EDGE VISION MODEL

PIPELINE DE MODELAGEM, AVALIAÇÃO E COMPARAÇÃO DE ARQUITETURAS DE DETECÇÃO DE OBJETOS

Relatório Técnico

Londrina, Dezembro de 2025

1. Introdução

O avanço dos modelos de aprendizado profundo aplicados à visão computacional tem possibilitado a construção de sistemas cada vez mais precisos e eficientes para tarefas de detecção de objetos. No entanto, a escolha adequada da arquitetura de modelagem envolve múltiplos fatores que vão além da acurácia isolada, incluindo custo computacional, viabilidade de execução em tempo real e restrições de hardware.

Nesse contexto, torna-se essencial a realização de uma análise comparativa sistemática entre diferentes arquiteturas de detecção, baseada em métricas objetivas e em um processo experimental reprodutível. Tal análise permite compreender os trade-offs entre desempenho e custo, subsidiando decisões técnicas fundamentadas.

Este relatório apresenta a documentação técnica do Edge Vision Model, uma Prova de Conceito (PoC) dedicada à preparação de dados, treinamento, avaliação e comparação de arquiteturas de detecção de objetos, utilizando um pipeline estruturado que contempla conversão e organização de datasets, avaliação padronizada no estilo COCO, análise de custo computacional e consolidação de resultados.

O sistema foi projetado com foco em clareza arquitetural, separação de responsabilidades e geração de evidências quantitativas que suportam decisões técnicas no contexto de aplicações de visão computacional, incluindo cenários de computação de borda (edge computing).

2. Objetivos

2.1 Objetivo Geral

Desenvolver um pipeline estruturado para preparação de dados, treinamento, avaliação e comparação de modelos de detecção de objetos, permitindo analisar simultaneamente métricas de desempenho e custo computacional de diferentes arquiteturas.

2.2 Objetivos Específicos

- Preparar e organizar datasets de detecção de objetos em formatos específicos para cada arquitetura.
- Treinar diferentes arquiteturas de detecção de objetos a partir de datasets preparados.
- Avaliar os modelos utilizando métricas padronizadas no estilo COCO.
- Comparar os modelos com base em métricas de precisão e recall.
- Medir o custo computacional dos modelos durante a inferência.
- Consolidar os resultados de qualidade e custo em artefatos únicos e reprodutíveis.
- Gerar visualizações comparativas que evidenciem trade-offs entre desempenho e eficiência.

3. Visão Geral do Sistema

O Edge Vision Model foi concebido como uma extensão natural da etapa de Análise Exploratória de Dados (EDA), atuando exclusivamente após a validação estrutural e estatística do dataset original. O sistema opera sobre um dataset externo e segue um fluxo bem definido, no qual cada etapa é executada de forma independente e explícita.

A pipeline é composta por três pontos de entrada principais:

- Preparação do dataset: responsável pela organização e cópia dos dados para estruturas específicas por arquitetura.
- Treinamento dos modelos: responsável pelo treinamento individual das arquiteturas.
- Avaliação: responsável pela mensuração padronizada do desempenho dos modelos.
- Comparação: responsável pela análise comparativa, avaliação de custo computacional e consolidação final dos resultados.

Essa separação garante previsibilidade, reprodutibilidade experimental e clareza operacional.

4. Arquitetura do Sistema

A arquitetura do Edge Vision Model reflete diretamente a organização real do repositório, sendo estruturada de forma modular para garantir baixo acoplamento, alta coesão e rastreabilidade experimental.

config/

Centraliza todas as configurações do projeto, incluindo:

- **settings.py:** paths globais, flags da pipeline e parâmetros gerais.
- **dataset.yaml:** referência ao dataset externo.
- **metrics.yaml:** definição formal das métricas de avaliação.
- **models/:** arquivos YAML com configurações específicas de cada arquitetura.

core/

Contém o núcleo funcional da pipeline:

- **config_validator.py:** validação estrutural das configurações.
- **dataset_preparer.py:** preparação determinística do dataset para cada arquitetura.
- **evaluation_runner.py:** coordenação da execução dos avaliadores.

trainers/

Responsável pelo treinamento dos modelos:

- yolo_trainer.py
- ssd_trainer.py
- faster_rcnn_trainer.py

Cada trainer executa o treinamento de uma arquitetura específica e persiste os pesos e metadados em artifacts/models.

evaluators/

Responsável pela avaliação padronizada:

- base_coco_evaluator.py
- yolo_evaluator.py
- ssd_evaluator.py
- faster_rcnn_evaluator.py

Os resultados são exportados em CSV para artifacts/metrics.

comparator/

Responsável pela análise comparativa:

- **model_comparator_metrics.py:** comparação de métricas de qualidade.
- **model_comparator_cost.py:** avaliação de custo computacional.
- **model_comparator_merge.py:** consolidação final dos resultados.

viz/

Utilitários de visualização:

- **plots.py:** geração de gráficos comparativos a partir do CSV final consolidado.

artifacts/

Diretório de saída de todos os resultados:

- dados preparados
- modelos treinados
- métricas
- comparações
- gráficos

utils/

- **logging_global.py:** configuração centralizada de logging.

Essa arquitetura garante clareza estrutural, reprodutibilidade e facilidade de manutenção.

5. Metodologia de Desenvolvimento

A metodologia adotada no Edge Vision Model foi definida de forma a garantir equidade na avaliação, reprodutibilidade experimental e comparabilidade direta entre arquiteturas desenvolvidas com frameworks distintos. Todas as etapas descritas a seguir foram implementadas explicitamente no código do projeto, não havendo procedimentos manuais ou ajustes externos não documentados.

5.1 Preparação de Dados

A preparação dos dados é realizada a partir de um dataset externo previamente validado na etapa de EDA. Essa etapa é executada de forma independente antes do treinamento e:

- Não modifica o dataset original.
- Organiza e copia os dados para estruturas específicas por arquitetura.
- Gera versões preparadas específicas para YOLO, SSD e Faster R-CNN.

Para os modelos baseados em TorchVision (SSD e Faster R-CNN), os dados são organizados de acordo com o formato esperado pelas APIs de detecção do framework. Para o YOLO, os dados são preparados no formato nativo do modelo, com arquivos de anotação no padrão YOLO.

5.2 Treinamento dos Modelos

Foram treinadas três arquiteturas de detecção de objetos:

- YOLO, utilizando implementação nativa do framework YOLO.
- SSD, utilizando implementação baseada em TorchVision.
- Faster R-CNN, utilizando implementação baseada em TorchVision.

Cada arquitetura possui um módulo de treinamento dedicado, responsável por:

- Carregar os dados preparados.
- Inicializar o modelo com pesos apropriados.
- Executar o treinamento supervisionado.
- Salvar o melhor modelo e o modelo final.
- Registrar metadados do treinamento.

Os pesos treinados são armazenados em diretórios específicos dentro de artifacts/models, garantindo rastreabilidade entre modelo, arquitetura e configuração utilizada.

5.3 Avaliação Padronizada dos Modelos

Embora os modelos sejam treinados utilizando frameworks distintos, toda a avaliação é realizada de forma padronizada, utilizando métricas no estilo COCO. Para garantir comparabilidade direta:

- As predições do YOLO são convertidas para o formato COCO antes da avaliação.
- As predições dos modelos baseados em TorchVision já seguem estrutura compatível com COCO.

Esse processo assegura que todas as arquiteturas sejam avaliadas sob o mesmo critério, evitando vieses decorrentes de diferenças de framework ou formato de saída.
As métricas calculadas incluem AP e AR em múltiplos limiares de IoU, conforme definido no padrão COCO.

5.4 Comparação de Métricas de Qualidade

As métricas de avaliação geradas são consolidadas por meio de um comparador dedicado, responsável por:

- Agregar métricas por modelo.
- Gerar rankings por métrica.
- Exportar os resultados em formatos estruturados.

Essa etapa opera exclusivamente sobre os artefatos de métricas previamente gerados na etapa de avaliação, não realizando inferência nem reavaliação dos modelos.

5.5 Avaliação de Custo Computacional

O custo computacional dos modelos é avaliado durante a inferência, utilizando um subconjunto fixo do conjunto de teste. As métricas de custo incluem:

- Tempo médio de inferência por imagem.
- Frames por segundo (FPS).
- Pico de uso de memória de GPU (VRAM).

Essa abordagem reflete o uso real dos modelos em cenários operacionais e permite avaliar sua viabilidade em ambientes com restrições de hardware.

5.6 Consolidação Final dos Resultados

Os resultados de desempenho e custo computacional são integrados em um único artefato final, que reúne todas as métricas relevantes por modelo. Esse artefato serve como base única para análises comparativas e visualizações.
Esse processo é realizado por um módulo dedicado de merge, responsável por integrar métricas de qualidade e custo computacional em um único artefato final.

5.7 Geração de Visualizações

A partir do artefato final consolidado, o sistema gera gráficos comparativos que representam:

- Métricas de qualidade (AP e AR).
- Métricas de custo computacional (tempo, FPS e VRAM).

As visualizações possuem caráter analítico e exploratório, não realizando qualquer interpretação automática dos resultados.
Os gráficos são gerados exclusivamente a partir do artefato final consolidado, sem acesso direto aos dados de treinamento, avaliação ou inferência.

6. Registro e Monitoramento

O sistema utiliza logging centralizado para registrar eventos relevantes durante a execução da pipeline, incluindo:

- Inicialização das etapas.
- Execução de treinamento e avaliação.
- Geração de comparações e plots.
- Encerramento do processo.

Isso garante rastreabilidade completa e suporte à depuração.
Os logs seguem o padrão edge_vision_model_YYYY-MM-DD.log, garantindo rastreabilidade temporal das execuções.

7. Resultados Obtidos

A execução do Edge Vision Model resultou em um conjunto consolidado de métricas que permitem comparar, de forma objetiva, o desempenho e o custo computacional das arquiteturas avaliadas.

7.1 Tabela-Resumo dos Resultados

Modelo	AP_50_95	AP_50	AP_75	AR_50_95	Tempo Médio (ms)	FPS	Pico de VRAM (MB)
Faster R-CNN	0.728	0.907	0.846	0.781	201.22	4.97	472.09
SSD	0.701	0.879	0.840	0.755	31.95	31.29	348.68
YOLO	0.751	0.853	0.789	0.780	9.73	102.75	39.04

Essa consolidação evidencia claramente os trade-offs entre qualidade de detecção e eficiência computacional entre as arquiteturas analisadas.

7.2 Análise Gráfica das Métricas de Qualidade

As análises a seguir são realizadas exclusivamente a partir das métricas consolidadas e visualizações geradas automaticamente pelo sistema, sem aplicação de heurísticas externas.

Os gráficos apresentados nesta seção permitem uma análise visual comparativa do desempenho dos modelos sob diferentes critérios de qualidade, considerando métricas padronizadas no estilo COCO.

Conforme observado na Figura 1, o gráfico de AP_50_95 (Desempenho Global) indica que o modelo YOLO apresentou o maior valor médio considerando múltiplos limiares de IoU, seguido de perto pelo Faster R-CNN. O modelo SSD apresentou desempenho inferior nesse critério, indicando menor consistência global na detecção.

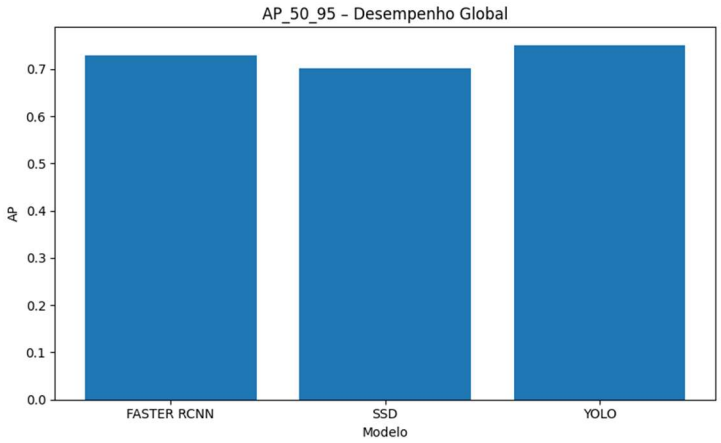

Figura 1 – Comparação do desempenho global dos modelos (AP_50_95).

A Figura 2 apresenta a métrica AP_75 (Precisão Espacial), que enfatiza a qualidade da localização das bounding boxes em cenários de maior exigência

espacial. Observa-se que Faster R-CNN e SSD obtiveram valores superiores ao YOLO, comportamento coerente com arquiteturas baseadas em propostas de região.

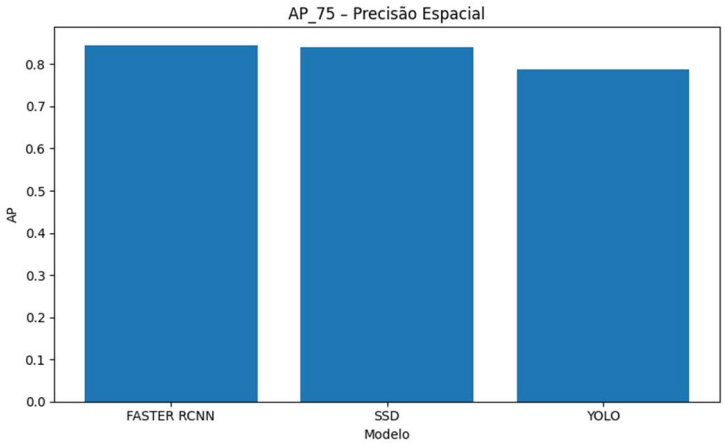

Figura 2 – Comparação da precisão espacial dos modelos (AP_75).

Na Figura 3, referente ao AR_50_95 (Recall), observa-se desempenho semelhante entre YOLO e Faster R-CNN, ambos superiores ao SSD, indicando maior capacidade de recuperação dos objetos presentes no dataset avaliado.

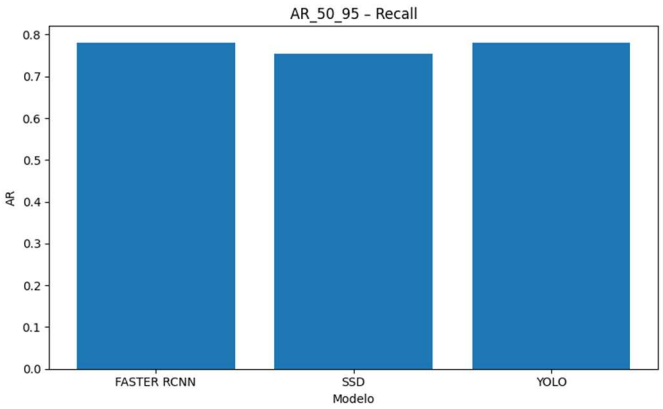

Figura 3 – Comparação do recall médio dos modelos (AR_50_95).

Esses resultados demonstram que diferentes arquiteturas apresentam vantagens distintas conforme o critério de avaliação adotado.

7.3 Análise Gráfica do Custo Computacional

Os gráficos apresentados nesta seção evidenciam o custo computacional associado à inferência dos modelos avaliados, considerando métricas diretamente relacionadas à viabilidade operacional, especialmente em cenários de tempo real e computação de borda.

A Figura 4 apresenta a comparação do desempenho em frames por segundo (FPS). Observa-se que o YOLO apresenta desempenho significativamente superior, indicando elevada capacidade de processamento em tempo real. O SSD apresenta desempenho intermediário, enquanto o Faster R-CNN apresenta valores reduzidos de FPS.

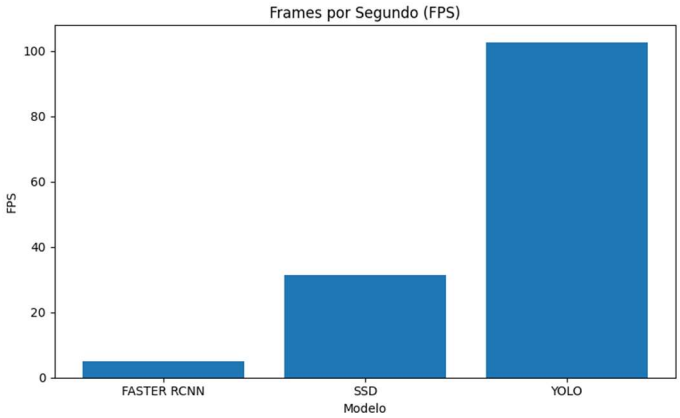

Figura 4 – Comparação do desempenho em frames por segundo (FPS) durante a inferência.

Na Figura 5, referente ao tempo médio de inferência por imagem, observa-se novamente a superioridade do YOLO, com tempos significativamente inferiores aos demais modelos. O SSD apresenta tempos intermediários, enquanto o Faster R-CNN apresenta o maior custo temporal.

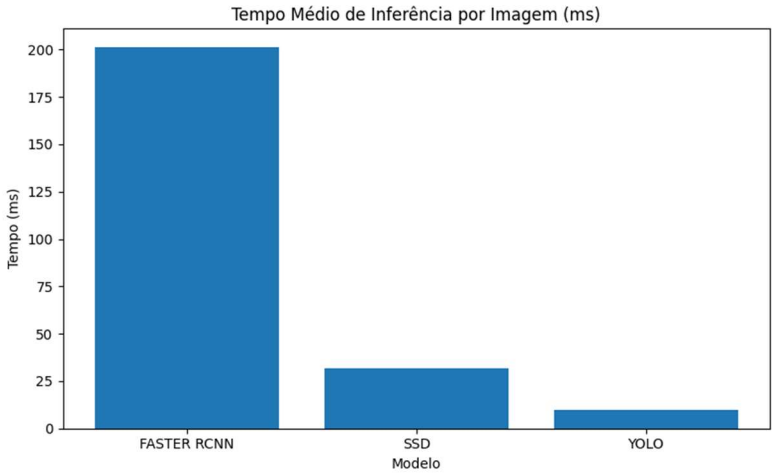

Figura 5 – Tempo médio de inferência por imagem para cada modelo avaliado.

A Figura 6 apresenta o pico de uso de memória de GPU (VRAM) durante a inferência. O YOLO destaca-se pelo baixo consumo de memória, enquanto SSD e Faster R-CNN apresentam consumo significativamente superior, sendo o Faster R-CNN o mais exigente nesse aspecto.

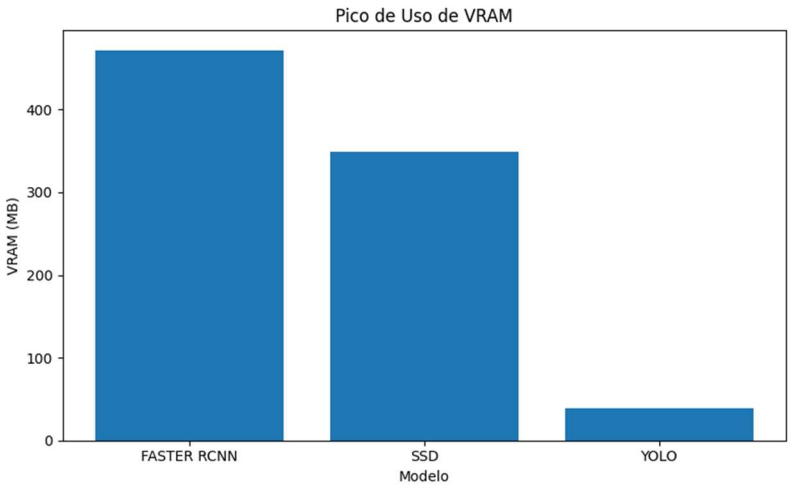

Figura 6 – Comparação do consumo máximo de memória de GPU (VRAM) entre os modelos.

Os resultados gráficos de custo computacional reforçam a adequação do YOLO para cenários com restrições de hardware e aplicações em tempo real, enquanto evidenciam as limitações operacionais das arquiteturas baseadas em propostas de região em ambientes de computação de borda.

7.4 Critérios de Avaliação e Escolha do Modelo

A definição dos critérios de escolha foi realizada a partir das métricas consolidadas e das visualizações apresentadas nas seções anteriores.
A escolha do modelo mais adequado depende diretamente do contexto de aplicação e dos requisitos do sistema.
Considerando qualidade de detecção, o Faster R-CNN apresenta vantagens em precisão espacial, enquanto o YOLO oferece desempenho global competitivo.
Sob a perspectiva de eficiência computacional, o YOLO destaca-se como a arquitetura mais adequada para cenários de tempo real e edge computing, devido ao alto FPS, baixo tempo de inferência e reduzido consumo de memória.
Dessa forma, para aplicações que exigem equilíbrio entre desempenho e custo computacional, o YOLO configura-se como a escolha mais apropriada no contexto avaliado neste projeto.

8. Limitações

- A comparação é baseada em um conjunto fixo de arquiteturas.
- As medições de custo computacional refletem um ambiente computacional específico, não contemplando variações de hardware. Não são avaliadas técnicas avançadas de otimização ou compressão de modelos.

Essas limitações não comprometem o objetivo principal do projeto.

9. Conclusão

O Edge Vision Model demonstrou a viabilidade de um pipeline completo e reprodutível para treinamento, avaliação e comparação de modelos de detecção de objetos. A arquitetura modular adotada permitiu separar claramente as responsabilidades, enquanto a consolidação de métricas de qualidade e custo computacional forneceu uma base objetiva para decisões técnicas.

A solução apresentada estabelece uma fundação consistente para estudos futuros envolvendo otimização, implantação em ambientes de borda e aplicações em tempo real. O sistema não tem como objetivo a implantação direta em produção, mas sim a geração de evidências técnicas para subsidiar decisões de modelagem em projetos subsequentes.