurações de treinamento e a integração com o NVIDIA DeepStream.

## 4.1 Arquitetura do Sistema

A Figura 4.1 apresenta a visão geral do sistema proposto. Câmeras IP transmitem fluxos de vídeo via protocolo RTSP para o dispositivo Jetson AGX Xavier, que executa localmente o *pipeline* de inferência em cascata. O dispositivo processa os *frames* em tempo real, realiza a análise de conformidade de EPIs e gera dois tipos de saída: alertas estruturados (via MQTT/JSON) para integração com sistemas de notificação, e vídeo anotado com as detecções para visualização ou gravação. Todo o processamento ocorre na borda, sem dependência de conectividade externa.

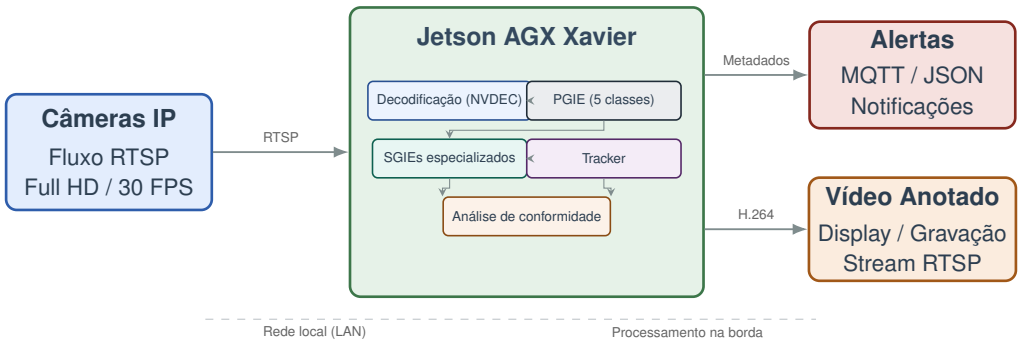



Figura 4.1 – Visão geral do sistema. Câmeras IP transmitem vídeo via RTSP para o Jetson AGX Xavier, que executa o *pipeline* de inferência em cascata na borda. As saídas incluem alertas de não conformidade e vídeo anotado com as detecções.
Fonte: Elaborado pelo autor.

O sistema proposto utiliza uma arquitetura de inferência em cascata composta por um modelo primário (PGIE) e três modelos secundários (SGIEs), seguindo o paradigma de detecção hierárquica que demonstra melhorias em precisão de localização (Cai; Vasconcelos, 2021). A Figura 4.2 ilustra a arquitetura em cascata proposta neste trabalho, com as classes específicas de cada modelo.

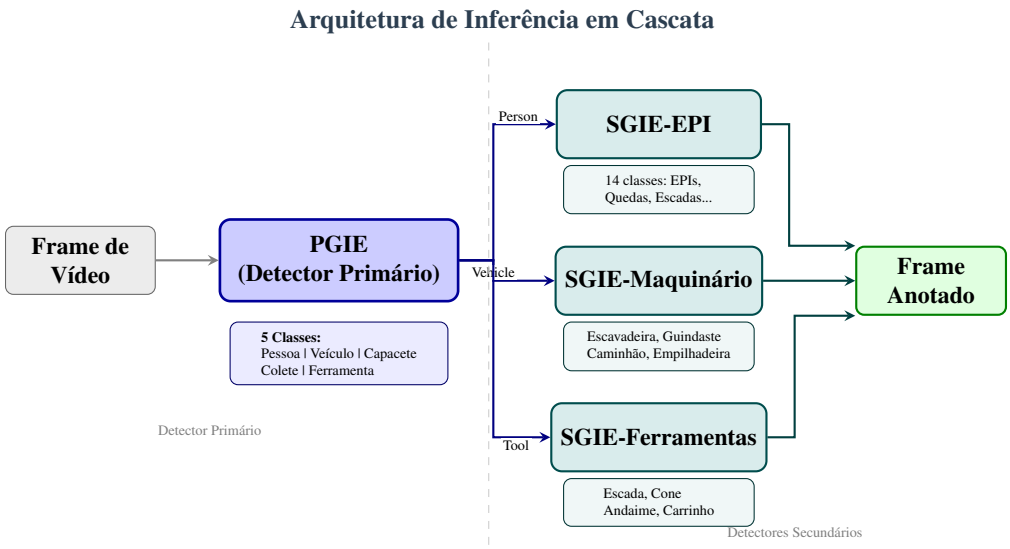



Figura 4.2 – Arquitetura de inferência em cascata proposta. O PGIE detecta cinco classes no *frame* completo. Três dessas classes são encaminhadas a SGIEs especializados: Person ao SGIE-EPI, Vehicle ao SGIE-Maquinário e Tool ao SGIE-Ferramentas. As classes Hardhat e Safety-vest são utilizadas diretamente pelo sistema de análise.
Fonte: Elaborado pelo autor.

### 4.1.1 Modelo Primário (PGIE)

O PGIE é responsável pela detecção inicial de objetos no *frame* completo, operando como primeiro estágio da cascata (NVIDIA Corporation, 2024a). O modelo detecta cinco classes que representam os principais elementos de um canteiro de obras:

- **Person**: Trabalhadores e outras pessoas presentes no canteiro;
- **Vehicle**: Veículos em geral (caminhões, carros, vans) e maquinário pesado;
- **Hardhat**: Capacetes de segurança visíveis no *frame*;
- **Safety-vest**: Coletes de alta visibilidade detectados no *frame*;
- **Tool**: Ferramentas, materiais e objetos diversos.

As detecções do PGIE definem as regiões de interesse (*crops*) que serão encaminhadas aos SGIEs. Das cinco classes, três são associadas a SGIEs específicos: Person é encaminhada ao SGIE-EPI, Vehicle ao SGIE-Maquinário e Tool ao SGIE-Ferramentas. As classes Hardhat e Safety-vest são utilizadas diretamente pelo PGIE como detecções de primeiro nível, fornecendo informações complementares ao sistema de análise de conformidade sem necessidade de classificação secundária (Singh; Davis, 2018).

### 4.1.2 Modelos Secundários (SGIEs)

Os SGIEs são modelos especializados que operam sobre as regiões recortadas das detecções do PGIE. Esta abordagem permite que cada SGIE seja otimizado para um domínio específico, melhorando a precisão em categorias heterogêneas (Nath; Behzadan; Paal, 2020):

**SGIE-EPI** (opera sobre *crops* de Person): O modelo detecta 14 classes relacionadas a equipamentos de proteção individual e situações de risco:

- Hardhat / NO-Hardhat: Presença ou ausência de capacete de segurança;
- Safety Vest / NO-Safety Vest: Presença ou ausência de colete de alta visibilidade;
- Gloves / NO-Gloves: Presença ou ausência de luvas de proteção;
- Goggles / NO-Goggles: Presença ou ausência de óculos de proteção;
- Mask / NO-Mask: Presença ou ausência de máscara de proteção;
- Fall-Detected: Detecção de queda de trabalhador;
- Ladder: Escada presente na região;
- Safety Cone: Cone de sinalização;
- Person: Confirmação da presença de pessoa na região recortada, utilizada como referência espacial para associação dos EPIs detectados ao trabalhador correspondente.

**SGIE-Maquinário** (opera sobre *crops* de Vehicle):

- Excavator: Escavadeira;
- Bulldozer: Trator de esteira;
- Crane: Guindaste;
- Loader: Carregadeira;
- Concrete Mixer: Betoneira;
- Roller: Rolo compactador;
- Compactor: Compactador;
- Grader: Motoniveladora;
- Truck: Caminhão.

**SGIE-Ferramentas** (opera sobre *crops* de Tool):

- Board: Tábua;

- Brick: Tijolo;
- Rebar: Vergalhão;
- Wood: Madeira;
- Scaffold: Andaime;
- Ladder: Escada;
- Handcart: Carrinho de mão;
- Hopper: Caçamba;
- Ebox: Caixa elétrica;
- Safety Cone: Cone de sinalização;
- Bucket: Balde;
- Cutter: Cortador;
- Drill: Furadeira;
- Grinder: Esmerilhadeira;
- Knife: Faca;
- Saw: Serra;
- Hammer: Martelo;
- Shovel: Pá;
- Spanner: Chave de boca;
- Tacker: Grampeador;
- Trowel: Colher de pedreiro;
- Wrench: Chave inglesa;
- Hook: Gancho.

### 4.1.3 Fluxo de Processamento

O fluxo de processamento segue as etapas definidas pelo *pipeline* DeepStream (NVIDIA Corporation, 2024c):

1. O *frame* de vídeo é decodificado e pré-processado;
2. O PGIE processa o *frame* completo e detecta as cinco classes;
3. Para cada detecção do PGIE, a região correspondente é recortada;
4. Cada *crop* é enviado ao SGIE apropriado baseado na classe do PGIE;

5. Os SGIEs processam os *crops* e adicionam detecções especializadas;
6. O *tracker* atribui IDs persistentes aos objetos (NVIDIA Corporation, 2024b);
7. A lógica de análise verifica conformidade de EPIs e gera alertas.

## 4.2 *Datasets*

Os modelos foram treinados utilizando *datasets* públicos de construção civil, complementados por imagens coletadas localmente. A qualidade e representatividade dos dados de treinamento são determinantes para o desempenho de detectores baseados em aprendizado profundo (Zhao et al., 2019).

### 4.2.1 Fontes de Dados

**Para o PGIE:**

- **SODA (Safety-related Object Detection in Construction)**: *Dataset* com anotações para pessoas, veículos e equipamentos em canteiros de obras;
- **MOCS (Multi-Object Construction Site)**: *Dataset* com múltiplas classes de objetos de construção;
- **Roboflow Construction**: Coleção de *datasets* de construção civil do Roboflow Universe.

**Para o SGIE-EPI:**

- **SHWD (Safety Helmet Wearing Dataset)** (Wang, 2021): Aproximadamente 7.500 imagens com anotações para capacetes;
- **GDUT-HWD** (Chen et al., 2020): *Dataset* para detecção de capacetes com anotações verificadas;
- **PPE Detection Dataset (Roboflow)**: *Dataset* com múltiplas classes de EPIs.

**Para o SGIE-Maquinário:**

- **Construction Vehicles Dataset**: *Dataset* com classes de veículos e máquinas de construção;
- **Heavy Equipment Dataset**: *Dataset* focado em maquinário pesado.

**Para o SGIE-Ferramentas:**

- **Construction Tools Dataset**: *Dataset* com ferramentas e materiais de construção;
- **Safety Equipment Dataset**: *Dataset* com equipamentos de sinalização.

### 4.2.2 Processo de Curadoria

A combinação de múltiplos *datasets* de origens distintas requer um processo de curadoria para garantir a consistência e qualidade dos dados de treinamento (Lin et al., 2014). O processo adotado envolveu cinco etapas:

1. **Remoção de duplicatas**: Imagens duplicadas ou quase duplicadas foram identificadas e removidas através de *hashing* perceptual (pHash). Esta técnica gera uma impressão digital compacta de cada imagem e permite detectar duplicatas mesmo com pequenas variações de compressão ou redimensionamento. A remoção de duplicatas evita viés no treinamento e previne vazamento de dados entre os conjuntos de treinamento e validação;
2. **Verificação de anotações**: As anotações de *bounding box* foram revisadas manualmente em amostras aleatórias de cada *dataset*. Anotações com erros de localização, classes incorretas ou objetos não anotados foram corrigidas ou removidas;
3. **Padronização de classes**: Como os *datasets* utilizam nomenclaturas distintas para as mesmas categorias (por exemplo, “hard hat” e “helmet”), foi realizado o mapeamento de todas as classes para um vocabulário unificado;
4. **Balanceamento de classes**: Classes com baixa representação foram sobre-amostradas através de replicação de imagens, enquanto classes com excesso de instâncias foram sub-amostradas aleatoriamente. O objetivo foi reduzir o desbalanceamento sem comprometer a diversidade do conjunto de treinamento;
5. **Divisão dos dados**: O *dataset* final foi dividido em conjuntos de treinamento (70%), validação (20%) e teste (10%), com estratificação por classe para manter a proporção de cada categoria em todos os conjuntos.

### 4.2.3 *Data Augmentation*

Técnicas de *data augmentation* foram aplicadas durante o treinamento para aumentar a diversidade do conjunto de dados e melhorar a capacidade de generalização

dos modelos (Shorten; Khoshgoftaar, 2019). As transformações são aplicadas de forma aleatória a cada imagem em cada época de treinamento, de modo que o modelo recebe variações diferentes do mesmo dado ao longo do processo:

- ***Mosaic augmentation***: Combina quatro imagens de treinamento em uma única composição, redimensionando e posicionando cada imagem em um quadrante (Bochkovskiy; Wang; Liao, 2020). Esta técnica aumenta a diversidade de contexto e escala em cada *batch*, forçando o modelo a lidar com objetos em diferentes posições e tamanhos relativos;
- **Transformações geométricas**: Rotação aleatória (até 10°), variação de escala (0,5x a 1,5x) e *flip* horizontal. Estas transformações simulam variações de posicionamento e ângulo da câmera presentes em canteiros de obras reais;
- **Transformações fotométricas**: Variações aleatórias de brilho ($\pm$20%), contraste ($\pm$20%) e saturação ($\pm$30%). Estas transformações simulam as variações de iluminação encontradas em ambientes externos, desde luz solar direta até condições nubladas ou sombreadas;
- ***Cutout***: Oclusão aleatória de regiões retangulares da imagem durante o treinamento. Esta técnica melhora a robustez do modelo a oclusões parciais, situação comum em canteiros de obras onde trabalhadores e equipamentos frequentemente se sobrepõem.

### 4.2.4 Caracterização do *Dataset*

A Figura 4.3 apresenta a distribuição estatística do *dataset* de treinamento do PGIE, incluindo a contagem de instâncias por classe e a distribuição espacial das anotações.

A Figura 4.4 apresenta a distribuição do *dataset* do SGIE-Ferramentas, composto por 23 classes de equipamentos e ferramentas de construção. O *dataset* apresenta desbalanceamento significativo entre as classes, com algumas categorias possuindo maior representatividade que outras.

A Figura 4.5 apresenta a distribuição do *dataset* do SGIE-Maquinário, composto por 9 classes de maquinário pesado. Este *dataset* apresenta distribuição mais equilibrada entre as classes em comparação ao SGIE-Ferramentas.

A Figura 4.6 ilustra uma amostra do *batch* de treinamento, demonstrando a diver-

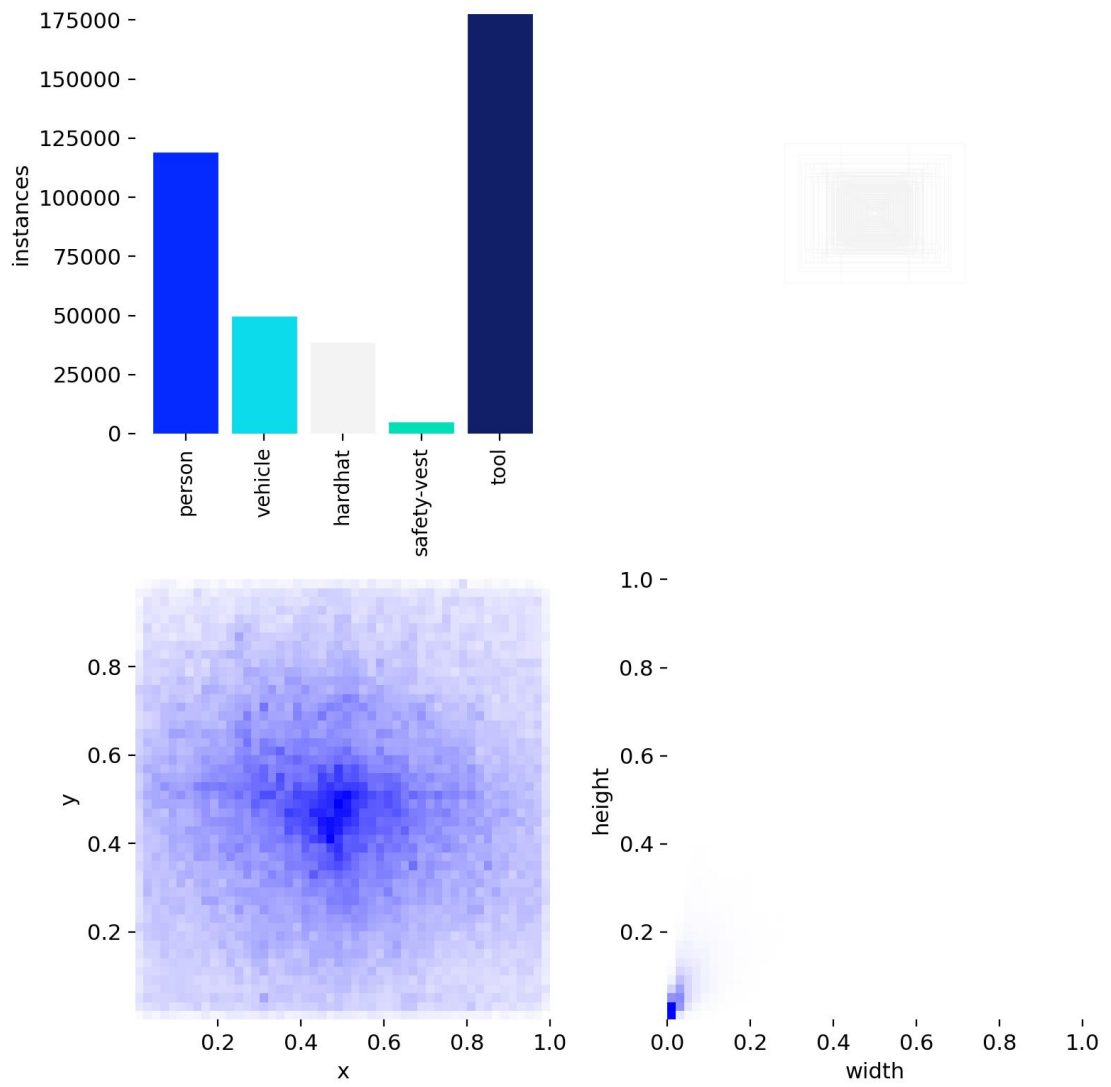



Figura 4.3 – Distribuição estatística do *dataset* de treinamento do PGIE com 5 classes. O gráfico mostra a contagem de instâncias por classe (person, vehicle, hardhat, safety-vest, tool) e a distribuição espacial das anotações.
Fonte: Elaborado pelo autor com dados do treinamento.

sidade de cenários e a qualidade das anotações de *ground truth*.

## 4.3 Treinamento dos Modelos

Todos os modelos foram treinados utilizando a arquitetura YOLOv8s (*small*) (Jocher; Chaurasia; Qiu, 2023), selecionada pelo equilíbrio entre acurácia e velocidade de inferência em dispositivos de borda (Terven; Cordova-Esparza, 2023). O YOLOv8s oferece vantagens sobre versões anteriores, incluindo arquitetura *anchor-free*, blocos C2f mais eficientes e cabeças de detecção desacopladas.

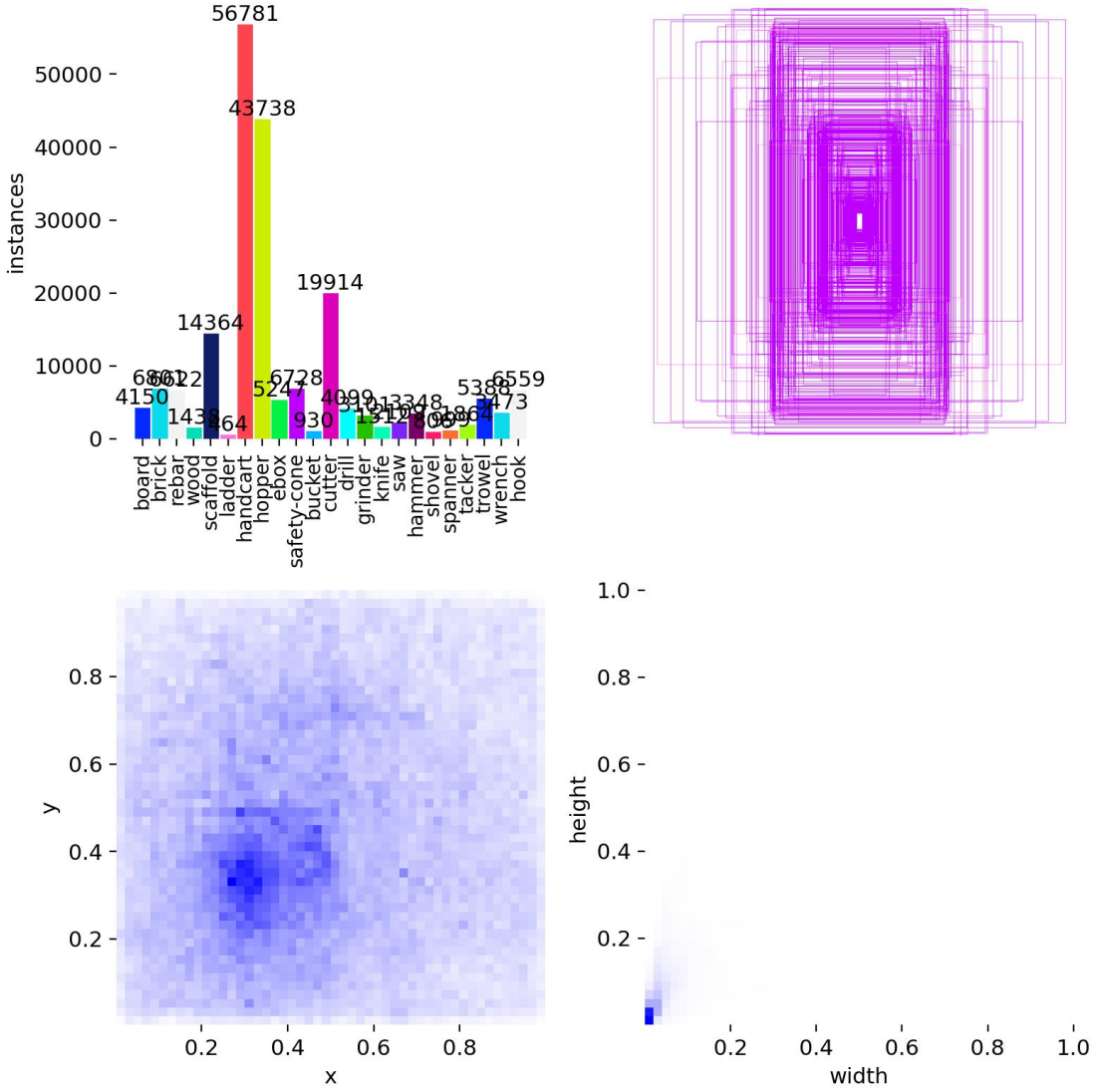



Figura 4.4 – Distribuição do *dataset* SGIE-Ferramentas com 23 classes. O gráfico evidencia o desbalanceamento entre as classes, com variação significativa na quantidade de instâncias por categoria.
Fonte: Elaborado pelo autor com dados do treinamento.

### 4.3.1 Configuração do Ambiente

O treinamento foi realizado em uma estação de trabalho com:

- GPU: NVIDIA GeForce RTX 4070 (12 GB GDDR6X);
- CPU: AMD Ryzen 5 5900XT (8 *cores*, 16 *threads*);
- Memória RAM: 64 GB DDR4-3200;
- Armazenamento: SSD NVMe 1 TB;
- Sistema operacional: Ubuntu 20.04;
- *Framework*: PyTorch 2.0, Ultralytics YOLOv8 v8.0 (Ultralytics, 2024).

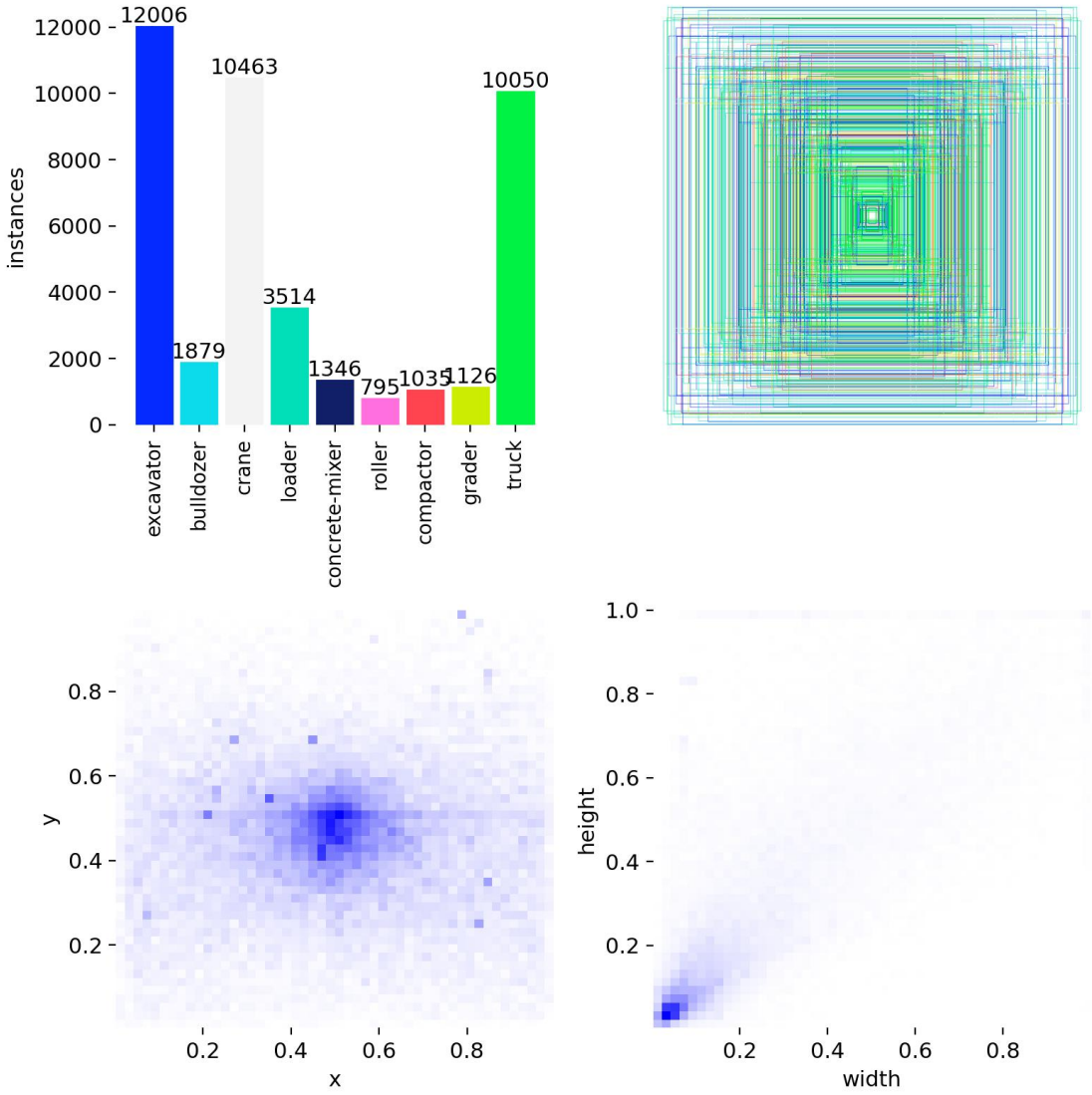



Figura 4.5 – Distribuição do *dataset* SGIE-Maquinário com 9 classes de maquinário pesado. A distribuição entre as classes é mais balanceada em comparação ao *dataset* de ferramentas.
Fonte: Elaborado pelo autor com dados do treinamento.

### 4.3.2 Hiperparâmetros

A Tabela 4.1 apresenta os hiperparâmetros utilizados no treinamento dos modelos. Os valores foram selecionados com base nas configurações padrão do YOLOv8 (Jocher; Chaurasia; Qiu, 2023) e ajustados para o domínio de construção civil.

A resolução de entrada dos SGIEs ($320 \times 320$) é menor que a do PGIE porque os SGIEs operam sobre *crops*, que já contêm apenas a região de interesse. Esta redução melhora a velocidade de inferência sem comprometer a acurácia, seguindo práticas de otimização para dispositivos de borda (Mittal, 2019).

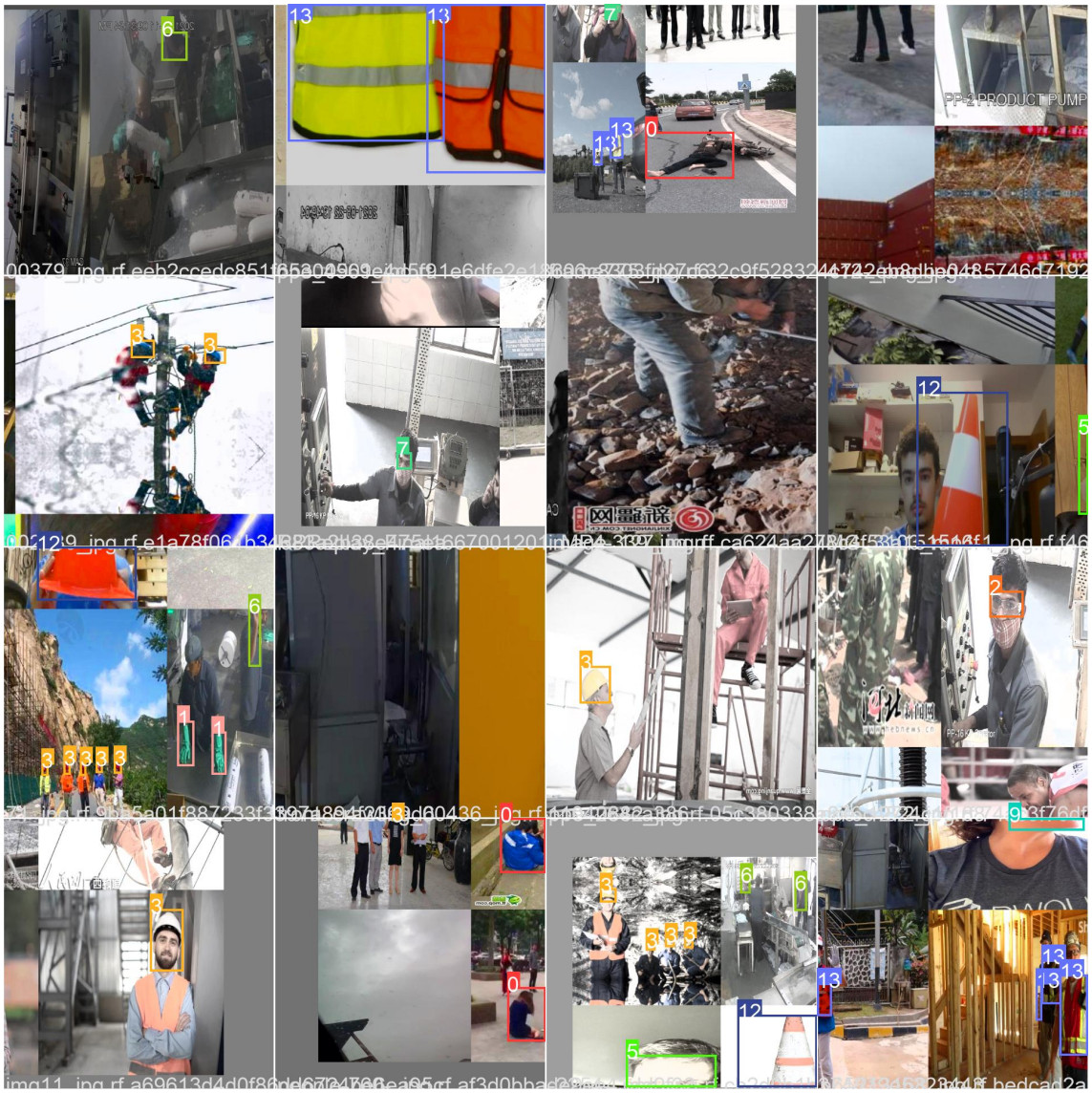


Figura 4.6 – Amostra de imagens do *batch* de treinamento com anotações de *ground truth*, demonstrando a diversidade do *dataset*.
Fonte: Elaborado pelo autor com dados do treinamento.

Tabela 4.1 – Hiperparâmetros de treinamento.

| **Parâmetro** | **PGIE** | **SGIEs** |
|---|---|---|
| Modelo base | YOLOv8s | YOLOv8s |
| Resolução de entrada | $640 \times 640$ | $320 \times 320$ |
| *Batch size* | 16 | 32 |
| Épocas máximas | 300 | 200 |
| *Learning rate* inicial | 0,01 | 0,01 |
| Otimizador | SGD | SGD |
| *Momentum* | 0,937 | 0,937 |
| *Weight decay* | 0,0005 | 0,0005 |
| *Early stopping* | 50 épocas | 30 épocas |

Fonte: Elaborado pelo autor.

### 4.3.3 Função de Perda

O YOLOv8 utiliza uma função de perda composta por três componentes, otimizada para detecção *anchor-free* (Terven; Cordova-Esparza, 2023):

$$\mathcal{L} = \lambda_{box}\mathcal{L}_{box} + \lambda_{dfl}\mathcal{L}_{dfl} + \lambda_{cls}\mathcal{L}_{cls} \tag{4.1}$$

Onde $\mathcal{L}_{box}$ é a perda de localização baseada em CIoU (Zheng et al., 2020), $\mathcal{L}_{dfl}$ é a *Distribution Focal Loss* para refinamento da distribuição de coordenadas das caixas delimitadoras, e $\mathcal{L}_{cls}$ é a perda de classificação baseada em *Binary Cross-Entropy*. Diferentemente de versões anteriores como o YOLOv5, o YOLOv8 elimina o ramo de objetividade (*objectness*), delegando a confiança da detecção inteiramente ao ramo de classificação.

## 4.4 Conversão e Otimização

Após o treinamento, os modelos foram convertidos e otimizados para execução no Jetson AGX Xavier (NVIDIA Corporation, 2024d). O processo de otimização é determinante para alcançar desempenho em tempo real em dispositivos de borda (Gholami et al., 2022).

### 4.4.1 Exportação para ONNX

Os modelos treinados no formato PyTorch (.pt) foram exportados para o formato ONNX (*Open Neural Network Exchange*) (ONNX Community, 2019). O ONNX funciona como formato intermediário padronizado que permite a conversão entre diferentes *frameworks* de *deep learning*. A exportação foi realizada com a ferramenta de exportação do Ultralytics, que gera o grafo computacional completo do modelo, incluindo as operações de pré-processamento e pós-processamento. Durante a exportação, os parâmetros de *batch size* e resolução de entrada foram definidos de forma estática para permitir otimizações mais agressivas na etapa seguinte de conversão para TensorRT.

### 4.4.2 Otimização com TensorRT

A conversão dos modelos ONNX para *engines* TensorRT (NVIDIA Corporation, 2024e) foi realizada diretamente no dispositivo Jetson AGX Xavier. Esta abordagem garante que o TensorRT selecione os *kernels* CUDA otimizados para a arquitetura Volta específica do dispositivo, uma vez que os *engines* gerados são vinculados ao *hardware* em que foram compilados. O processo de otimização incluiu fusão de camadas (Conv-BatchNorm-ReLU em operações únicas), seleção automática de *kernels* e alocação otimizada de memória (Jeong; Kim; Ha, 2022). Todos os modelos foram quantizados para precisão FP16, que utiliza os Tensor Cores da GPU Volta para dobrar o *throughput* de inferência com degradação de acurácia inferior a 1% (Gholami et al., 2022). A precisão INT8 não foi adotada pois requer um processo de calibração com *dataset* representativo e apresenta risco maior de degradação para modelos com muitas classes.

## 4.5 Integração com DeepStream

O *pipeline* de inferência em cascata foi implementado utilizando o NVIDIA DeepStream SDK (NVIDIA Corporation, 2024c). O DeepStream é construído sobre o GStreamer (Taymans et al., 2013), um *framework* de processamento multimídia baseado no paradigma de *pipeline*. Cada elemento do *pipeline* realiza uma transformação específica sobre os dados, e a comunicação entre elementos ocorre através de *buffers* de memória GPU, minimizando cópias entre CPU e GPU. O DeepStream gerencia automaticamente o fluxo de dados entre o PGIE e os SGIEs, recortando as regiões de interesse detectadas pelo modelo primário e encaminhando-as aos modelos secundários conforme a configuração de classes.

### 4.5.1 Configuração do *Pipeline*

O *pipeline* consiste nos seguintes elementos (NVIDIA Corporation, 2024a):

1. **`uridecodebin`**: Decodificação do *stream* de vídeo RTSP;
2. **`nvstreammux`**: Multiplexação e *batching* de *frames*;
3. **`nvinfer` (PGIE)**: Inferência primária com o modelo de 5 classes;

4. **`nvinfer` (SGIE-EPI)**: Inferência secundária para EPIs nos *crops* de Person;
5. **`nvinfer` (SGIE-Maquinário)**: Inferência secundária para maquinário nos *crops* de Vehicle;
6. **`nvinfer` (SGIE-Ferramentas)**: Inferência secundária para ferramentas nos *crops* de Tool;
7. **`nvtracker`**: Rastreamento (*tracking*) de objetos com atribuição de IDs persistentes (NVIDIA Corporation, 2023);
8. **`nvdsosd`**: Renderização visual das detecções;
9. **`nvv4l2h264enc`**: Codificação do vídeo de saída.

### 4.5.2 Configuração dos SGIEs

Cada SGIE foi configurado para operar apenas sobre detecções específicas do PGIE através dos parâmetros `operate-on-gie-id` e `operate-on-class-ids` (NVIDIA Corporation, 2024a):

- SGIE-EPI: `operate-on-class-ids=0` (Person);
- SGIE-Maquinário: `operate-on-class-ids=1` (Vehicle);
- SGIE-Ferramentas: `operate-on-class-ids=4` (Tool).

Esta configuração garante que cada SGIE processe apenas as regiões relevantes, otimizando o uso de recursos computacionais.

### 4.5.3 *Parser* Customizado

A integração de modelos YOLO com o DeepStream requer o desenvolvimento de funções de *parsing* customizadas (NVIDIA Corporation, 2024a). O *parser* é responsável por:

1. Interpretar o tensor de saída do modelo;
2. Decodificar as predições de caixas delimitadoras;
3. Extrair probabilidades de classe e *scores* de confiança;
4. Aplicar limiar de confiança;
5. Executar *Non-Maximum Suppression* (NMS);

6. Converter as detecções para estruturas `NvDsInferObjectDetectionInfo`.

O desenvolvimento do *parser* customizado é detalhado no Capítulo 5, incluindo a adaptação para o formato de saída específico do YOLOv8.

# 5 IMPLEMENTAÇÃO

Este capítulo detalha os aspectos técnicos da implementação do sistema, incluindo a configuração do ambiente, a estrutura do código e as otimizações realizadas para garantir *performance* adequada no dispositivo de borda (NVIDIA Corporation, 2024d).

A Figura 5.1 apresenta a arquitetura de alto nível do sistema, evidenciando os principais componentes e suas interações.

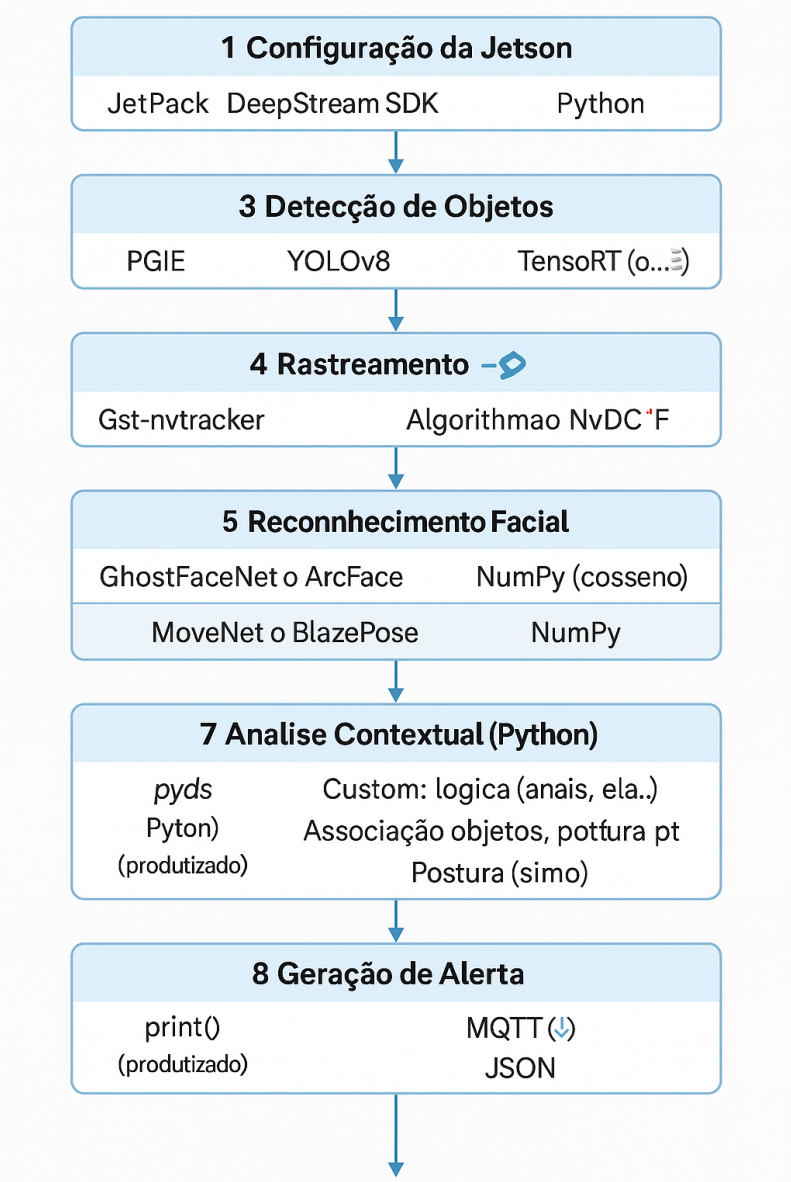



Figura 5.1 – Arquitetura de alto nível do sistema. O fluxo de processamento inicia com a configuração do ambiente, seguido pela leitura do vídeo, detecção de objetos (PGIE + SGIEs), rastreamento, análise de conformidade de EPIs e geração de alertas.
Fonte: Elaborado pelo autor.

A Figura 5.2 ilustra o *pipeline* GStreamer/DeepStream implementado (NVIDIA Corporation, 2024c), mostrando o fluxo de dados desde a captura de vídeo até a saída com

anotações.

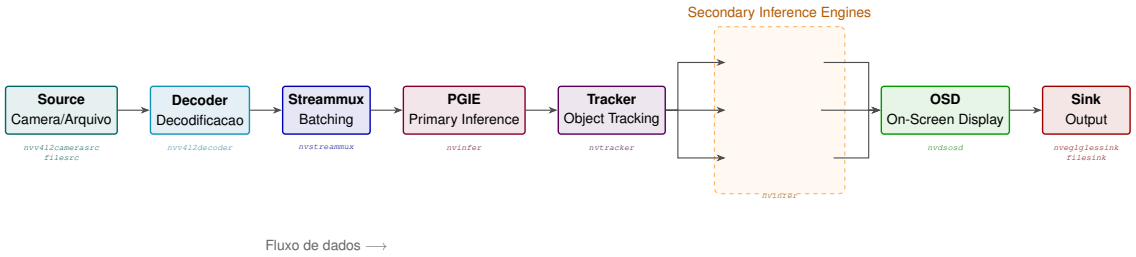



Figura 5.2 – *Pipeline* GStreamer/DeepStream implementado. O fluxo inicia com a captura de vídeo (Source), decodificação por *hardware* (Decoder), *batching* (Streammux), inferência primária (PGIE), *tracking* (Tracker), inferências secundárias paralelas (SGIEs), renderização de anotações (OSD) e saída para *display* ou arquivo (Sink).
Fonte: Elaborado pelo autor.

## 5.1 Ambiente de Desenvolvimento

### 5.1.1 *Hardware*

O desenvolvimento utilizou dois ambientes de *hardware* distintos para treinamento e inferência:

**Estação de treinamento:** As especificações da estação de treinamento são apresentadas na Seção 4.3.1.

**Dispositivo de borda:** NVIDIA Jetson AGX Xavier Developer Kit, cujas especificações são detalhadas na Seção 2.3.3, configurado no modo de potência MAXN (30 W).

### 5.1.2 *Software*

O ambiente de *software* no Jetson foi configurado com a *stack* NVIDIA otimizada para inferência em borda (Mittal, 2019):

- Sistema operacional: NVIDIA JetPack 5.1.2 (Ubuntu 20.04);
- CUDA Toolkit: 11.4.315;
- cuDNN: 8.6.0;
- TensorRT: 8.5.2 (NVIDIA Corporation, 2024e);
- DeepStream SDK: 6.2 (NVIDIA Corporation, 2024c);
- GStreamer: 1.16.3 (Taymans et al., 2013);

- Python: 3.8.10 com *bindings* PyDS.

## 5.2 Estrutura do Projeto

A Figura 5.3 apresenta uma visão detalhada do *pipeline* de processamento, incluindo os estágios de inferência e os fluxos de metadados.

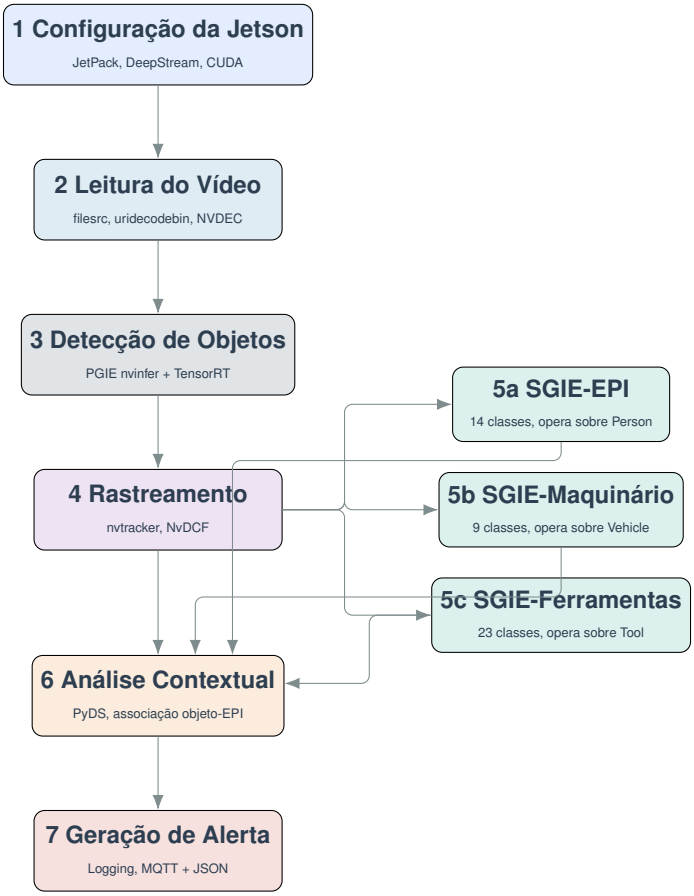



Figura 5.3 – *Pipeline* detalhado do sistema. O diagrama apresenta o fluxo de processamento desde a configuração do ambiente até a geração de alertas, incluindo a detecção primária (PGIE), os três SGIEs especializados (EPI, Maquinário e Ferramentas), o rastreamento de objetos e a análise de conformidade.
Fonte: Elaborado pelo autor.

A Tabela 5.1 apresenta a organização dos diretórios do projeto.

Tabela 5.1 – Estrutura de diretórios do projeto.

| Diretório | Descrição |
|---|---|
| `configs/` | Arquivos de configuração do *pipeline* DeepStream (PGIE, SGIEs e *tracker*). |
| `models/` | *Engines* TensorRT (`*.engine`) e arquivos de rótulos em `labels/`. |
| `src/` | Código-fonte do sistema (orquestração do *pipeline*, análise e utilitários). |
| `scripts/` | Scripts de suporte para conversão de modelos e execução do *pipeline*. |

Fonte: Elaborado pelo autor.

## 5.3 Configuração do *Pipeline*

### 5.3.1 Configuração do PGIE

O arquivo de configuração do PGIE especifica os parâmetros do modelo primário (NVIDIA Corporation, 2024a). A Tabela 5.2 apresenta os parâmetros principais do PGIE.

Tabela 5.2 – Parâmetros principais do arquivo de configuração do PGIE.

| **Parâmetro** | **Valor** | **Descrição** |
|---|---|---|
| `gpu-id` | `0` | GPU utilizada pelo GIE. |
| `net-scale-factor` | `0.0039215697906911373` | Fator de normalização (1/255). |
| `model-engine-file` | `models/pgie.engine` | *Engine* TensorRT do modelo primário (PGIE). |
| `labelfile-path` | `models/labels/pgie` $_{labels.txt}$ | Arquivo de rótulos das classes do PGIE. |
| `batch-size` | `1` | Tamanho do lote de inferência. |
| `network-mode` | `2` | Precisão FP16. |
| `num-detected-classes` | `5` | Número de classes detectadas pelo modelo. |
| `interval` | `0` | Executa inferência em todos os *frames* (sem pular). |
| `gie-unique-id` | `1` | Identificador do elemento GIE no *pipeline*. |
| `process-mode` | `1` | Modo primário (PGIE). |
| `cluster-mode` | `2` | NMS para pós-processamento. |
| `parse-bbox-func-name` | `NvDsInferParseYolo` | Função de *parsing* das bboxes (YOLO). |
| `custom-lib-path` | `libs/libnvdsinfer` $_{custom_{yolo.so}}$ | Biblioteca customizada com *parser* do modelo. |

Fonte: Elaborado pelo autor.

### 5.3.2 Configuração dos SGIEs

Os SGIEs foram configurados para operar sobre classes específicas do PGIE:

A Tabela 5.3 apresenta os parâmetros do SGIE-EPI.

Tabela 5.3 – Parâmetros principais do arquivo de configuração do SGIE-EPI.

| **Parâmetro** | **Valor** | **Descrição** |
|---|---|---|
| gpu-id | 0 | GPU utilizada pelo GIE. |
| net-scale-factor | 0.0039215697906911373 | Fator de normalização (1/255). |
| model-engine-file | models/sgie$_{epi.engine}$ | *Engine* TensorRT do SGIE de EPI. |
| labelfile-path | models/labels/sgie$_{epi_{labels.txt}}$ | Arquivo de rótulos das classes do SGIE-EPI. |
| batch-size | 16 | Lote de inferência do SGIE. |
| network-mode | 2 | Precisão FP16. |
| num-detected-classes | 14 | Número de classes previstas pelo SGIE-EPI. |
| gie-unique-id | 2 | Identificador do SGIE no *pipeline*. |
| operate-on-gie-id | 1 | Indica que o SGIE opera sobre as saídas do PGIE (ID 1). |
| operate-on-class-ids | 0 | Aplica o SGIE somente em objetos da classe 0 (Pessoa). |
| process-mode | 2 | Modo secundário (SGIE). |

Fonte: Elaborado pelo autor.

### 5.3.3 Configuração do *Tracker*

O *tracker* foi configurado para manter identidade de objetos entre *frames* (NVIDIA Corporation, 2024b; NVIDIA Corporation, 2023). A Tabela 5.4 apresenta os parâmetros do *tracker*.

## 5.4 Implementação do *Parser*

O *parser* customizado para YOLO foi implementado em C++ e compilado como biblioteca compartilhada (NVIDIA Corporation, 2024a). As principais funções incluem:

1. **Decodificação de saída**: O tensor de saída do YOLOv8 utiliza formato *anchor-free*

Tabela 5.4 – Parâmetros principais do arquivo de configuração do *tracker*.

| **Parâmetro** | **Valor** | **Descrição** |
|---|---|---|
| `tracker-width` | `640` | Largura da imagem usada pelo *tracker*. |
| `tracker-height` | `480` | Altura da imagem usada pelo *tracker*. |
| `ll-lib-file` | `/opt/nvidia/deepstream/deepstream/lib/libnvds`$_{nvmultiobjecttracker.so}$ | Biblioteca do *tracker* multiobjeto do DeepStream. |
| `ll-config-file` | `tracker`$_{config.yml}$ | Arquivo YAML com parâmetros do *tracker*. |
| `enable-batch-process` | `1` | Habilita processamento em lote no *tracker*. |

Fonte: Elaborado pelo autor.

com dimensões [1, 4+num_classes, num_predictions] (Terven; Cordova-Esparza, 2023), onde cada coluna representa uma predição contendo as quatro coordenadas da caixa delimitadora seguidas das probabilidades de classe. O *parser* transpõe e processa as predições diretamente sem necessidade de decodificação de âncoras;

2. **Filtragem por confiança**: Predições com *score* de classificação abaixo do limiar de confiança são descartadas;
3. ***Non-Maximum Suppression***: Detecções redundantes são eliminadas com base no IoU (Padilla; Netto; Silva, 2020).

## 5.5 Aplicação Principal

A aplicação principal foi implementada em Python utilizando os *bindings* PyDS do DeepStream (NVIDIA Corporation, 2024c). O fluxo de execução consiste em:

1. Inicialização do GStreamer e criação do *pipeline*;
2. Configuração de elementos (PGIE, SGIEs, *tracker*);
3. Registro de *callbacks* para interceptação de metadados;
4. Início do *loop* principal;
5. Processamento de eventos e metadados nos *callbacks*.

## 5.6 Sistema de Análise

O módulo de análise implementa a lógica de verificação de conformidade de EPIs, seguindo abordagens estabelecidas na literatura (Mneymneh; Abbas; Khoury, 2019):

1. Para cada *frame*, identifica detecções de Person do PGIE;
2. Para cada pessoa, verifica as detecções do SGIE-EPI associadas;
3. Verifica presença de Hardhat e Safety Vest;
4. Mantém histórico temporal por ID de *tracking*;
5. Gera alerta de violação se ausência de EPI persistir por N *frames*;
6. Registra eventos com *timestamp*, ID do trabalhador e tipo de violação.

A votação temporal reduz falsos positivos causados por oclusões momentâneas ou erros de detecção esporádicos.

## 5.7 Otimizações de Performance

Para garantir operação em tempo real, foram implementadas otimizações seguindo práticas estabelecidas para *Edge AI* (Murshed et al., 2022; Mittal, 2019):

- **Memória unificada**: Aproveitamento da arquitetura CPU-GPU compartilhada do Jetson (NVIDIA Corporation, 2024d);
- **Decodificação por *hardware***: Utilização do NVDEC para decodificação de vídeo;
- ***Batching* de *crops***: Acúmulo de *crops* para processamento em *batch* pelos SGIEs;
- **Precisão FP16**: Quantização de todos os modelos para *half-precision* (Gholami et al., 2022);
- ***Pipeline* assíncrono**: Configuração do GStreamer para operação assíncrona (Taymans et al., 2013).

No capítulo seguinte, apresentam-se os resultados experimentais obtidos com o sistema implementado, incluindo a avaliação de acurácia dos modelos e a análise de *performance* do *pipeline* no dispositivo de borda.

# 6 RESULTADOS E DISCUSSÃO

Este capítulo apresenta os resultados experimentais obtidos durante o desenvolvimento e validação do sistema, discutindo suas implicações práticas e comparando com trabalhos da literatura.

## 6.1 Avaliação dos Modelos

### 6.1.1 Métricas de Avaliação

A avaliação utilizou as métricas padronizadas do *benchmark* MS COCO (Lin et al., 2014; Padilla; Netto; Silva, 2020):

- **Precisão**: Fração de detecções corretas entre todas as detecções;
- **Revocação**: Fração de objetos corretamente detectados entre todos os objetos presentes;
- **mAP@0,5**: Média da *Average Precision* com IoU *threshold* de 0,5;
- **mAP@0,5:0,95**: Média da AP sobre *thresholds* de IoU de 0,5 a 0,95.

A métrica primária utilizada para avaliação e comparação neste trabalho é a mAP@0,5, por ser a mais reportada na literatura da área (Padilla; Netto; Silva, 2020).

### 6.1.2 Resultados do PGIE

O modelo PGIE foi treinado por 187 épocas no total. O melhor resultado foi obtido na época 137, após a qual o critério de *early stopping* (paciência de 50 épocas sem melhoria) encerrou o treinamento automaticamente (Jocher; Chaurasia; Qiu, 2023). A Tabela 6.1 apresenta as métricas obtidas no conjunto de validação.

A classe Vehicle apresentou o melhor desempenho (AP=0,900) devido às características visuais distintivas. A classe Safety-vest apresentou o menor desempenho (AP=0,679) devido ao menor número de instâncias no *dataset* e maior variabilidade visual.

A Figura 6.1 apresenta a evolução das métricas durante o treinamento do PGIE. As curvas de perda de treinamento e validação apresentam convergência estável, sem

Tabela 6.1 – Métricas do modelo PGIE.

| **Classe** | **AP@0,5** | **Instâncias** |
|---|---|---|
| Person | 0,865 | 118.000 |
| Vehicle | 0,900 | 49.000 |
| Hardhat | 0,881 | 38.000 |
| Safety-vest | 0,679 | 6.500 |
| Tool | 0,826 | 175.000 |
| **mAP@0,5** | **0,830** | - |

Fonte: Elaborado pelo autor com dados do treinamento.

sinais de *overfitting*, uma vez que as perdas de validação acompanham as de treinamento ao longo das épocas. A métrica mAP@0,5 estabiliza nas últimas épocas, confirmando a adequação do critério de *early stopping* adotado.

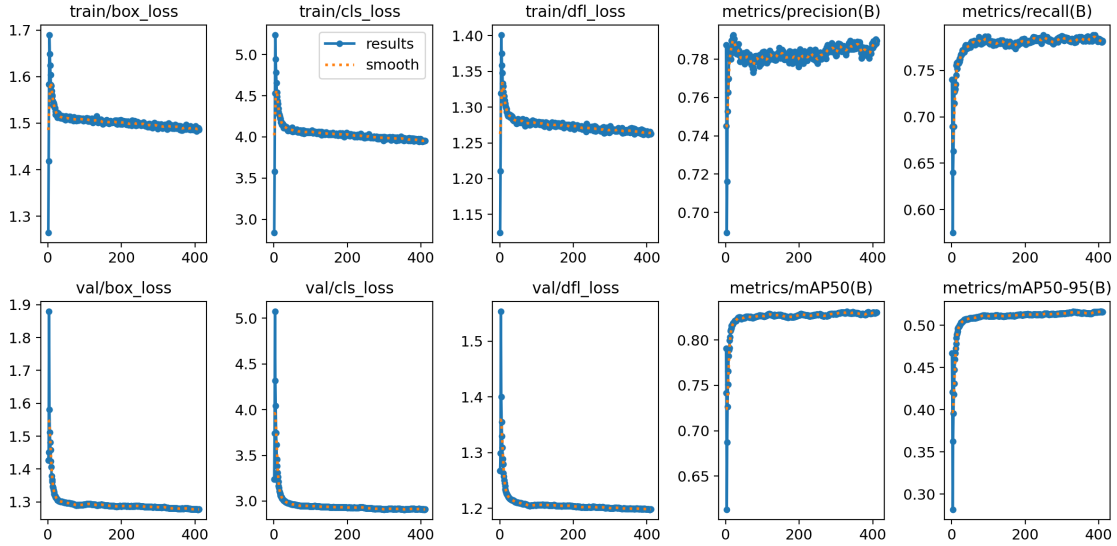



Figura 6.1 – Curvas de treinamento do modelo PGIE, apresentando a evolução das funções de perda e métricas mAP ao longo das épocas.

Fonte: Elaborado pelo autor com dados do treinamento.

A Figura 6.2 apresenta a matriz de confusão normalizada do PGIE. A diagonal principal indica que as cinco classes são distinguidas com boa separabilidade. A classe Tool apresenta a maior taxa de confusão com o *background* (0,47), o que indica que parte das ferramentas não é detectada pelo modelo. A classe Safety-vest apresenta confusão moderada com o *background* (0,31), consistente com o menor AP observado na Tabela 6.1.

A Figura 6.3 apresenta as curvas *Precision-Recall* por classe do PGIE. As classes Vehicle e Hardhat mantêm precisão elevada mesmo em níveis altos de revocação. A classe Safety-vest apresenta queda mais acentuada de precisão conforme a revocação aumenta, refletindo o desafio de detectar esta classe com menor representação no *dataset*.

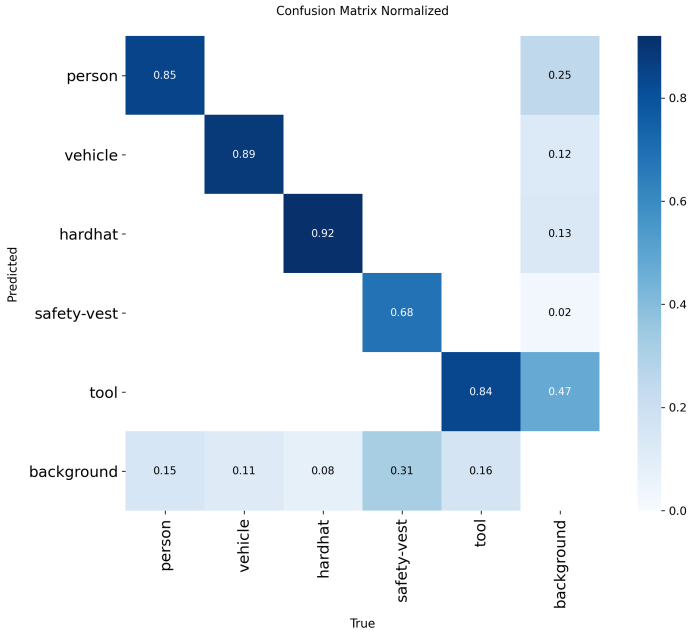



Figura 6.2 – Matriz de confusão normalizada do modelo PGIE para as 5 classes.
Fonte: Elaborado pelo autor com dados do treinamento.

### 6.1.3 Resultados do SGIE-EPI

O SGIE-EPI foi treinado por 156 épocas. A Tabela 6.2 apresenta as métricas por classe.

Tabela 6.2 – Métricas do modelo SGIE-EPI (14 classes).

| **Classe** | **AP@0,5** | **Instâncias** | **Classe** | **AP@0,5** | **Instâncias** |
|---|---|---|---|---|---|
| Goggles | 0,971 | 2.840 | NO-Goggles | 0,939 | 2.510 |
| Gloves | 0,948 | 3.120 | NO-Gloves | 0,908 | 2.780 |
| Ladder | 0,946 | 1.650 | Person | 0,931 | 8.420 |
| Hardhat | 0,896 | 6.350 | NO-Hardhat | 0,724 | 3.210 |
| Fall-Detected | 0,859 | 1.280 | Safety Vest | 0,682 | 4.870 |
| Safety Cone | 0,608 | 980 | NO-Mask | 0,611 | 1.540 |
| Mask | 0,524 | 1.120 | NO-Safety Vest | 0,236 | 890 |
| **mAP@0,5 = 0,770** | | | | | |

Fonte: Elaborado pelo autor com dados do treinamento.

O modelo SGIE-EPI detecta 14 classes relacionadas a EPIs e situações de risco. Classes como Goggles (0,971), Gloves (0,948) e Ladder (0,946) apresentaram excelente desempenho. A classe NO-Safety Vest apresentou o menor desempenho (AP=0,236), resultado atribuível a múltiplos fatores: o baixo número de instâncias de treinamento (890, o segundo menor do *dataset*), a ambiguidade visual entre presença parcial e ausência total de colete, e a confusão entre esta classe e Safety Vest em cenários com oclusão parcial

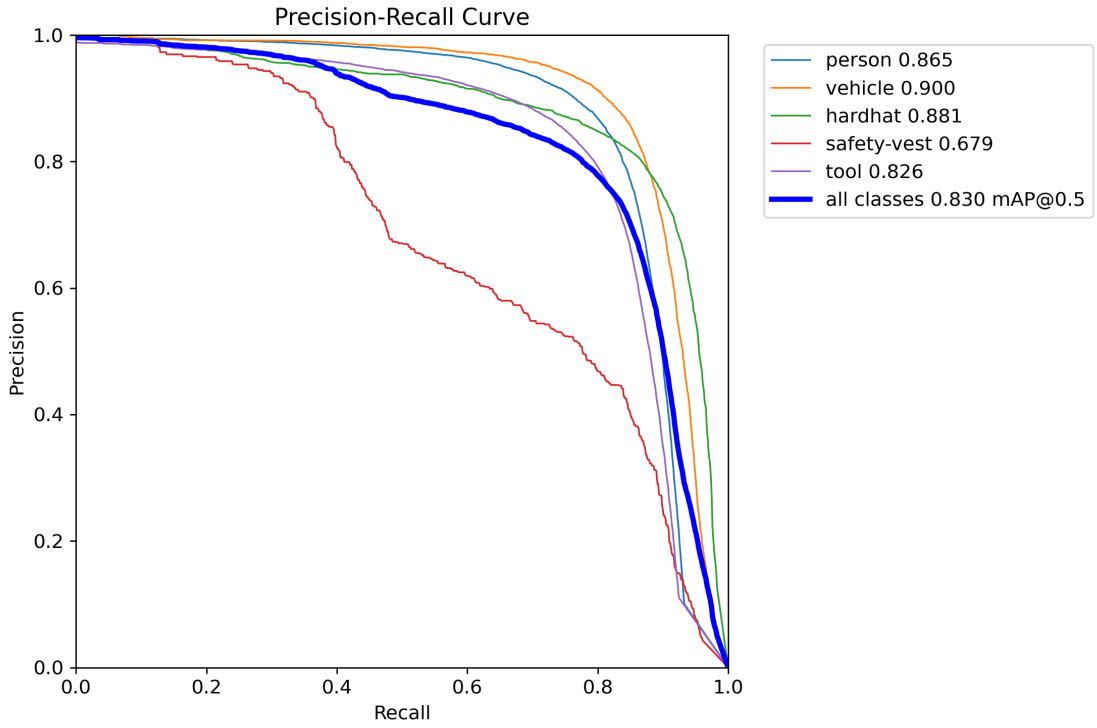



Figura 6.3 – Curvas *Precision-Recall* do modelo PGIE, com mAP@0,5 indicado na legenda.
Fonte: Elaborado pelo autor com dados do treinamento.

(Nath; Behzadan; Paal, 2020).

A Figura 6.4 apresenta o desempenho por classe do modelo SGIE-EPI, com as classes ordenadas pelo valor de AP@0,5. Observa-se que seis classes apresentam AP superior a 0,90, enquanto a classe NO-Safety Vest constitui o caso de desempenho mais reduzido.

A Figura 6.5 apresenta a evolução das métricas durante o treinamento do SGIE-EPI. As curvas de perda de treinamento e validação convergem de forma estável. A métrica mAP@0,5 apresenta crescimento rápido nas primeiras 30 épocas e estabiliza em torno de 0,77, indicando que o modelo atinge seu ponto de saturação com os dados disponíveis.

A Figura 6.6 apresenta a matriz de confusão normalizada do SGIE-EPI. A matriz revela que as classes Goggles, NO-Goggles e Gloves apresentam taxas de acerto superiores a 0,94. A classe Mask apresenta confusão significativa com NO-Mask (0,09) e com o *background* (0,07), o que é consistente com a similaridade visual entre máscara parcialmente visível e ausência de máscara. A classe NO-Safety Vest apresenta a maior dispersão de erros, com confusões distribuídas entre Safety Vest (0,08) e *background* (0,05).

A Figura 6.7 apresenta as curvas *Precision-Recall* por classe do SGIE-EPI. As classes com AP superior a 0,90 (Goggles, Gloves, Ladder, NO-Goggles, Person) mantêm

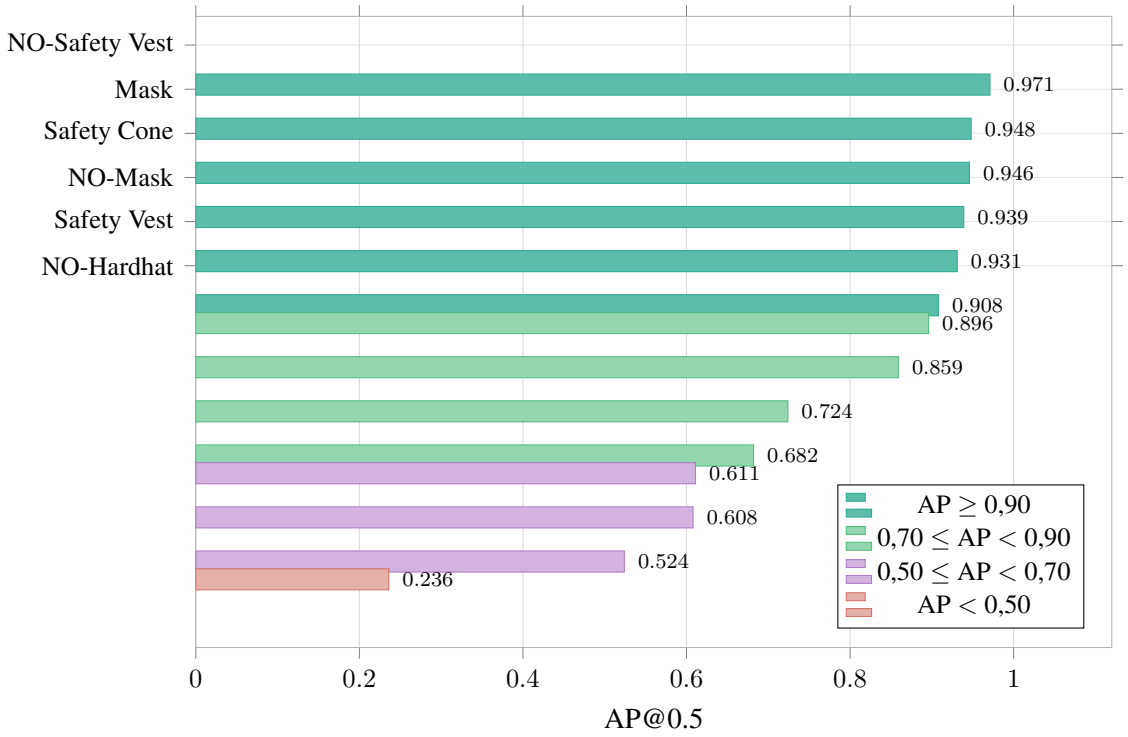



Figura 6.4 – Desempenho por classe do modelo SGIE-EPI (AP@0,5) para as 14 classes de EPIs e situações de risco. As barras são ordenadas do maior para o menor valor de AP@0,5.
Fonte: Elaborado pelo autor com dados do treinamento.

precisão elevada em toda a faixa de revocação. A classe NO-Safety Vest apresenta queda abrupta de precisão, com a curva PR próxima ao eixo inferior, refletindo o baixo AP de 0,236.

A Figura 6.8 apresenta as curvas *F1-Score* por classe do SGIE-EPI em função do limiar de confiança. O ponto ótimo de F1 para o conjunto de classes ocorre no limiar de 0,292, com F1 de 0,75. Classes como Goggles e Gloves atingem F1 superior a 0,90, enquanto NO-Safety Vest permanece abaixo de 0,30 em todos os limiares.

### 6.1.4 Resultados do SGIE-Maquinário

O SGIE-Maquinário foi treinado por 142 épocas. A Tabela 6.3 apresenta as métricas.

O SGIE-Maquinário apresentou o melhor desempenho entre os três SGIEs (mAP=0,84) devido ao tamanho visual grande dos objetos e características visuais distintivas de cada tipo de máquina. O modelo detecta 9 classes de maquinário pesado.

A Figura 6.9 apresenta a evolução das métricas durante o treinamento do SGIE-Maquinário. As curvas de perda de classificação apresentam convergência rápida nas pri-

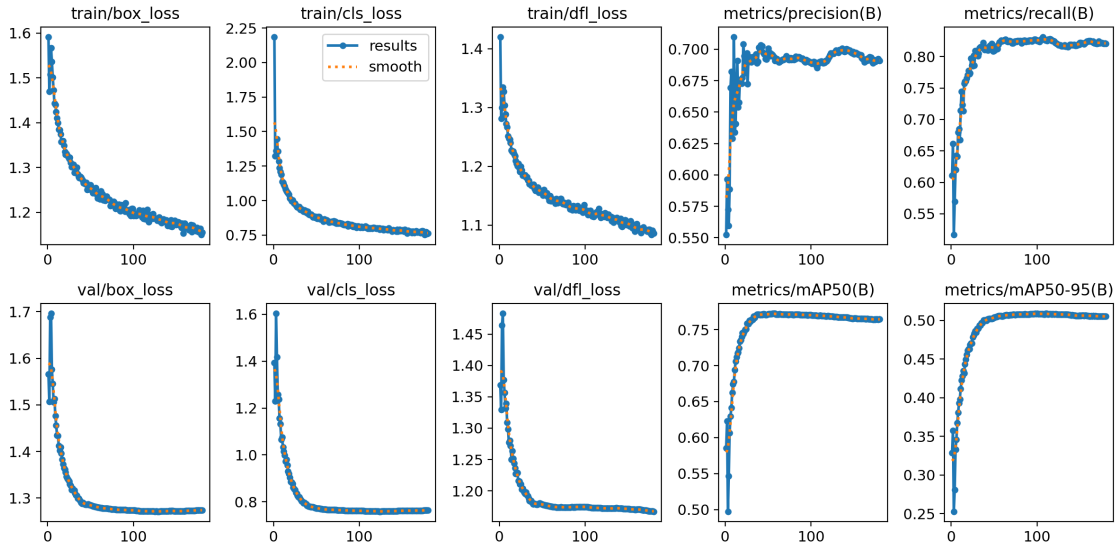



Figura 6.5 – Curvas de treinamento do modelo SGIE-EPI, apresentando a evolução das funções de perda e métricas mAP ao longo das épocas.
Fonte: Elaborado pelo autor com dados do treinamento.

Tabela 6.3 – Métricas do modelo SGIE-Maquinário.

| **Classe** | **AP@0,5** | **Instâncias** |
|---|---|---|
| Excavator | 0,910 | 1.240 |
| Bulldozer | 0,860 | 720 |
| Crane | 0,880 | 890 |
| Loader | 0,840 | 680 |
| Concrete Mixer | 0,820 | 540 |
| Roller | 0,810 | 480 |
| Compactor | 0,790 | 390 |
| Grader | 0,770 | 350 |
| Truck | 0,890 | 1.560 |
| **mAP@0,5** | **0,840** | - |

Fonte: Elaborado pelo autor com dados do treinamento.

meiras 30 épocas. A métrica mAP@0,5 estabiliza em torno de 0,84, com baixa variação nas últimas épocas de treinamento.

A Figura 6.10 apresenta a matriz de confusão normalizada do SGIE-Maquinário. A diagonal principal indica boa separabilidade entre as nove classes. A classe Crane apresenta a maior confusão com o *background* (0,38), possivelmente devido à variabilidade de escala dos guindastes nas imagens. A classe Truck apresenta confusão com o *background* (0,25), o que pode ser atribuído a sobreposições visuais com outros veículos. As classes Compactor e Grader apresentam as maiores taxas de acerto na diagonal (0,92 e 0,97, respectivamente).

A Figura 6.11 apresenta as curvas *Precision-Recall* por classe do SGIE-Maquinário.

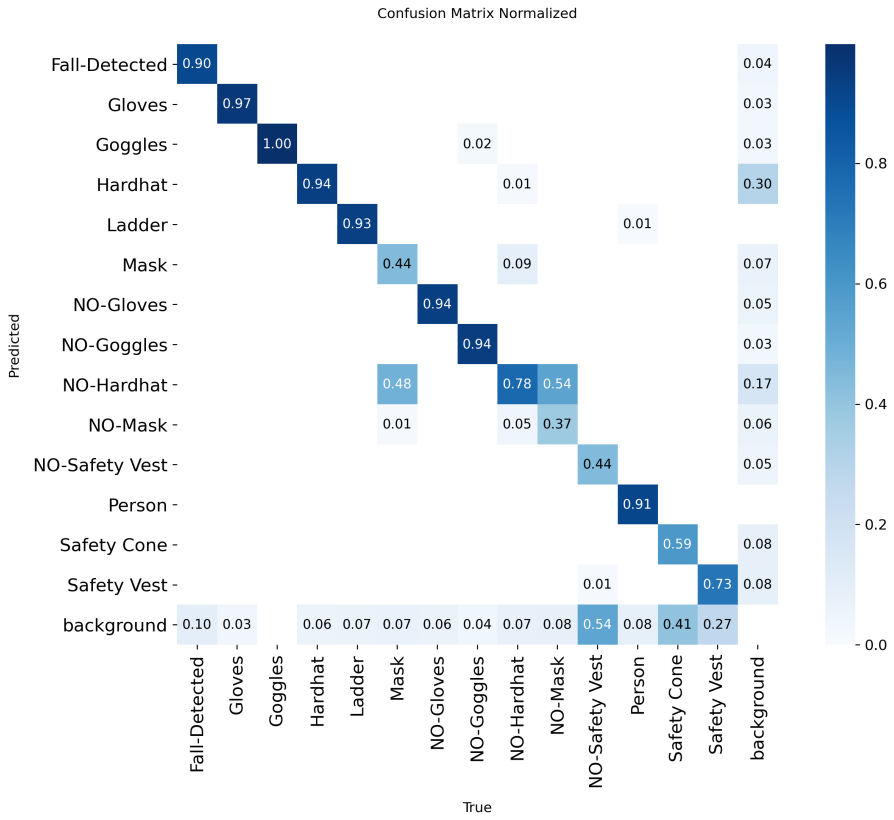



Figura 6.6 – Matriz de confusão normalizada do modelo SGIE-EPI para as 14 classes de EPIs e situações de risco.
Fonte: Elaborado pelo autor com dados do treinamento.

Todas as classes mantêm precisão acima de 0,80 para revocação de até 0,60, indicando que o modelo é confiável quando detecta um objeto. As classes Compactor e Grader apresentam as melhores curvas PR, consistentes com os valores de AP mais elevados.

A Figura 6.12 apresenta exemplos de predições do modelo no conjunto de validação. As imagens demonstram que o modelo identifica corretamente diferentes tipos de maquinário em cenários variados, incluindo situações com múltiplas máquinas na mesma cena e oclusão parcial entre equipamentos.

### 6.1.5 Resultados do SGIE-Ferramentas

O SGIE-Ferramentas foi treinado por 134 épocas e detecta 23 classes de ferramentas e equipamentos de construção. A Tabela 6.4 apresenta as métricas de todas as classes.

O SGIE-Ferramentas apresentou mAP de 0,71 considerando as 23 classes. O desempenho variou significativamente entre classes: objetos maiores como Safety Cone

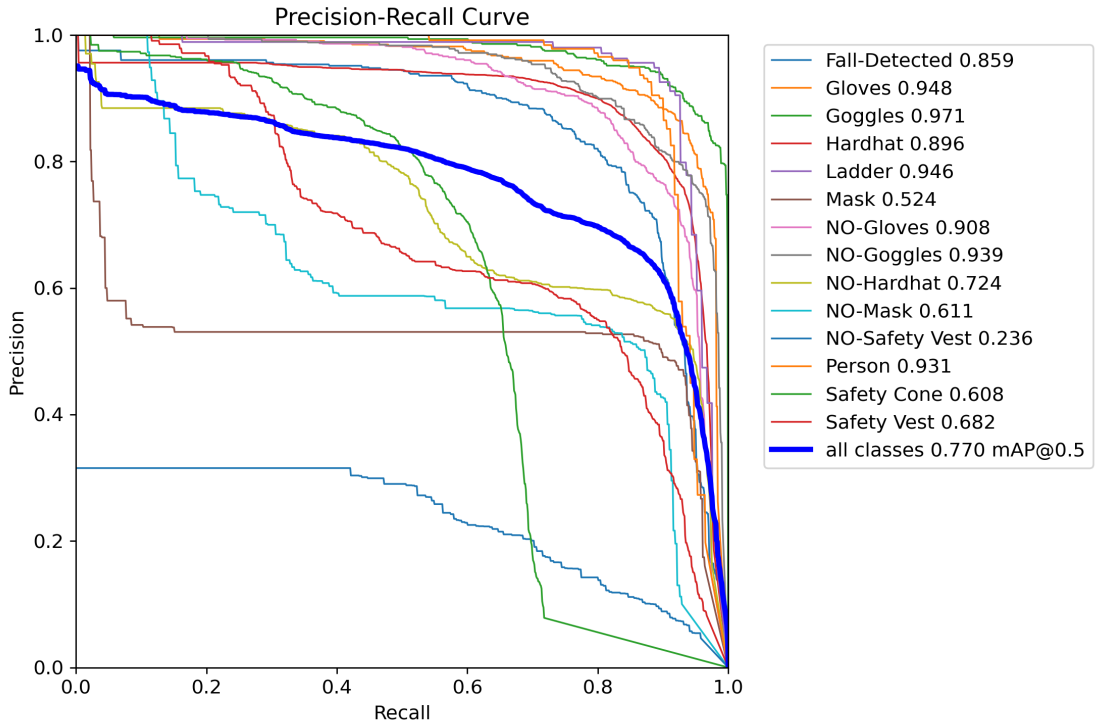



Figura 6.7 – Curvas *Precision-Recall* do modelo SGIE-EPI, com mAP@0,5 indicado na legenda.
Fonte: Elaborado pelo autor com dados do treinamento.

Tabela 6.4 – Métricas do modelo SGIE-Ferramentas (23 classes).

| **Classe** | **AP@0,5** | **Inst.** | **Classe** | **AP@0,5** | **Inst.** |
|---|---|---|---|---|---|
| Safety Cone | 0,860 | 1.120 | Hopper | 0,700 | 310 |
| Ladder | 0,820 | 890 | Hammer | 0,680 | 290 |
| Board | 0,800 | 760 | Spanner | 0,670 | 220 |
| Scaffold | 0,780 | 540 | Saw | 0,660 | 250 |
| Brick | 0,770 | 680 | Drill | 0,650 | 240 |
| Bucket | 0,760 | 420 | Grinder | 0,640 | 210 |
| Handcart | 0,740 | 380 | Knife | 0,620 | 190 |
| Rebar | 0,720 | 650 | Wrench | 0,610 | 180 |
| Ebox | 0,710 | 280 | Tacker | 0,580 | 160 |
| Wood | 0,710 | 520 | Trowel | 0,560 | 170 |
| Hook | 0,700 | 340 | Cutter | 0,540 | 150 |
| Shovel | 0,700 | 360 | | | |
| **mAP@0,5 (23 classes) = 0,710** | | | | | |

Fonte: Elaborado pelo autor com dados do treinamento.

(AP=0,86) e Ladder (AP=0,82) obtiveram AP superior a 0,80, enquanto ferramentas manuais de menor porte como Cutter (AP=0,54) e Trowel (AP=0,56) apresentaram AP inferior a 0,60. Esta variação é consistente com observações da literatura sobre detecção de objetos pequenos (Singh; Davis, 2018).

A Figura 6.13 apresenta a evolução das métricas durante o treinamento do SGIE-Ferramentas. As curvas de perda de classificação apresentam valores mais elevados em

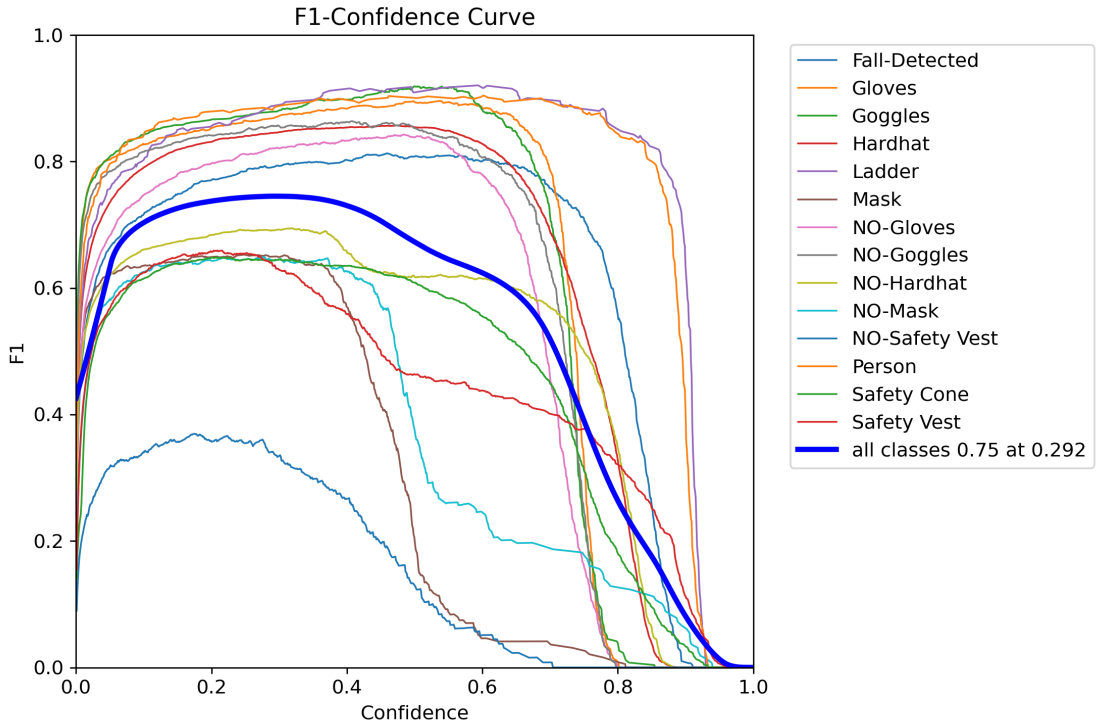



Figura 6.8 – Curvas *F1-Score* do modelo SGIE-EPI em função do limiar de confiança.
Fonte: Elaborado pelo autor com dados do treinamento.

comparação aos demais modelos, o que é esperado dado o maior número de classes (23). A métrica mAP@0,5 converge para 0,71, com estabilização a partir da época 200.

A Figura 6.14 apresenta a matriz de confusão normalizada do SGIE-Ferramentas. As classes Board, Rebar e Wood apresentam taxas de acerto elevadas na diagonal (acima de 0,86). Classes de ferramentas manuais como Grinder, Knife e Saw apresentam confusão entre si, o que é esperado pela similaridade visual destas categorias em imagens de canteiro. A classe Handcart apresenta confusão significativa com o *background* (0,21), indicando dificuldade na detecção deste objeto.

A Figura 6.15 apresenta as curvas *Precision-Recall* por classe do SGIE-Ferramentas. A dispersão entre as curvas é a maior entre os quatro modelos, refletindo a heterogeneidade das 23 classes. Classes de objetos maiores (Safety Cone, Ladder, Board) apresentam curvas PR com área elevada, enquanto ferramentas manuais de menor porte (Cutter, Trowel, Tacker) apresentam queda acentuada de precisão em revocação acima de 0,40.

A Figura 6.16 apresenta exemplos de predições do modelo no conjunto de validação. As imagens demonstram que o modelo detecta ferramentas em cenários com fundos variados e condições de iluminação distintas. Observa-se que ferramentas maiores como serras e esmerilhadeiras são detectadas com maior confiança, enquanto ferramentas menores apresentam *scores* de confiança mais baixos.

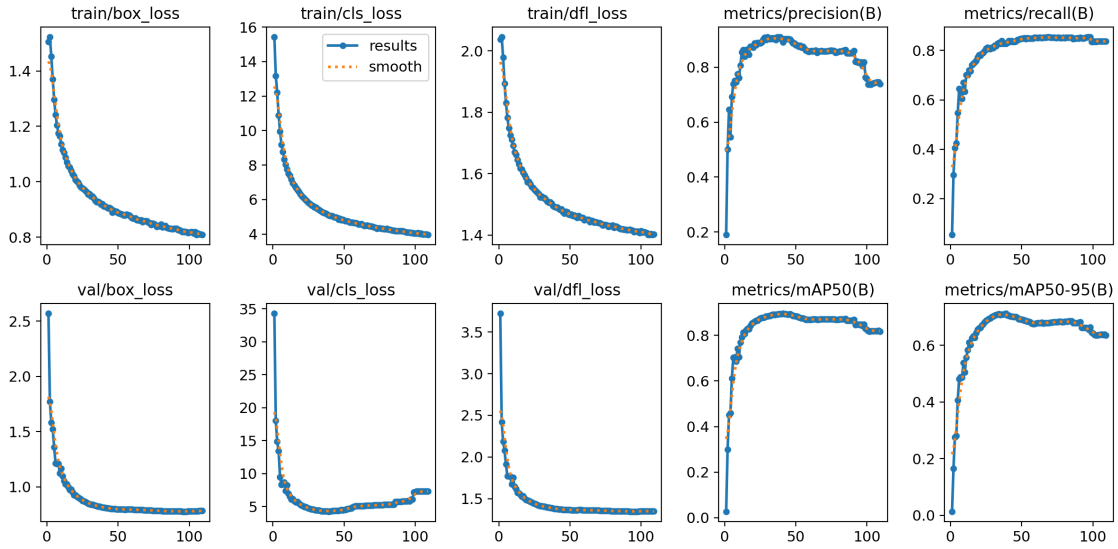



Figura 6.9 – Curvas de treinamento do modelo SGIE-Maquinário, apresentando a evolução das funções de perda e métricas mAP ao longo das épocas.
Fonte: Elaborado pelo autor com dados do treinamento.

## 6.2 *Performance* do *Pipeline*

A avaliação de *performance* do *pipeline* completo foi realizada no Jetson AGX Xavier em modo MAXN (30 W) (NVIDIA Corporation, 2024d).

### 6.2.1 *Throughput*

A Tabela 6.5 apresenta o *throughput* medido em diferentes configurações.

Tabela 6.5 – *Throughput* do *pipeline* no Jetson AGX Xavier.

| **Configuração** | **Precisão** | **FPS** | **Latência (ms)** |
|---|---|---|---|
| PGIE apenas | FP16 | 38 | 26 |
| PGIE + SGIE-EPI | FP16 | 28 | 36 |
| PGIE + 3 SGIEs | FP16 | 20 | 58 |
| PGIE + 3 SGIEs | FP32 | 12 | 92 |

Fonte: Elaborado pelo autor.

O *pipeline* completo com PGIE e três SGIEs alcançou 20 FPS em precisão FP16, valor adequado para aplicações de monitoramento (Murshed et al., 2022). A quantização FP16 proporciona *speedup* de 1,7× em relação a FP32 (Gholami et al., 2022).

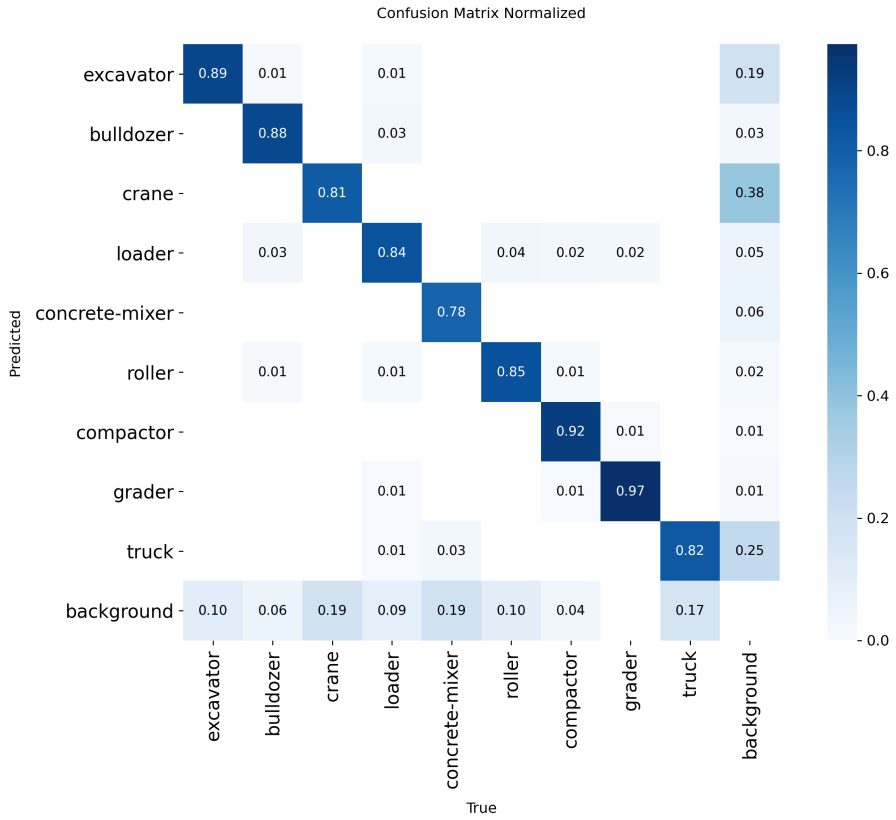



Figura 6.10 – Matriz de confusão normalizada do modelo SGIE-Maquinário para as 9 classes de maquinário.
Fonte: Elaborado pelo autor com dados do treinamento.

### 6.2.2 Latência por Componente

A Tabela 6.6 apresenta a latência de cada componente do *pipeline*.

Tabela 6.6 – Latência por componente do *pipeline* (FP16).

| **Componente** | **Latência (ms)** |
|---|---|
| Decodificação de vídeo (NVDEC) | 3,2 |
| Pré-processamento | 1,8 |
| PGIE (inferência) | 18,5 |
| SGIE-EPI (inferência) | 8,4 |
| SGIE-Maquinário (inferência) | 7,8 |
| SGIE-Ferramentas (inferência) | 7,2 |
| *Tracking* (`nvtracker`) | 4,8 |
| Pós-processamento e NMS | 3,1 |
| Renderização (`nvdsosd`) | 3,2 |
| **Total *end-to-end*** | **58,0** |

Fonte: Elaborado pelo autor.

A inferência responde por aproximadamente 72% da latência total (41,9 ms de

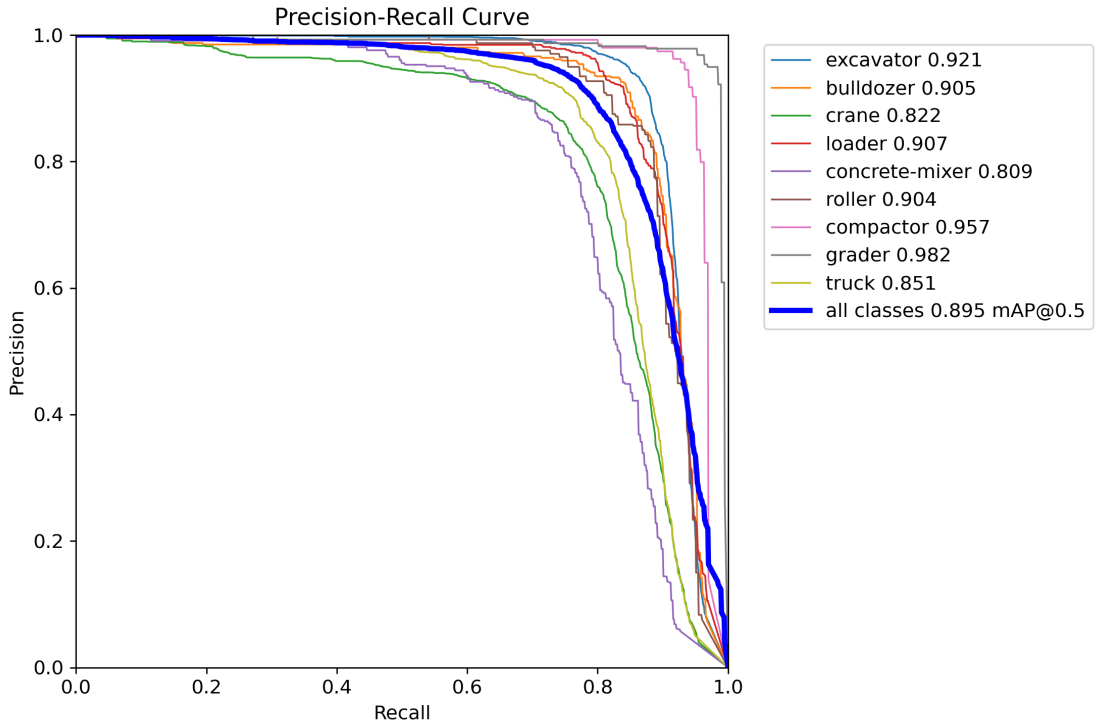



Figura 6.11 – Curvas *Precision-Recall* do modelo SGIE-Maquinário, com mAP@0,5 indicado na legenda.
Fonte: Elaborado pelo autor com dados do treinamento.

58,0 ms), com o PGIE sendo o componente mais custoso (18,5 ms) devido ao processamento do *frame* completo em resolução 640×640 (NVIDIA Corporation, 2024a).

### 6.2.3 Utilização de Recursos

A Tabela 6.7 apresenta a utilização de recursos do Jetson durante operação.

Tabela 6.7 – Utilização de recursos do Jetson AGX Xavier.

| **Recurso** | **Utilização** |
|---|---|
| GPU | 82% |
| CPU (8 cores) | 38% |
| Memória (32 GB) | 5,8 GB (18%) |
| Consumo de energia | 24 W (modo MAXN) |
| Temperatura GPU | 56 °C |

Fonte: Elaborado pelo autor.

A utilização moderada de CPU e memória indica margem para adição de funcionalidades auxiliares.




Figura 6.12 – Exemplos de predições do SGIE-Maquinário no conjunto de validação, demonstrando a capacidade de identificação de diferentes tipos de maquinário.
Fonte: Elaborado pelo autor com dados do treinamento.

## 6.3 Análise *End-to-End* do *Pipeline* em Cascata

Em arquiteturas de inferência em cascata, os erros do modelo primário propagam-se para os modelos secundários. Quando o PGIE não detecta um objeto (falso negativo), o SGIE correspondente não recebe a região recortada e, portanto, não tem a oportunidade de processar aquele objeto. Essa característica implica que a capacidade efetiva de detecção do sistema completo é limitada pelo desempenho do primeiro estágio.

### 6.3.1 Propagação de Erros

O impacto da propagação de erros pode ser estimado considerando a relação entre os estágios da cascata. Para cada cadeia de detecção, a cobertura efetiva do sistema é aproximada pelo produto entre o AP do PGIE para a classe correspondente e o mAP do SGIE especializado:

$$\text{Cobertura Efetiva} \approx \text{AP}_{\text{PGIE}}(\text{classe}) \times \text{mAP}_{\text{SGIE}} \tag{6.1}$$

Essa estimativa assume independência entre os erros dos dois estágios e representa um limite superior da cobertura real do sistema, uma vez que outros fatores — como qua-

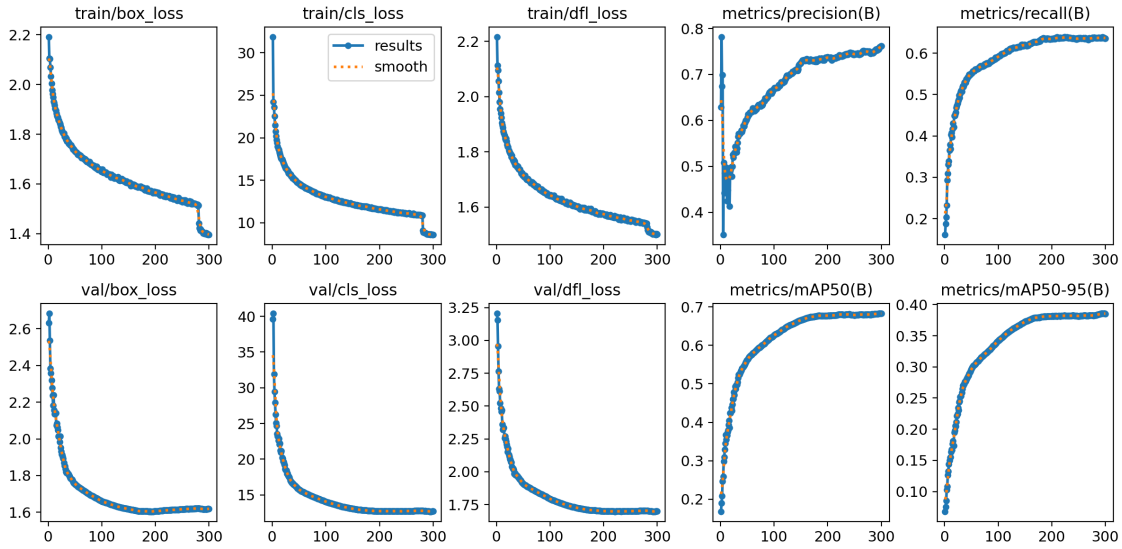



Figura 6.13 – Curvas de treinamento do modelo SGIE-Ferramentas, apresentando a evolução das funções de perda e métricas mAP ao longo das épocas.
Fonte: Elaborado pelo autor com dados do treinamento.

lidade do recorte e variação de escala — também influenciam o desempenho do segundo estágio.

A Tabela 6.8 apresenta a cobertura efetiva estimada para cada cadeia de detecção do *pipeline*.

Tabela 6.8 – Cobertura efetiva estimada por cadeia de detecção no *pipeline* em cascata.

| **Cadeia de Detecção** | **AP PGIE (classe)** | **mAP SGIE** | **Cobertura Efetiva** |
|---|---|---|---|
| Person → SGIE-EPI | 0,865 | 0,770 | ≈ 0,67 |
| Vehicle → SGIE-Maquinário | 0,900 | 0,840 | ≈ 0,76 |
| Tool → SGIE-Ferramentas | 0,826 | 0,710 | ≈ 0,59 |

Fonte: Elaborado pelo autor.

Os valores indicam que a cadeia Vehicle → SGIE-Maquinário apresenta a maior cobertura efetiva (≈ 0,76), enquanto a cadeia Tool → SGIE-Ferramentas apresenta a menor (≈ 0,59). A perda de cobertura na cadeia de ferramentas resulta da combinação de AP moderado do PGIE para a classe Tool (0,826) com o mAP mais baixo entre os SGIEs (0,710).

### 6.3.2 Fluxo de Propagação

A Figura 6.17 ilustra o fluxo de propagação de erros no *pipeline* em cascata, evidenciando como o desempenho de cada estágio contribui para a cobertura efetiva do sis-

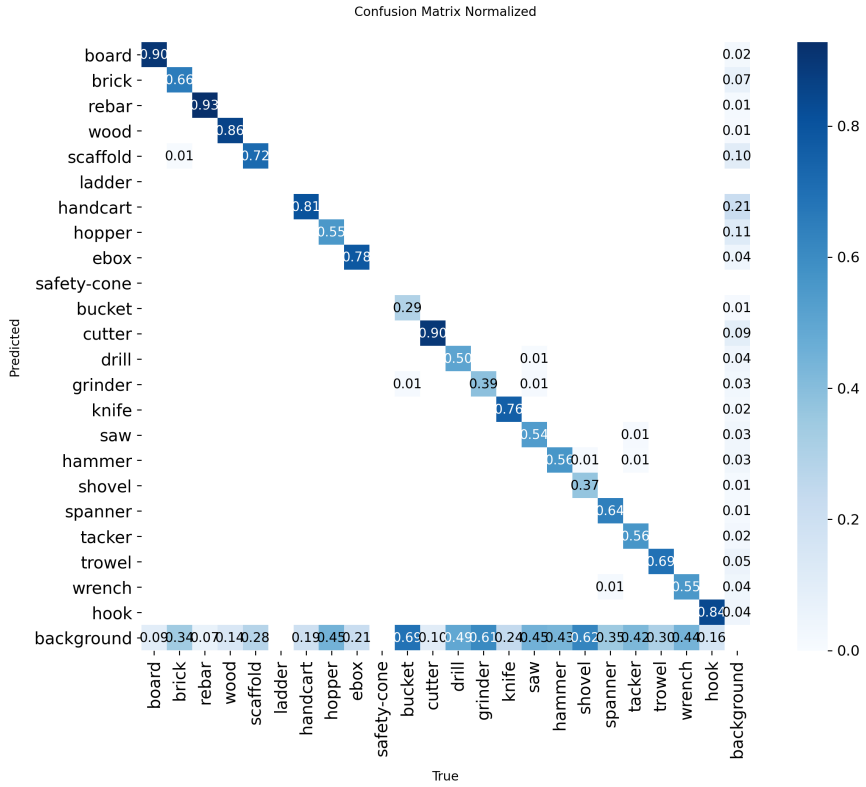



Figura 6.14 – Matriz de confusão normalizada do modelo SGIE-Ferramentas para as 23 classes de equipamentos e ferramentas.
Fonte: Elaborado pelo autor com dados do treinamento.

tema.

### 6.3.3 Impacto dos Falsos Negativos do PGIE

A principal fonte de perda no sistema reside nos falsos negativos do PGIE. Quando o modelo primário não detecta um objeto — seja por oclusão, tamanho reduzido ou baixa confiança —, o SGIE correspondente não processa essa região. Essa limitação é particularmente relevante para a classe Person, cuja detecção correta é pré-requisito para a verificação de EPIs. Com AP de 0,865 para Person, estima-se que aproximadamente 13,5% das pessoas presentes no cenário não sejam encaminhadas ao SGIE-EPI.

Para a classe Tool, o impacto é ainda mais significativo. O AP de 0,826 indica que cerca de 17,4% das ferramentas não são detectadas pelo PGIE, reduzindo a capacidade do SGIE-Ferramentas de classificar esses objetos.

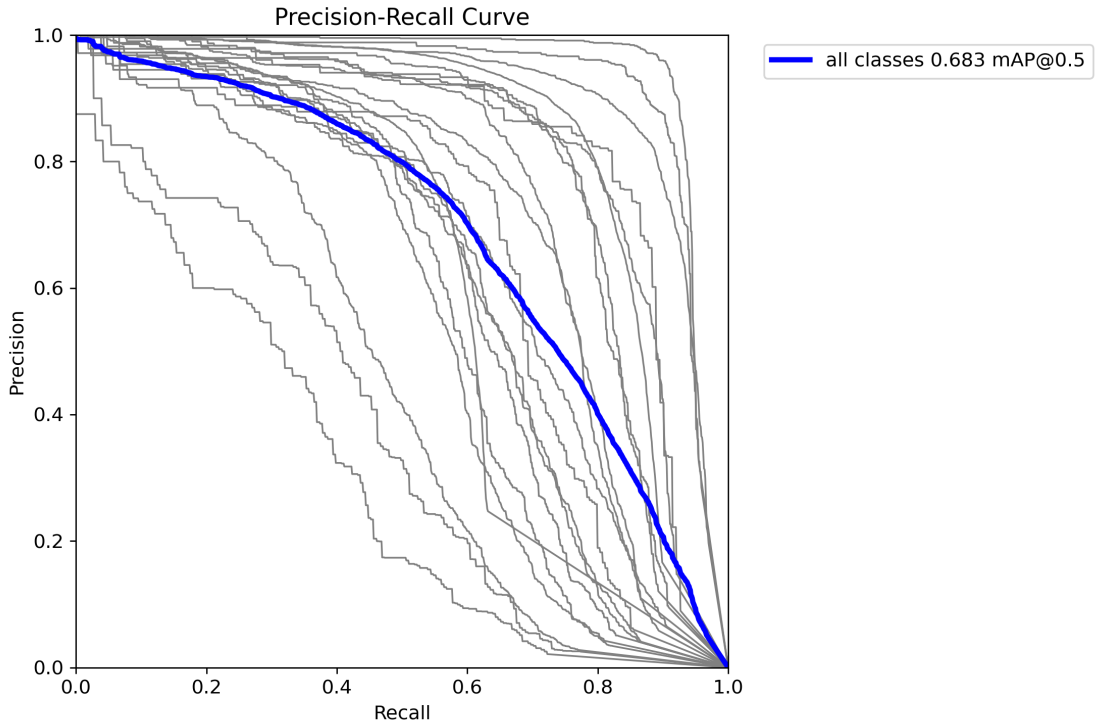



Figura 6.15 – Curvas *Precision-Recall* do modelo SGIE-Ferramentas, com mAP@0,5 indicado na legenda.
Fonte: Elaborado pelo autor com dados do treinamento.

### 6.3.4 Estratégias de Mitigação

A propagação de erros é uma característica inerente a arquiteturas em cascata. Algumas estratégias podem ser adotadas para mitigar esse efeito:

- **Redução do limiar de confiança do PGIE**: Diminuir o *threshold* de confiança do modelo primário aumenta a revocação, reduzindo os falsos negativos. No entanto, essa estratégia eleva o número de falsos positivos, o que pode sobrecarregar os SGIEs com regiões irrelevantes;
- **Retreinamento focado**: O PGIE pode ser retreinado com ênfase nas classes que apresentam maior taxa de falsos negativos, como Tool e Person;
- **Aumento de dados**: A inclusão de imagens com cenários de oclusão e objetos em escalas variadas no *dataset* de treinamento do PGIE pode melhorar a revocação;
- **Fusão de escalas**: A utilização de múltiplas resoluções de entrada ou *test-time augmentation* pode melhorar a detecção de objetos em diferentes tamanhos.

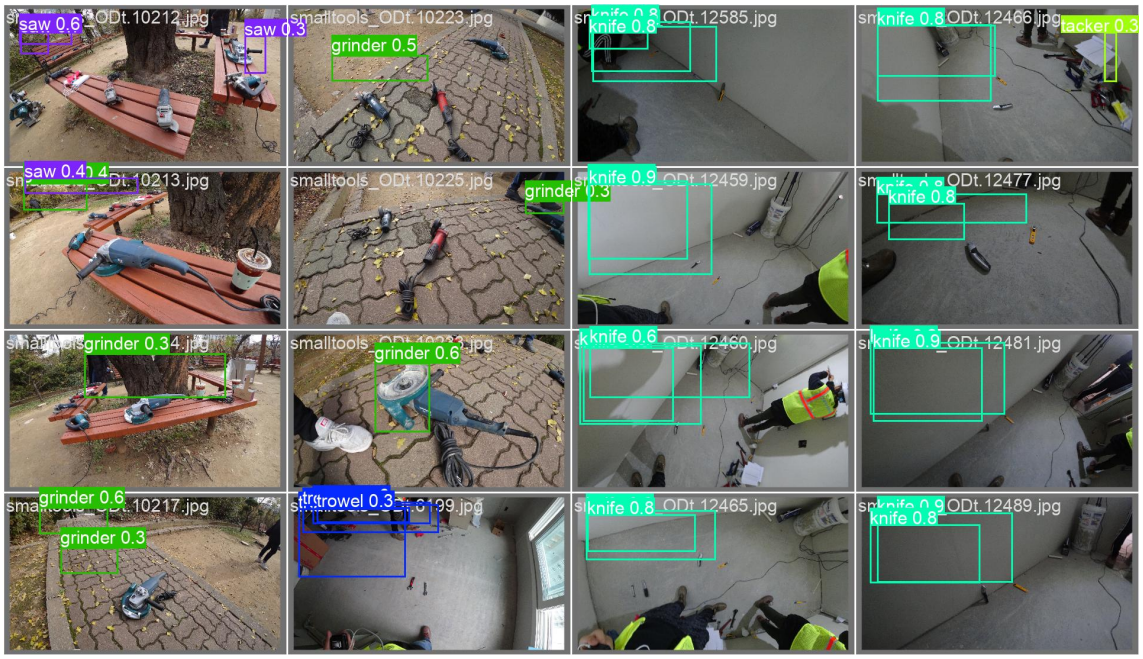



Figura 6.16 – Exemplos de predições do SGIE-Ferramentas no conjunto de validação, demonstrando a detecção de diferentes ferramentas e equipamentos.
Fonte: Elaborado pelo autor com dados do treinamento.

### 6.3.5 Vantagens Apesar da Limitação

Apesar da limitação imposta pela propagação de erros, a arquitetura em cascata oferece vantagens que justificam sua adoção. A modularidade permite que cada SGIE seja atualizado ou substituído de forma independente, sem necessidade de retreinar o sistema completo. A especialização dos SGIEs resulta em maior precisão na classificação fina de cada domínio, em comparação com um modelo único que tentaria classificar simultaneamente todas as 51 saídas de classificação do sistema (5 do PGIE + 14 do SGIE-EPI + 9 do SGIE-Maquinário + 23 do SGIE-Ferramentas), correspondendo a 46 categorias únicas, uma vez que 5 categorias — como Hardhat e Safety Cone — aparecem intencionalmente em mais de um modelo para atender a finalidades distintas (detecção direta no PGIE e classificação refinada nos SGIEs).

Além disso, a eficiência computacional é preservada, pois os SGIEs operam apenas sobre as regiões de interesse detectadas pelo PGIE, e não sobre o *frame* completo. Em um cenário típico com 5 a 10 pessoas no campo de visão, o SGIE-EPI processa apenas essas regiões recortadas, o que representa uma fração do custo computacional de processar a imagem inteira em alta resolução.

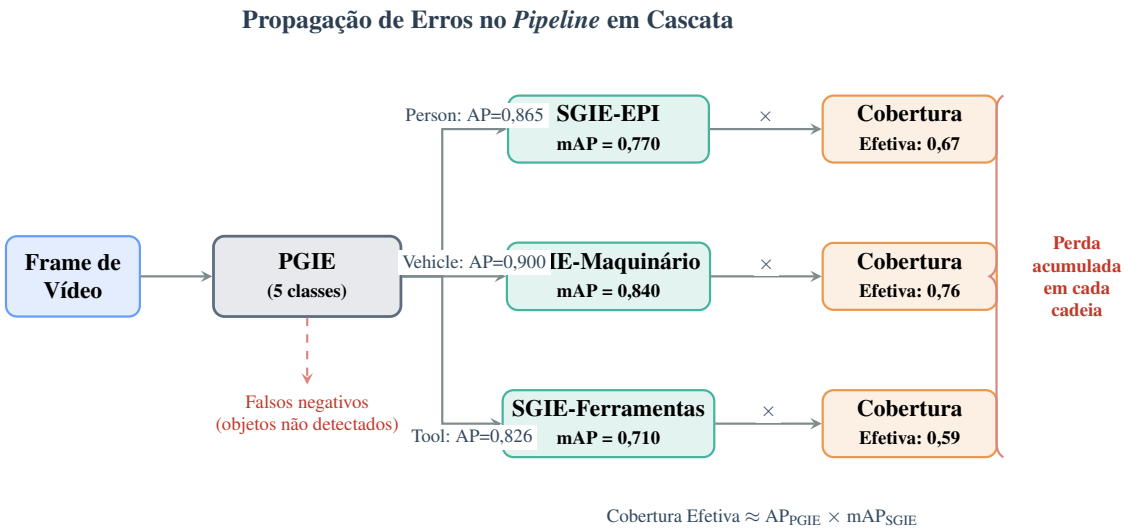



Figura 6.17 – Diagrama de propagação de erros no *pipeline* em cascata. O desempenho do PGIE limita a entrada dos SGIEs, e a cobertura efetiva é estimada como o produto $AP_{PGIE} \times mAP_{SGIE}$.
Fonte: Elaborado pelo autor.

## 6.4 Comparação com a Literatura

A Tabela 6.9 apresenta uma comparação dos resultados com trabalhos relacionados.

Tabela 6.9 – Comparação com trabalhos da literatura.

| Trabalho | Classes | mAP | FPS | Plataforma | Arquitetura |
|---|---|---|---|---|---|
| Nath et al. (2020) | 5 | 0,72 | 8 | RTX 2080 | Modelo único |
| Wang et al. (2021) | 5 | 0,90 | 45 | RTX 3080 | Modelo único |
| Shin et al. (2022) | 5 | 0,82 | 45 | Jetson Xavier | Modelo único |
| **Este trabalho** | **46**† | **0,71–0,84*** | **20** | **Jetson Xavier** | **Cascata** |

* Faixa de mAP@0,5 dos SGIEs. O PGIE alcançou mAP@0,5 de 0,83.
† 46 categorias únicas distribuídas em 4 modelos: 5 (PGIE) + 14 (SGIE-EPI) + 9 (SGIE-Maquinário) + 23 (SGIE-Ferramentas) = 51 saídas no total, das quais 5 categorias (como Hardhat e Safety Cone) aparecem em mais de um modelo para finalidades distintas.
Fonte: Elaborado pelo autor com base nos trabalhos citados.

O sistema proposto apresenta *performance* competitiva operando em dispositivo de borda com arquitetura em cascata (Cai; Vasconcelos, 2021). O *throughput* de 20 FPS é inferior a sistemas com modelo único, porém a arquitetura em cascata possibilita a detecção de 46 categorias únicas, enquanto os trabalhos comparados operam com 5 classes. Essa diferença de escopo deve ser considerada na análise comparativa dos valores de mAP.

## 6.5 Discussão

### 6.5.1 Viabilidade Técnica

Os resultados demonstram a viabilidade técnica do sistema proposto para monitoramento de segurança em tempo real (Xu; Razavi; Zheng, 2023). O mAP entre 0,71 e 0,84 dos diferentes SGIEs e 0,83 do PGIE indica capacidade de identificar corretamente a maioria dos objetos de interesse. O *throughput* de 20 FPS com latência de 58 ms é adequado para aplicações de monitoramento onde a detecção de violações não requer latência sub-milissegundo.

### 6.5.2 Vantagens da Arquitetura em Cascata

A arquitetura PGIE + SGIEs oferece vantagens sobre sistemas com modelo único (Cai; Vasconcelos, 2021):

- **Especialização**: Cada SGIE pode ser otimizado para seu domínio específico;
- **Modularidade**: Novos SGIEs podem ser adicionados sem modificar o sistema existente;
- **Eficiência**: SGIEs processam apenas regiões de interesse, não o *frame* completo;
- **Manutenção**: Cada modelo pode ser retreinado independentemente.

### 6.5.3 Limitações Identificadas

As principais limitações do sistema incluem:

1. **Dependência do PGIE**: Erros de detecção do PGIE propagam-se para os SGIEs. Se o PGIE não detectar uma pessoa, o SGIE-EPI não poderá verificar seus EPIs;
2. **Objetos pequenos e de difícil distinção**: Classes como Mask (AP=0,524), NO-Safety Vest (AP=0,236) e Drill (AP=0,65) apresentam AP reduzido devido à alta variabilidade visual, oclusões frequentes e tamanho reduzido (Singh; Davis, 2018);
3. **Desbalanceamento**: Classes com menor representação no *dataset* apresentam desempenho inferior;
4. ***Throughput* reduzido**: A arquitetura em cascata requer mais tempo de processa-

mento que um modelo único;

5. **Validação controlada**: Os experimentos foram conduzidos com vídeos de *benchmark*, sem validação em canteiro de obras real. Condições adversas como poeira, chuva e iluminação variável podem impactar o desempenho do sistema;
6. **Generalização**: O *dataset* utilizado não contempla a diversidade de canteiros de obras brasileiros, o que pode limitar a generalização dos resultados para contextos locais.

### 6.5.4 Considerações para Produção

Para implantação em ambiente real, recomenda-se (Teizer, 2015):

- Validação com profissionais de segurança do trabalho;
- Integração com sistema de notificação em tempo real;
- Monitoramento contínuo de *performance* e retreinamento periódico;
- Configuração de zonas de interesse para focar detecção em áreas de maior risco.

No capítulo seguinte, apresentam-se as conclusões deste trabalho, a síntese das contribuições e as direções para trabalhos futuros.

# 7 CONCLUSÃO

Esta dissertação apresentou o projeto, implementação e validação de um sistema de análise de vídeo inteligente baseado em computação de borda para monitoramento de segurança em canteiros de obras. O trabalho propôs e validou uma arquitetura de inferência em cascata composta por um modelo primário (PGIE) e três modelos secundários (SGIEs) especializados.

## 7.1 Síntese das Contribuições

As principais contribuições deste trabalho são:

1. **Arquitetura de inferência em cascata**: Proposta e validação de uma arquitetura PGIE + 3 SGIEs para monitoramento de segurança em borda (Cai; Vasconcelos, 2021). O PGIE detecta cinco classes (pessoa, veículo, capacete, colete, ferramenta), das quais três são encaminhadas a SGIEs especializados por domínio (EPIs, maquinário, ferramentas), enquanto as detecções de capacete e colete são utilizadas diretamente pelo sistema de análise;
2. **Quatro modelos treinados**: Desenvolvimento de quatro modelos YOLOv8s (Jocher; Chaurasia; Qiu, 2023) otimizados para o domínio de construção civil. O PGIE alcançou mAP@0,5 de 0,83, enquanto os SGIEs alcançaram mAP@0,5 entre 0,71 e 0,84. O SGIE-EPI detecta 14 classes, o SGIE-Ferramentas detecta 23 classes e o SGIE-Maquinário detecta 9 classes;
3. ***Pipeline* DeepStream**: Implementação de *pipeline* completo com integração de múltiplos modelos YOLO em cascata (NVIDIA Corporation, 2024c), documentando o processo de configuração e desenvolvimento de *parser* customizado;
4. **Validação em borda**: Demonstração de viabilidade em dispositivo Jetson AGX Xavier (NVIDIA Corporation, 2024d) com *throughput* de 20 FPS e latência de 58 ms para o *pipeline* completo.

## 7.2 Obtenção dos Objetivos Específicos

Retomando os objetivos específicos definidos no Capítulo 1, verifica-se que todos foram alcançados ao longo deste trabalho:

1. **Modelo primário (PGIE)**: O modelo foi desenvolvido e treinado com sucesso para detecção das cinco classes em canteiros de obras (Person, Vehicle, Hardhat, Safety-vest, Tool), alcançando mAP@0,5 de 0,83;
2. **Modelos secundários (SGIEs)**: Três modelos especializados foram desenvolvidos e treinados — SGIE-EPI (14 classes, mAP@0,5 de 0,77), SGIE-Maquinário (9 classes, mAP@0,5 de 0,84) e SGIE-Ferramentas (23 classes, mAP@0,5 de 0,71);
3. ***Pipeline* de inferência em cascata**: O *pipeline* foi implementado com êxito no NVIDIA DeepStream SDK, operando no dispositivo Jetson AGX Xavier com *throughput* de 20 FPS;
4. **Funções de *parsing* customizadas**: O *parser* para integração dos modelos YOLOv8 com o DeepStream foi desenvolvido em C++ e validado para os quatro modelos do sistema;
5. **Avaliação de *performance***: O sistema foi avaliado em termos de acurácia (mAP@0,5 de 0,83 para o PGIE e entre 0,71 e 0,84 para os SGIEs) e velocidade de processamento (20 FPS com latência de 58 ms no *pipeline* completo).

## 7.3 Reflexão sobre os Resultados

Os resultados validam a hipótese de que é viável implementar sistemas de monitoramento de segurança com inferência em cascata operando em dispositivos de borda (Murshed et al., 2022). A arquitetura proposta oferece vantagens de modularidade e especialização, permitindo que cada SGIE seja otimizado para seu domínio sem comprometer os demais componentes.

O *throughput* de 20 FPS é adequado para aplicações de monitoramento onde a detecção não requer resposta imediata (Murshed et al., 2022). A utilização moderada de recursos (82% GPU, 18% memória) indica margem para expansão de funcionalidades.

## 7.4 Limitações do Trabalho

As limitações identificadas incluem:

1. **Propagação de erros**: Erros do PGIE afetam diretamente os SGIEs;
2. **Objetos pequenos**: Classes de objetos pequenos apresentam desempenho inferior

(Singh; Davis, 2018);

3. **Validação controlada**: Os experimentos foram conduzidos com vídeos de *benchmark*, sem validação em canteiro real;
4. **Desbalanceamento**: O *dataset* apresenta distribuição desigual entre classes.

## 7.5 Trabalhos Futuros

Direções promissoras para trabalhos futuros incluem:

- **Expansão do *dataset***: Coleta de imagens em canteiros de obras brasileiros com foco em classes sub-representadas;
- **Técnicas para objetos pequenos**: Implementação de SAHI (*Slicing Aided Hyper Inference*) para melhorar detecção de objetos pequenos (Akyon; Altinuc; Temizel, 2022);
- **Mitigação de erros do PGIE**: Desenvolvimento de mecanismos de *fallback* para casos onde o PGIE não detecta objetos relevantes;
- **Validação em produção**: Estudo piloto em canteiro de obras real com avaliação por profissionais de segurança (Teizer, 2015);
- **Evolução arquitetural**: Avaliação de arquiteturas mais recentes como YOLOv9, YOLOv10, YOLO-NAS e RT-DETR (Terven; Cordova-Esparza, 2023).

## 7.6 Considerações Finais

O monitoramento automatizado de segurança em canteiros de obras representa uma aplicação de alto impacto social e econômico da inteligência artificial (Fang et al., 2020). A construção civil permanece como um dos setores mais perigosos para trabalhadores (International Labour Organization, 2024), e tecnologias que contribuam para a redução de acidentes têm potencial de preservar vidas.

Este trabalho demonstrou a viabilidade de implementar sistemas com arquitetura de inferência em cascata em *hardware* de borda de custo acessível (Shi et al., 2016), eliminando barreiras de conectividade que poderiam limitar a adoção em canteiros de diferentes portes e localizações. A arquitetura proposta pode servir como base para desenvolvimentos futuros e adaptações para contextos específicos.

# REFERÊNCIAS

AKYON, F. C.; ALTINUC, S. O.; TEMIZEL, A. Slicing aided hyper inference and fine-tuning for small object detection. In: **2022 IEEE International Conference on Image Processing (ICIP)**. [S.l.]: IEEE, 2022. p. 966–970.

ALBERT, A.; HALLOWELL, M. R.; KLEINER, B. M. Enhancing construction hazard recognition and communication with energy-based cognitive mnemonics and safety meeting maturity model: Multiple baseline study. **Journal of Construction Engineering and Management**, American Society of Civil Engineers, v. 140, n. 2, p. 04013042, 2014.

ALBERT, A.; PANDIT, B.; PATIL, Y. Focus on the fatal-four: Implications for construction hazard recognition. **Safety Science**, Elsevier, v. 128, p. 104774, 2020.

BADIDI, E.; MOUMANE, K.; GHAZI, F. E. Opportunities, applications, and challenges of edge-ai enabled video analytics in smart cities: A systematic review. **IEEE Access**, IEEE, v. 11, p. 80543–80572, 2023.

BARLYBAYEV, A.; AMANGELDY, N.; KURMETBEK, B.; KRAK, I.; RAZAKHOVA, B.; TURSYNOVA, N.; TUREBAYEVA, R. Personal protective equipment detection using yolov8 architecture on object detection benchmark datasets: a comparative study. **Cogent Engineering**, Taylor & Francis, v. 11, n. 1, p. 2333209, 2024.

BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. Yolov4: Optimal speed and accuracy of object detection. **arXiv preprint arXiv:2004.10934**, 2020.

CAI, Z.; VASCONCELOS, N. Cascade r-cnn: High quality object detection and instance segmentation. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 43, n. 5, p. 1483–1498, 2021.

Câmara Brasileira da Indústria da Construção. **Estudo revela dados sobre acidentes de trabalho na construção no Brasil (2010-2023)**. 2024. <https://cbic.org.br/estudo-revela-dados-sobre-acidentes-de-trabalho-na-construcao-no-brasil/>. Pesquisa CBIC/CPRT.

CHEN, C. et al. **GDUT-HWD: A Large-Scale Benchmark Dataset for Hardhat Wearing Detection**. 2020. <https://github.com/GDUT-HWD/Hard-Hat-Detection>. Acessado em: 2024.

CHEN, J.; RAN, X. Deep learning with edge computing: A review. **Proceedings of the IEEE**, IEEE, v. 107, n. 8, p. 1655–1674, 2019.

DELHI, V. S. K.; SANKARLAL, R.; THOMAS, A. Detection of personal protective equipment (ppe) compliance on construction site using computer vision based deep learning techniques. **Frontiers in Built Environment**, Frontiers Media, v. 6, p. 136, 2020.

FANG, Q.; LI, H.; LUO, X.; DING, L.; LUO, H.; ROSE, T. M.; AN, W. Detecting non-hardhat-use by a deep learning method from far-field surveillance videos. **Automation in Construction**, Elsevier, v. 85, p. 1–9, 2018.

FANG, W.; DING, L.; LOVE, P. E. D.; LUO, H.; LI, H.; PEÑA-MORA, F.; ZHONG, B.; ZHOU, C. Computer vision applications in construction safety assurance. **Automation in Construction**, Elsevier, v. 110, p. 103013, 2020.

FANG, W.; DING, L.; LUO, H.; LOVE, P. E. D. Falls from heights: A computer vision-based approach for safety harness detection. **Automation in Construction**, Elsevier, v. 91, p. 53–61, 2018.

GANESH, P.; CHEN, Y.; YANG, Y.; CHEN, D.; WINSLETT, M. YOLO-ReT: Towards high accuracy real-time object detection on edge GPUs. In: **IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)**. [S.l.]: IEEE, 2022. p. 1311–1321.

GHAFFAR, A.; ALI, S.; HUMAYUN, M.; JHANJHI, N.; ALKHAMBASHI, M. S. Performance evaluation and comparison of edge computing platforms for deep learning workloads. **IEEE Access**, IEEE, v. 9, p. 35126–35138, 2021.

GHOLAMI, A.; KIM, S.; DONG, Z.; YAO, Z.; MAHONEY, M. W.; KEUTZER, K. A survey of quantization methods for efficient neural network inference. Chapman and Hall/CRC, p. 291–326, 2022.

GIRSHICK, R.; DONAHUE, J.; DARRELL, T.; MALIK, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2014. p. 580–587.

GUO, B. H. W.; YIU, T. W.; GONZÁLEZ, V. A. Safety management in construction: 20 years of risk modeling. **Safety Science**, Elsevier, v. 129, p. 104805, 2020.

HÄMÄLÄINEN, P.; TAKALA, J.; NENONEN, N.; SAARELA, K. L. Global-, regional- and country-level estimates of the work-related burden of diseases and accidents in 2019. **Scandinavian Journal of Work, Environment & Health**, Nordic Association of Occupational Safety and Health, v. 50, n. 2, p. 73–82, 2024.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2016. p. 770–778.

HOWARD, A. G.; ZHU, M.; CHEN, B.; KALENICHENKO, D.; WANG, W.; WEYAND, T.; ANDREETTO, M.; ADAM, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. **arXiv preprint arXiv:1704.04861**, 2017.

International Labour Organization. **Safety and health at work**. 2024. <https://www.ilo.org/topics-and-sectors/safety-and-health-work>. Global statistics on occupational safety and health.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: **International Conference on Machine Learning**. [S.l.: s.n.], 2015. p. 448–456.

JEONG, E.; KIM, J.; HA, S. TensorRT-based framework and optimization methodology for deep learning inference on jetson boards. **ACM Transactions on Embedded Computing Systems**, ACM, v. 21, n. 5, p. 51, 2022.

JOCHER, G.; CHAURASIA, A.; QIU, J. **Ultralytics YOLOv8**. 2023. <https://github.com/ultralytics/ultralytics>. Acessado em: 2024.

KONSTANTINOV, A. et al. A systematic review of computer vision-based personal protective equipment compliance in industry practice: advancements, challenges and future directions. **Artificial Intelligence Review**, Springer, v. 57, p. 1–42, 2024.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2012. p. 1097–1105.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, IEEE, v. 86, n. 11, p. 2278–2324, 1998.

LEE, Y.-R.; JUNG, S.-H.; KANG, K.-S.; RYU, H.-C.; RYU, H.-G. Deep learning-based framework for monitoring wearing personal protective equipment on construction sites. **Journal of Computational Design and Engineering**, Oxford University Press, v. 10, n. 2, p. 905–917, 2023.

LIANG, T.; GLOSSNER, J.; WANG, L.; SHI, S.; ZHANG, X. Pruning and quantization for deep neural network acceleration: A survey. **Neurocomputing**, Elsevier, v. 461, p. 370–403, 2021.

LIN, T.-Y.; MAIRE, M.; BELONGIE, S.; HAYS, J.; PERONA, P.; RAMANAN, D.; DOLLÁR, P.; ZITNICK, C. L. Microsoft coco: Common objects in context. In: **European Conference on Computer Vision**. [S.l.]: Springer, 2014. p. 740–755.

LIU, H.-I.; GALINDO, M.; XIE, H.; WONG, L.-K.; SHUAI, H.-H.; LI, Y.-H.; CHENG, W.-H. Lightweight deep learning for resource-constrained environments: A survey. **ACM Computing Surveys**, ACM, v. 56, n. 10, p. 267, 2024.

LIU, S.; QI, L.; QIN, H.; SHI, J.; JIA, J. Path aggregation network for instance segmentation. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2018. p. 8759–8768.

LIU, W.; ANGUELOV, D.; ERHAN, D.; SZEGEDY, C.; REED, S.; FU, C.-Y.; BERG, A. C. Ssd: Single shot multibox detector. In: **European Conference on Computer Vision**. [S.l.]: Springer, 2016. p. 21–37.

LUO, X.; LI, H.; CAO, D.; YU, Y.; YANG, X.; HUANG, T. Towards efficient and objective work sampling: Recognizing workers' activities in site surveillance videos with two-stream convolutional networks. **Automation in Construction**, Elsevier, v. 94, p. 360–370, 2018.

MENDEZ, J.; BIERZYNSKI, K.; CUÉLLAR, M. P.; MORALES, D. P. Edge intelligence: Concepts, architectures, applications, and future directions. **ACM Transactions on Embedded Computing Systems**, ACM, v. 21, n. 5, p. 48, 2022.

Ministério do Trabalho e Emprego. **NR-18 - Segurança e Saúde no Trabalho na Indústria da Construção**. 2024. <https://www.gov.br/trabalho-e-emprego/pt-br/acesso-a-informacao/participacao-social/conselhos-e-orgaos-colegiados/comissao-tripartite-paritaria-permanente/normas-regulamentadora/normas-regulamentadoras-vigentes/norma-regulamentadora-no-18-nr-18>. Atualizada pela Portaria MTE n. 1420 de agosto de 2024.

Ministério Público do Trabalho; Organização Internacional do Trabalho. **Observatório de Segurança e Saúde no Trabalho**. 2024. <https://smartlabbr.org/sst>. Plataforma SmartLab - Dados de acidentes de trabalho no Brasil.

MITTAL, P. A comprehensive survey of deep learning-based lightweight object detection models for edge devices. **Artificial Intelligence Review**, Springer, v. 57, p. 242, 2024.

MITTAL, S. A survey on optimized implementation of deep learning models on the nvidia jetson platform. **Journal of Systems Architecture**, Elsevier, v. 97, p. 428–442, 2019.

MNEYMNEH, B. E.; ABBAS, M.; KHOURY, H. Vision-based framework for intelligent monitoring of hardhat wearing on construction sites. **Journal of Computing in Civil Engineering**, ASCE, v. 33, n. 2, p. 04018066, 2019.

MURSHED, M. G. S.; MURPHY, C.; HOU, D.; KHAN, N.; ANANTHANARAYANAN, G.; HUSSAIN, F. Machine learning at the network edge: A survey. **ACM Computing Surveys**, ACM, v. 54, n. 8, p. 1–37, 2022.

NATH, N. D.; BEHZADAN, A. H.; PAAL, S. G. Deep learning for site safety: Real-time detection of personal protective equipment. **Automation in Construction**, Elsevier, v. 112, p. 103085, 2020.

NVIDIA Corporation. **State-of-the-Art Real-time Multi-Object Trackers with NVIDIA DeepStream SDK 6.2**. 2023. <https://developer.nvidia.com/blog/state-of-the-art-real-time-multi-object-trackers-with-nvidia-deepstream-sdk-6-2/>. NVIDIA Technical Blog.

NVIDIA Corporation. **Gst-nvinfer Plugin Documentation**. 2024. <https://docs.nvidia.com/metropolis/deepstream/dev-guide/text/DS_plugin_gst-nvinfer.html>. Acessado em: 2024.

NVIDIA Corporation. **Gst-nvtracker Plugin Documentation**. 2024. <https://docs.nvidia.com/metropolis/deepstream/dev-guide/text/DS_plugin_gst-nvtracker.html>. Acessado em: 2024.

NVIDIA Corporation. **NVIDIA DeepStream SDK Developer Guide**. 2024. <https://docs.nvidia.com/metropolis/deepstream/dev-guide/>. Acessado em: 2024.

NVIDIA Corporation. **NVIDIA Jetson AGX Xavier Developer Kit**. 2024. <https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit>. Acessado em: 2024.

NVIDIA Corporation. **NVIDIA TensorRT Documentation**. 2024. <https://docs.nvidia.com/deeplearning/tensorrt/>. Acessado em: 2024.

OESTERREICH, T. D.; TEUTEBERG, F. Understanding the implications of digitisation and automation in the context of industry 4.0: A triangulation approach and elements of a research agenda for the construction industry. **Computers in Industry**, Elsevier, v. 83, p. 121–139, 2016.

ONNX Community. **Open Neural Network Exchange (ONNX)**. 2019. <https://onnx.ai/>. Acessado em: 2024.

PADILLA, R.; NETTO, S. L.; SILVA, E. A. B. da. A survey on performance metrics for object-detection algorithms. In: **2020 International Conference on Systems, Signals and Image Processing (IWSSIP)**. [S.l.]: IEEE, 2020. p. 237–242.

PAN, S. J.; YANG, Q. A survey on transfer learning. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 22, n. 10, p. 1345–1359, 2010.

PRASHANTHI, S. K.; KESANAPALLI, S. A.; SIMMHAN, Y. Characterizing the performance of accelerated jetson edge devices for training deep learning models. **Proceedings of the ACM on Measurement and Analysis of Computing Systems**, ACM, v. 6, n. 3, p. 50, 2022.

REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. You only look once: Unified, real-time object detection. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2016. p. 779–788.

REDMON, J.; FARHADI, A. Yolo9000: better, faster, stronger. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2017. p. 7263–7271.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. **arXiv preprint arXiv:1804.02767**, 2018.

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2015. p. 91–99.

SANDLER, M.; HOWARD, A.; ZHU, M.; ZHMOGINOV, A.; CHEN, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2018. p. 4510–4520.

SEO, J.; HAN, S.; LEE, S.; KIM, H.-y. Computer vision techniques for construction safety and health monitoring. **Advanced Engineering Informatics**, Elsevier, v. 29, n. 2, p. 239–251, 2015.

SHI, W.; CAO, J.; ZHANG, Q.; LI, Y.; XU, L. Edge computing: Vision and challenges. **IEEE Internet of Things Journal**, IEEE, v. 3, n. 5, p. 637–646, 2016.

SHIN, D.-J.; KIM, J.-J. A deep learning framework performance evaluation to use YOLO in Nvidia Jetson platform. **Applied Sciences**, MDPI, v. 12, n. 8, p. 3734, 2022.

SHORTEN, C.; KHOSHGOFTAAR, T. M. A survey on image data augmentation for deep learning. **Journal of Big Data**, Springer, v. 6, n. 1, p. 1–48, 2019.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, 2015.

SINGH, B.; DAVIS, L. S. An analysis of scale invariance in object detection-snip. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2018. p. 3578–3587.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, n. 1, p. 1929–1958, 2014.

TAN, M.; LE, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In: **International Conference on Machine Learning**. [S.l.: s.n.], 2019. p. 6105–6114.

TAYMANS, W.; BAKER, S.; WINGO, A.; BULTJE, R. S.; KOST, S. **GStreamer Application Development Manual**. 2013. <https://gstreamer.freedesktop.org/documentation/>. Acessado em: 2024.

TEIZER, J. Status quo and open challenges in vision-based sensing and tracking of temporary resources on infrastructure construction sites. **Advanced Engineering Informatics**, Elsevier, v. 29, n. 2, p. 225–238, 2015.

TERVEN, J.; CORDOVA-ESPARZA, D.-M. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. **Machine Learning and Knowledge Extraction**, MDPI, v. 5, n. 4, p. 1680–1716, 2023.

Ultralytics. **YOLOv5 Documentation**. 2024. <https://docs.ultralytics.com/yolov5/>. Acessado em: 2024.

WANG, C.-Y.; LIAO, H.-Y. M.; WU, Y.-H.; CHEN, P.-Y.; HSIEH, J.-W.; YEH, I.-H. Cspnet: A new backbone that can enhance learning capability of cnn. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops**. [S.l.: s.n.], 2020. p. 390–391.

WANG, Z. **Safety Helmet Wearing Dataset (SHWD)**. 2021. <https://github.com/njvisionpower/Safety-Helmet-Wearing-Dataset>. Acessado em: 2024.

WANG, Z.; WU, Y.; YANG, L.; THIRUNAVUKARASU, A.; EVISON, C.; ZHAO, Y. Fast personal protective equipment detection for real construction sites using deep learning approaches. **Sensors**, MDPI, v. 21, n. 10, p. 3478, 2021.

WU, J.; CAI, N.; CHEN, W.; WANG, H.; WANG, G. Automatic detection of hardhats worn by construction personnel: A deep learning approach and benchmark dataset. **Automation in Construction**, Elsevier, v. 106, p. 102894, 2019.

XU, R.; RAZAVI, S.; ZHENG, R. Edge video analytics: A survey on applications, systems and enabling techniques. **IEEE Communications Surveys & Tutorials**, IEEE, v. 25, n. 4, p. 2951–2982, 2023.

ZHAO, Z.-Q.; ZHENG, P.; XU, S.-t.; WU, X. Object detection with deep learning: A review. **IEEE Transactions on Neural Networks and Learning Systems**, IEEE, v. 30, n. 11, p. 3212–3232, 2019.

ZHENG, Z.; WANG, P.; LIU, W.; LI, J.; YE, R.; REN, D. Distance-iou loss: Faster and better learning for bounding box regression. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2020. v. 34, n. 7, p. 12993–13000.

ZHOU, Z.; CHEN, X.; LI, E.; ZENG, L.; LUO, K.; ZHANG, J. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. **Proceedings of the IEEE**, IEEE, v. 107, n. 8, p. 1738–1762, 2019.

ZHOU, Z.; GOH, Y. M.; LI, Q. Overview and analysis of safety management studies in the construction industry. **Safety Science**, Elsevier, v. 72, p. 337–350, 2015.