Insper

Trabalho de Conclusão de Curso

Relatório do Projeto Final – Capstone

Diego Baptista Daurea Machado

Douglas Pablo Braçal Celestino

Gustavo Mendes da Silva

Pedro Ivo de Rizzo

Solução Full-Stack de machine learning para Visão Computacional em Ambientes Industriais: Classificação, Versionamento e Treinamento Local de Modelos

São Paulo

2025

Douglas Pablo Bracale Celestino

Diego Baptista Daurea Machado

Pedro Ivo de Rizzo

Gustavo Mendes da Silva

Solução Full-Stack de machine learning para Visão Computacional em Ambientes Industriais: Classificação, Versionamento e Retreinamento Local de Modelos

Relatório apresentado aos respectivos cursos de Graduação em Engenharia de Computação, e Mecatrônica, como requisito parcial para a obtenção do título de Bacharel no respectivo curso.

Orientador: Prof. Dr. Carlos Magno de Oliveira Valente.

Coorientador: Prof. Dr. Marcelo Hashimoto

Mentor: Tiago Prata, MSc (R&D Engineer, Software Engineer and Talent Ambassador at ABB)

Coordenador Capstone: Prof. Dr. Luciano P. Soares

São Paulo

2025

Douglas Pablo Bracale Celestino

Diego Baptista Daurea Machado

Pedro Ivo de Rizzo

Gustavo Mendes da Silva

Solução Full-Stack de machine learning para Visão Computacional em Ambientes Industriais: Classificação, Versionamento e Retreinamento Local de Modelos

Relatório apresentado aos respectivos cursos de Graduação em Engenharia Computação e Mecatrônica, como requisito parcial para a obtenção do título de Bacharel no respectivo curso.

Orientador: Prof. Carlos Magno de Oliveira Valente

Coorientador: Prof. Marcelo Hashimoto

Banca Examinadora

Prof. Dr. Carlos Magno de Oliveira Valente

Prof. Dr. Luiz Fernando Cardoso dos Santos Durão

Prof. Dr. Maciel Calebe Vidal

Insper

Resumo

Este projeto dedica-se a estabelecer uma solução full-stack de machine learning para algoritmos de visão computacional, com versionamento, reclassificação e retreinamento de modelos, operando integralmente via Intranet para facilitar seu uso em plantas industriais. Tal abordagem visa atender qualquer processo, permitindo ao usuário realizar o upload de mídias, escolher as *Labels* que deseja classificar e anotar manualmente *bounding boxes* para rotulagem supervisionada e depois treinamento e retreinamento de modelos de visão computacional com as mídias rotuladas. Para isso, empregou-se Python e o modelo de detecção de objetos YOLO, associadas a um fluxo de versionamento (DVC/Git) que permite realimentar o modelo conforme surgem correções manuais de classificação. O frontend foi desenvolvido utilizando Next.js e Node.js, proporcionando uma interface interativa e eficiente para interação com os modelos. Além disso, toda a arquitetura foi dockerizada utilizando Docker, garantindo portabilidade, escalabilidade e facilidade de implantação em ambientes industriais. O resultado é um sistema robusto que permite evolução dos modelos e datasets sem depender de soluções em nuvem, garantindo escalabilidade e adequação às necessidades industriais locais.

Palavras-chave: Classificação de Imagens, Visão Computacional, Versionamento de Modelos, Arquitetura de software, Arquitetura full-stack

Abstract

This project aims to establish a full-stack machine learning solution for computer vision algorithms, with versioning, reclassification, and model retraining capabilities, operating entirely over an Intranet to facilitate its use in industrial mining plants. This approach is designed to support any process, allowing users to upload media, select the labels they want to classify, and manually annotate bounding boxes for supervised labeling and then train or retrain computer vision models. The system leverages Python and the YOLO object detection model, combined with a versioning flow (DVC/Git) that enables retraining as manual classification corrections are implemented. The frontend was developed using Next.js and Node.js, providing an interactive and efficient interface for interacting with the models. Furthermore, the entire architecture is containerized using Docker, ensuring portability, scalability, and ease of deployment in industrial environments. The result is a robust system that allows the evolution of models and datasets without relying on cloud-based solutions, providing scalability and alignment with local industrial needs.

Keywords: Image Classification, Mining, Computer Vision, Model Versioning, Software Architecture, Full-Stack architecture

Resumo 4
Abstract 4
1. Planejamento e Fundamentação do Projeto 7
1.1 Origem do Desafio 7
1.2 Objetivo do Projeto 7
1.3 Escopo do Projeto 8
1.4 Pesquisa de Trabalhos Relacionados 10
1.4.1 Soluções para classificação de imagens 10
1.4.2 Soluções de Arquitetura de sistemas 15
2. Metodologia 18
2.1 Fases de desenvolvimento do projeto 18
2.2 Organização de atividades do projeto 20
3. Desenvolvimento 23
3.1 Definição do Escopo 23
3.2 Design - Pesquisa e Estrutura 27
3.3 Implementação 33
3.3.1 Implementação do frontend 33
3.3.2 Implementação do backend 41
3.3.3 Implementação do classificador 43
3.3.4 Implementação do DVC 44
3.4 Testes e Validações 45
3.5 Documentação 47
4. Resultados 48
5. Discussão dos Resultados e Próximos Passos 55
6. Referências Bibliográficas 56
7. Anexos 58
7.1 Questões Éticas e Profissionais 58
7.2 Normas Técnicas 58
7.3 Riscos Envolvidos 59
7.4 Recursos Necessários 60

7.5 Cronograma 61
7.6 Mapeamento dos Stakeholders 63
7.7 Conteinerização e Deploy 63
7.8 Guia de instalação do DVC 69
7.9 Guia de uso do DVC 74

1. Planejamento e Fundamentação do Projeto

1.1 Origem do Desafio

A ABB é uma empresa multinacional suíço-sueca, com sede em Zurique, Suíça, que atua nos setores de automação e eletrificação em diversas indústrias, incluindo mineração, automotiva, química, alimentícia e ferroviária. O presente projeto tem como foco as demandas da indústria de mineração, um setor no qual a ABB possui forte atuação em soluções voltadas para controle, eficiência e segurança de processos industriais.

O desafio deste projeto surgiu da necessidade da ABB em desenvolver uma arquitetura de visão computacional robusta, personalizável e capaz de operar integralmente dentro de uma Intranet industrial, sem depender de soluções em nuvem. A empresa já possui um sistema de monitoramento contínuo baseado em câmeras, denominado Camera Connect, utilizado em campo para diversas aplicações — como a verificação do uso de Equipamentos de Proteção Individual (EPIs), detecção de fumaça em chaminés, análise de flotação e avaliação da granulometria de minérios. No entanto, esse sistema atualmente realiza apenas inferência alarmes sobre as imagens captadas, e dispara alarmes sem fornecer mecanismos para reconfiguração, retreinamento ou validação dinâmica dos modelos utilizados.

A proposta do projeto Capstone é justamente preencher essa lacuna, integrando ao sistema atual uma plataforma web completa que permita ao usuário interagir diretamente com os modelos de visão computacional. O sistema desenvolvido possibilita o upload de imagens e vídeos capturados nas operações industriais, a anotação supervisionada com bounding boxes para correção de erros de classificação, o retreinamento personalizado dos modelos com controle de parâmetros como número de épocas, e o versionamento dos dados e modelos via integração com Git e DVC. Além disso, o usuário pode comparar diferentes versões de modelos por meio de métricas como mAP, precisão e recall, promovendo o melhor modelo a ativo para uso em campo.

Essa abordagem busca proporcionar um ciclo contínuo de melhoria, permitindo que engenheiros de automação e visão computacional colaborem diretamente na curadoria de dados e no aprimoramento de modelos. O sistema adapta-se melhor à planta industrial, com foco em facilidade de uso, manutenção autônoma e possibilidade de escalar a precisão dos modelos conforme as demandas e uso locais, sem comprometer os requisitos de conectividade e segurança do ambiente.

1.2 Objetivo do Projeto

O principal objetivo deste projeto é desenvolver uma arquitetura completa de visão computacional voltada para aplicações industriais, operando de forma totalmente integrada via Intranet. A proposta visa assegurar controle total, escalabilidade e eficiência no

armazenamento e na gestão dos modelos de visão computacional, bem como de seus respectivos datasets e processos de retreinamento, eliminando a necessidade de conexão com serviços de nuvem. Por fim, como forma de exemplificar a usabilidade da solução foi utilizado um modelo de classificação de EPIs (capacete, óculos, luvas e colete) previamente treinado.

1.3 Escopo do Projeto

O projeto abrange o desenvolvimento de uma solução full-stack de machine learning para visão computacional, projetada para operar via Intranet em ambientes industriais de mineração. A solução inclui o fluxo completo de operação de um modelo: upload de imagens e vídeos para composição de dataset (do zero ou como adição a datasets existentes), escolha de labels, classificação manual e treinamento do modelo. Além disso, os algoritmos estão incluídos em um sistema de versionamento e retreinamento, permitindo a melhoria contínua dos modelos com base em feedback manual. Além disso, sua arquitetura web foi desenhada para ser implementável no sistema de controle distribuído (DCS) da ABB, o 800XA, de acordo com as necessidades passadas pelo mentor técnico da empresa.

O escopo contempla a criação de uma infraestrutura modular e escalável, garantindo que a solução seja testável e confiável. Embora haja a possibilidade de se escolher as Labels e criar modelos do zero, a entrega atual se concentra na exemplificação do uso da solução para um modelo de visão com o treinamento mais avançado de detecção de EPIs.

O algoritmo desenvolvido como exemplo consiste na verificação do uso correto de Equipamentos de Proteção Individual (EPIs) em ambientes industriais, oferecendo maior segurança operacional e conformidade com regulamentações ambientais e trabalhistas.

Imagem 1- Exemplo de detecção de EPI

A arquitetura do sistema foi projetada para ser modular e expansível, facilitando sua adaptação para novos desafios industriais. O versionamento acoplado às soluções garante um fluxo contínuo de atualização dos modelos, automatizando o retreinamento com base em correções manuais e melhorando a precisão das soluções, com base nas novas imagens reclassificadas. O escopo apresenta os módulos que o cliente solicitou: módulo de upload de imagens e vídeos, solução de versionamento, solução de treinamento e retreinamento de modelos e ter todos esses módulos organizados em uma interface fácil de usar.

Apesar de o sistema proposto estar alinhado com as necessidades da empresa parceira e possuir compatibilidade conceitual com os sistemas de controle distribuído indicados pela ABB, o projeto não contempla o desenvolvimento nem a integração direta com hardware industrial, como o DCS 800xA. Essa etapa, por exigir acesso a infraestruturas específicas, protocolos industriais proprietários e validações complexas, foi considerada fora do escopo deste trabalho. Outras funcionalidades não abordadas envolvem a integração com sistemas de alarme automáticos e com sensores físicos, que dependeriam de infraestrutura externa e acordos operacionais.

1.4 Pesquisa de Trabalhos Relacionados

1.4.1 Soluções para classificação de imagens

A classificação de imagens com base em redes neurais artificiais tem ganhado destaque nos últimos anos, principalmente devido aos avanços no campo de deep learning. Nessa abordagem, modelos como as Convolutional Neural Networks (CNNs) são capazes de analisar uma imagem em diferentes níveis, extraindo padrões e características visuais de forma hierarquizada (LECUN et al., 1998). Esse processo viabiliza a identificação de objetos em diversos contextos, desde cenas mais simples até imagens de alta resolução adquiridas por satélites ou câmeras industriais (GOODFELLOW; BENGIO; COURVILLE, 2016).

Na prática, as CNNs executam convoluções sobre a imagem, identificando bordas, formas e texturas, o que permite a detecção e classificação de objetos específicos. Em problemas mais complexos de object detection, em que é preciso localizar onde o objeto se encontra dentro da imagem, surgiram métodos como o You Only Look Once (YOLO), que realiza a previsão em apenas uma etapa, tornando-se bastante rápido e aplicável a cenários de tempo real (REDMON; FARHADI, 2018). Versões mais recentes, como a YOLOv8, trazem melhorias tanto em precisão quanto em velocidade, empregando arquiteturas de backbone mais profundas e estratégias de ancoragem livre (anchor-free), permitindo a detecção de alvos menores e mais variados em resoluções maiores (TERVEN; CORDOVA-ESPARZA, 2023).

A aplicação dessas técnicas é ampla e diversa, conforme demonstrado em diferentes estudos recentes. Por exemplo, na pesquisa de Santos (2023), explora-se o uso de redes neurais para detecção de Equipamentos de Proteção Individual (EPIs) em tempo real. Nesse contexto, utiliza-se uma abordagem baseada em CNNs e versões da rede YOLO para identificar se os trabalhadores estão usando capacetes, óculos de proteção e luvas corretamente, auxiliando na automação do processo de fiscalização. Com isso, obtém-se maior rapidez, redução de custos e mais confiabilidade, pois o sistema pode funcionar 24 horas por dia e emitir alertas quando detecta alguém sem o devido EPI. Essa aplicação reflete diretamente os avanços discutidos no artigo investigado, que também aborda o uso de redes neurais para detecção de EPIs, destacando a importância do pré-processamento e da escolha da arquitetura da rede (NATH; BEHZADAN; PAAL, 2020).

Em outro estudo, Park et al. (2023) demonstraram a eficácia de uma combinação entre YOLOv8 e uma rede ResNet para regressão, aplicada à identificação e mensuração de chaminés em imagens de satélite de alta resolução. A partir de dados dos satélites coreanos KOMPSAT-3 e KOMPSAT-3A, os autores avaliaram o desempenho do modelo em diferentes países (Coreia do Sul e Tailândia), comparando também conjuntos de dados balanceados e desbalanceados. O resultado evidencia a capacidade de CNNs em lidar com problemas de variação de tamanho e formato dos objetos, além de demonstrar que imagens de melhor resolução podem aumentar a acurácia tanto na detecção quanto no cálculo de dimensões. Esse estudo complementa os achados anteriores, mostrando que a combinação de diferentes

arquiteturas, como YOLO e ResNet, pode ser uma estratégia poderosa para melhorar a precisão e a eficiência em tarefas específicas.

Além disso, pesquisas recentes têm explorado a integração de Transformers com redes convolucionais para superar limitações tradicionais. Um exemplo notável é a introdução da YOLOv5s-gnConv, que incorpora um mecanismo adicional de modelagem espacial baseado em autoatenção dot-product, inspirado pelas Transformers. Essa abordagem permitiu capturar informações de longo alcance de maneira mais eficiente, resultando em uma precisão de 97,32% e uma taxa de quadros por segundo (FPS) de 56,12. Esses resultados demonstram que a combinação de diferentes paradigmas, como CNNs e Transformers, pode levar a modelos mais rápidos e precisos, abrindo novas possibilidades para aplicações em tempo real, como a detecção de EPIs e outras tarefas de segurança.

Tais exemplos ilustram como a classificação de imagens apoiada em redes neurais convolucionais é bem-sucedida em distintos cenários, desde a segurança do trabalho até a gestão ambiental e controle de emissões. A flexibilidade das CNNs em aprender padrões complexos e a robustez de modelos como YOLO as tornam adequadas para aplicações onde se exige alta precisão e desempenho em tempo real. Além disso, conforme destacado no artigo investigado, a utilização de técnicas como Region of Interest (ROI) pode aprimorar ainda mais a eficiência desses sistemas, permitindo que as redes neurais concentrem seus esforços apenas nas áreas relevantes das imagens, reduzindo o processamento desnecessário e aumentando a precisão geral (CHARAYA; JINDAL; KAUR, 2016).

Escolher o modelo adequado de detecção de objetos pode ser uma tarefa desafiadora, isso porque o modelo que se adequa melhor a cada situação pode variar muito, dentro de um mesmo escopo.

Diferentemente de modelos baseados em Proposta de Região - técnica utilizada para detectar regiões de interesse dentro de uma imagem antes de classificá-las – a arquitetura do Yolo, desenvolvida pela Utralytics, realiza essas duas tarefas ao mesmo tempo. É por isso que o YOLO recebe possuí esse nome (YOU ONLY LOOK ONCE) e tem uma velocidade avançada, processando as imagens em tempo real. O YOLO se destaca pela sua capacidade de ter uma enorme velocidade de inferência sem comprometer significantemente sua acurácia, enquanto muitos modelos exageram na velocidade ou na precisão, o YOLO encontra um equilíbrio, (HI-TECH BPO).

Aplicações do YOLO:

- Gerenciamento de tráfego de veículos: A capacidade de processamento em tempo real do YOLO o torna excelente para detectar objetos no tráfego de veículos de uma cidade.
- Vigilância em tempo real: Sistemas de segurança utilizam YOLO para detecção de intrusos, acesso não autorizado e monitoramento de áreas restritas. A velocidade do YOLO garante que sistemas de segurança detectam e direcionam ameaças instantaneamente.

- Análise de esportes: mais um cenário em que a velocidade do YOLO o torna adequado para identificar padrões e estratégias.

O YOLO possui diferentes versões, sendo as mais recentes: v8, v9, v10, v11 e v12. De acordo com o canal de YouTube da própria Ultralytics, é difícil generalizar a comparação entre as diferentes versões, pois, em um caso específico, uma pode se adequar melhor do que outra, enquanto, em um cenário diferente, a escolha ideal pode mudar dependendo das necessidades e restrições do projeto. No entanto, o posicionamento da empresa sobre o teste em um cenário específico demonstrado no vídeo e, também, de uma forma mais generalizada foi:

- O YOLO v12 é construído a partir de seus predecessores, incrementando a acurácia enquanto mantém uma velocidade de tempo real. Em contrapartida, ainda possuí uma menor velocidade de inferência que as versões anteriores;
- O YOLO v11 não tem a mesma acurácia que o v8, especialmente para objetos menores, mas se o objetivo é detectar objetos grandes e obter a maior velocidade de inferência possível, ele pode ser a solução;
- O YOLO v11 é significantemente melhor que o YOLO v10;
- O YOLO v8 e o v9 são extremamente similares.

Outra ferramenta analisada foi o EfficientDet, um modelo inovador de detecção de objetos que enfatiza a eficiência sem comprometer a performance. Ele é construído sobre o método de escalonamento composto, o qual escala uniformemente a resolução, profundidade e largura de rede. Uma das inovações arquitetônicas do EfficientDet é a introdução de uma rede piramidal de recursos bidirecional ponderada (BiFPN). Essa arquitetura aprimora a capacidade do modelo detectar objetos de diferentes tamanhos. Outro recurso de destaque do EfficientDet é sua capacidade de prover uma performance robusta sem demandar muitos recursos, se estabelecendo como uma das principais escolhas para tarefas de detecção de objetos em cenários de vida real com recursos limitados por conseguir alcançar o um ótimo equilíbrio entre velocidade e acurácia, (HI-TECH BPO, 2025).

Aplicações do EfficientDet:

- Aplicações mobile com recursos computacionais limitados devido aos seus poucos parâmetros de treinamento. Pode-se utilizá-lo para detecção de objetos, aplicações de realidade aumentada e mais.
- Vigilância com o uso de drones: drones geralmente têm recursos computacionais limitados, portanto o EfficientDet pode ser uma solução otimizada.
- Sistemas Embarcados.

Em uma comparação da própria ultralytics do YOLO11 a frente do EfficientDet, os benefícios do EfficientDet são apontados como: elevada eficiência, escalabilidade e desempenho equilibrado, enquanto o YOLO necessita de mais recursos computacionais e seu desempenho pode variar com base em hardware específico e cenários de implantação. No entanto o YOLO apresenta uma melhor velocidade de inferência (ULTRALYTICS, 2025).

O SSD, ou Single Shot MultiBox Detector, é outro modelo de detecção de objetos revolucionário que opera usando uma única rede neural profunda. Ao contrário de detectores de duas fases, que geram propostas de objetos e depois classificam-nos, o SSD faz ambos de uma vez só, o que o torna altamente eficiente. O modelo garante uma detecção eficiente de objetos com diferentes tamanhos e formas, (HI-TECH BPO, 2025).

Aplicações do SSD:

- Vigilância em tempo real: Dado a sua capacidade de processamento em tempo real, o SSD é amplamente escolhido para sistemas de detecção de objetos em vigilância, garantindo segurança.
- Gerenciamento de Tráfego de Veículos: Em sistemas inteligentes de trânsito, o SSD pode ser uma boa escolha para detectar veículos, identificar violações de regras de trânsito e prever congestionamento.
- Aplicações interativas onde o feedback em tempo real é essencial.

O ponto forte deste modelo é sua habilidade de prover detecção de objetos em tempo real sem comprometer sua acurácia e sua capacidade de detectar objetos de tamanhos que variam.

Em suma, o SSD tem mais desempenho que o YOLO v8 no equilíbrio entre velocidade de inferência e acurácia. Apesar do YOLO v8 ganhar na velocidade, ele fica atrás do SSD quando se trata de acurácia, (KEYLABS.AI, 2023).

O Faster R-CNN incorpora uma Rede de Proposta de Região (RPN), a qual busca por regiões de interesse em uma imagem para que, depois, seja feita a classificação de objetos. Essa junção entre detectar regiões de interesse na imagem junto com a classificação de objetos faz com que este modelo trabalhe com uma eficiência excepcional, (HI-TECH BPO, 2025).

Aplicações do Faster R-CNN:

- Pesquisas: Devido as suas capacidades avançadas, o R-CNN é uma escolha popular para projetos de pesquisa.
- Vigilância baseada em Drones
- Cenários de alta acurácia e precisão: Em situações em que a acurácia e a precisão são imprescindíveis, como no setor da medicina, este modelo se torna uma das opções mais fidedignas.

Em comparação com modelos como YOLO v8 e SSD, o Faster R-CNN tem melhor desempenho na acurácia, porém ele fica atrás na velocidade e eficiência, além de requerer mais esforço computacional (KEYLABS.AI, 2023).

Mask R-CNN é a evolução do Faster R-CNN, feito para ir além de somente detecção de objetos. Enquanto o Faster R-CNN foca na predição de bounding boxes, o Mask R-CNN introduz um feature adicional dedicado a predição de máscaras de segmentação para cada

região e interesse (RoI). Este modelo detecta objetos na imagem de forma eficiente, ao mesmo tempo que gera segmentação de máscaras de alta qualidade, (HI-TECH BPO, 2025).

Aplicações do Mask R-CNN:

- Imagens no Setor Médico: Nesta área, a segmentação precisa é crucial. A habilidade deste modelo de prover uma segmentação detalhada torna-o excelente para tarefas como detecção de tumores e delineação de órgãos.
- Segmentação de Vídeos
- Segmentação de instância detalhada: Em cenários onde uma simples bounding box não é suficiente e uma segmentação detalhada é necessária, o Mask R-CNN brilha. Isso inclui aplicações em robótica e realidade aumentada.

O ponto forte deste modelo é a proposta de juntar a detecção de objetos com a segmentação de imagens, enquanto outros modelos apenas fornecem a detecção de bounding boxes, (HI-TECH BPO, 2025).

Retina net é um modelo de detecção de objetos que é construído a partir dos fundamentos do Faster R-CNN. O que o diferencia é a integração de uma funcionalidade de rede de pirâmide no backbone do modelo, permitindo-o processar imagens de múltiplas escalas ao mesmo tempo. Mas a verdadeira mudança significativa é a sua função de perda focal, criada para tratar o problema do desbalanceamento de classes que constantemente prejudica o desempenho de modelos em cenários onde a presença de uma classe é exorbitantemente maior que as outras. Em modelos onde a função de perda de entropia cruzada (cross-entropy) é utilizada, o desbalanceamento de classes é um impasse. Apesar de ter uma performance inferior a modelos como o YOLO e o EfficientDet no dataset COCO — um conjunto de dados famoso muito utilizado para comparar modelos entre si —, o RetinaNet se firmou pela sua característica função de perda focal. Outra vantagem do modelo é conseguir detectar objetos de tamanhos muito diferentes, desde veículos até pequenos defeitos com a mesma proficiência, (HI-TECH BPO, 2025).

Aplicações do RetinaNet:

- Manufatura: Quando o controle de qualidade é a meta, este modelo é uma boa solução para encontrar defeitos em produtos
- Imagens no setor médico: A proeza do modelo de detectar pequenos objetos o torna extremamente adequado para detectar tumores e lesões.
- Imagens de satélite

A geração de valor do RetinaNet está voltada para sua habilidade de detectar pequenos e grandes objetos com a mesma proficiência. O que o torna menos suscetível a ser escolhido é sua performance reduzida no geral, porém ele ainda pode ser aplicado para certos cenários, assim como foi supracitado, (HI-TECH BPO 2025).

Concluindo a pesquisa, pode-se dizer que, assim como mencionado na comparação entre as versões do modelo YOLO, apesar de existirem algumas equiparações generalizadas

sobre os modelos, a partir de treinamentos em datasets modelos, cada caso cenário pode ter uma solução que mais se adequa a ele. Com isso, o ideal é sempre concluir na prática qual modelo será utilizado. Em outras palavras, fazer, em cada cenário, um treinamento para cada modelo. Isso tornaria muito útil um recurso que permite a troca entre diferentes modelos na aplicação do projeto. No entanto, trata-se de uma tarefa muito complexa, que pode exigir mais tempo do que o disponível para o grupo. Sendo assim, como os cases apresentados pelo gerente técnico da ABB requerem tanto uma boa acurácia quanto uma velocidade de inferência, começar testando o YOLO v8 na aplicação pode ser um bom caminho.

1.4.2 Soluções de Arquitetura de sistemas

Agora nas pesquisas sobre a arquitetura do sistema, uma boa prática é manter o frontend (interface do usuário) desacoplado do backend (serviços de inferência Machine Learning). Essa separação de responsabilidades segue o princípio de que a separação de preocupações, permite que cada componente evolua e escale independentemente. Em aplicações de visão computacional com modelos pesados (como YOLO para detecção de objetos), delegar a inferência ao backend alivia o frontend de operações computacionalmente intensivas. Segundo Pankaj (2025), ao construir um serviço de detecção de objetos com YOLO, é recomendável ter um serviço dedicado apenas à interface do usuário e outro exclusivamente para o modelo de IA, o que isola a lógica de apresentação da lógica de processamento. Isso também viabiliza reutilizar o serviço de detecção em múltiplos projetos e dispositivos, reforçando a portabilidade e reuso da solução.

Do ponto de vista de manutenibilidade e escalabilidade, essa divisão é benéfica. Paguthaniya et al. (2024) destacam que a integração de modelos de ML em sistemas de produção traz desafios de escalabilidade e manutenção, exigindo planejamento cuidadoso na arquitetura. Ao manter o frontend e backend separados, a equipe pode atualizar a interface ou o modelo independentemente, facilitando a manutenção contínua sem interromper todo o sistema. Além disso, pode-se alocar recursos de hardware distintos para cada parte: o backend de inferência pode ser escalado horizontalmente (mais instâncias do serviço de ML) conforme a carga de processamento, sem precisar escalar também o frontend. Essa independência contribui para atingir baixa latência, pois o backend pode ser otimizado para desempenho e escalado conforme a demanda de inferência, ao mesmo tempo em que preserva uma experiência responsiva no frontend.

Para o frontend, em aplicações que envolvem entrada de vídeo ou imagem ao vivo e exigem monitoramento ou resposta instantânea (como detecção de objetos em tempo real), a utilização de WebSockets mostra-se uma abordagem superior ao HTTP tradicional. Diferente do modelo requisição-resposta do HTTP, o WebSocket estabelece uma conexão persistente bidirecional entre cliente e servidor, permitindo troca contínua de mensagens em baixa latência. Isso é especialmente útil em cenários de visão computacional contínua, porque em um sistema de detecção de objetos ao vivo, o WebSocket possibilita streaming contínuo

de frames de vídeo do cliente para o servidor para processamento, enquanto envia de volta os resultados das detecções quase instantaneamente, à medida que cada frame é analisado. Essa comunicação full-duplex elimina a sobrecarga de abrir múltiplas conexões HTTP e reduz o tempo ocioso, fundamental quando cada milissegundo conta para identificação em tempo real.

Trabalhos recentes comparam as duas abordagens e reforçam a adequação do WebSocket para esse propósito. Borgohain (2024) explica que, embora o HTTPS seja indispensável para transmissões seguras e possa lidar com envios individuais (como uma imagem para classificação), o WebSocket se destaca em cenários que demandam interação contínua e imediata, como análise de esportes ao vivo ou veículos autônomos recebendo streams de sensores. Na prática, a arquitetura proposta se beneficia disso ao monitorar uma fonte de vídeo, já que o frontend Next.js/React pode capturar quadros sucessivos e enviá-los via WebSocket ao backend FastAPI, onde o modelo YOLO processa cada quadro. Conforme cada inferência é concluída, o backend empurra pelo WebSocket os resultados de volta ao frontend, sem o cliente precisar consultar repetidamente. Um caso de uso documentado por Alpha2phi (2021) exemplifica essa dinâmica: ele implementou uma aplicação React que envia imagens capturadas da webcam através de um WebSocket para um servidor FastAPI rodando YOLOv5, recebendo de volta as predições em tempo real.

O framework FastAPI foi estudado para o backend devido ao seu desempenho e recursos adequados a aplicações de inferência de visão computacional e pela familiaridade da ABB pelo framework. Trabalhos técnicos recentes enfatizam que o FastAPI é um dos frameworks web Python mais velozes, comparável em performance a Node.js e Go, tornando-o excepcional para aplicações de machine learning em tempo real, que exigem alta taxa de requisições com baixa latência. Essa alta performance é atribuída ao suporte nativo a programação assíncrona (via async/await), o que permite tratar múltiplas requisições concorrentes de forma eficiente. Em um contexto como classificação de imagens ou detecção de objetos, onde cada requisição envolve processamento pesado de GPU/CPU, a capacidade de lidar com I/O e chamadas concorrentes é crucial para manter o throughput do sistema elevado.

Grandes empresas já demonstraram na prática os benefícios do FastAPI em sistemas de inferência. Um exemplo notável é o Uber Michelangelo, plataforma de ML do Uber, que adotou FastAPI para servir modelos de previsão em tempo real globalmente. De acordo com relato da empresa, o FastAPI foi selecionado por ser projetado com foco em alta performance e baixa latência, atendendo às rigorosas exigências de tempo de resposta de aplicações críticas. A arquitetura assíncrona do framework permitiu ao Uber manter serviços responsivos mesmo sob demanda massiva, servindo previsões de ML em milissegundos. Além disso, o FastAPI traz facilidades como documentação automática e integração com Pydantic para validação de dados, o que reforça a confiabilidade do sistema. No caso de um modelo YOLO servindo classificações/detecções, essas características se traduzem em benefícios diretos: capacidade de processar streams de imagens com rapidez, garantindo baixa latência para o usuário final; suporte a múltiplas conexões simultâneas sem degradação

significativa, útil em cenários de monitoramento com vários clientes; estruturação clara do código (via data models do Pydantic e rotas do FastAPI) facilitando a manutenção e futuras extensões da API. Em suma, conforme sintetiza Shokrzad (2023), o FastAPI alia desempenho de ponta a uma sintaxe moderna e fácil, posicionando-se como escolha ideal para desdobrar modelos de ML em produção de forma rápida, eficiente e segura.

A adoção do Docker no contexto dessa arquitetura foi motivada principalmente pela necessidade de padronizar, isolar e escalar os serviços de forma ágil, tanto durante o desenvolvimento quanto na produção. Como destacam Andrade (2016) e Da Silva (2016), uma das grandes vantagens do Docker é a capacidade de empacotar aplicações em contêineres leves, sem precisar replicar todo um sistema operacional para cada instância. Isso reduz significativamente o uso de recursos e diminui o tempo de inicialização. Em cenários com diversos micro serviços, essa abordagem facilita bastante o trabalho colaborativo, já que cada equipe consegue rodar localmente o mesmo ambiente que será usado em produção (Gomes; Souza, 2015).

Além disso, projetos que dependem fortemente de dados – como treinamento ou retreinamento de modelos – se beneficiam muito do Docker. Ele garante que as bibliotecas e dependências de cada módulo estejam corretamente configuradas e versionadas ((Romero, 2016). Turnbull (2014)) destaca que o Docker também simplifica o versionamento e a distribuição de imagens, o que se alinha perfeitamente com práticas de integração e entrega contínua (CI/CD). Com isso, o fluxo de trabalho fica mais eficiente: qualquer alteração em um serviço pode ser empacotada numa imagem Docker, testada em um ambiente igual ao de produção e, só depois, lançada com segurança (Veras; Carissimi, 2015).

Por fim, a leveza e portabilidade dos contêineres Docker – especialmente quando comparados a máquinas virtuais tradicionais – permitem executar os serviços em diferentes infraestruturas e facilitam o monitoramento via orquestradores externos. Essa flexibilidade é fundamental para lidar com picos de demanda, como no caso da classificação ao vivo, onde câmeras podem gerar cargas imprevisíveis. Outro ponto importante é que componentes individuais podem ser atualizados ou substituídos sem afetar o sistema como um todo (Vitalino; Castro, 2016). Assim, o Docker atende às necessidades de isolamento dos micros serviços, além de garantir rapidez na entrega e consistência nos ambientes de desenvolvimento e produção.

2. Metodologia

2.1 Fases de desenvolvimento do projeto

Para garantir organização, visibilidade e coerência no progresso do trabalho, o desenvolvimento do projeto foi estruturado em fases distintas, como exemplificado na imagem 2. Essa divisão segue um fluxo comum de gestão de projetos, permitindo que cada etapa tenha objetivos claros, entregas esperadas e métricas de acompanhamento. As fases foram planejadas de forma sequencial e iterativa, favorecendo a tomada de decisões conforme a evolução do entendimento técnico e das necessidades da empresa parceira.

Imagem 2 – fases de desenvolvimento do projeto

A fase de Entendimento do Escopo será dedicada à investigação das necessidades da empresa e à identificação dos possíveis casos de uso da solução. Inicialmente, será explorado um escopo amplo, conduzindo testes preliminares com modelos de visão computacional e avaliando diferentes abordagens. Também serão planejadas interações com a empresa para alinhar expectativas e compreender as restrições de infraestrutura, permitindo uma definição mais precisa dos requisitos do projeto.

A fase de Design terá como objetivo estudar e definir a melhor forma de estruturar a arquitetura da solução. Serão analisadas diferentes tecnologias para viabilizar o desenvolvimento do projeto em todas as suas frentes, explorando alternativas e suas vantagens e desvantagens de acordo com o escopo.

Na fase de Implementação, serão programados os módulos conforme arquitetura escolhida. Será seguido um modelo baseado na metodologia Kanban, permitindo acompanhar o progresso e distribuir tarefas conforme as especialidades da equipe. O desenvolvimento será realizado de forma iterativa, com testes constantes e estudo de novas viabilidades conforme problemas ou novas ideias forem surgindo.

A fase de Teste e Validação Final será destinada à garantia de que a solução funcione conforme os requisitos definidos e possa ser demonstrada como uma solução integral. Pretende-se validar o sistema por meio de testes unitários e iterativos, testes de integração e demonstrações práticas. Também está no planejamento apresentar um protótipo funcional à banca avaliadora e ao representante da empresa, visando coletar feedbacks para refinamentos e ajustes finais antes da entrega final do projeto.

Por fim, a fase de Documentação será estruturada para registrar todos os detalhes do projeto, assegurando que a solução possa ser facilmente compreendida, replicada e aprimorada no futuro.

2.2 Organização de atividades do projeto

Imagem 3 – WBS do projeto

Para organizar e estruturar as atividades do projeto, foi inicialmente elaborado um Work Breakdown Structure (WBS) (imagem 3), que serviu como base tanto para o planejamento estratégico quanto para a execução das tarefas técnicas. O WBS foi elaborado com foco nos principais módulos funcionais da aplicação, como a captura ao vivo da câmera, reclassificação de imagens, retreinamento do modelo, exibição de logs, seleção de versões e dockerização. Cada um desses blocos foi decomposto em subtarefas específicas, contemplando desde o design da interface até a implementação de rotas e integração com ferramentas como DVC e Docker. A granularidade do WBS nos permitiu visualizar claramente o escopo técnico de cada funcionalidade, além de facilitar a transição dessas etapas para o quadro Kanban do Teams.

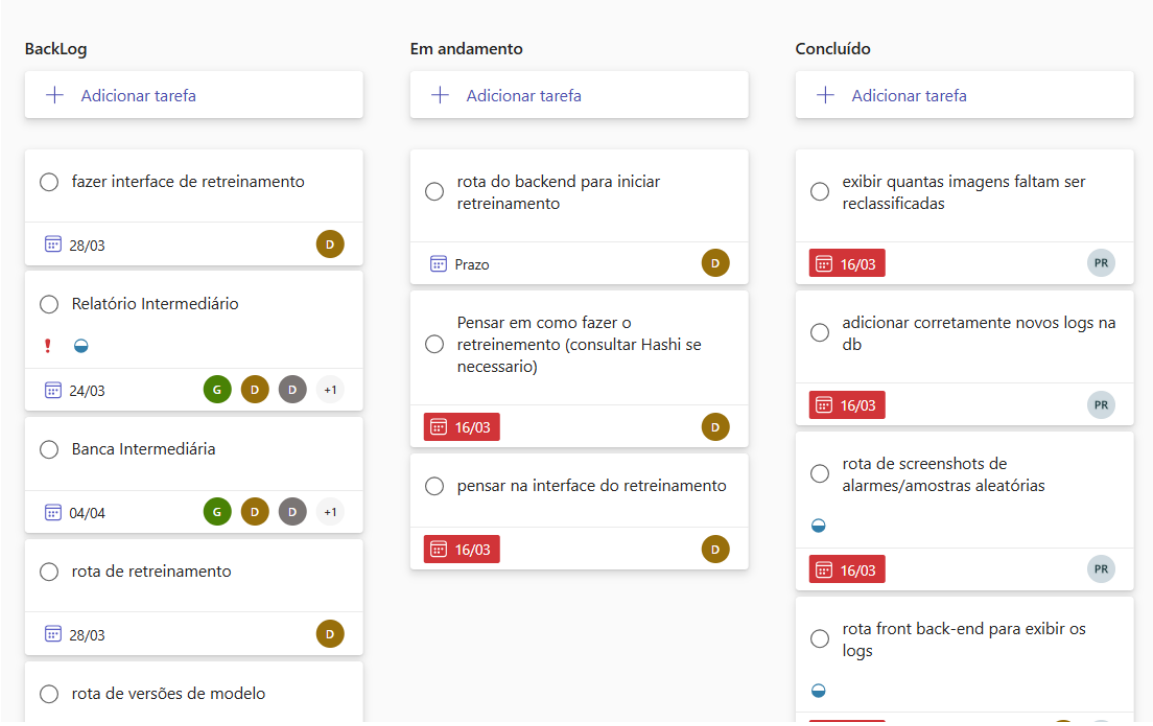


Imagem 4 – Organização do Kanban na interface do Teams

A organização das tarefas foi baseada nas principais áreas macro do sistema definido, dividido em tarefas menores complementares. Além disso, o GitHub foi usado como repositório de código e versionamento, enquanto a comunicação entre os membros da equipe foi facilitada por mensagens no WhatsApp e chamadas de vídeo, tanto internas quanto com a empresa, pelo Microsoft Teams.

Para organizar e acompanhar o progresso das atividades do projeto, foi adotado um quadro Kanban, estruturado em três colunas principais: Backlog, Em andamento e Concluído. A coluna Backlog foi utilizada para listar todas as tarefas planejadas, mas ainda não iniciadas, funcionando como um repositório de atividades futuras. Já a seção Em andamento concentrava as tarefas que estavam sendo desenvolvidas naquele momento, permitindo visibilidade sobre o que cada integrante da equipe estava executando. Por fim, a coluna

Concluído armazenava todas as tarefas finalizadas, servindo como registro histórico do progresso obtido ao longo do tempo.

Cada funcionalidade importante do projeto — como, por exemplo, criar a rota de upload de imagens no backend, implementar a tela de retreinamento no frontend ou configurar o container do banco de dados PostgreSQL — foi transformada em um cartão no Kanban. Esses cartões continham descrições detalhadas da tarefa, subtarefas específicas para orientar o desenvolvimento, possíveis dependências com outras atividades e os responsáveis designados por sua execução.

3. Desenvolvimento

3.1 Definição do Escopo

No primeiro contato com a empresa, foi apresentada a infraestrutura de automação industrial que já está em uso nas plantas atendidas pela ABB, o sistema ABB 800xA (Extended Automation). Trata-se de uma plataforma de automação distribuída (DCS – Distributed Control System) que integra controle de processos, instrumentação, sistemas de segurança (Imagem 5), comunicação e interfaces operacionais em uma única arquitetura digital. Seu grande diferencial está na capacidade de consolidar diferentes sistemas e dados operacionais em uma visão única e acessível para os operadores, facilitando o monitoramento em tempo real e a tomada de decisão.

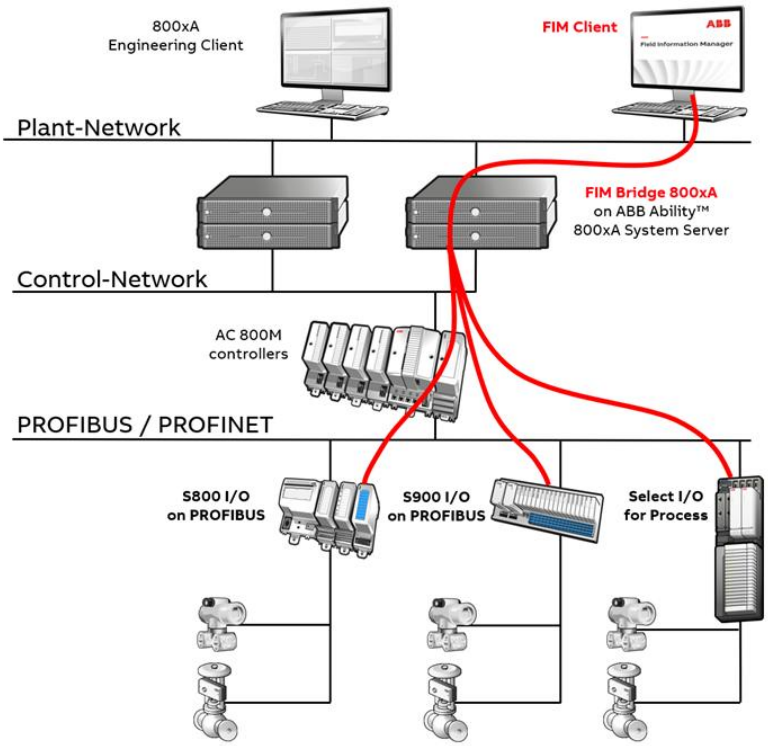


Imagem 5 – Estrutura de implementação do 800xA

Dentro dessa plataforma, foi apresentado o módulo Camera Connect, que amplia as funcionalidades do 800xA ao transformar câmeras industriais — originalmente instaladas para vigilância — em sensores inteligentes, com capacidade de realizar análises de imagem em tempo real (video analytics). Esse módulo utiliza algoritmos de visão computacional para gerar alertas e acionar automaticamente funções do sistema de controle, como alarmes, parada de motores ou gravação de trechos específicos. Como ilustra a Imagem 6, essa inteligência descentralizada pode ocorrer tanto localmente quanto em nuvem, e permite diversas

aplicações práticas, como: Detecção de EPIs em áreas restritas; Monitoramento de temperatura via imagem térmica; Análise de bolhas no processo de flotação de minérios; Análise granulométrica de rochas em esteiras de transporte.

Imagem 6 – Possíveis usos do Camera Connect

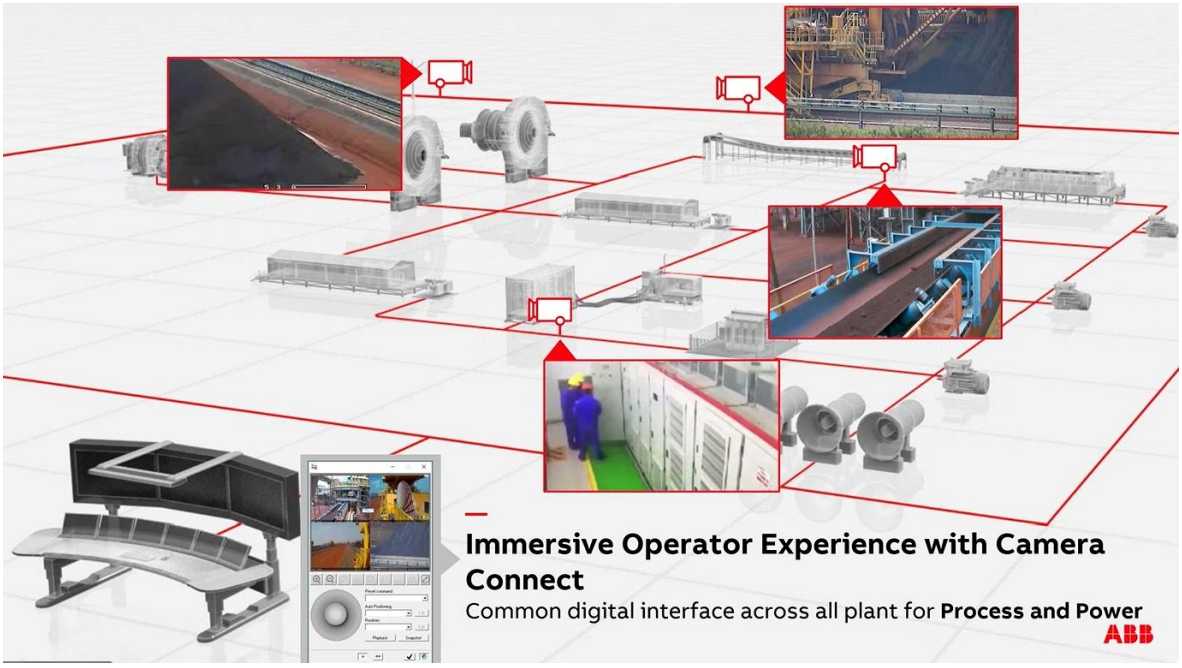


Imagem 7- Exemplo da Estrutura do Câmera Connect na planta de controle

A Imagem 7 ilustra como essas câmeras estão distribuídas ao longo das unidades industriais, cobrindo diferentes pontos críticos do processo produtivo. Essas imagens são enviadas ao Camera Connect Server, que se comunica com o sistema 800xA para gerar alertas e eventos. Também é possível visualizar as imagens diretamente no painel do operador, de forma imersiva, dentro da mesma interface digital onde os controles do processo são gerenciados.

Esse sistema representa, portanto, uma base sobre a qual o nosso projeto foi idealizado. A proposta do nosso trabalho é desenvolver uma solução modular que possa se acoplar a esse ecossistema, permitindo o ciclo completo de coleta, reclassificação, retreinamento e versionamento dos modelos de visão computacional — tudo isso respeitando a restrição de funcionamento em rede local (Intranet), que é uma exigência padrão em ambientes industriais por questões de segurança.

Assim, chegou-se à reunião seguinte trazendo dois protótipos iniciais: um modelo de detecção de fumaça em chaminés (usando um dataset criado a partir do recolhimento de imagens da internet, com técnicas de aumentação implementadas) e outro de EPIs (baseado em dataset aberto, porém mais extenso). Essas demonstrações ajudaram a empresa a enxergar a viabilidade dos modelos e serviram como ponto de partida para refinar o que seria do projeto.

A partir das reuniões seguintes, o escopo foi gradualmente refinado, definindo a necessidade de um sistema de classificação em cascata (feedback e reclassificação manual), e com isso a realimentação das imagens reclassificadas para o dataset. Surgiu também a demanda de um sistema local (sem internet), uma vez que, por mais que o Camera Connect tenha suporte para um serviço de Cloud, muitas plantas não possuem conexão de internet estável e requerem uma segurança maior de dados e equipamentos, mantendo-os em uma rede local. A possibilidade de retreinamento por lote de imagens foi outro tópico importante, reforçando a necessidade de controlar versões de dataset e modelo.

Com esse escopo definido, três personas/usuários foram desenhados como na imagem 8. Três perfis principais interagem com o sistema: o engenheiro de visão computacional da ABB, o engenheiro de automação da ABB e o alunos do PFE.

Engenheiro de Visão Computacional da ABB: responsável pelo desenvolvimento, treinamento e ajuste dos modelos de análise de imagem. Atua na melhoria contínua dos algoritmos de detecção, garantindo que o sistema entregue resultados consistentes e relevantes para o contexto industrial.

Engenheiro de Automação da ABB: responsável por integrar os resultados da análise de imagem ao sistema de controle da planta (como o 800xA). Define ações automáticas com base nas saídas do modelo, como alarmes, comandos de start/stop e registro de eventos.

Alunos do PFE: grupo responsável pelo desenvolvimento e evolução da plataforma como parte do projeto acadêmico. Atuam na implementação de funcionalidades, testes e documentação do sistema, além de colaborar com os engenheiros da ABB para validar soluções.

O processo tem início com o engenheiro de visão computacional, que atua diretamente sobre o módulo de classificação. A partir desse módulo, ele realiza o treinamento dos modelos de machine learning. Uma vez treinados, os modelos passam por uma etapa de validação em laboratório. Caso os resultados ainda não sejam satisfatórios, o fluxo retorna ao módulo de classificação para ajustes e refinamentos. Quando os testes de laboratório são concluídos com sucesso, os modelos são então levados para validação em campo, onde são testados em condições reais de operação.

Uma vez que a aplicação está disponível, ela passa a ser utilizada pelo engenheiro de automação da ABB, que opera a aplicação diretamente no ambiente industrial, utilizando-a para monitorar os fluxos de produção em tempo real. Ele acompanha as detecções realizadas pelo modelo de visão computacional, valida os alarmes emitidos e consulta os logs de eventos. Seu foco está na utilização prática e operacional da ferramenta, garantindo que ela se integre ao contexto produtivo e atenda aos requisitos de segurança e eficiência da planta.

Além disso, os estudantes atuam tanto como usuários quanto como desenvolvedores da solução. Eles interagem com o sistema por meio do fluxo de retreinamento e classificação, realizando anotações manuais para alimentar o dataset, monitorando a performance dos modelos e iniciando retreinamentos conforme necessário. Além disso, são responsáveis por aprimorar a infraestrutura técnica do projeto, implementar novas funcionalidades, corrigir eventuais falhas e garantir a evolução contínua da plataforma. Seu papel é essencial para a manutenção técnica e expansão da solução, além de fornecer suporte aos engenheiros da ABB quando mudanças são necessárias.

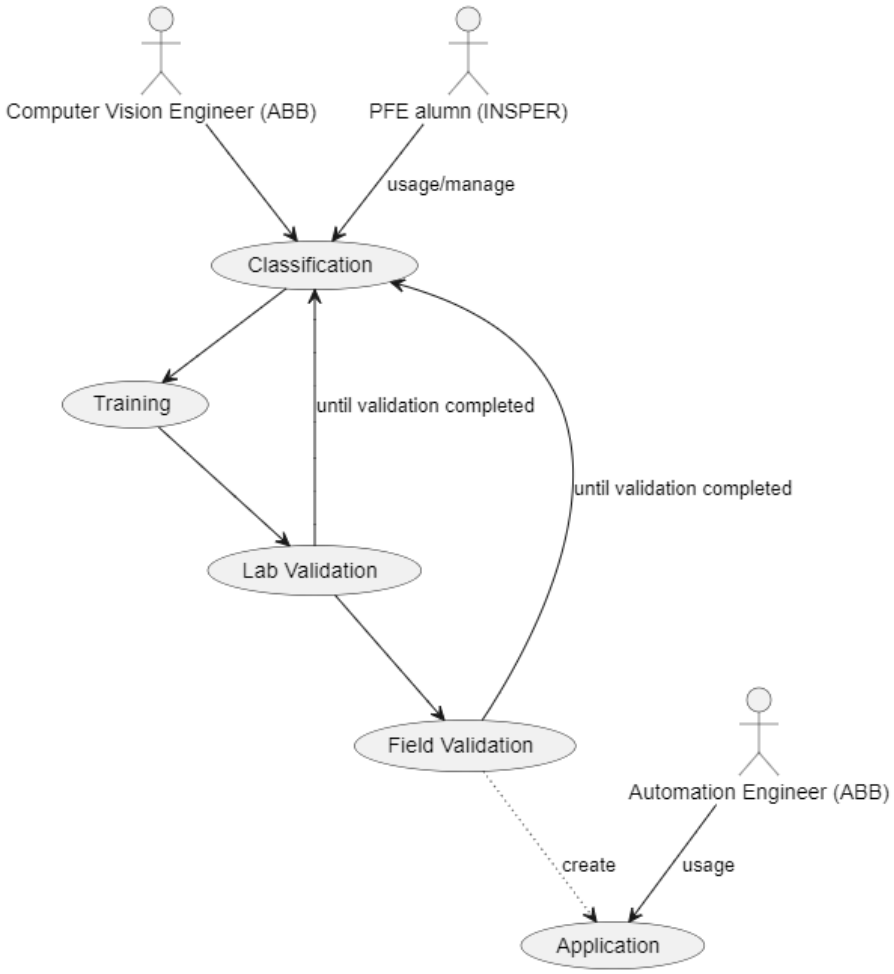


Imagem 8 - Fluxo de uso e desenvolvimento do sistema

3.2 Design - Pesquisa e Estrutura

Diante do avanço do escopo e com as várias definições tomando forma, foram necessárias pesquisas para que a escolha das ferramentas que compõem nossa arquitetura fosse feita da forma mais acertada possível.

Durante as discussões técnicas, a equipe, junto a indicação da empresa parceira concluiu que uma API em Python seria a forma mais viável de receber e processar as requisições de inferência e retreinamento. O FastAPI demonstrou ser leve, de fácil implementação e fornecer suporte necessário para a validação de dados. Assim, a integração com o frontend é fácil e se alinha com a direção dada pela empresa, que usa a mesma ferramenta em suas aplicações industriais.

Com a decisão de manter a aplicação acessível via navegador em intranets industriais e com ela podendo ser implantada em formato web, foi escolhido o desenvolvimento uma interface web de fácil acesso e formatação. O Next.js, baseado em React, mostrou-se vantajoso pela renderização híbrida (SSR/SPA), boa organização de rotas e suporte a integrações com bibliotecas de UI e styling, como o TailWind. Isso favoreceu um desenvolvimento mais rápido e a criação de páginas mais dinâmicas.

Em nossa pesquisa sobre modelos de detecção de objetos (YOLO, EfficientDet, SSD, Faster R-CNN, etc.), descrita na parte 1.11 chegou-se à conclusão de que o YOLOv8 apresentava um equilíbrio desejado entre velocidade de inferência e acurácia. Apesar de haver outras bibliotecas com vantagens pontuais (como maior precisão ou menor custo computacional), o YOLOv8 se mostrou suficientemente versátil para atender os cenários definidos pelo grupo.

Para a curadoria dos datasets, dada a possibilidade do upload de imagens, caso o usuário estiver aplicando técnicas de data augmentation, como rotações, espelhamentos ou alterações de brilho, são geradas novas imagens que são variações diretas de uma imagem original. Se uma dessas variações for colocada no conjunto de treino e outra no conjunto de validação, o modelo pode acabar "memorizar" padrões específicos dessa imagem, resultando em uma validação artificialmente otimista, um fenômeno conhecido como data leakage. Para mitigar esse risco, o sistema implementado exige que todas as variações de uma mesma imagem (como cat_aug1.jpg, cat_aug2.jpg) compartilhem o mesmo prefixo (cat) e sejam agrupadas durante o split dos dados. Assim, todas as versões daquela imagem serão atribuídas ao mesmo conjunto (treinamento ou validação), mantendo a independência entre os dados usados para treinar o modelo e os usados para testá-lo.

Para armazenar o histórico dos datasets utilizados no treinamento de modelos, tornou-se necessário buscar uma solução mais eficiente para lidar com grandes volumes de dados. Embora o Git seja amplamente utilizado para o versionamento de código, sua aplicação no controle de versões de datasets pode não ser a alternativa mais adequada, pois pode resultar em lentidão extrema ou até mesmo em falhas no sistema devido ao grande tamanho dos arquivos.

Diante desse desafio, foi identificado o DVC (Data Version Control) como uma solução otimizada para o versionamento de datasets. O DVC utiliza um mecanismo de hashing MD5 para rastrear as versões dos arquivos, garantindo um controle eficiente das alterações.

Após uma análise detalhada e comparação entre diversas ferramentas de versionamento de datasets, como Neptune.ai, Delta Lake, Git LFS, Pachyderm, Dolt e lakeFS, chegou-se à conclusão de que o DVC (Data Version Control) é a solução mais adequada para o projeto. A decisão foi fundamentada em vários fatores que tornam o DVC a opção mais vantajosa, especialmente em termos de facilidade de implementação, flexibilidade e a presença de uma interface de linha de comando (console UI), o que facilita a integração no fluxo de trabalho de desenvolvimento da aplicação.

O DVC se destaca pela sua simplicidade de uso, permitindo o versionamento e o controle eficiente de datasets sem exigir uma curva de aprendizado complexa. Em contraste, as outras ferramentas apresentam desvantagens significativas que poderiam impactar negativamente o progresso do projeto, como:

- O Delta Lake exige o uso de um formato de dados dedicado, o que o torna menos flexível e não agnóstico aos formatos atuais.
- O Git LFS não é feito para escalar e requer servidores dedicados para ser utilizado.
- O Pachyderm requer uma curva de aprendizado muito complexa para ser utilizado, além de ser difícil de integrar na infraestrutura de um projeto.
- O Dolt ainda é uma solução imatura em comparação com as outras, assim como o lakeFS.
- O Neptune.ai não possui interface de usuário de console.

Para viabilizar a implantação local nas plantas industriais dos clientes da ABB e propiciar uma comunicação mais simples e modular entre os serviços, decidiu-se containerizar cada componente, backend, frontend e modelo, utilizando o Docker. Com essa abordagem, cada serviço é executado de forma independente em um contêiner, o que simplifica consideravelmente a configuração e diminui conflitos de dependências entre as bibliotecas utilizadas. Além disso, o uso de contêineres favorece a manutenção e amplia a escalabilidade do sistema: novos módulos, como um contêiner adicional de OpenCV para processamento específico de imagens, podem ser incorporados ao ecossistema sem impactar o restante da aplicação. Por fim, a conteinerização permite uniformizar a criação dos ambientes de desenvolvimento e produção, assegurando que todos os micros serviços, desde o repositório de dados até o modelo de detecção, estejam configurados de forma idêntica em diferentes máquinas ou redes industriais.

Como grande parte da aplicação exige informações sobre imagens recebidas, reclassificações manuais, logs de alarme e versões do modelo, a equipe optou por um banco de dados relacional confiável e de fácil manutenção local. O PostgreSQL atende bem a esse cenário, pois suporta funções avançadas, tem boa performance e integra-se facilmente ao FastAPI com bibliotecas de ORM (Object-Relational Mapping), como, no caso do projeto, o SQLAlchemy.

A abordagem geral — FastAPI no backend, Next.js no frontend, YOLOv8 como modelo de detecção, DVC para versionar dados e Docker para implantação — portanto, foi decidida com base nos requisitos refinados durante a descoberta do escopo, e quando feito um benchmarking com ferramentas parecidas, se destacaram como as melhores opções.

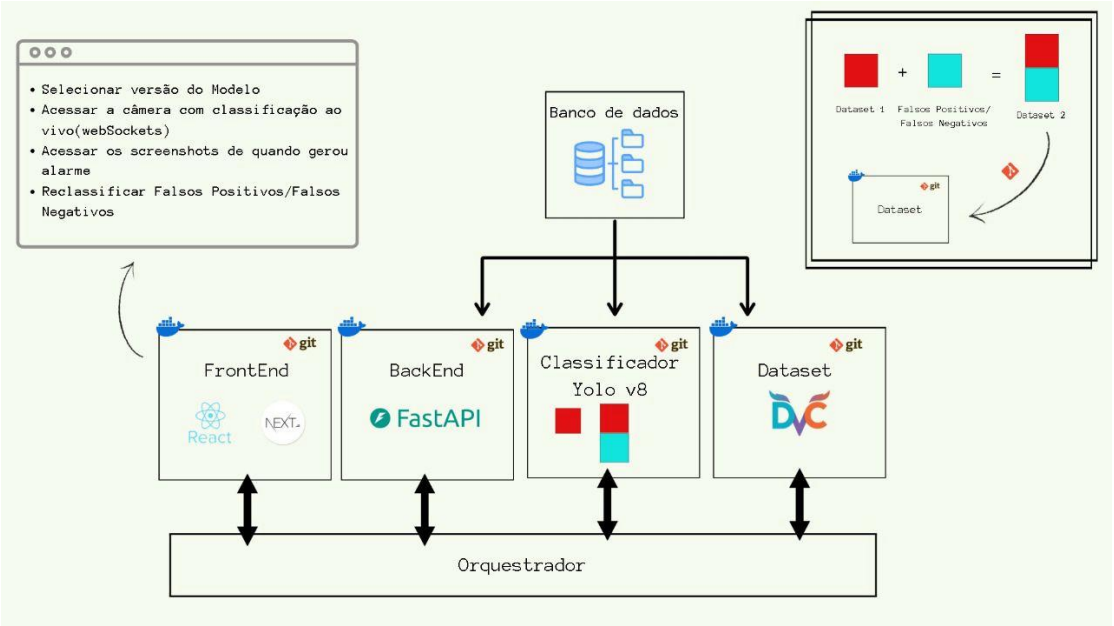


Imagem 9 – Esquemático da arquitetura completa

Em suma, a arquitetura projetada (ilustrada na imagem 9) reflete as escolhas que foram feitas para atender às exigências técnicas e às demandas operacionais do projeto. No frontend, Next.js foi usado para construir uma interface web acessível via intranet, na qual o operador pode visualizar a câmera em tempo real, reclassificar imagens e acompanhar o retreinamento. O backend, desenvolvido em FastAPI, age como intermediário entre o frontend e os serviços de retreinamento, recebendo requisições de upload, inferência e logs, além de se comunicar com o banco de dados PostgreSQL para armazenar informações sobre as imagens e o histórico de operações. O modelo de detecção escolhido, o YOLOv8, roda em um contêiner específico, possibilitando inferência e, futuramente, retreinamento local.

O sistema oferece quatro opções distintas de treinamento:

1. Treinamento incremental com dataset fixo: treina-se um novo modelo a partir dos pesos de uma versão anterior, utilizando exatamente o mesmo dataset que foi usado na versão anterior do modelo.

2. Treinamento incremental com dataset expandido: o modelo é treinado a partir dos pesos de uma versão anterior, utilizando um dataset atualizado que expande o conjunto de dados usado no treinamento anterior. Essa nova versão do dataset inclui imagens recentemente classificadas e/ou reclassificadas, incorporando novos exemplos que aprimoram a capacidade do modelo de generalizar e corrigir inferências incorretas observadas anteriormente.

3. Treinamento do zero com base model e dataset anterior: utiliza-se o modelo base do YOLOv8, sem herança de pesos anteriores, treinando-o com um dataset já existente no sistema.

4. Treinamento do zero com base model e dataset expandido: parte-se do modelo base do YOLOv8 e um dataset expandido com novas imagens classificadas e/ou reclassificadas, selecionado pelo usuário.

Caso uma reclassificação seja feita, o novo rótulo é registrado no banco de dados PostgreSQL, o que permite ao usuário visualizar o histórico de reclassificações — essencial para compreender as falhas do modelo ao longo do tempo. Paralelamente, as imagens classificadas e reclassificadas são incorporadas ao conjunto de dados gerenciado pelo DVC (Data Version Control), garantindo o rastreamento rigoroso de versões dos datasets e dos modelos ao longo do ciclo de vida do sistema. O esquema da imagem 10 é um resumo do fluxo de uso do usuário e foi no que o grupo se baseou para a implementação completa do sistema.

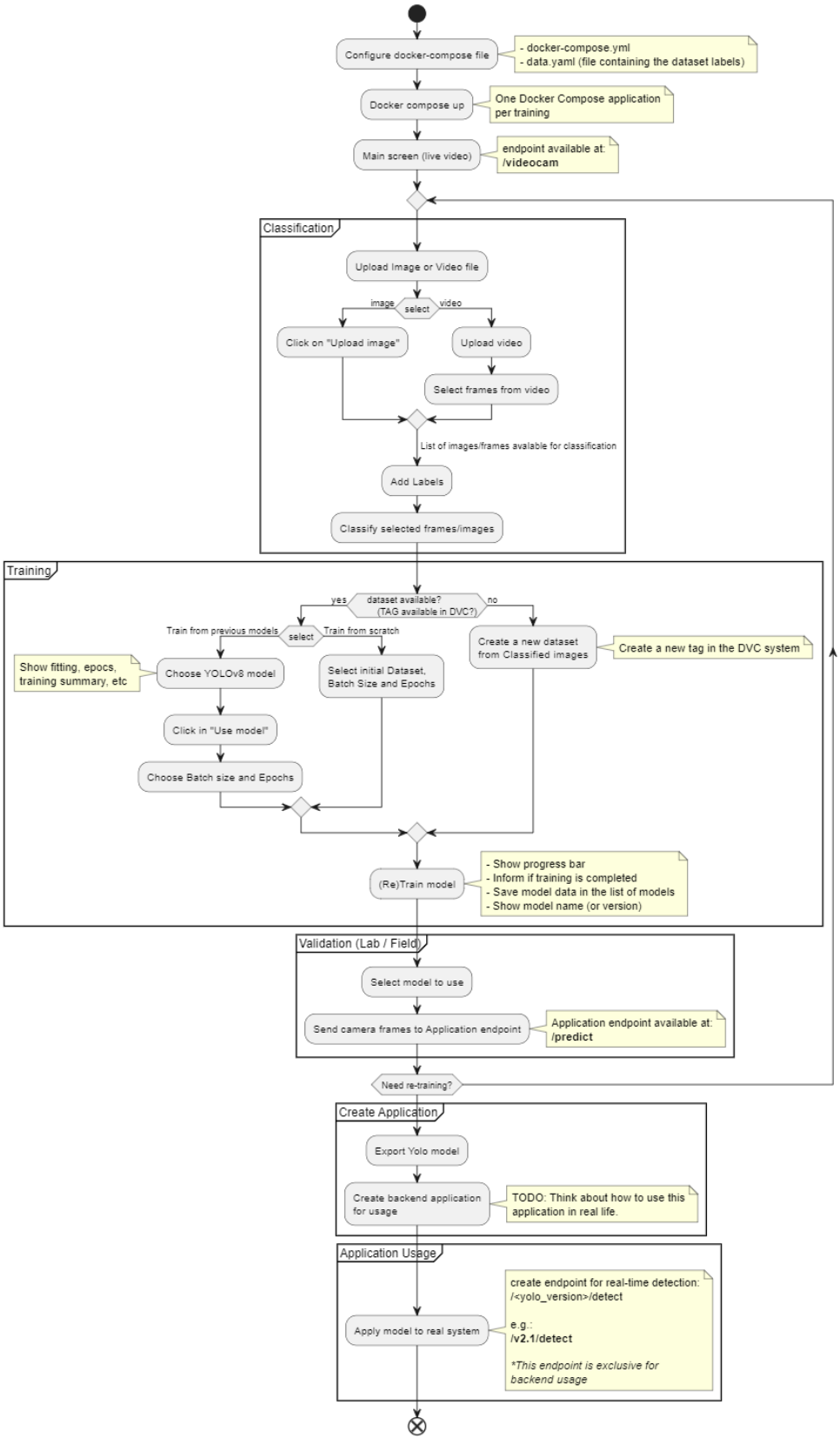


Imagem 10 – Fluxograma da jornada do usuário

3.3 Implementação

3.3.1 Implementação do frontend

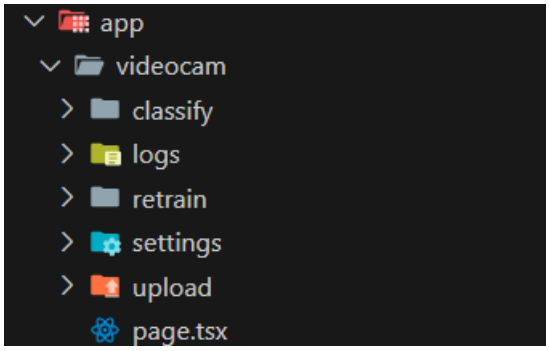


Imagem 11 – Diretório do frontend

O front-end foi desenvolvido em Next.js 14 (React + Typescript) e organizado por funcionalidades: cada tela possui a sua pasta com páginas, componentes reutilizáveis, hooks e testes. O estilo visual é unificado com Tailwind CSS e componentes shadcn-ui/Radix-UI, mantendo a paleta oficial da ABB (vermelho #e31f26, cinzas neutros) e garantindo responsividade nativa.

A organização do código segue uma estrutura clara (imagem 11), onde cada funcionalidade é encapsulada em sua própria pasta. Por exemplo:

- /videocam: Responsável pela exibição de vídeo e execução do modelo em tempo real.
- /classify: Gerencia a reclassificação manual de imagens capturadas.
- /upload: Lida com o upload de imagens e vídeos para classificação assistida.
- /logs: Exibe os registros de reclassificações realizadas.
- /retrain: Interface para o re-treinamento de modelos.
- /settings: Gerencia as versões de modelos e datasets disponíveis.

A interface /videocam exibe o stream ao vivo da câmera industrial usando WebSocket. O vídeo é processado pelo modelo YOLO, que retorna bounding boxes sobre objetos detectados. O botão “toggle” ativa ou desativa dois tipos de capturas automáticas: Alarm: captura um frame quando o modelo detecta um capacete com probabilidade > 80%. Random: captura frames aleatórios a cada minuto. Essas imagens são armazenadas em diretórios locais (screenshots/ e

screenshots_w_bboxes/) e enviadas para posterior reclassificação manual na interface /classify. A imagem 12 mostra a interface e o botão toggle circulado em verde.

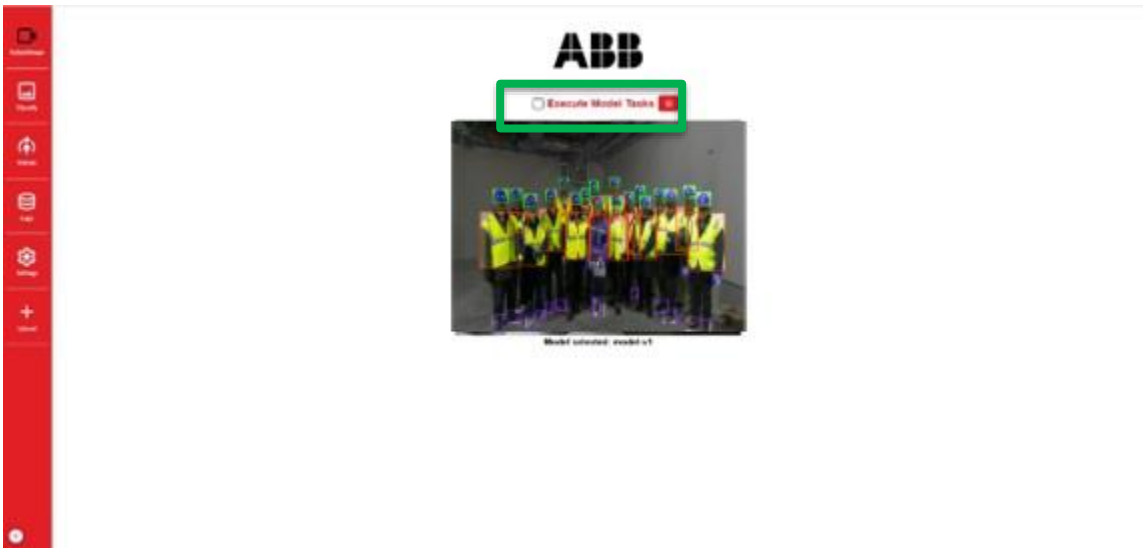


Imagem 12 – Página /videocam

Quando o "toggle" está ativo as capturas automáticas (Alarm / Random) ficam disponíveis na página de /classify para o usuário reclassificá-las manualmente. As ações de reclassificação são enviadas para o back-end, que registra os dados em reclassify_logs (PostgreSQL) e salva imagem e anotação no diretório versionado pelo DVC.

Caso não haja imagens disponíveis para reclassificação, o usuário é avisado da falta de imagens – "Whenever the model detects an unwanted situation, a screenshot of the moment will be taken. Additionally, to monitor the model's predictions, a random screenshot will be captured periodically." (imagem 13).

Imagem 13 – Página /classify quando não há imagens para reclassificação

Em /upload o operador inicia o fluxo de classificação assistida: há abas “Images” e “Videos”; para imagens, pode-se selecionar as imagens dos arquivos do computador e visualizá-las em uma grade de miniaturas (Imagem 14). Para vídeos, o usuário seleciona os vídeos (Imagem 15) e depois seleciona os frames que serão classificados (Imagem 16) e, quando confirmado, eles são apresentados em miniaturas.

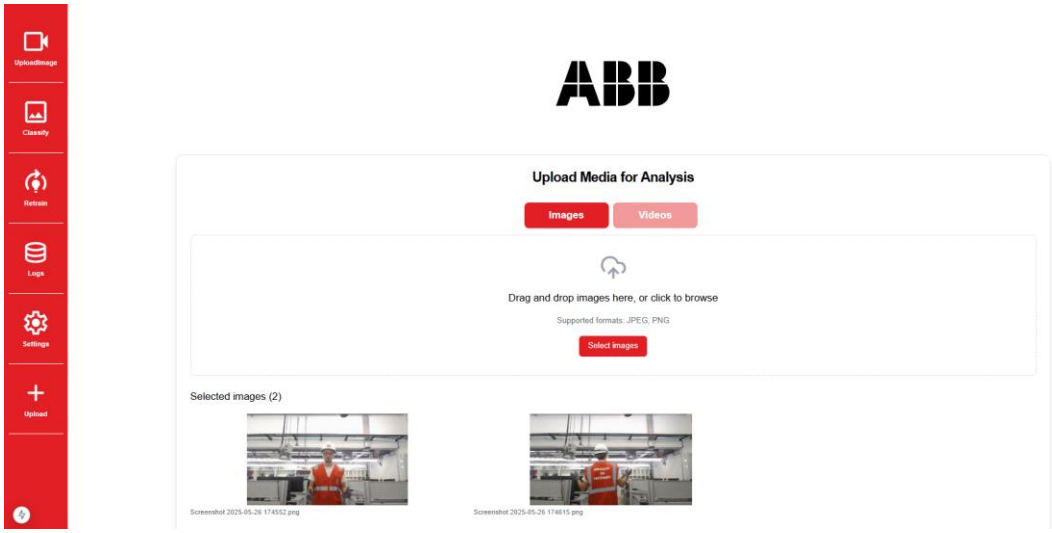


Imagem 14 – Página /upload com algumas fotos selecionadas

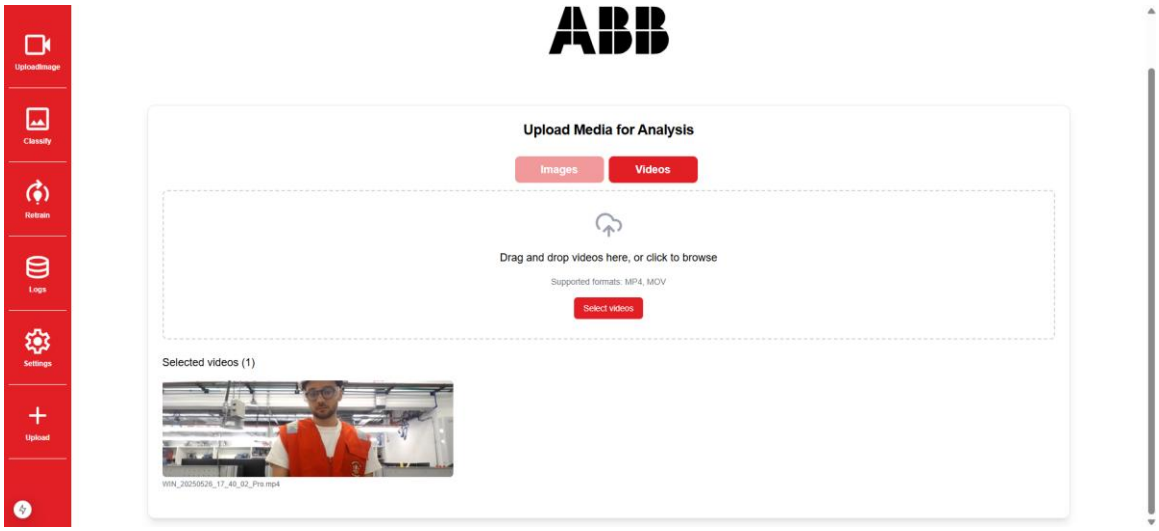


Imagem 15 – Página /upload com um vídeo selecionado

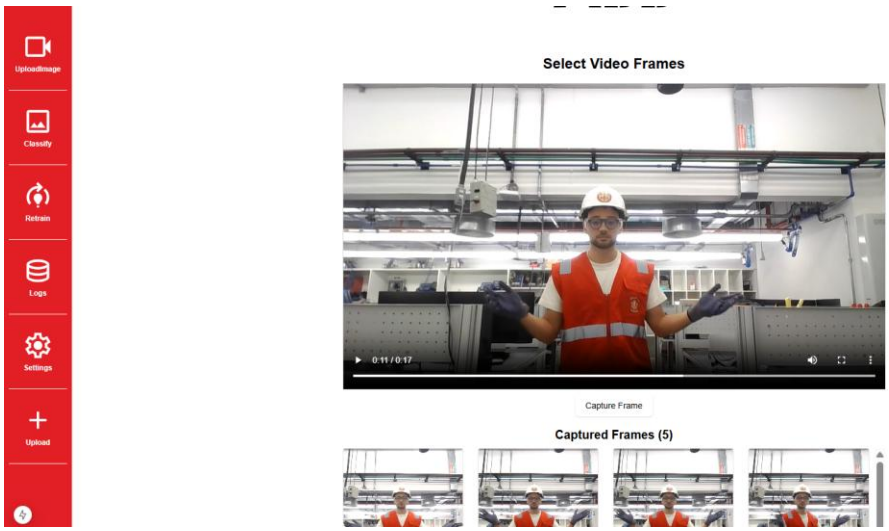


Imagem 16 – Página /upload selecionando os frames do vídeo

Quando o usuário abre o BoundingBoxAnnotator (Imagem 17), ele adiciona as Labels que quer utilizar para classificar as imagens, desenha caixas e pressiona Save. O componente converte as anotações para o formato YOLO (class, x_center, y_center, w, h) e envia um JSON {route, filename, image_base64, annotation} para o endpoint /upload_annotation. O mesmo fluxo é feito quando há imagens disponíveis na página classify.

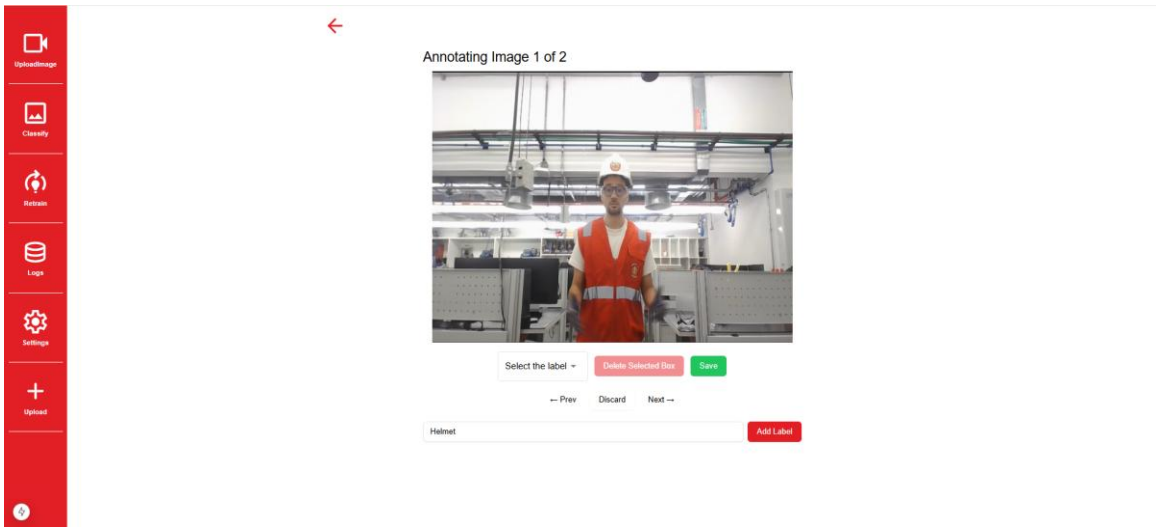


Imagem 17 – Página de BoundingBox Annotator para a classificação manual das imagens ou frames selecionados

Cada reclassificação gera um registro na tabela reclassify_logs do PostgreSQL e copia o par imagem/rotulagem para o diretório versionado pelo DVC. A página /logs consome esses dados via SWR e mantém-se atualizada em tempo real graças a um canal WebSocket; as colunas mostram timestamp, labels e se o caso foi falso-positivo ou falso-negativo, com link para reabrir o anotador (Imagem 18).

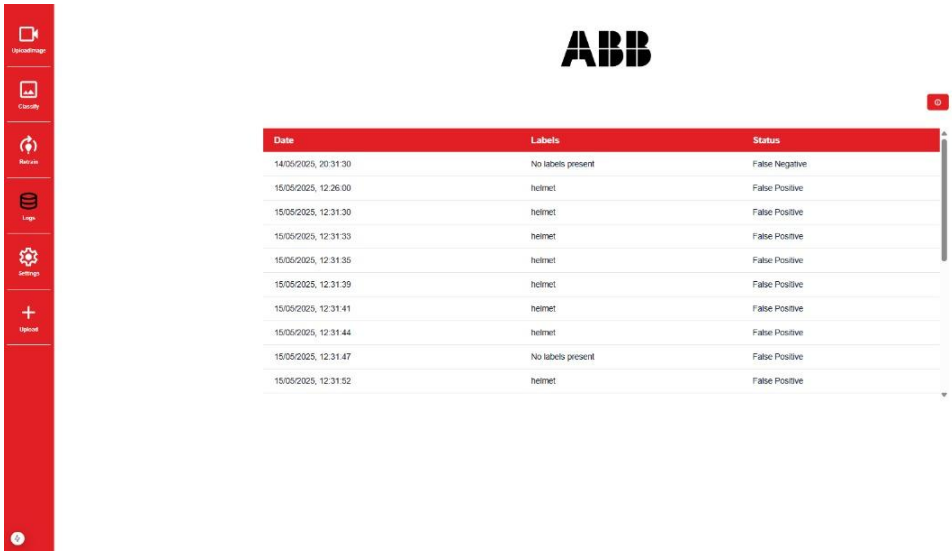


Imagem 18 – Página de /logs

Já em /retrain o operador primeiro define se irá treinar modelos já existentes ou um modelo do zero (Imagem 19). Caso decida treinar modelos já existentes define qual modelo irá usar, podendo visualizar informações sobre o desempenho do modelo: Best Train Loss, Best Val Loss, Best mAP50 e o dataset usado, além do gráfico de perdas por épocas treinadas (Imagem 20).

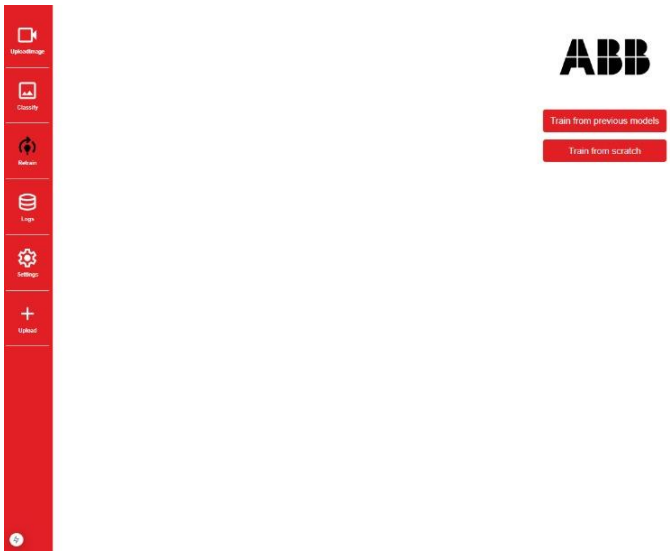


Imagem 19 – Página de /reclassify para escolher modo de treino

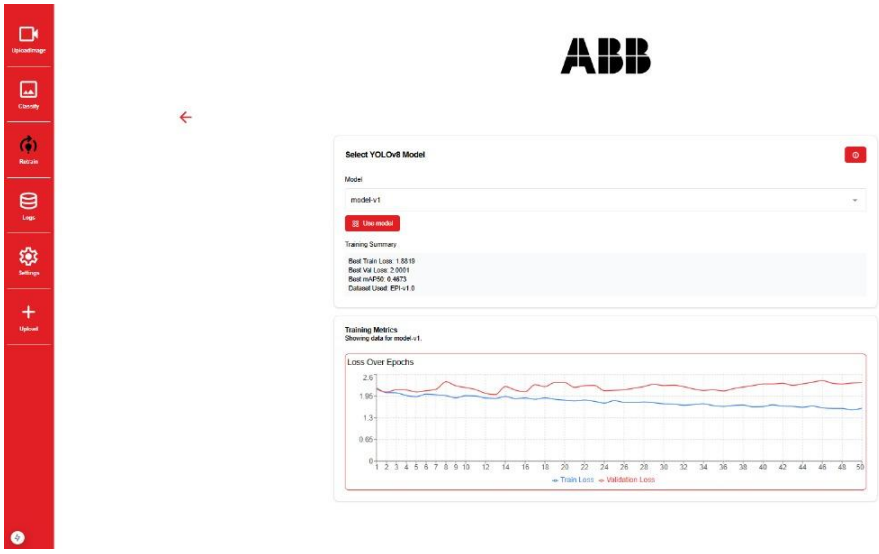


Imagem 20 – Página de seleção de Modelos

Depois de escolher o modelo já existente ou decidir por treinar um do zero, o operador passa para a seleção de dataset. Ele pode usar um dataset existente (Imagem 21) ou usar um novo (Imagem 22), que consiste em adicionar um certo de número de imagens, alterando o batch-size, considerando o número de imagens disponíveis.

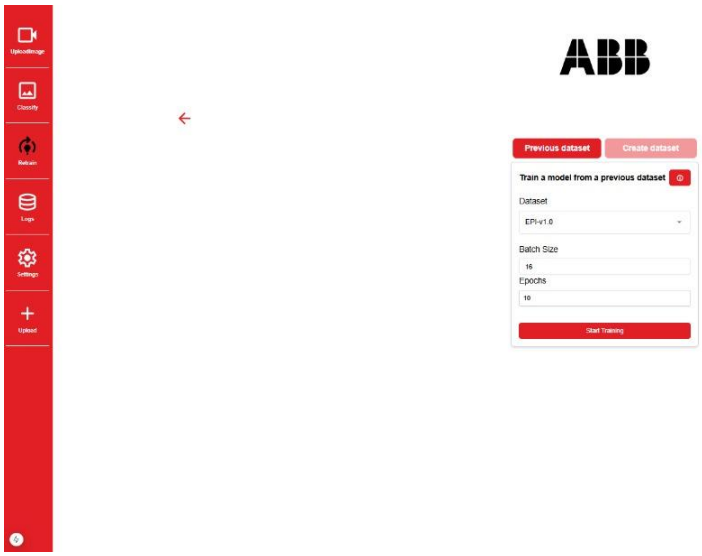


Imagem 21 – Página de seleção de Dataset já existente

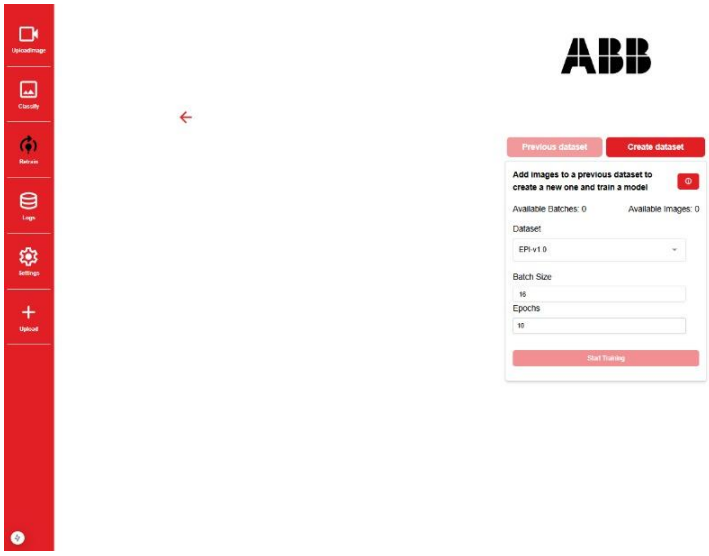


Imagem 22 – Página de seleção de Dataset já existente

Assim, selecionado o batch size e o número de Epochs, ao clicar retreinar o front-end dispara um POST /train_model_new. Durante o treinamento, um segundo WebSocket envia {epoch, loss, val_loss}, atualizando uma barra de progresso e um gráfico de curvas de perda/validação; ao término chega o status “completed”.

Na página de /settings são listadas todas as versões dos modelos armazenadas no DVC, exibindo métricas agregadas (mAP, precision, recall) lidas de best_model_metrics.json. Nela, o usuário pode ver se o modelo está em uso, baixar o modelo ou exclui-lo (Imagem 23).

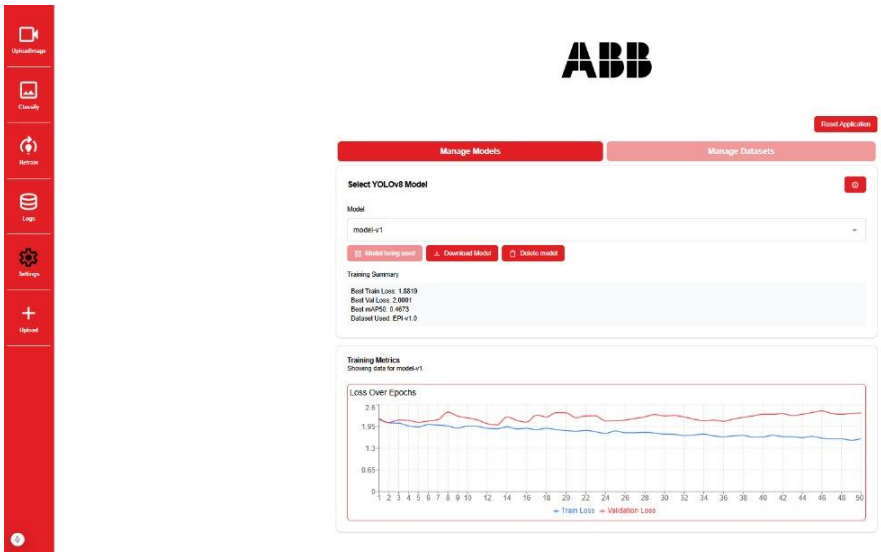


Imagem 23 – Página de /settings com as informações das versões dos modelos

A aba ao lado mostra os datasets utilizados pelo operador, buscando as tags de retornadas de /dataset_version, e permitindo o download ou remoção de datasets (Imagem 24).

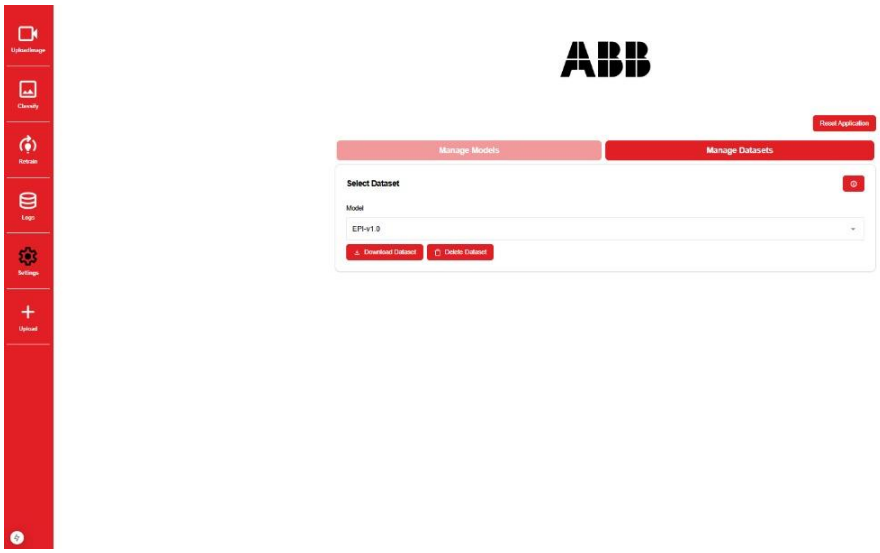


Imagem 24 – Página de /settings com as versões dos datasets

Além de todas essas implementações, o sistema conta com algumas caixas de diálogo em que caso clicadas dispõe de uma explicação exclusiva daquela função. Na imagem 25, é descrita a função de upload de imagens e a particularidade sobre upload de imagens aumentadas, já que como lê-se na caixa: ''Caso esteja enviando imagens aumentadas, certifique-se de que elas seguem o padrão de nomenclatura indicado para evitar vazamento de dados entre os conjuntos de treino e validação. Cada imagem aumentada deve ter o mesmo prefixo da original e terminar com _aug1, _aug2 e assim por diante. Exemplo: cat.jpg, cat_aug1.jpg, cat_aug2.jpg. Dessa forma, o sistema entenderá que esses arquivos são variações da mesma imagem e os tratará adequadamente durante o treinamento. Após o upload das imagens, clique

em “Continue to BBox Annotations” para desenhar as caixas delimitadoras (bounding boxes) nas regiões de interesse de cada imagem.’’

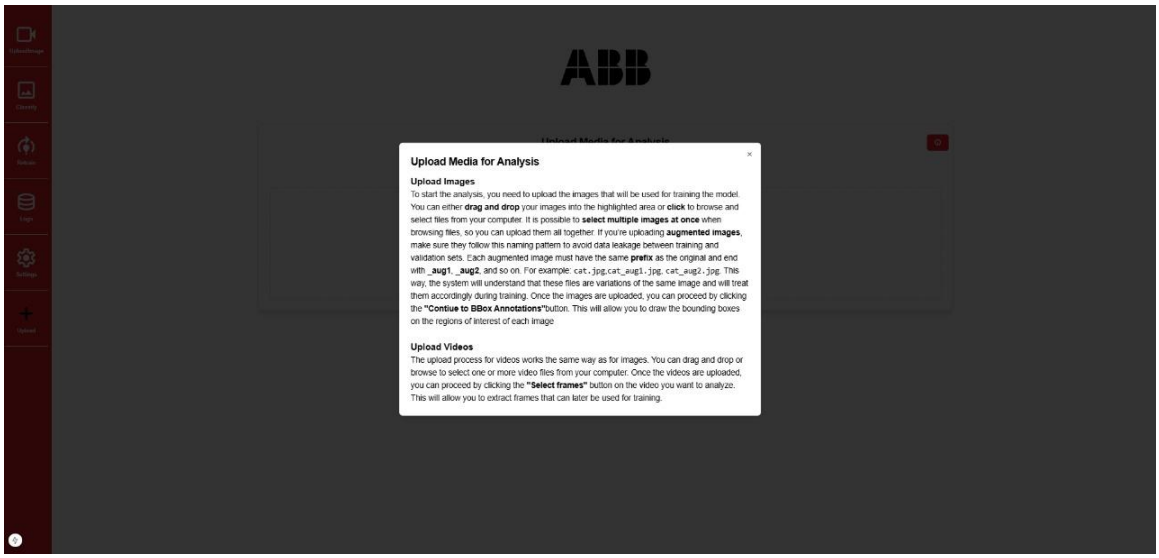


Imagem 25 – Caixa de diálogo da página de upload de imagens/vídeos

3.3.2 Implementação do backend

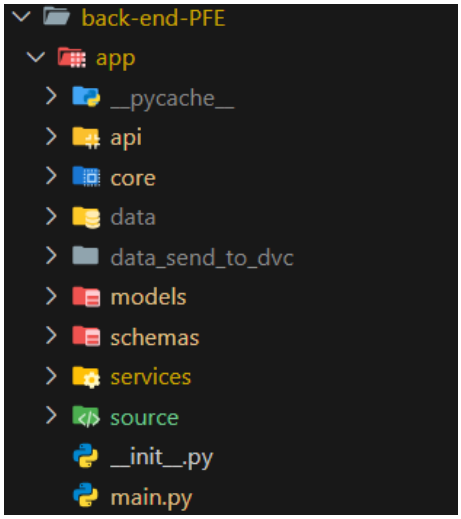


Imagem 26 – Diretório do backend

O backend foi desenvolvido em FastAPI, framework escolhido por aliar alto desempenho assíncrono, geração automática de documentação OpenAPI e integração nativa

com o sistema de validação do Pydantic. A aplicação organiza-se em quatro camadas lógicas: routes, schemas, services e core. Essa separação garante baixo acoplamento: as rotas expõem somente a interface HTTP/WebSocket; as regras residem em services; os objetos de dados (DTOs) vivem em schemas; infraestrutura de persistência e inicialização está em core.

A pasta api/routes agrupa nove arquivos, cada qual responsável por um domínio funcional. Em image.py reside a parte de transmissão em tempo real. A função process_frame captura quadros da webcam com OpenCV; se o arquivo current_model.json indicar um peso ativo, o YOLOv8 é carregado e executa inferência diretamente sobre o frame. As bounding boxes e rótulos são desenhados e o resultado é convertido para Base64, sendo transmitido continuamente pelo WebSocket mantido por websocket_endpoint. Nessa mesma função, duas regras governam a persistência de screenshots: quando o modelo detecta "helmet" com confiança > 0,80 é considerado um cenário de alarme e de minuto em minuto, para formar uma amostragem aleatória que permita identificar falsos negativos. O estado que habilita ou desabilita essas capturas é alterado pelo endpoint /update_checkbox_status, permitindo controle dinâmico a partir do frontend.

Em logs.py encontra-se o mecanismo de rastreamento de reclassificações. O endpoint reclassified_image interpreta o sufixo do arquivo (_ALARM ou _RANDOM) para classificar se o erro do modelo foi falso-positivo ou falso-negativo. Os rótulos são extraídos via expressão regular e o registro é armazenado na tabela reclassify_logs.logs, definida em models/reclassify_logs.py. Esse log, consultável pelo endpoint /logs, é fundamental para monitorar a qualidade do modelo ao longo do tempo.

Os arquivos models.py e set_version.py controlam o ciclo de vida dos pesos treinados. A primeira lista todos os diretórios model-v* existentes, permite ativar um deles gravando o nome em current_model.json e ainda remove versões antigas sob demanda. Já set_version.py sincroniza código e dados para uma tag específica: executa git checkout <tag> e, em seguida, dvc checkout <tag>, de modo que o backend, o dataset e o modelo permaneçam sempre alinhados à mesma revisão.

O versionamento de datasets é encapsulado em version.py: um endpoint devolve todas as tags Git/DVC existentes e outro elimina uma versão, realizando o cleanup tanto no repositório Git quanto no remoto DVC. Dessa forma, a equipe pode auditar ou descartar rapidamente conjuntos de dados obsoletos.

O arquivo mais extenso, retrain.py, implementa o ciclo MLOps completo. O endpoint /send_data_to_dvc varre as pastas de screenshots e anotações, agrupa arquivos originais e aumentados, embaralha os conjuntos e separa-os em 80 % treino / 20 % validação. Depois move cada item para a árvore exigida pelo YOLO (train/images, train/labels, valid/...) dentro do repositório de datasets, atualiza dinamicamente ao data.yaml e executa a sequência dvc add, git add, git commit, git tag e dvc push, garantindo que cada rodada de dados fique registrada de forma reprodutível. Já os endpoints /train_model_new e /train_model_previous disparam o script train.py com os hiperparâmetros enviados pelo operador. A chamada é feita em background com asyncio.create_task, e o progresso é consultado via /training_status. Quando a

execução termina, a API grava o CSV de resultados e o JSON de métricas para que o frontend exiba curvas de perda, mAP, precisão e recall.

A gestão de screenshots fica a cargo de screenshots.py. A rota /screenshots produz um dicionário em que a chave é o nome base da captura e os valores são URLs para as versões com e sem bounding boxes, além de um contador total. Operações de leitura, deleção e upload de novas anotações completam o ciclo: no upload, a imagem ou anotação recebida em Base64 é decodificada e copiada para as pastas data_send_to_dvc/images e data_send_to_dvc/annotations, preparando-a automaticamente para o próximo retreinamento.

No core/database.py. Ao iniciar a aplicação, a função init_db consulta um arquivo control_db.json: se não houver banco provisionado, cria um novo database PostgreSQL com sufixo incremental (abb_db_1, abb_db_2, …). Em seguida, monta o schema reclassify_logs e gera as tabelas definidas nos modelos SQLAlchemy. O generator get_db() é injetado via dependência em cada rota que precise de sessão

A camada schemas padroniza todos os payloads de entrada e saída. ImageRequest, RetrainYoloPayload, ReclassifiedImage, ScreenshotsResponse e outras classes Pydantic fornecem validação automática, garantindo que erros de tipagem sejam detectados na borda do sistema, antes de chegar às regras de negócio. Com isso, ao chamar um endpoint como /train_model_new, o desenvolvedor tem a certeza de que num_epochs é inteiro, model_version é string e campos opcionais foram devidamente tratados.

Os services concentram utilidades reutilizáveis. image_service.py oferece conversões entre PIL e Base64; data_dir_service.py cria a estrutura de diretórios na fase de startup, evitando falhas de I/O em produção; user_service.py isola a lógica de criação de usuários, permitindo que a rota user.py se limite a expor a operação HTTP. Esse padrão de Service Layer facilita testes unitários e manutenção futura.

Por fim, o arquivo main.py reúne todos os routers sob o prefixo /api/v1, aplica CORS para permitir chamadas do frontend e define uma função lifespan que, no boot da aplicação, inicializa o banco, garante a existência dos diretórios de dados e screenshots e registra o schema no PostgreSQL.

3.3.3 Implementação do classificador

O módulo Classificador completa o ciclo iniciado pelo backend, transformando as anotações já versionadas no DVC em novos pesos de rede prontos para produção. Quando o usuário clica em “Start Retraining”, o backend despacha, de forma assíncrona, uma chamada para o script train.py, localizado em Classificador/Scripts. Esse disparo acontece dentro da rotina run_training, em retrain.py, que reúne os argumentos recebidos da interface – número de épocas, tamanho de lote, versão do dataset e modelo de partida – e cria um novo processo Python sem bloquear o servidor FastAPI.

O script train.py começa determinando qual será o ponto de partida para os pesos. Se o operador optou por “scratch”, ele faz o download do artefato público yolov8n.pt; caso contrário, carrega o best.pt do último modelo selecionado, armazenado em Classificador/models/<nome>/weights. Em seguida, inicializa o objeto YOLO da biblioteca Ultralytics e chama o método train, passando como entrada o caminho do data.yaml que o backend acabara de atualizar com os diretórios train e val. O treinamento roda no modo projeto ../models, e o helper get_next_model_version garante que a pasta de resultados siga o padrão incremental model-vN, evitando colisões quando vários retreinamentos ocorrem em sequência.

Concluídas as épocas, o script remove arquivos supérfluos – checkpoints intermediários, gráficos de curva e pesos redundantes – por meio da função clean_directory. Mantêm-se apenas três artefatos: o best.pt, o results.csv gerado automaticamente pela Ultralytics e o best_model_metrics.json. Esse último é construído pela função generate_metrics_json, que realiza a leitura do results.csv com pandas e calcula a “fitness” de cada época com base ponderada nas métricas metrics/mAP50(B) (peso 0.1) e metrics/mAP50-95(B) (peso 0.9). Essa fórmula segue a recomendação da própria Ultralytics, conforme descrito no issue #14137 (ULTRALYTICS, 2024), privilegiando a generalização do modelo além do mAP@50. A linha de melhor fitness é então selecionada com. idxmax(), e suas métricas de train_loss, val_loss, mAP_50 e dataset_used são serializadas como JSON.

Terminada a fase de treino, a lógica de versionamento do backend aproveita o diretório já limpo. Se o operador aprovar a nova versão, o endpoint /set_model grava o nome da pasta em current_model.json. Cada vez que o WebSocket de vídeo reinicia, ele lê esse arquivo e carrega o best.pt correspondente, garantindo que a próxima inferência ao vivo utilize o modelo recentemente aprovado. Enquanto isso, o dataset que alimentou o treino permanece rastreado pelo DVC sob a tag criada no passo /update_dvc, o que assegura reprodutibilidade plena: por meio de git checkout <tag> e dvc checkout é possível reconstruir exatamente o conjunto de dados e o modelo de qualquer ciclo anterior.

3.3.4 Implementação do DVC

Para garantir rastreabilidade e reprodutibilidade dos dados e dos pesos de modelo dentro da intranet da planta, todo o pipeline de visão computacional foi acoplado ao Data Version Control (DVC). O DVC reproduz o fluxo de versionamento do Git para arquivos volumosos – imagens, rótulos e checkpoints – sem exigir serviços em nuvem, exatamente como recomenda o guia corporativo adotado pelo time.

O repositório dvc-ABB é inicializado com dvc init, criando a pasta oculta .dvc/ onde vivem o cache MD5 e os arquivos de lock. Em seguida adicionou-se um remote local, hospedado em um volume de rede interno; assim diferentes instâncias do backend podem enviar ou recuperar datasets sem sair da intranet.

A convenção de pastas segue o layout exigido pelo YOLO v8: dentro de data/ cada dataset é armazenado como dataset_<projeto>/train/{images,labels} e dataset_<projeto>/valid/{images,labels}. Essa hierarquia é criada programaticamente na função split_dataset() (app/api/routes/retrain.py). Nela o backend: lê todas as imagens reclassificadas e suas anotações; remove sufixos de augmentation e agrupa arquivos originais + aumentados; embaralha e divide 80 % para treino e 20 % para validação; copia os pares imagem/label para as pastas correspondentes dentro do repositório DVC; executa dvc add data, git add -A, git commit e dvc push, encapsulando toda a operação atrás do endpoint REST /send_data_to_dvc.

Cada commit recebe uma tag semântica – por exemplo EPI-v3.0 – para que o endpoint /dataset_version liste versões disponíveis, enquanto /delete_dataset_version remove tags obsoletas quando necessário.

Quando o usuário, no frontend, aciona Start Retraining, o backend primeiro organiza o novo snapshot de dados (chamada a /send_data_to_dvc) e logo em seguida dispara /train_model_new. Esse segundo endpoint invoca assincronamente o script Scripts/train.py. O script carrega o arquivo data.yaml (atualizado pela rota anterior), escolhe se parte de pesos existentes ou de "scratch" e grava o treinamento em CLASSIFICADOR/models/<model-vN>/. Ao final ele cria best_model_metrics.json, salvando métricas-chave junto com a versão do dataset usada, fechando o elo dados→pesos→resultado.

Durante inferência, o WebSocket definido em image.py consulta current_model.json para carregar, via Ultralytics, o modelo que o engenheiro marcou como ativo. Caso seja necessário usar um dataset antigo para auditoria ou para reproduzir um bug de campo, a rota /update_dvc executa: git checkout <tag-antiga>, dvc checkout –force e reconfigura automaticamente o data.yaml, garantindo que qualquer retreinamento subsequente use exatamente aquele conjunto de imagens.

3.4 Testes e Validações

Adotou-se uma abordagem iterativa e de integração contínua para a realização dos testes, conforme planejado na metodologia. À medida que cada componente da aplicação (frontend, backend, banco de dados etc.) era desenvolvido, testes parciais e preliminares foram realizados para verificar a conformidade com os requisitos funcionais. Após a consolidação da arquitetura, o plano de testes foi ampliado para cobrir exaustivamente os pontos mais críticos do ciclo captura → inferência → reclassificação → versionamento → retreinamento. A validação passou a seguir três camadas complementares: testes unitários, testes de integração local (frontend ↔ backend) e testes de orquestração Docker

Esses testes iniciais serviram como base para ajustes contínuos, frequentemente orientados por feedbacks recebidos nas reuniões semanais com o orientador acadêmico e com

o mentor técnico da empresa. Esse processo iterativo permitiu corrigir falhas de lógica, ajustar a interface e revisar fluxos de dados de forma ágil. A estrutura do Kanban e o uso disciplinado de commits organizados no Git, permitiram rastrear funcionalidades já testadas, organizar pendências e priorizar melhorias.

Além disso, foram realizados testes de usabilidade repetindo o fluxo "upload → classificação → reclassificação → retrain", apontando ajustes de UX que se converteram em issues rastreadas no Kanban. A validação do fluxo completo de retreinamento com versionamento integrado via DVC, garantindo que os novos datasets sejam corretamente registrados, que os modelos gerados sejam reaproveitáveis e que todas as versões estejam disponíveis para consulta e comparação é fundamental e ainda deve ser realizada. Outro ponto de atenção será a verificação do funcionamento da aplicação com a câmera conectada em tempo real, classificação e registro de imagens em um cenário contínuo na estrutura containerizada via Docker, assegurando que todos os serviços (frontend, backend, modelo, banco de dados) operem de forma orquestrada.

Além dos testes regulares, foi conduzido um processo de validação com o objetivo de consolidar um protótipo funcional, permitindo sua apresentação ao orientador acadêmico e ao representante da empresa parceira. Essas reuniões tiveram como finalidade verificar se o sistema desenvolvido atende aos requisitos estabelecidos na fase de definição do escopo, bem como coletar sugestões referentes à usabilidade das interfaces e ao fluxo de retreinamento do modelo.

Durante essa etapa, foram validados os seguintes aspectos:

- Design da interface de frontend;
- Definição da arquitetura e das bibliotecas utilizadas;
- Funcionalidades implementadas, tais como:
    - Exibição de imagens da câmera em tempo real;
    - Reclassificação manual de imagens;
    - Testes preliminares de versionamento de datasets.
    - Versionamento de modelos e datasets;
    - Processo completo de retreinamento dos modelos;
    - Upload de imagens para possibilitar treinamento a partir do zero;
    - Alternância entre diferentes versões de modelos treinados, de acordo com o cenário de aplicação.
- Funcionamento completo da arquitetura dockerizada;

Depois da entrega intermediária todas as verificações planejadas foram concluídas e o protótipo evoluiu para um ciclo de validação integral, executado em ambiente controlado do Insper. Para tanto, foram reunidos voluntários no Laboratório de Automação Industrial no Insper que vestiram os EPIs a serem analisados pelo modelo: Luvas, óculos, colete e capacete. A partir disso, foram tiradas fotos de cada voluntário, simulando situações e movimentos reais no laboratório e mantendo a câmera em uma posição fixa para todos. Assim, foi composto um dataset completo para ser usado nos treinamentos subsequentes.

3.5 Documentação

Paralelamente ao desenvolvimento técnico, foi elaborada a documentação do projeto por meio deste relatório, contemplando a arquitetura da solução, as etapas de desenvolvimento e os resultados obtidos. Também foram criados um guia do usuário, explicando as funcionalidades acessíveis do sistema e instruções de uso por meio de um vídeo explicativo, e um guia rápido sobre a utilização do DVC, voltado à manutenção e ao versionamento de datasets e modelos.

Além disso, foi elaborado um guia de instalação da arquitetura dockerizada, permitindo a execução modular e portável da aplicação em diferentes ambientes. Para desenvolvedores que desejam rodar o sistema localmente sem utilizar Docker, foi preparado um guia alternativo de execução sem Docker, mais adequado para o desenvolvimento e testes rápidos, evitando a necessidade de subir imagens constantemente. No entanto, essa abordagem possui pequenas diferenças estruturais em relação ao código da versão dockerizada, sendo indicada apenas para fins de desenvolvimento e não para produção.

No repositório de código, adotou-se uma organização padronizada de branches, em que a branch main de cada repositório contém a versão completa da aplicação com Docker e seus respectivos arquivos de orquestração, enquanto a branch release contém a versão configurada para execução local sem Docker, mas com as mesmas funcionalidades principais. Essa separação permite flexibilidade no ambiente de execução e facilita o trabalho de desenvolvedores com diferentes níveis de familiaridade com contêineres.

Além disso, os códigos Python foram comentados utilizando a biblioteca Sphinx, permitindo a geração automática de documentação técnica a partir das docstrings. Essa documentação foi estruturada em formato web, hospedada na pasta docs, e oferece uma visão completa dos módulos, funções e classes do backend. O uso da extensão autodoc garante que as anotações feitas diretamente nas funções sejam refletidas dinamicamente na documentação final, promovendo clareza e facilitando a manutenção por futuros desenvolvedores.

Por fim, os commits e branches foram mantidos de forma organizada, promovendo rastreabilidade do progresso e facilitando revisões. Os comentários explicativos no código-fonte detalham o funcionamento de funções-chave, o que contribui para a compreensão coletiva do sistema e estabelece uma base sólida para futuras manutenções ou evoluções, seja por membros atuais da equipe ou por terceiros.

4. Resultados

O resultado do projeto é como desenhado: uma Solução Full-Stack de machine learning para Visão Computacional em Ambientes Industriais: Classificação, Versionamento e Retreinamento Local de Modelos. Para demonstrar isso, realizamos o processo em três etapas sequenciais, espelhando o fluxo real de uso do sistema:

Primeiro, obteve-se um dataset público de EPI (capacete, colete, luva e óculos) e o realizou-se o upload de todas as imagens desse dataset pela função de upload (Imagem 27). Com as imagens disponíveis, foi realizada a classificação manual de 150 das imagens, por meio da escolha das Labels e desenho das Bounding Boxes (Imagem 28). Nota-se aqui também que o dataset utilizava-se de data augmentation, portanto para não haver vazamento de dados os nomes dos arquivos foram corrigidos conforme já foi explicado.

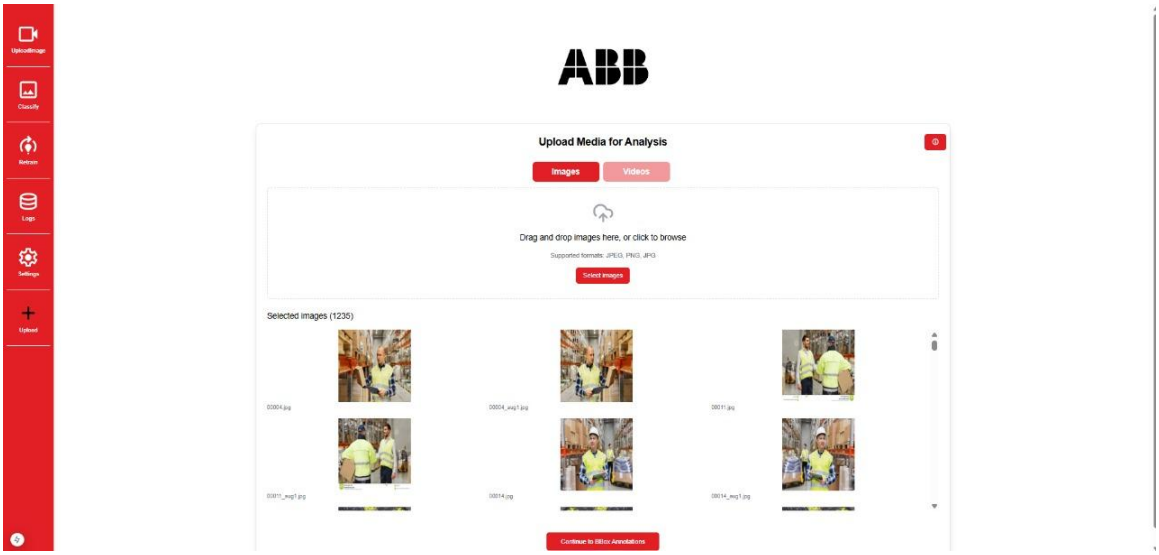


Imagem 27 – Upload de imagens do dataset público

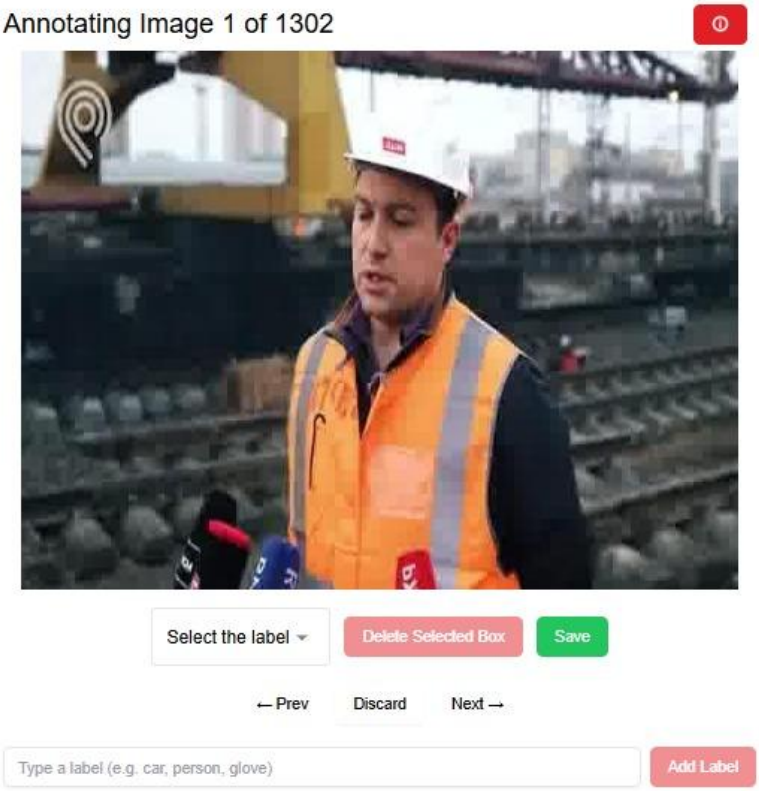


Imagem 28 – classificação de imagens do dataset público

Para a aquisição e rotulagem próprias, em laboratório controlado do Insper, montou-se um cenário com voluntários usando EPIs reais (luvas, óculos, colete e capacete). A página UploadImage recebeu as 150 fotos tiradas (Imagem 29) e o annotator de bounding boxes foi utilizado para desenhar manualmente as regiões em que se encontravam as labels escolhidas e salvar os pares .jpg + .txt (Imagem 30).

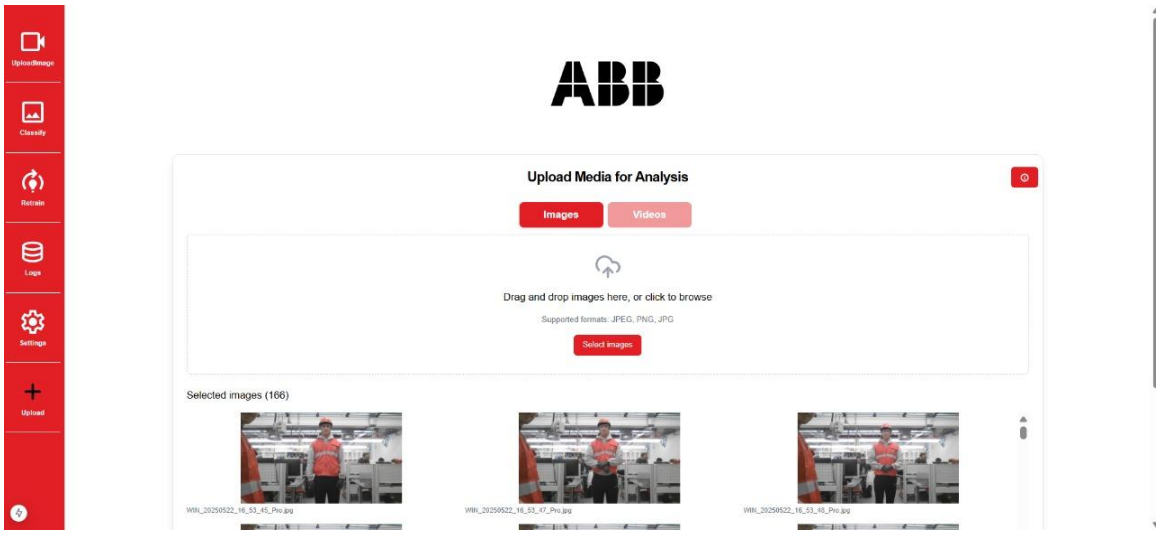


Imagem 29 – upload de imagens do dataset criado em laboratório

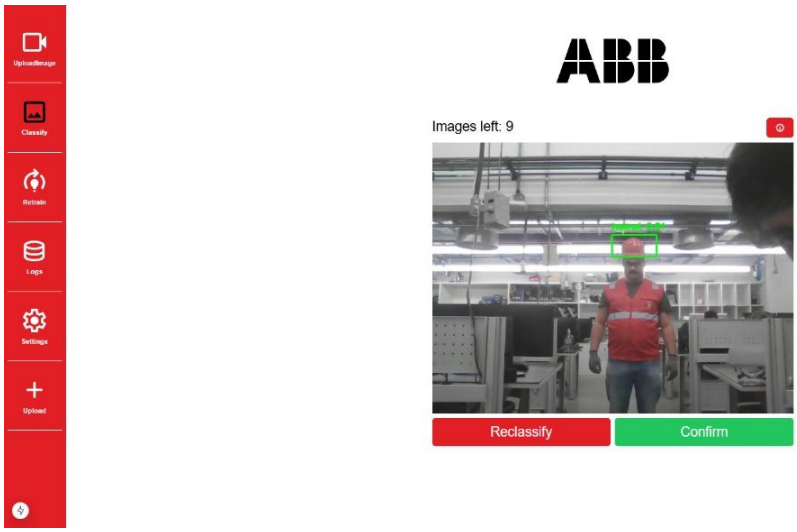


Imagem 30 – classificação de imagens do dataset criado em laboratório

Com isso, na página de retrain, foi escolhido treinar o modelo do zero e depois quais quantas das imagens realizadas em laboratório rotuladas vão ser incluídas no novo dataset e quantas epochs que o modelo seja treinado. Com isso, ele volta ao tipo de treinamento disponíveis, e pode escolher em incrementar o número de epochs ou não e escolher adicionar novas imagens ao dataset ou manter o dataset como está. Para meios de demonstração, fixamos o número de epochs em 50 e adicionamos 131 imagens ao novo dataset, criando-se o dataset-v1.

Com isso gerou-se o model-v1 (Imagem 31) e obteve-se as seguintes métricas:

Training Summary	
Best Train Loss	**1,8819**
Best Val Loss	**2,0001**
Best mAP50	**0,4673**

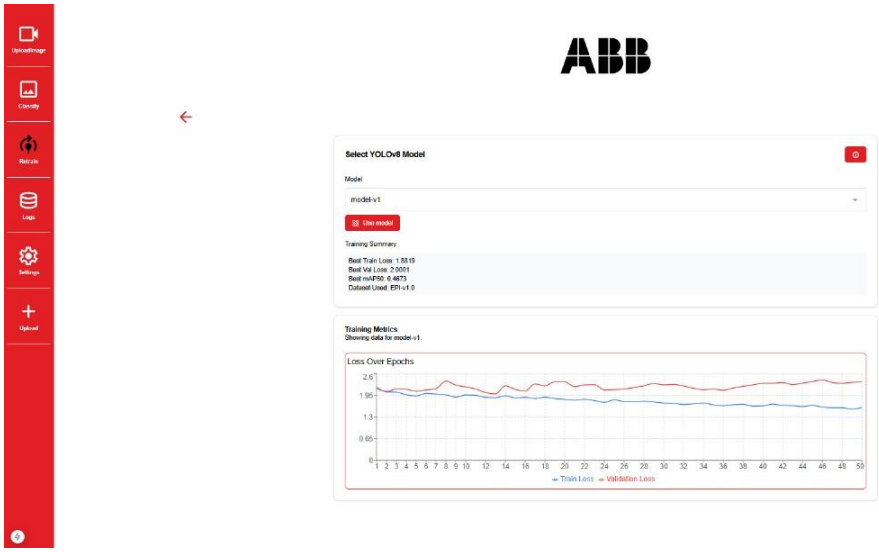


Imagem 31 – resultado do model-v1com o dataset EPI-v1

` Para validação final, o modelo foi retreinado com mais épocas, portanto o usuário escolher por treinar com um dataset já existente e definiu o número de épocas para 200, como na imagem 32. E o modelo foi colocado para retreinamento, como mostra o progresso na imagem 33.

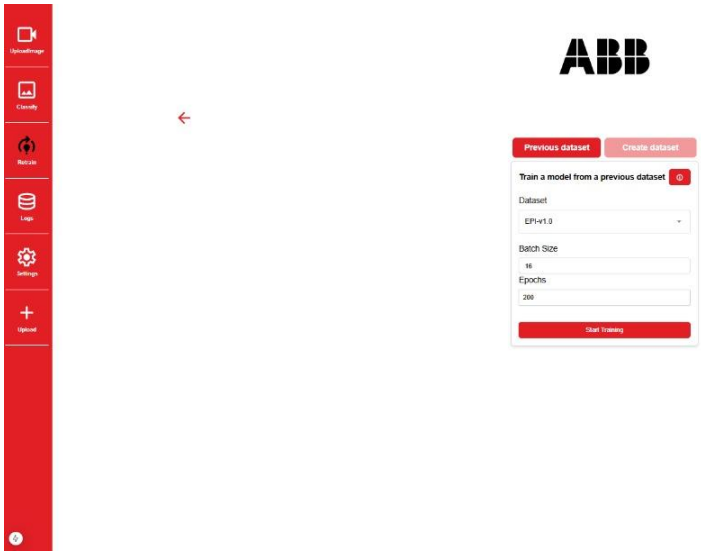


Imagem 32 – Retreinamento do model-v1 com o dataset EPI-v1 com 200 épocas

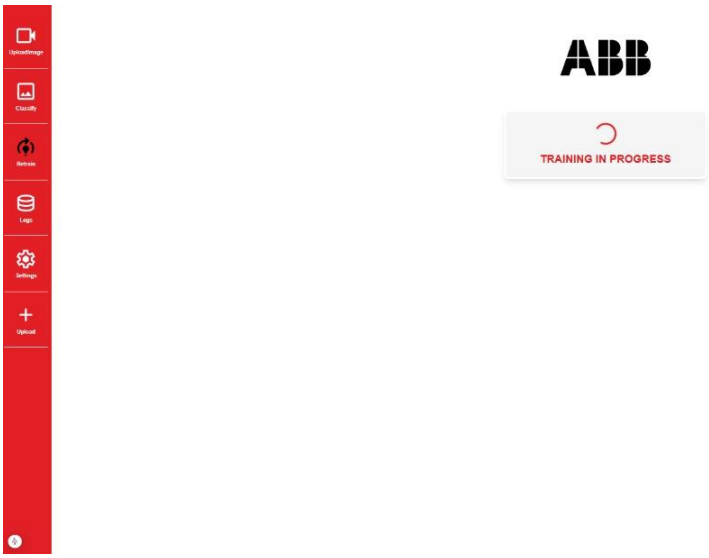


Imagem 33 – Progresso do retreinamento

Imagem 34 –Retreinamento finalizado

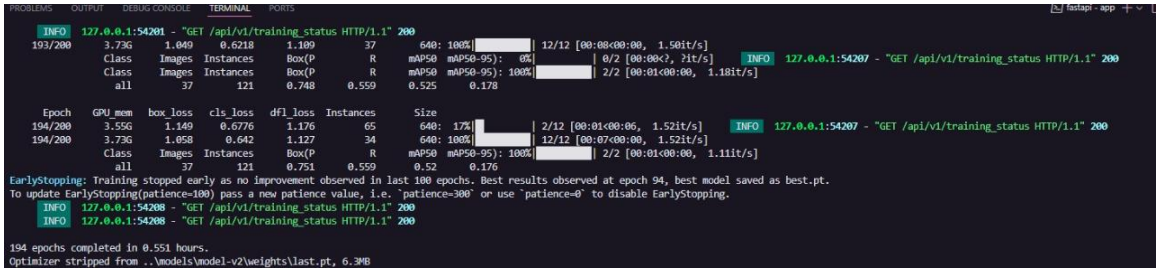


Imagem 35 – Resultados do retreinamento no terminal

O retreinamento resultou no que está na imagem 35, com os resultados do Yolo no terminal. O modelo, concluiu-se em 0,55 horas e foi interrompido automaticamente pelo callback de EarlyStopping ao detectar que, desde a época 94, nenhuma métrica de validação melhorava dentro da janela de paciência (100 épocas). O melhor ponto do ciclo foi, portanto, a época 94, cujo checkpoint foi salvo como best.pt e promovido para uso na interface (Imagem 36).

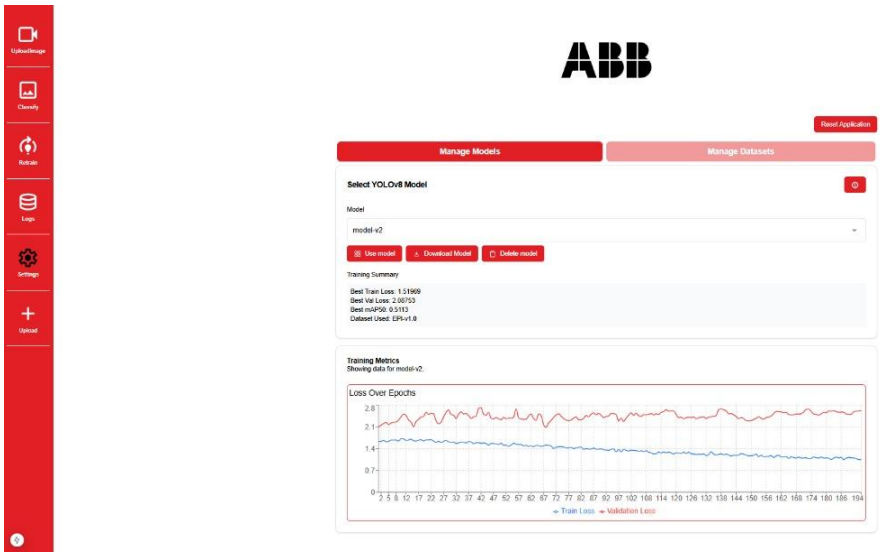


Imagem 36 – Resultados do retreinamento na interface

A validação da plataforma resultou na execução bem-sucedida do modelo de detecção em tempo real, como ilustrado na Imagem 37. Nesta tela, é possível observar o modelo YOLOv8 identificado como model-v2 operando ao vivo sobre o stream da câmera, realizando a detecção de capacetes e EPI's. A interface permite ativar ou desativar as tarefas do modelo via o botão "Execute Model Tasks", habilitando a funcionalidade de capturas automáticas por evento (alarme) e por intervalo de tempo (random). Esse teste confirma que o sistema está apto a operar de forma autônoma, realizando inferência contínua e servindo de base para as etapas seguintes de reclassificação e retreinamento.

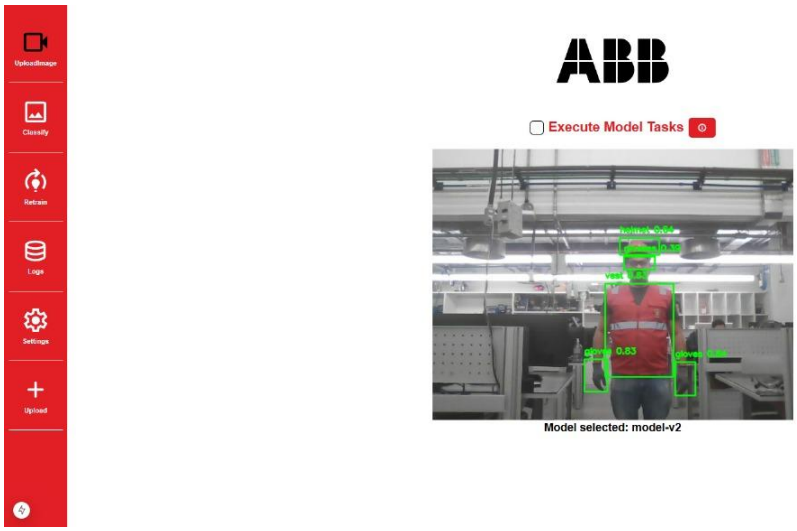


Imagem 37 – Modelo rodando na câmera ao vivo

Comparando as métricas, obteve-se um Best Train Loss–19 % em relação ao modelo-v1, um Best Val Loss relativamente igual e um Best mAP50: 0,517 10% melhor. Isso traz que, a incorporação das 131 imagens reais rotuladas em laboratório foi decisiva para elevar a sensibilidade do detector e que a diferença contida entre train e val loss indica que o modelo ainda não sofreu overfitting, portanto, novos ciclos poderão obter ganhos adicionais se trouxerem novos exemplos de casos. Mas acima de tudo tem-se que o fluxo esperado funcionou e o retreinamento foi concluído com sucesso e com melhora, confirmando o funcionamento da arquitetura.

5. Discussão dos Resultados e Próximos Passos

A análise dos resultados demonstra que a arquitetura full-stack proposta, composta por Next.js no frontend, FastAPI no backend, YOLO como motor de visão computacional, PostgreSQL para organização dos logs, DVC para versionamento de dados e Docker como camada de orquestração, cumpre o requisito central de funcionar integralmente dentro de uma intranet industrial. O ciclo completo, executado em laboratório no Insper, comprovaram o fluxo end-to-end: primeiro treinou-se um modelo inicial (model-v1) a partir de um dataset expandido de dataset público de EPIs, com a adição de imagens capturadas em cenário controlado, coletadas e rotuladas manualmente. Logo depois, prolongou-se o treinamento para duzentas épocas sobre esse mesmo conjunto, treinando o model-v2. Em todas as rodadas o sistema rastreou automaticamente as imagens e anotações via DVC, etiquetou código e dados na mesma tag Git, empacotou pesos e métricas no diretório correto e, graças ao canal WebSocket, atualizou em tempo real a interface do operador, que pôde promover ou rebaixar versões de modelo com um único clique.

Para a ABB o protótipo agrega valor imediato porque transforma a manutenção de modelos em um ciclo contínuo que pode ser executado pelo próprio engenheiro de automação, sem depender de acesso externo. A empresa passa a ter rastreabilidade fina de cada versão de dataset e de modelo, monitora em tempo real falsos positivos e negativos, e consegue reagir rapidamente a mudanças de processo ou iluminação treinando um novo modelo localmente.

Para a próxima fase, recomenda-se que o grupo responsável pela continuidade do projeto realize novas iterações da arquitetura e das tecnologias utilizadas, abrangendo possíveis alterações nos frameworks e na infraestrutura de hardware e software envolvida, com base no feedback da ABB. O novo projeto também deverá priorizar a ampliação das funcionalidades já existentes, como a integração simplificada com sistemas de automação e o desenvolvimento de novas interfaces e funções.

O projeto prova que é possível implementar, dentro do limite físico de uma intranet industrial, um ciclo de machine learning completo, auditável e sustentável. Essa base permitirá que a ABB amplie o alcance do Camera Connect, introduzindo novas aplicações de visão computacional sem renunciar a segurança, privacidade nem controle de versões, ao mesmo tempo em que oferece ao próximo grupo de PFE um terreno fértil para pesquisa e inovação incremental.

6. Referências Bibliográficas

1. ANDRADE, M. Usando o Docker. São Paulo: Novatec, 2016.
2. CHARAYA, P.; JINDAL, A.; KAUR, H. Region of interest extraction in images: A survey. International Journal of Advanced Research in Computer Science and Software Engineering, v. 6, n. 5, p. 1–7, 2016.
3. GOMES, R.; SOUZA, R. Docker – Infraestrutura como Código, com Autonomia e Replicabilidade. Universidade Federal da Bahia, 2015.
4. GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. Deep Learning. Cambridge: MIT Press, 2016.
5. LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. Proceedings of the IEEE, v. 86, n. 11, p. 2278–2324, 1998.
6. NATH, S.; BEHZADAN, V.; PAAL, S. G. Automated monitoring of personal protective equipment usage on construction sites using deep learning. Automation in Construction, v. 119, p. 103377, 2020.
7. PARK, J. et al. Chimney detection and measurement in high-resolution satellite imagery using YOLOv8 and ResNet. Remote Sensing, v. 15, n. 4, p. 1–18, 2023.
8. REDMON, J.; FARHADI, A. YOLOv3: An incremental improvement. arXiv preprint, arXiv:1804.02767, 2018.
9. ROMERO, D. Containers com Docker: do desenvolvimento à produção de contêineres. São Paulo: Casa do Código, 2016.
10. SANTOS, R. S. Detecção de EPIs em tempo real utilizando redes neurais convolucionais. Revista Brasileira de Tecnologia e Inovação, v. 10, n. 1, p. 45–60, 2023.
11. TERVEN, J.; CORDOVA-ESPARZA, D. M. YOLOv8: A comprehensive review. Journal of Artificial Intelligence Research, v. 20, n. 3, p. 1–25, 2023.
12. TURNBULL, J. The Docker Book: Containerization is the new virtualization. Creative Commons, 2014.
13. VERAS, M.; CARISSIMI, A. Virtualização de Servidores. Rio de Janeiro: RNP, 2015.
14. VITALINO, J. F. N.; CASTRO, M. A. N. Descomplicando o Docker. Rio de Janeiro: Brasport, 2016.
15. HI-TECH BPO. Top Object Detection Models. Disponível em: https://www.hitechbpo.com/blog/top-object-detection-models.php. Acesso em: 22 mar. 2025.
16. KEYLABS.AI. YOLOv8 vs SSD: Choosing the right object detection model. Disponível em: https://keylabs.ai/blog/yolov8-vs-ssd-choosing-the-right-object-detection-model/. Acesso em: 22 mar. 2025.
17. ULTRALYTICS. EfficientDet vs YOLOv11 - Ultralytics Docs. Disponível em: https://docs.ultralytics.com/pt/compare/efficientdet-vs-yolo11/. Acesso em: 22 mar. 2025.
18. ALPHA2PHI. Build a Web-based Object Detection System using YOLO. Medium, 16 fev. 2021. Disponível em: https://alpha2phi.medium.com/yolo-using-fastapi-websocket-and-react-2b2d28e9f7ed. Acesso em: 21 maio 2025.
19. BORGOHAIN, Rajdeep. Exploring HTTPS vs. WebSocket for Real-Time Model Inference in Machine Learning Applications. Inferless (Blog), 11 jun. 2024. Disponível em: https://www.inferless.com/learn/exploring-https-vs-websocket-for-real-time-model-inference-in-machine-learning-applications. Acesso em: 21 maio 2025.
20. DATA SCIENCE COUNCIL OF AMERICA (DASCA). Effortless Data and Model Versioning with DVC. 14 nov. 2024. Disponível em: https://www.dasca.org/world-of-data-science/article/effortless-data-and-model-versioning-with-dvc. Acesso em: 21 maio 2025.

21. PAGUTHANIYA, Sajid Ali et al. Integration of Machine Learning Models into Backend Systems: Challenges and Opportunities. IOSR Journal of Computer Engineering, v.26, n.3, p.33-39, 2024.
22. PANKAJ, Anil. Building a Scalable Microservice for Object Detection with YOLOv8, FastAPI and Docker. Medium, 5 fev. 2025. Disponível em: https://medium.com/@anilpankaj3/building-a-scalable-microservice-for-object-detection-with-yolov8-a320b1436da6. Acesso em: 21 maio 2025.
23. **SHOKRZAD, Reza.** *FastAPI: The Modern Toolkit for Machine Learning Deployment*. Medium, 26 nov. 2023. Disponível em: https://medium.com/@reza.shokrzad/fastapi-the-modern-toolkit-for-machine-learning-deployment-af31d72b6589. Acesso em: 21 maio 2025.
24. *How Uber Leverages FastAPI For Scalable Machine Learning Inference With Michelangelo*. The ML Engineer – Newsletter (Vintage Global), edição 209, 2024. Disponível em: https://www.linkedin.com/pulse/how-uber-leverages-fastapi-scalable-machine-learning-bds7e. Acesso em: 21 maio 2025.
25. **"5 Advantages of Microservices."** Atlassian Blog, 2021. Disponível em: https://www.atlassian.com/microservices/microservices-architecture/advantages-of-microservices. Acesso em: 21 maio 2025.
26. ULTRALYTICS. *YOLOv8: Selecting best epoch for training result*. GitHub Issues, 2024. Disponível em: https://github.com/ultralytics/ultralytics/issues/14137. Acesso em: 26 maio 2025.

7. Anexos

7.1 Questões Éticas e Profissionais

O desenvolvimento deste projeto envolve responsabilidades éticas e profissionais essenciais, considerando tanto a aplicação tecnológica quanto o cumprimento das diretrizes legais e regulamentares. Como a solução será utilizada em um ambiente industrial, seu impacto direto na segurança do trabalho, privacidade de dados e equidade na detecção de EPIs exige um compromisso com boas práticas de desenvolvimento.

O modelo de identificação deve evitar penalizações indevidas ou falhas na detecção de riscos reais. Além disso, o algoritmo deve ser imparcial, garantindo que a identificação dos EPIs ocorra independentemente de características físicas, como altura, peso, cor de pele ou vestimentas específicas. Esse cuidado evita viés algorítmico e assegura que o sistema seja justo e confiável para todos os trabalhadores.

Outro aspecto crítico do projeto é a privacidade dos dados e o cumprimento da Lei Geral de Proteção de Dados (LGPD). O sistema foi projetado para processar apenas os aspectos visuais relevantes para a análise industrial, sem exposição indevida das informações dos trabalhadores e todo o armazenamento e processamento seguem protocolos de segurança e privacidade estabelecidos pela empresa parceira.

A infraestrutura desenvolvida e os algoritmos aplicados devem respeitar os limites definidos no escopo do projeto, garantindo que a solução atenda estritamente às necessidades da organização parceira. A equipe se comprometeu a não compartilhar dados ou modelos fora do contexto do Capstone e da empresa parceira, garantindo que o conhecimento gerado seja utilizado de forma ética e responsável.

7.2 Normas Técnicas

Diferente de um projeto voltado para a implementação de processos normativos específicos, este trabalho não se baseia diretamente em normas técnicas da ABNT ou ISO para definir sua arquitetura ou algoritmos. No entanto, algumas diretrizes gerais foram consideradas para garantir que o desenvolvimento seguisse boas práticas e respeitasse exigências da organização parceira.

A principal referência normativa aplicada ao projeto foi a Lei Geral de Proteção de Dados (LGPD - Lei nº 13.709/2018), que orientou as decisões sobre armazenamento, processamento e acesso às imagens utilizadas no treinamento dos modelos de visão computacional. Como parte do compromisso com a proteção de dados, foram adotadas restrições quanto à divulgação das informações processadas, e os repositórios de código e datasets foram mantidos em ambientes internos controlados, sem acesso público.

Além disso, o desenvolvimento do projeto seguiu as regras e diretrizes da organização parceira para o uso de softwares e bibliotecas. Essas diretrizes impactaram diretamente na escolha das ferramentas utilizadas para implementação dos modelos e da infraestrutura, garantindo que a solução estivesse alinhada com as práticas e padrões já estabelecidos dentro do ambiente corporativo da empresa.

7.3 Riscos Envolvidos

Durante o planejamento e a execução do projeto, foram identificados diversos riscos que poderiam impactar seu sucesso, sendo classificados em riscos técnicos, regulatórios e organizacionais. Para minimizar esses impactos, foram adotadas estratégias específicas ao longo do desenvolvimento.

Entre os riscos técnicos, destaca-se a possibilidade de baixa acurácia dos algoritmos de visão computacional, o que poderia comprometer a confiabilidade da solução. Outro fator relevante é a limitação de hardware e infraestrutura, uma vez que a aplicação opera em uma arquitetura local via Intranet, o que impõe restrições na capacidade de processamento e armazenamento. Para reduzir esse risco, o sistema foi projetado para ser modular e eficiente, garantindo que os modelos sejam processados de forma otimizada dentro das limitações existentes.

Outro risco importante identificado foi a possível falta de imagens para o treinamento dos modelos, uma vez que a empresa parceira poderia não fornecer um volume suficiente de dados. Para evitar essa limitação, a equipe utilizou datasets públicos para validação preliminar dos modelos.

No que diz respeito aos riscos regulatórios, um dos principais desafios está na conformidade com a Lei Geral de Proteção de Dados (LGPD), visto que o sistema lida com imagens de operários e ambientes industriais. Para mitigar esse risco, o projeto foi desenvolvido seguindo diretrizes de proteção de dados, assegurando que as imagens fossem tratadas de forma anônima e armazenadas de acordo com os protocolos de segurança estabelecidos pela empresa parceira.

Outro risco regulatório identificado foi o uso indevido de bibliotecas de software, uma vez que algumas dependências utilizadas no desenvolvimento poderiam apresentar restrições para uso comercial. Para evitar problemas legais, a equipe seguiu as diretrizes da empresa parceira, garantindo que apenas bibliotecas licenciadas ou de uso liberado para ambientes industriais fossem empregadas, além de estipular possíveis assinaturas de bibliotecas, como o YOLO. Além disso, os repositórios internos do Insper foram utilizados.

Por fim, entre os riscos organizacionais, destaca-se a possibilidade de atrasos no desenvolvimento e validação dos modelos, uma vez que o tempo limitado do Capstone impõe desafios para o cumprimento do cronograma. Para reduzir esse risco, foram estabelecidos marcos intermediários que permitiram o monitoramento contínuo do progresso do projeto, possibilitando ajustes sempre que necessário. A equipe também adotou uma abordagem ágil de

gestão, utilizando o Kanban no Microsoft Teams para organizar as tarefas e garantir que as entregas ocorressem dentro dos prazos estipulados.

Outro fator organizacional relevante foi o risco de desalinhamento com as expectativas da organização parceira, o que poderia comprometer a aplicabilidade da solução final. Para mitigar esse problema, a equipe manteve uma comunicação constante com o Mentor, realizando reuniões periódicas para alinhar as necessidades e garantir que o desenvolvimento estivesse em conformidade com as demandas da organização.

7.4 Recursos Necessários

A execução do projeto requer recursos acessíveis, sendo a maior parte das atividades realizadas utilizando os próprios computadores da equipe. No entanto, para atender às demandas computacionais do treinamento de modelos de visão computacional, foi disponibilizada uma GPU dedicada NVIDIA GeForce RTX 4070 Ti, com 11 GB de VRAM, instalada em um computador com processador AMD Ryzen 7 5800X (8 núcleos, 3.8 GHz) e 64 GB de RAM. Esta configuração garante maior poder de processamento, possibilitando a execução de modelos mais complexos e exigentes. A GPU dedicada oferece vantagens significativas em tarefas de machine learning, como a aceleração do treinamento de redes neurais, permitindo a execução de cálculos massivos de forma paralela e eficiente. Além disso, a RTX 4070 Ti é altamente otimizada para lidar com grandes volumes de dados, como imagens e vídeos de alta resolução, que são fundamentais em modelos de visão computacional. A capacidade de processamento paralelo e o armazenamento de dados na VRAM dedicada tornam esse hardware uma solução ideal para a execução de modelos mais pesados e complexos, garantindo maior agilidade e eficiência no desenvolvimento de soluções baseadas em IA.

Durante as etapas iniciais do projeto, a equipe utilizou datasets públicos da internet para realizar testes preliminares no modelo de EPIS e validar a viabilidade técnica dos algoritmos propostos. Essa estratégia permitiu comprovar conceitos e ajustar os modelos antes de obter acesso às imagens reais da empresa parceira. Em uma fase posterior, as imagens captadas diretamente no ambiente industrial serão incorporadas ao processo de treinamento, permitindo a adaptação dos modelos ao contexto específico do sistema de controle distribuído (DCS). Para viabilizar essa etapa, será utilizada uma câmera dedicada à aquisição das imagens, garantindo a captura de dados compatíveis com as condições reais da aplicação. Para o treinamento do modelo de chaminés, o dataset foi criado a partir de imagens da internet, junto com um algoritmo de aumentação de imagens.

Além do hardware necessário, o desenvolvimento do projeto depende de uma variedade de bibliotecas, frameworks e recursos computacionais para a implementação dos modelos e da arquitetura do sistema. Entre as principais ferramentas utilizadas estão o YOLO para detecção de padrões em imagens, DVC e Git para versionamento e reclassificação dos modelos, além de Next.js e Node.js para a construção da interface interativa. O ambiente de execução foi estruturado para rodar em infraestrutura local e dockerizada, permitindo escalabilidade e portabilidade na implantação.

7.5 Cronograma

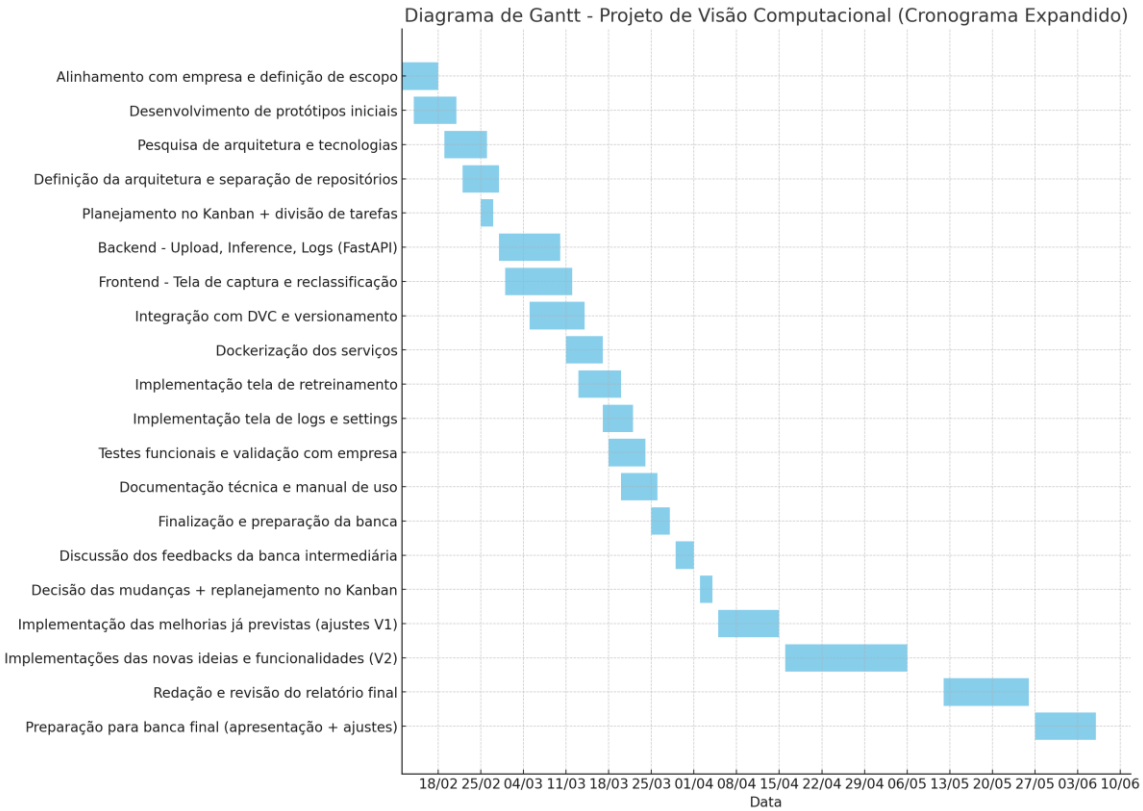


Imagem 3- Diagrama de Grantt do cronograma do grupo

Task	Start	End	Sprint	Responsáveis	Duração (dias)
Alinhamento com empresa e definição de escopo	12/02/2025	18/02/2025	1	Todos	7
Desenvolvimento de protótipos iniciais	14/02/2025	21/02/2025	1	Douglas, Pedro	8
Pesquisa de arquitetura e tecnologias	19/02/2025	26/02/2025	2	Todos	8
Definição da arquitetura e separação de repositórios	22/02/2025	28/02/2025	2	Todos	7
Planejamento no Kanban + divisão de tarefas	25/02/2025	27/02/2025	2	Todos	3
Backend - Upload, Inference, Logs (FastAPI)	28/02/2025	10/03/2025	3	Todos	11

Frontend - Tela de captura e reclassificação	01/03/2025	12/03/2025	3	Pedro	12
Integração com DVC e versionamento	05/03/2025	14/03/2025	3	Pedro	10
Dockerização dos serviços	11/03/2025	17/03/2025	4	Douglas, Gustavo	7
Implementação tela de retreinamento	13/03/2025	20/03/2025	4	Diego	8
Implementação tela de logs e settings	17/03/2025	22/03/2025	4	Pedro	6
Testes funcionais e validação com empresa	18/03/2025	24/03/2025	5	Todos	7
Documentação técnica e manual de uso	20/03/2025	26/03/2025	5	Diego, Pedro	7
Finalização e preparação da banca	25/03/2025	28/03/2025	5	Todos	4
Discussão dos feedbacks da banca intermediária	29/03/2025	01/04/2025	6	Todos	4
Decisão das mudanças + replanejamento no Kanban	02/04/2025	04/04/2025	6	Todos	3
Implementação das melhorias já previstas (ajustes V1)	05/04/2025	15/04/2025	7	Todos	11
Upload de imagens e vídeos – front e backend	16/04/2025	06/05/2025	8	Pedro e Diego	21
Debugging	06/05/2025	12/05/2025	8	Todos	7
Dockerizacao – integração da câmera	16/04/2025	28/04/2025	8	Douglas e Gustavo	12
Dockerizacao integração dos contêiners individuais	28/04/2025	16/05/2025	8	Douglas e Gustavo	18
Redação e revisão do relatório final	12/05/2025	26/05/2025	9	Todos	14
Elaboração dos Guias de implementação e uso	05/05/2025	26/05/2025	9	Diego	21
Elaboração do Vídeo	05/05/2025	28/05/2025	9	Diego	23

Preparação para banca final (apresentação + ajustes)	27/05/2025	06/06/2025	10	Todos	11

Tabela 1 – Cronograma do Projeto

7.6 Mapeamento dos Stakeholders

O mapeamento dos stakeholders do projeto envolve os membros da organização parceira. Esse mapeamento é essencial para alinhar expectativas, facilitar a comunicação e garantir que todas as partes interessadas sejam devidamente consideradas durante o desenvolvimento do projeto.

Stakeholder	Posição	Papel no Projeto	Expectativas
Tiago Prata	Gerente Técnico (ABB)	Supervisão técnica do projeto, fornecimento de orientações e validação dos algoritmos desenvolvidos.	Receber uma solução eficiente e integrada ao sistema DCS da ABB.
Fernanda Janguas	RH (ABB)	Gerenciamento do contato administrativo e organizacional, facilitando a comunicação entre a equipe acadêmica e a empresa.	Garantir uma boa relação entre a equipe do Insper e a ABB.

Tabela 2 – Mapeamento dos Stakeholders

7.7 Conteinerização e Deploy

Documentação de Conteinerização e Deploy

1. Visão Geral

Este documento descreve a *stack* conteinerizada do projeto PFE, composta por FastAPI (backend), Next.js (frontend), PostgreSQL, Classificador (PyTorch + Ultralytics) e DVC.

---

2. Estrutura do Repositório

```
Arquitetura-PFE
│
├── docker-compose.yml
├── .env.sample                    # Substituir variáveis de ambiente exemplo Configurações
│
├── back-end-PFE                   # serviço FastAPI
│   ├── Dockerfile
│   ├── app
│   ├── .env                       # Imutável, configurações padrão
│   ├── control_db.json            # Inicialmente null
│   ├── project_name.json          # Inicialmente null
│   └── ...
│
├── abb-web                        # serviço Next.js
│   ├── Dockerfile
│   ├── src
│   └── ...
│
├── Classificador                  # serviço de IA
│   ├── Dockerfile
│   ├── Scripts
│   ├── data.yaml                  # Inicialmente Vazio
│
└── dvc-ABB                        # Repositório DVC
    └── Dockerfile
```

3. Serviços e Dockerfiles

Serviço	Porta	Highlights do Dockerfile
backend	8000	Base python:3.11-slim → instala deps → uvicorn --reload (dev) / gunicorn ... (prod)
frontend	3000	Build multi-stage: node:20 compila; node:20-alpine roda yarn start
db	5432	Imagem oficial postgres:15
classificador	N/A	python:3.11 + PyTorch, volume /models read-only

dvc-abb	N/A	Debian slim com DVC + Git

---

4. Ambiente Local

1. Clone todos os repositórios do projeto

```
git clone https://github.com/pfeinsper/abb-web.git

git clone https://github.com/pfeinsper/back-end-PFE.git

git clone https://github.com/pfeinsper/Classificador.git

git clone https://github.com/pfeinsper/dvc-ABB.git
```

2. Crie o arquivo .env na raiz do projeto baseando-se no .env.sample, definindo as credenciais que serão utilizadas pelo banco de dados e variáveis git como:

- User_db
- Pass_db
- url de conexão
- Usuário git
- Email git

3. Suba a stack (build completo na primeira vez)

```
docker compose up -d --build
```

4. Acesso às Endpoints

- Frontend : http://localhost:3000

- API docs : http://localhost:8000/docs

5. Publicação das Imagens no Amazon ECR

Para permitir que o projeto seja executado facilmente em qualquer máquina ou ambiente de nuvem, todas as imagens Docker utilizadas na stack foram publicadas no Amazon Elastic Container Registry (ECR), utilizando repositórios privados.

5.1. Repositórios criados

Os seguintes repositórios privados foram criados na conta AWS do grupo (533267158323), na região us-east-2, com os nomes:

Serviço	Repositório ECR	Tag
Backend	abb/infraestrutura-backend	latest
Frontend	abb/infraestrutura-frontend	latest
Classificador	abb/infraestrutura-classificador	latest
Dvc	abb/infraestrutura-dvc	latest
Banco de Dados	abb/db-postgres	15

5.2. Etapas para subir as imagens

1. Autenticação via AWS CLI com SSO

aws configure sso

Respostas:

- SSO start URL: https://d-9067fda013.awsapps.com/start
- SSO region: us-east-2
- AWS account ID: 533267158323
- Role: AdministratorAccess

2. Login no Amazon ECR

aws ecr get-login-password --region us-east-2 --profile AdministratorAccess-533267158323 |
`

docker login --username AWS --password-stdin 533267158323.dkr.ecr.us-east-2.amazonaws.com

3. Tag das imagens locais com os URIs do ECR

docker tag infraestrutura-backend:latest 533267158323.dkr.ecr.us-east-2.amazonaws.com/abb/infraestrutura-backend:latest

docker tag infraestrutura-frontend:latest 533267158323.dkr.ecr.us-east-2.amazonaws.com/abb/infraestrutura-frontend:latest

docker tag infraestrutura-classificador:latest 533267158323.dkr.ecr.us-east-2.amazonaws.com/abb/infraestrutura-classificador:latest

docker tag infraestrutura-dvc-abb:latest 533267158323.dkr.ecr.us-east-2.amazonaws.com/abb/infraestrutura-dvc:latest

docker tag postgres:15 533267158323.dkr.ecr.us-east-2.amazonaws.com/abb/db-postgres:15

4. Envio das imagens com docker push

docker push 533267158323.dkr.ecr.us-east-2.amazonaws.com/abb/infraestrutura-backend:latest

docker push 533267158323.dkr.ecr.us-east-2.amazonaws.com/abb/infraestrutura-frontend:latest

docker push 533267158323.dkr.ecr.us-east-2.amazonaws.com/abb/infraestrutura-classificador:latest

docker push 533267158323.dkr.ecr.us-east-2.amazonaws.com/abb/infraestrutura-dvc:latest

docker push 533267158323.dkr.ecr.us-east-2.amazonaws.com/abb/db-postgres:15

5.3. Atualização do docker-compose.yml

O docker-compose.yml foi ajustado para usar as imagens do ECR diretamente sem pré configurações:

```yaml
services:
  ▷Run Service
  backend:
    image: 533267158323.dkr.ecr.us-east-2.amazonaws.com/abb/infraestrutura-backend:latest
  ▷Run Service
  frontend:
    image: 533267158323.dkr.ecr.us-east-2.amazonaws.com/abb/infraestrutura-frontend:latest
  ▷Run Service
  classificador:
    image: 533267158323.dkr.ecr.us-east-2.amazonaws.com/abb/infraestrutura-classificador:latest
  ▷Run Service
  dvc:
    image: 533267158323.dkr.ecr.us-east-2.amazonaws.com/abb/infraestrutura-dvc:latest
  ▷Run Service
  db:
    image: 533267158323.dkr.ecr.us-east-2.amazonaws.com/abb/db-postgres:15
```

5.4. Como clonar e subir a stack com imagens do ECR

Para um novo colaborador:

aws configure sso

aws ecr get-login-password --region us-east-2 | docker login --username AWS --password-stdin 533267158323.dkr.ecr.us-east-2.amazonaws.com

git clone https://github.com/pfeinsper/abb-web.git

git clone https://github.com/pfeinsper/back-end-PFE.git

git clone https://github.com/pfeinsper/Classificador.git

git clone https://github.com/pfeinsper/dvc-ABB.git

cd abb-web

docker compose pull

docker compose up

---

6. Variáveis de Ambiente (referência)

Chave	Descrição
CLASSIFIER_PATH	Caminho interno do classificador (/app/Classificador)
DATA_YAML_PATH	/datasets/data/data.yaml
DVC_REPO_PATH	/datasets/
NEXT_PUBLIC_API_URL	URL que o Next.js utiliza para acessar a API
POSTGRES_*	Configurações do banco

Todas as variáveis ficam em .env; nunca faça commit de senhas reais.

---

7. Persistência de Dados

- postgres_data → volume nomeado para /var/lib/postgresql/data (SGBD).
- dataset_volume → snapshots do DVC e imagens anotadas.
- models_volume → modelos treinados pelo classificador.

7.8 Guia de instalação do DVC

Installation Guide – VisionOps

How to run the application locally

ABB

São Paulo
2025

1. **Basic Requirements**

The following list includes software that is easy to install and used as standard tools to run the application locally during development.

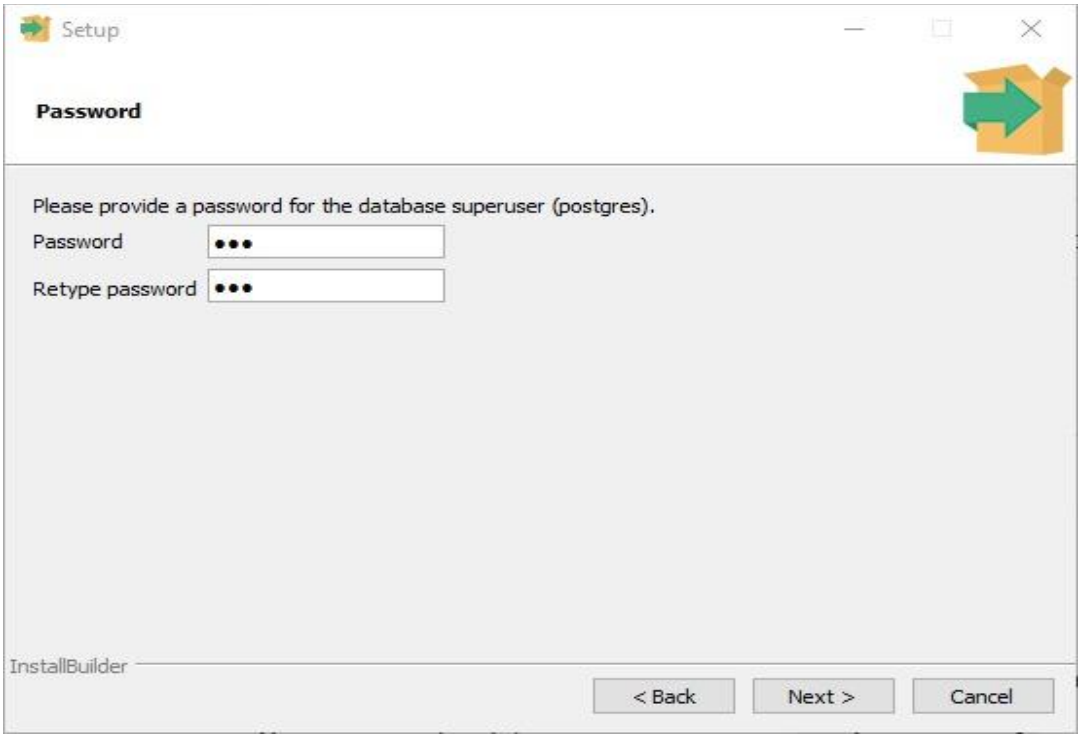
- Python
- Node.js
- Yarn
- Git – Install and Setup (https://docs.github.com/en/get-started/git-basics/set-up-git)
- DVC (https://dvc.org/doc/install)
- If you have a GPU, install cuda toolkit (https://developer.nvidia.com/cuda-downloads)

2. PostgreSQl

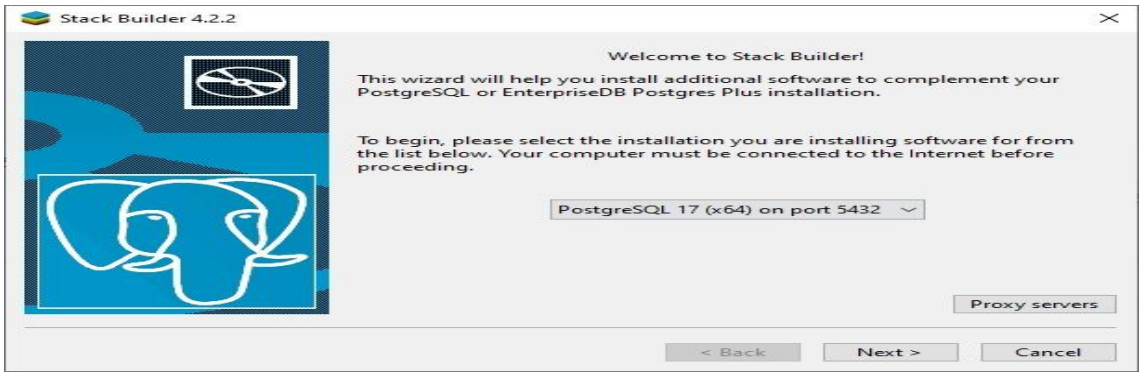
1. Go to the official PostgreSQL download page to initialize the installation:

   https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

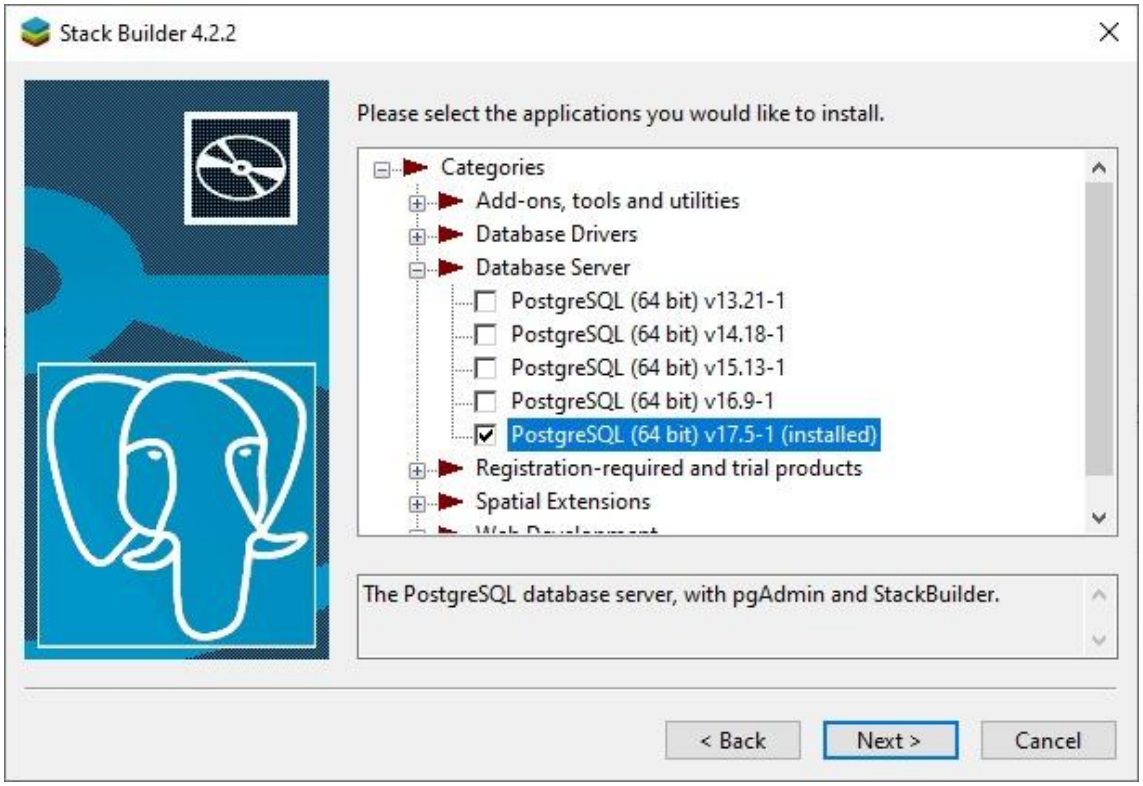
2. Once the installer is open, make sure to set a simple password, for example 123:



3. Select this option (we recommend choosing port 5432):



4. Select this option:



3. Step by Step Initialization

In this section, you will find a step-by-step guide on how to initialize the application — from cloning the repositories to running the application locally.

1. **Create an empty folder**
   Create a new empty folder where all the Git repositories will be placed.
2. **Open the terminal and navigate to the folder path**
   Use the terminal to navigate to the folder you just created:
   cd path/to/your/folder
3. **Ask for an admin for access**
   Request access to the repositories by asking an admin to send you an invitation.
4. **Clone the repositories into the folder**

After receiving access, run the following commands to clone each of the required repositories:
git clone https://github.com/DouglasSimpala/back-end-PFE
git clone https://github.com/Gustavoms7/Classificador
git clone https://github.com/Pedroivo1600/dvc-ABB
git clone https://github.com/pfeinsper/abb-web

5. **Run the back-end repository**
   1. Use the terminal to navigate to the back-end repository and then go to the branch "release":

      cd path/to/your/folder/back-end-PFE

      git checkout origin/release

   2. Run the following command on terminal to install the dependencies

      pip install -r requirements.txt

   3. On the root of the back-end repository, create a ".env" file, and add the following environment variables:

```
ADMIN_DATABASE_URL =
"postgresql://postgres:{password}@localhost:{port}/postgres"

#On the path above, you must use the password and the port
#configured during the PostgreSQL installation. For example:
#postgresql://postgres:123@localhost:5432/postgres

DATA_SEND_TO_DVC = "path/to/your/folder/back-end-
PFE/app/data_send_to_dvc"

DATA_SCREENSHOTS = "path/to/your/folder/back-end-PFE/app/data"
ANNOTATIONS_DIR = "path/to/your/folder/back-end-
PFE/app/data_send_to_dvc/annotations"

IMAGES_DIR = "path/to/your/folder/back-end-
PFE/app/data_send_to_dvc/images"

DVC_REPO_PATH = "path/to/your/folder/dvc-ABB"

CLASSIFIER_PATH = "path/to/your/folder/Classificador"

DATA_YAML_PATH = "path/to/your/folder/Classificador/data.yaml"

DATA_DVC_PATH = "path/to/your/folder/dvc-ABB/data"
```

4. Use the terminal to navigate to the repository where the model is trained and then go to the branch “release”:

   cd path/to/your/folder/Classificador

   git checkout origin/release

5. On the root of the “Classificador” repository, create a “.env” file, and add the following environment variables:

```
CLASSIFIER_PATH = "path/to/your/folder/Classificador"

DATA_YAML_PATH = "path/to/your/folder/Classificador/data.yaml"
```

6. Navigate back to the back-end repository:

   cd path/to/your/folder/back-end-PFE

7. Run the following command:

   uvicorn main:app –-reload

8. Navigate back to the back-end repository and go to the app folder using the following command:

   cd path/to/your/folder/back-end-PFE/app

9. Run the following command to run the application:

   fastapi dev .\main.py

6. **Run the front-end repository**
   1. Open another terminal to navigate to the front-end repository and go to the branch “release”:

      cd path/to/your/folder/abb-web

      git checkout origin/release

   2. Run the following command on terminal to install the dependencies:

      yarn install

3. As soon as the dependencies are installed, run the following command to run the front-end application:

   yarn dev

4. Open your browser and go to the application URL, which should be localhost:3000

7.9 Guia de uso do DVC

DVC Quick Usage Guide – VisionOps

Learn the basics of DVC

ABB

São Paulo
2025

1. Introduction

This document provides a basic introduction to **DVC (Data Version Control)**, a powerful tool designed to simplify data and model management in machine learning projects. Whether you're working alone or as part of a team, DVC helps you organize large datasets, track model versions, and structure your workflows in a reproducible way.

Throughout this guide, you'll learn how to set up DVC in a project, track files that are too large for Git, connect to a local storage, and version your datasets. No prior experience with DVC is required — this document is intended for developers working on the **VisionOps** project. The versioning logic of the software was designed based on the core features of **DVC** demonstrated throughout this guide.

2. Step by Step of How to Initialize DVC and Version Datasets

In this section, we provide a step-by-step guide on how to initialize DVC and begin versioning datasets.

1. **Create an empty folder where you will learn how to use DVC**

2. **Open the terminal and navigate to the folder path**
   cd path/to/your/folder

3. **Install dvc (you can use pip):**
   https://dvc.org/doc/install

4. **Run the following command to initialize GIT:**

git init

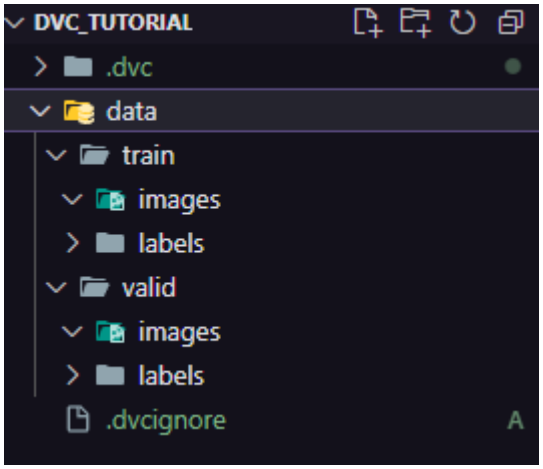
5. **Run the following command to initialize DVC:**
   dvc init
6. **Create an empty folder where your files will be stored locally**

7. **Run the following command to create a local storage:**
   dvc remote add -d local-remote “path/to/your/folder”

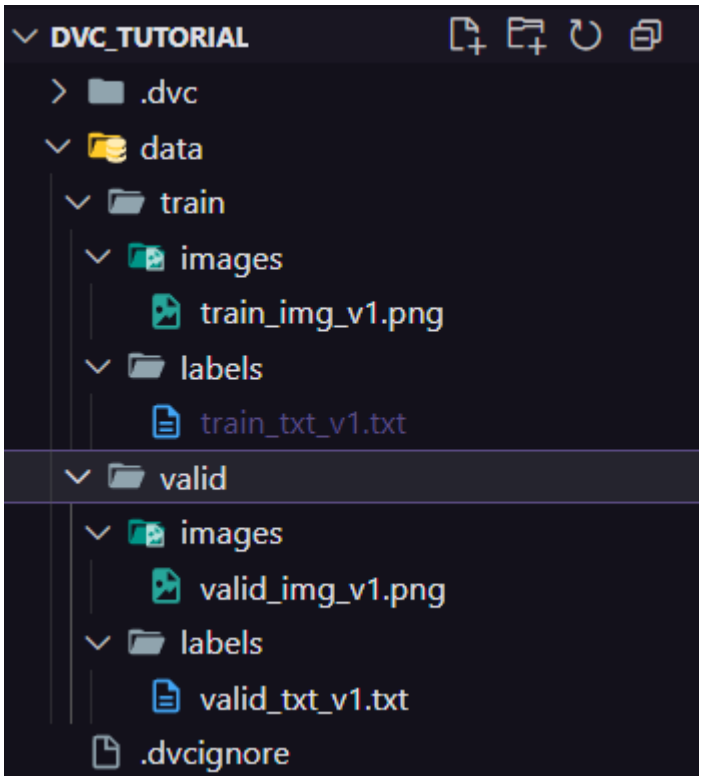
8. **On the root of your folder, create a new folder named “data”.**

9. **Let´s simulate the creation of a dataset used to train a YOLO model:**
   - Inside your data folder, create a new folder named “train”
   - Again, inside your data folder, create a new folder named “valid”
   - Inside both “train” and “valid” folders, create a folder named “images” and another one named “labels”.

10. **After finishing the previous steps, your project should look like this:**



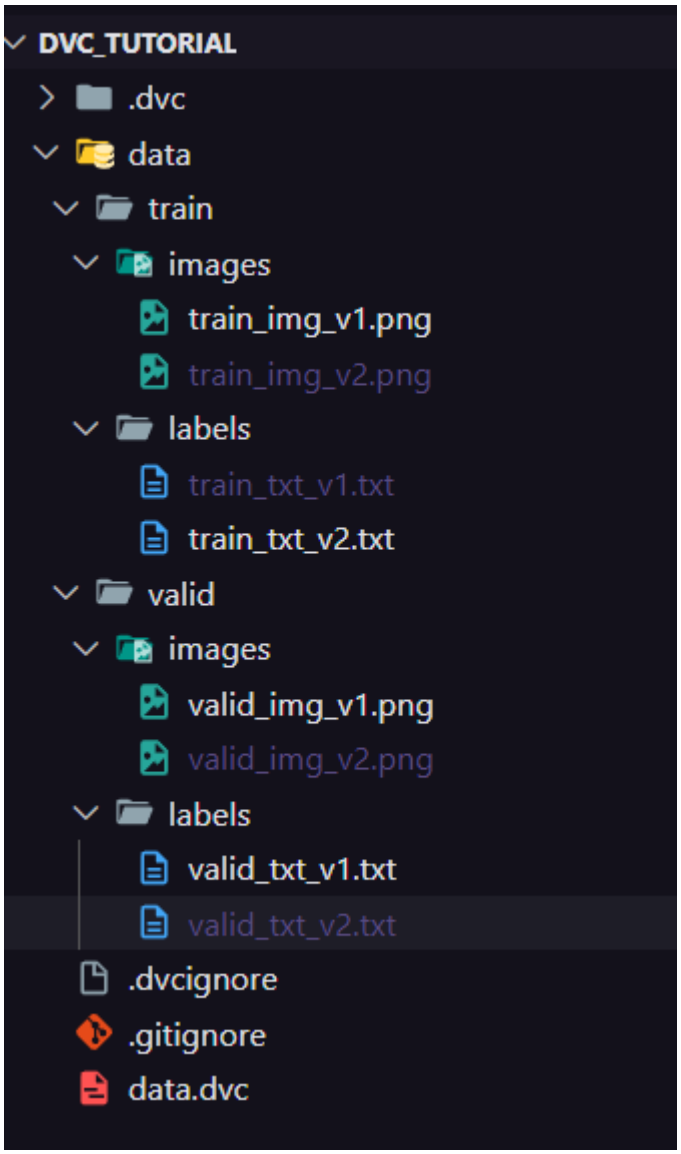
11. **In each folder, add a file, for example:**



12. **Run the following command to start tracking the data folder:**
    dvc add data

13. **Run the following commands (in order) to commit the changes and create a first version of the dataset:**
    - git add – A
    - git commit -m “v1.0 dataset”
    - git tag -a dataset-v1.0 -m “v1.0 dataset”
    - dvc push

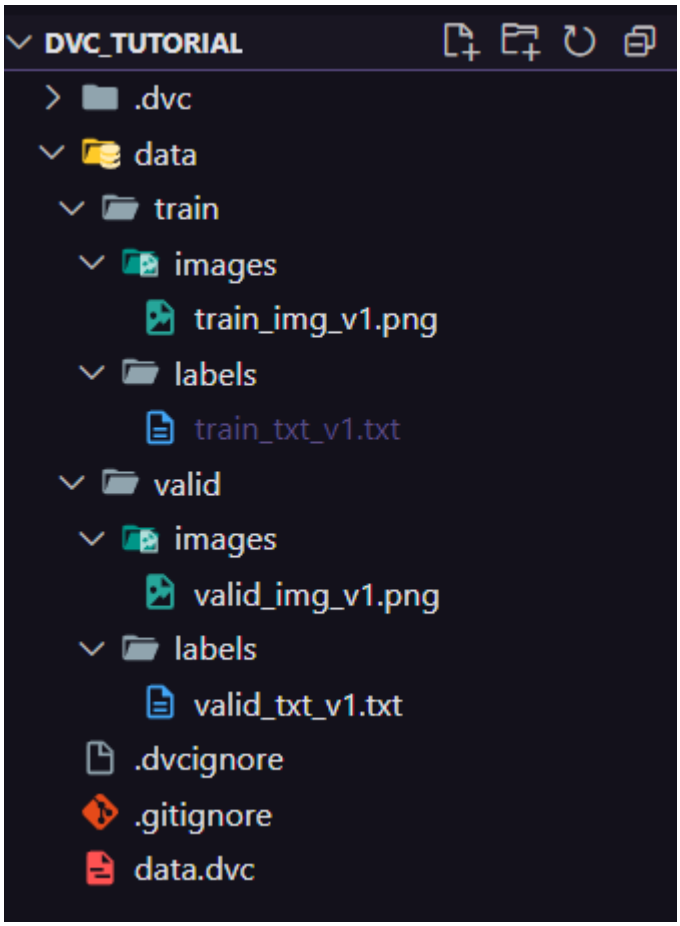
14. **Add new files to the dataset folders to simulate the creation of a second version of your dataset:**



15. **Run the following commands (in order) to commit the changes and create a second version of the dataset:**
    - git add – A
    - git commit -m "v2.0 dataset"
    - git tag -a dataset-v1.0 -m "v1.0 dataset"
    - dvc push

16. **Now, both versions v1.0 and v2.0 are created. To navigate between versions dataset-v1.0 and dataset-v2.0, use the following commands:**
    - git checkout dataset-v1.0
    - dvc checkout

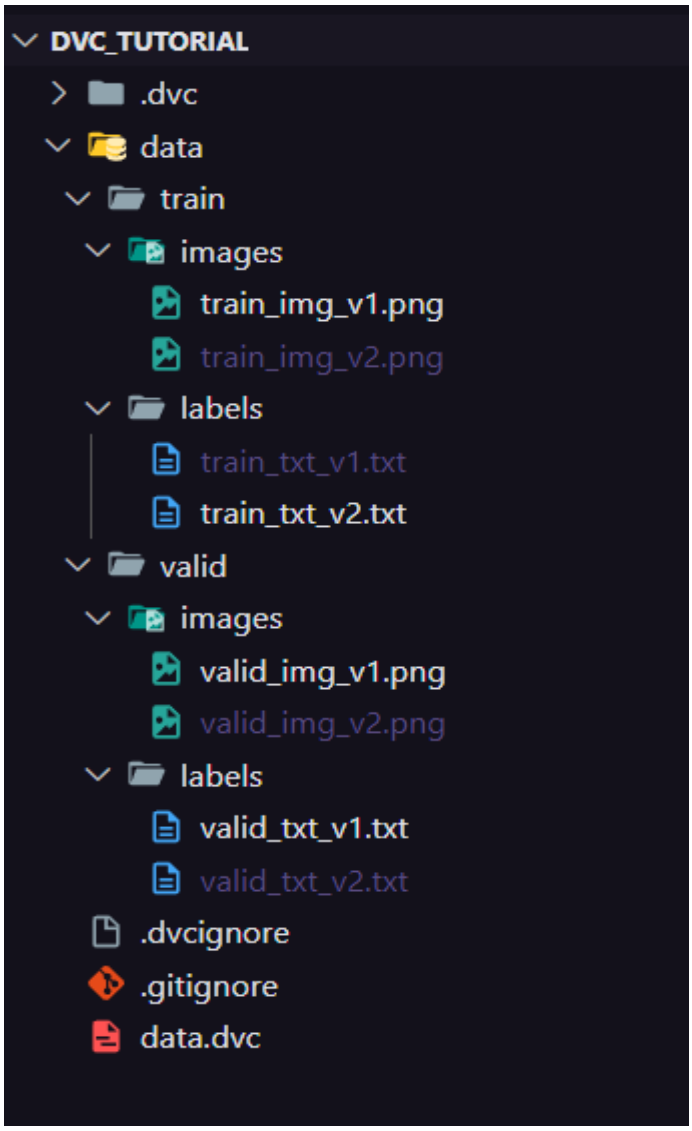
    After running the commands above, your files will look like this:



Now, to go to the version dataset-v2.0, run:

- git checkout dataset-v2.0
- dvc checkout

Your files will now look like this:



17. If you run the command "git tag", you will be able to see all the versions of your dataset

```
(base) PS C:\Users\pedro\OneDrive\dvc_tutorial> git tag
dataset-v1.0
dataset-v2.0
```

3. Conclusion

In this document, we covered the fundamentals of how to version datasets using DVC. You learned how to initialize DVC in a project, add datasets to version control, and navigate between different dataset versions using DVC commands. The file data.dvc plays a key role in tracking the data files, storing metadata that links your data to specific versions.

For more advanced usage and deeper understanding, refer to the official DVC documentation: https://dvc.org/doc.