

«Título do relatório»

«Nome da organização»

20XX / 20XX

«Número(s)» «nome(s) do(s) aluno(s)»



«Título do relatório»

«Nome da organização»

200X / 200X

«Número(s)» «nome(s) do estudante»



Licenciatura em Engenharia Informática

«Mês e Ano de apresentação»

Orientador ISEP: **«nome(s)»**

Supervisor Externo: **«nome(s)»**

«*Dedicatória*»

Agradecimentos

Nesta secção (opcional) colocam-se notas de agradecimento às pessoas que contribuíram para a realização da tarefa. No caso de terem usufruído de financiamento via bolsa ou qualquer outro mecanismo formal, deverá ser incluído um agradecimento.

Resumo

O resumo do relatório (que só deve ser escrito após o texto principal do relatório estar completo) é uma apresentação abreviada e precisa do trabalho, sem acrescento de interpretação ou crítica, escrita de forma impessoal, podendo ter, por exemplo, as seguintes três partes:

1. Um parágrafo inicial de introdução do contexto e do problema/objetivo do trabalho.
2. Resumo dos aspetos mais importantes do trabalho descrito no presente relatório, que por sua vez documenta abordagem adotada e sistematiza os aspetos relevantes do trabalho realizado durante o estágio. Deve mencionar tudo o que foi feito, por isso deve concentrar-se no que é realmente importante e ajudar o leitor a decidir se quer ou não consultar o restante do relatório.
3. Um parágrafo final com as conclusões do trabalho realizado.

**Palavras-chave (Tema):** Incluir 3 a 6 palavras/expressões chave que caraterizem o projeto do ponto de vista de tema/área de intervenção.

**Palavras-chave (Tecnologias):** Incluir 3 a 6 palavras/expressões chave que caraterizem o projeto do ponto de vista de tecnologias utilizadas.

**(O Resumo só deve ocupar 1 página, cerca de 20 linhas**)

Índice

[1 Introdução 1](#_Toc5805924)

[1.1 Enquadramento/Contexto 1](#_Toc5805925)

[1.2 Descrição do Problema 1](#_Toc5805926)

[1.3 Estrutura do relatório 2](#_Toc5805927)

[2 Estado da arte 3](#_Toc5805928)

[2.1 Trabalhos relacionados 3](#_Toc5805929)

[2.2 Tecnologias existentes 3](#_Toc5805930)

[3 Análise e desenho da solução 5](#_Toc5805931)

[3.1 Domínio do problema 5](#_Toc5805932)

[3.2 Requisitos funcionais e não funcionais 5](#_Toc5805933)

[3.3 Desenho 5](#_Toc5805934)

[4 Implementação da Solução 7](#_Toc5805935)

[4.1 Descrição da implementação 7](#_Toc5805936)

[4.2 Testes 7](#_Toc5805937)

[4.3 Avaliação da solução 7](#_Toc5805938)

[5 Conclusões 9](#_Toc5805939)

[5.1 Objetivos concretizados 9](#_Toc5805940)

[5.2 Limitações e trabalho futuro 9](#_Toc5805941)

[5.3 Apreciação final 9](#_Toc5805942)

[Referências 11](#_Toc5805943)

[Anexo A Conteúdo em anexos 13](#_Toc5805944)

[Anexo B Regras de Conteúdo e Estrutura 14](#_Toc5805945)

[B.1 Linguagem 14](#_Toc5805946)

[B.2 Formatação 15](#_Toc5805947)

[B.3 Imagens e tabelas 15](#_Toc5805948)

[B.4 Referências 17](#_Toc5805949)

[Anexo C Plágio 19](#_Toc5805950)

Índice de Figuras

Figura 1 - Exemplo de imagens a) difícil leitura; b) fácil leitura (Sousa 2002) 16

Figura 2 - Exemplo de lista de Referências (Sousa, 2002) 18

**(O documento não pode exceder as 70 páginas, contando desde a primeira página da Introdução até à última página da Conclusão)**

Índice de Tabelas

Tabela 1 - Exemplo de tabela 16

Notação e Glossário

|  |  |
| --- | --- |
| **CPU** | Processador |
| **PHP** | Linguagem de programação... |
| **SO** | Sistema operativo |
| **WS** | Serviço Web (do inglês *Web service*) |

«Esta secção (glossário de termos) apresenta os conceitos ordenados alfabeticamente, símbolos matemáticos e acrónimos **utilizados** no corpo do relatório.»

1. Introdução

A poluição atmosférica constitui um dos maiores desafios ambientais e de saúde pública da atualidade. O aumento da industrialização, do tráfego urbano e da queima de combustíveis fósseis contribui de forma significativa para a libertação de gases tóxicos e partículas em suspensão. A monitorização rigorosa destes poluentes torna-se essencial para prevenir riscos à saúde humana e reduzir impactos ambientais. [1]

A investigação recente tem explorado o uso de sensores de baixo custo, técnicas de visão computacional e métodos de inteligência artificial como soluções promissoras para a deteção e caracterização da poluição. Este trabalho insere-se neste contexto, propondo uma abordagem híbrida que combina dados captados por sensores ambientais com imagens associadas a cada recolha. O objetivo é melhorar a precisão da deteção de fontes de poluição visível, como fumo ou emissões industriais, recorrendo a modelos de aprendizagem profunda. [2]

Adotou-se uma abordagem baseada em visão computacional, complementada com variáveis contextuais recolhidas por sensores, permitindo a criação de um modelo multimodal. Os principais contributos incluem: Integração de sensores físicos e dados visuais num único pipeline; Desenvolvimento de um modelo de aprendizagem automática capaz de correlacionar leituras ambientais e imagens; Proposta de um sistema que combina simplicidade de implementação e potencial de expansão.

* 1. Enquadramento/Contexto

O projeto insere-se na área da monitorização ambiental e do uso de tecnologias digitais emergentes para avaliação da qualidade do ar. A motivação surge da necessidade crescente de soluções acessíveis e escaláveis para a deteção precoce de fontes de poluição. O uso de sensores de baixo custo permite recolher dados em tempo real, enquanto a integração com modelos de visão computacional aumenta a capacidade de caracterização das fontes poluentes.

A escolha desta temática resulta da relevância social e ambiental do problema e da oportunidade de contribuir para a criação de ferramentas que possam ser aplicadas em diferentes contextos urbanos ou industriais. Ao nível pessoal, os tópicos abordados contribuem para uma possibilidade de expansão de conhecimento sobre tecnologias de visão computacional e possibilidade de desenvolver um trabalho com sistemas integrados e circuitos eletrônicos. Esses factos aliados com a capacidade de expansão do projeto numa solução inovadora tornam o projecto muito apelativo.

* 1. Descrição do Problema

A monitorização da poluição encontra limitações quando baseada apenas em sensores químicos, dado que estes não permitem identificar visualmente a origem das emissões. Por outro lado, a visão computacional aplicada de forma isolada carece de dados complementares que permitam contextualizar as imagens e evitar interpretações ambíguas. O problema em estudo centra-se, assim, na ausência de soluções integradas que combinem leituras sensoriais com informação visual, assegurando uma análise mais robusta da qualidade do ar.

* + 1. Objetivos
* Desenvolver um sistema capaz de capturar leituras ambientais (CO₂, CO, PM2.5, temperatura, humidade, VOCs) e associá-las a imagens do local.
* Criar um conjunto de dados multimodal com exemplos reais para treino e validação de modelos de aprendizagem automática.
* Treinar e otimizar um modelo de visão computacional multimodal para identificação de fontes de poluição.
* Avaliar o desempenho do modelo em diferentes cenários, comparando-o com abordagens tradicionais baseadas apenas em sensores ou apenas em imagens.
* Integrar os dados numa interface.
* Elaborar um tutorial para configuração do sistema.
  + 1. Abordagem

A abordagem assenta na recolha de dados através de um sistema híbrido composto por sensores ambientais (ligados a Raspberry Pi e Arduino) e imagens capturadas de forma síncrona. Foi adotada uma metodologia de aprendizagem multimodal, em que os dados de sensores são integrados com as representações extraídas das imagens através de redes neurais convolucionais. [3] [4]

O modelo de base segue práticas descritas na literatura, utilizando arquiteturas pré-treinadas (MobileNetV2) para a componente visual e camadas densas para integração das variáveis sensoriais. [3] [4] Esta abordagem combina a robustez da visão computacional com a precisão dos sensores, reduzindo erros de classificação e aumentando a interpretabilidade dos resultados.

* + 1. Contributos

Os principais contributos deste trabalho incluem:

* Proposta de uma solução multimodal inédita no contexto de monitorização de poluição visível.
* Desenvolvimento de um dataset original que integra imagens e dados sensoriais de outros datasets com dados originais.
* Criação de um modelo leve e otimizado para execução em dispositivos de baixo custo.
* Demonstração do valor acrescentado da integração de sensores e visão computacional para deteção de poluição.
* Potencial contributo social e ambiental, fornecendo uma ferramenta acessível para monitorização e sensibilização.
  + 1. Planeamento do trabalho

O planeamento do trabalho foi organizado da seguinte forma (actividade, data de inicio, data de fim):

* Pesquisa e revisão bibliográfica 23/02/2025 09/03/2025
* Aquisição de equipamento 09/03/2025 28/07/2025
* Calibração de sensores e configuração das câmaras para captura de imagens 28/07/2025 25/08/2025
* Desenvolvimento do modelo de visão computacional 25/08/2025 30/08/2025
* Integração de sensores e IA 30/08/2025 31/08/2025
* Desenvolvimento da interface 31/08/2025 02/09/2025
* Testes com sensores e câmaras em diferentes ambientes 31/08/2025 03/09/2025
* Avaliação e otimização do modelo de IA 31/08/2025 03/09/2025
* Documentação 25/08/2025 06/09/2025
  1. Estrutura do relatório

Para além da introdução, esta dissertação contém mais 5 capítulos. No capítulo 2, é descrito o estado da arte e são apresentados trabalhos relacionados. No capítulo 3, é realizado o desenho e esquematização da solução proposta, com metodologia, técnicas a usar e equipamentos. No capítulo 4, é avaliado o modelo criado e os resultados obtidos da análise em tempo real da solução obtida. No capítulo 5, é realizada uma conclusão que engloba e resume todas as conclusões obtidas do trabalho final.

1. Estado da arte

A monitorização visual de poluição, particularmente fumo, plumas de chaminés e emissões visíveis, tem um leque de aplicações. Nomeadamente, em conformidade regulatória, detecção precoce de incêndios, controlo industrial e vigilância ambiental urbana. Os métodos tradicionais (observação humana, protocolos como EPA 9/22 ou o artigo 192º e 218º da UE) provêm da necessidade histórica de quantificar a opacidade e frequência de emissões. O avanço em visão computacional e a disponibilidade de sistemas de baixo custo (câmaras, Raspberry Pi, sensores de qualidade do ar low-cost) permitem automatizar a observação contínua e a fusão de sinais visuais com leituras de sensores, com ganhos em sensibilidade, custo e escalabilidade.

* 1. Trabalhos relacionados

As preocupações crescentes com as alterações climáticas, a poluição industrial e a saúde pública têm exigido o desenvolvimento de sistemas de monitorização ambiental robustos, escaláveis e em tempo real. Os métodos tradicionais, que frequentemente dependem de sensores físicos (ex.: estações de monitorização da qualidade do ar in-situ), enfrentam limitações de cobertura espacial, custo e velocidade de implementação. Neste contexto, a visão computacional (VC) emergiu como uma ferramenta transformadora, permitindo a análise passiva, de uma área ampla e cada vez mais quantitativa de fenómenos ambientais. Ao obter e processar dados de uma variedade de plataformas de imagem, desde circuitos fechados de televisão (CCTV), veículos aéreos não tripulados (VANTs/drones) e até satélites, as técnicas de VC estão a revolucionar a forma como detetamos, quantificamos e rastreamos as emissões, a deteção de fogos e poluentes.

Uma aplicação bem estabelecida da VC na área da conformidade ambiental é a monitorização de fontes pontuais industriais, como chaminés. A prática comum envolve a utilização de câmaras fixas para capturar vídeo ou imagens de intervalómetro (time-lapse) das plumas. Algoritmos avançados segmentam depois a pluma do fundo, analisando as suas propriedades visuais para quantificar a opacidade, um indicador-chave das emissões de partículas, e para detetar eventos anómalos como fugas ou nuvens de poeira. Isto fornece aos reguladores e operadores de centrais dados contínuos e objetivos para relatórios de conformidade e deteção precoce de fugas, ultrapassando as observações humanas esporádicas. [3] [4]

A gestão de incêndios florestais foi significativamente avançada pela visão computacional, focando-se na deteção precoce de fumo e chamas para minimizar os tempos de resposta. A prática comum aproveita redes de câmaras CCTV e PTZ (Pan-Tilt-Zoom) colocadas estrategicamente, que fornecem uma cobertura contínua de paisagens propensas a fogos. Modelos de VC, frequentemente baseados em redes neuronais convolucionais (CNNs), são treinados para distinguir plumas de fumo de nuvens, nevoeiro ou poeira, e para identificar a assinatura visual única das chamas. [5] Adicionalmente, VANTs/drones equipados com câmaras são utilizados para vigilância aérea, fornecendo dados em tempo real críticos em terrenos complexos. Esta área é suportada por comunidades de investigação ativas e benchmarks que facilitam o desenvolvimento e comparação de algoritmos de deteção. A intersecção entre a IA e as alterações climáticas, destacada por iniciativas como a Climate Change AI, realça a importância destas tecnologias na mitigação de desastres de origem climática. [6]

Uma área de investigação inovadora e em crescimento envolve a utilização de VC para inferir a qualidade do ar urbano, particularmente a poluição relacionada com o tráfego (TRAP). A prática comum, sob o paradigma emergente de "câmaras como sensores", utiliza fluxos de vídeo da infraestrutura urbana existente (ex.: câmaras de tráfego) para estimar os níveis de poluentes. Em vez de medir partículas diretamente, os modelos de VC correlacionam características visuais como a neblina, a degradação da visibilidade, os efeitos de dispersão da luz e até a densidade do tráfego com dados de referência de sensores físicos. [7]

À escala regional e global, a deteção remota por satélite representa a aplicação mais extensiva da VC para a monitorização ambiental. A prática comum envolve a análise de imagens multiespectrais e hiperespectrais de satélites como o Sentinel-2, PlanetScope e MODIS. Os algoritmos de VC são aptos a mapear fenómenos de larga escala: podem monitorizar o transporte de plumas de fumo através de continentes, delinear cicatrizes de queimadas de incêndios florestais, identificar áreas de combustão ativa e estimar volumes de emissões. [8]

Os atuais desenvolvimentos utilizam modelos de *Deep Learning* (DL) para detetar e analisar as imagens obtidas e as bases de dados coletadas através dos mecanismos explicados anteriormente. Os modelos têm como base classificação, deteção de objectos, segmentação, séries temporais, transformadores e modelos fundamentais.

A classificação de imagem completa constitui a abordagem mais direta e rápida, ideal para a triagem (*screening*) inicial de grandes volumes de dados oriundos de câmaras de vigilância ou satélites. As arquiteturas ligeiras e eficientes, como MobileNet e ResNet, mantêm-se extremamente úteis para implementação nos dispositivos de *edge computing*, onde os recursos computacionais são limitados. [9]

Para aplicações que exigem não só a deteção mas, também a localização espacial precisa da pluma dentro de uma imagem, a deteção de objetos é a tarefa preferencial. Esta é a abordagem mais comum em sistemas baseados em CCTV e UAVs (drones), onde é crucial saber onde está o fumo. Modelos da família YOLO (You Only Look Once), nas suas versões v5, v8 e a mais recente v11, dominam o panorama devido ao seu equilíbrio excecional entre velocidade e precisão. De acordo com os estudos, a adaptação do YOLOv8 para a deteção específica de fumo industrial, demonstra uma performance superior em comparação com outros detetores como SSD (Single Shot MultiBox Detector) e Faster R-CNN em contextos de tempo real. [9]

Quando o objetivo é a quantificação precisa ou a análise morfológica da pluma, a segmentação semântica, que gera uma máscara pixel a pixel, é o método mais apropriado. Arquiteturas encoder-decoder como U-Net e DeepLab têm sido o pilar desta tarefa. Contudo, o panorama atual foi revolucionado pela introdução dos Modelos de Base (Foundation Models\*) para segmentação. O Segment Anything Model (SAM) da Meta, e a sua variante eficiente FastSAM, emergiram como bases universais de segmentação. [10] [11] Vários estudos realizados demonstram o *fine-tuning* e aplicação específica do FastSAM para a segmentação de alta precisão de plumas de fumo, aproveitando a sua robustez e capacidade de generalização. [10] [11]

A natureza dinâmica do fumo torna a análise temporal crucial para reduzir falsos positivos, particularmente para distinguir plumas de fumo de elementos visuais estáticos ou em movimento lento, como nuvens e nevoeiro. Para tal, arquiteturas capazes de modelar a informação espaço-temporal são essenciais. Abordagens que combinam CNNs 3D ou redes CNN-LSTM (Long Short-Term Memory) são amplamente utilizadas para extrair características espaciais e modelar a sua evolução temporal. Um exemplo paradigmático é a arquitetura SmokeyNet, especificamente concebida e treinada em conjuntos de dados como o FIgLib (Fire Ignition Library), que demonstra uma elevada eficácia na deteção de incêndios incipientes através da análise do movimento do fumo em sequências vídeo. [12]

Existem diversos estudos que estudam e aplicam os métodos e técnicas abordados. Um dos estudos propõe pipelines com CNNs e etapas de machine learning para classificar/estimar níveis de poluentes a partir de fotos urbanas. [13] Outro trabalho utiliza vídeos de tráfego para extrair variáveis relacionadas ao fluxo veicular como proxies para poluentes relacionados ao tráfego (TRAP), produzindo previsões em tempo real. [14] Outro estudo aplica arquiteturas profundas (ex.: ResNet) a imagens móveis (smartphones) para estimar valores de PM e NO₂. [15] Um estudo realizado em Nanjing, usa câmeras de vigilância que exploram informação espaço-temporal contígua para melhorar robustez das estimativas de monitorização da poluição do ar. [16] Um estudo com tecnologia mais dispendiosa consiste em sistemas baseados em UAVs que visam alta resolução espacial através de sensoriamento móvel e discutem desafios de calibração e fusão com medidas de referência. [17]

Ao comparar os estudos para procurar semelhanças nas técnicas e métodos usados verifica-se que os artigos cobrem três categorias de entrada: imagens estáticas, vídeo contínuo e imagens móveis (UAV/smartphone). Imagens estáticas são simples de recolher mas muito sensíveis a variações de iluminação, hora do dia e condições meteorológicas. Vídeo oferece continuidade temporal que ajuda a filtrar ruído instantâneo e a capturar picos relacionados a eventos (ex.: engarrafamentos), ao custo de maior processamento. UAVs e dispositivos móveis aumentam a cobertura espacial, mas introduzem variações de ângulo e de calibração que exigem pipeline de normalização complexa. [13] [14] [15] [16] [17]

Em termos de modelos e arquiteturas, predominam CNNs para extração de características visuais; alguns trabalhos combinam CNNs com componentes temporais (p.ex. ConvLSTM, LSTM) para gravar dependências temporais. Em termos de maturidade, observa-se a mesma evolução vista noutros domínios: arquiteturas puramente espaciais cedem lugar a modelos híbridos spatio-temporais ou multimodais quando se deseja produção operacional. [13] [14] [15] [16] [17]

Ao nível de supervisão dos modelos, todos os estudos dependem de medidas de referência (estações fixas, sensores low-cost) para rotular imagens. A sincronização temporal e espacial entre a imagem e a medição é uma fraqueza repetida: delays, diferenças de distância e variações microclimáticas podem introduzir ruído ou viés nos rótulos, prejudicando a generalização. [13] [14] [15] [16] [17]

Em termos de vantagens e desvantagens, a infra-estrutura de câmeras urbanas ou smartphones pode reduzir os custos em cenários com poucas estações de referência. As imagens e vídeo capturam efeitos visuais (partículas em suspensão) que não aparecem diretamente em satélite de baixa resolução. Com pipelines de vídeo/edge computing podem fornecer estimativas locais em quase tempo real, úteis para alertas imediatos. Algumas das limitações são a relação entre indicadores visuais (visibilidade, tonalidade do céu) e concentrações químicas (PM2.5, NO₂) é imperfeita. A existência de diferentes condições meteorológicas podem produzir sinais visuais semelhantes para concentrações díspares. Alguns dos modelos treinados numa cidade/estação frequentemente degradam noutras localidades por diferenças arquitetónicas, clima, e composição das fontes. [13] [14] [15] [16] [17]

A prática dominante atual combina redes de estações de referência, sensores low-cost, dados satelitais e modelos spatio-temporais (por exemplo GCNs, ConvLSTM) para mapear e prever poluição. Os estudos baseados exclusivamente em imagens apresentam provas de conceito valiosas mas, na literatura mais avançada, as imagens tendem a ser usadas como **uma** das várias entradas num sistema multimodal e não como substituto direto das medições.

* 1. Tecnologias existentes

O objetivo é monitorizar, avaliar e quantificar os índices de poluição atráves de VC. Uma outra vertente dos objectivos propostos é fazê-lo com uma solução de baixo custo económico e tecnológico para ser fácil de implementar. Nesta vertente, existem alguns trabalhos realizados que evidênciam as tecnologias existentes e utilizadas em casos de sistemas de baixo custo.

Bogdanffy et al. (2025) e Zhao et al. (2023) destacam-se pela sua validação em ambiente real e atenção aos desafios de engenharia. Bogdanffy et al. propuseram uma plataforma para monitorização urbana que combina de forma pragmática a contagem de veículos via object detection (usando bibliotecas standard) com leituras de sensores de PM de baixo custo, implementada em nodes de edge computing (ex: Raspberry Pi). [18] A sua principal contribuição reside na demonstração de custo-efetividade e na arquitetura de pipeline robusta (aquisição local para agregação para cloud), validada através de comparação com estações oficiais. Da mesma forma, Zhao et al. focaram-se no ambiente interior, projetando e implementando um detector multi-parâmetro estável para poluentes como CO₂, PM e VOC. [19] A sua força reside na minuciosa avaliação da estabilidade a longo prazo (um ano) e no design modular de hardware. Contudo, uma limitação comum a ambos os estudos é o foco na plataforma de aquisição e fusão de dados, em detrimento da exploração de algoritmos de visão computacional de estado da arte (SOTA) para análise aprofundada da imagem (ex.: segmentação de plumas).

O trabalho seminal de Zhang et al. (2020) forneceu a prova de conceito crucial, demonstrando que funcionalidades extraídas por CNNs de imagens de telemóvel podiam ser correlacionadas com medições de PM2.5. [20] A sua abordagem leve, utilizando transfer learning, abriu caminho para aplicações crowdsourced. No entanto, a dependência de uma imagem única torna-o vulnerável a variáveis de confusão como condições de iluminação e meteorologia.

Avançando significativamente nesta linha, o mais recente trabalho de 2025 propõe o AQP-Mamba, um modelo de deep learning espaço-temporal que representa o estado da arte em métodos video-based. [21] Ao contrário de Zhang et al., que usavam imagens estáticas, o AQP-Mamba processa sequências de vídeo, capturando a dinâmica temporal do haze e do movimento das partículas. A sua arquitetura inovadora, que integra um Structured Selective State Space Model (SSM), é especificamente desenhada para modelar dependências de longo prazo em séries temporais vídeo, melhorando substancialmente a robustez sobre métodos baseados em imagens únicas. A sua principal força é a demonstração de que a riqueza da informação temporal no vídeo pode superar as limitações de iluminação e oferecer estimativas mais fiáveis de AQ, porventura aproximando-se da precisão de sistemas híbridos. [21] A principal contrapartida é o custo computacional inerente ao processamento de vídeo, levantando desafios para a implantação em edge sem otimizações avançadas.

Tabela 1: Comparação entre diferentes estudos e solução proposta.



A tabela 1 compara os estudos de equipamentos de baixo custo com a nossa solução proposta. A abordagem multimodal adotada na nossa solução destaca-se pela simplicidade e eficácia comprovada. Através de uma estratégia de late fusion, combinam-se funcionalidades extraídas de imagens por uma rede MobileNetV2 com dados de sensores de PM, CO₂, CO e NO3 processados por uma MLP. Esta combinação permite aproveitar sinais complementares: em casos onde a imagem é ambígua (e.g., distinção entre nuvem, vapor ou fumo), as leituras dos sensores tendem a esclarecer a natureza da pluma, aumentando a robustez do sistema.

A arquitetura é leve e executável em dispositivos de edge computing, como o raspberry pi. A utilização de uma MobileNetV2 pré-treinada e congelada para extração de características, seguida de um classificador simples, garante eficiência computacional. A conversão do modelo para TorchScript permite uma execução razoavelmente rápida num Raspberry Pi mesmo sem utilização de GPU. Esta eficiência contrasta favoravelmente com soluções baseadas em vídeo e modelos espaço-temporais mais complexos, como o AQP-Mamba mencionado na literatura, que exigem substancialmente mais recursos computacionais.

O pipeline implementado é prático e baseado em hardware acessível. A solução recorre a um Arduino para aquisição de dados de sensores analógicos, uma opção robusta e económica que evita a necessidade de conversores ADC adicionais, e a um Raspberry Pi para aquisição de imagem e inferência. Esta configuração hardware é amplamente validada em trabalhos similares, como os descritos por Bogdanffy et al. e Zhao et al., demonstrando ser suficiente para prototipagem e implementação de sistemas integrados de monitorização.

Em termos de limitações, relaciona-se com o não aproveitamento de informação temporal de vídeo. Trabalhos recentes, como o AQP-Mamba, recorrem a State Space Models (SSMs) ou redes 3D-CNN para modelar clips de vídeo, capturando a dinâmica temporal do movimento das plumas. Esta modelação temporal pode reduzir significativamente os falsos positivos, particularmente em casos onde o movimento é um fator discriminativo crucial (e.g., fumo vs. névoa). Para cenários com câmaras capazes de capturar vídeo e recursos computacionais superiores, a incorporação de temporalidade seria uma melhoria natural.

Outra limitação reside na ausência de segmentação pixel-wise. Enquanto a solução se foca na classificação de imagem completa ou regressão de valores, abordagens baseadas em U-Net, DeepLab, ou modelos de base como o Segment Anything Model (SAM) permitem obter máscaras de segmentação precisas da pluma. Estas máscaras são indispensáveis para estimativas quantitativas de parâmetros como opacidade ou área de dispersão, aproximando-se dos protocolos standard como o Método 9 da EPA. Se a métrica de opacidade for um requisito, a segmentação é incontornável.

Adicionalmente, a dependência de sensores de baixo custo introduz um desafio de calibração. Tal como destacado nos estudos analisados, a fiabilidade das leituras destes sensores está diretamente ligada a processos rigorosos de calibração e comparação com equipamentos de referência. Sem esta etapa, o modelo de machine learning pode aprender enviesamentos sistemáticos presentes nos dados brutos, comprometendo a generalização e precisão das previsões. É, portanto, imperativo implementar rotinas de calibração e normalização dos sinais dos sensores antes do treino.

A opção da solução escolhida privilegia a simplicidade, eficiência e integração multimodal e está diretamente alinhada com o objetivo principal do projeto: criar uma solução prática, operável num Raspberry Pi, e de implementação rápida. O trade-off consciente entre complexidade e desempenho é explicitamente recomendado pela literatura para sistemas de edge computing com restrições de recursos.

1. Análise e desenho da solução

Este capítulo permite demonstrar a idealização da solução criada e explicação detalhada da sua construção, implementação e otimização. Serão ainda descritas todas as metodologias e técnicas utilizadas, assim como, todo o equipamento e software.

* 1. Lista de equipamentos, componentes e software

Os equipamentos utilizados na nossa solução são os descritos:

* Raspberry Pi 4 Model B (4GB).
* Arduino Uno, alimenta sensores analógicos e comunica por USB com o Pi.
* Cartão microSD (32GB+), sistema operativo, armazenamento local com SQLite/CSV para logs.
* Fonte de alimentação, 5V para Raspberry Pi e fonte externa 7–12V para Arduino e bateria de 5V para potenciar ligaçao a um sensor.
* Cabos USB A-B (Arduino para Pi), cabos jumper dupont, fios para alimentação.

Ao nível de sensores:

* DFRobot SEN0159 (MG811) — Sensor de CO₂
* DFRobot SEN0132 (MQ7) — Sensor de CO
* DFRobot SEN0248 (BME680) — Sensor de VOC/Temp/Hum/Press
* DFRobot SEN0233 — Sensor de PM2.5
* Raspberry Pi Camera v2.0— captura imagens do local (imagem única por evento ou frames quando necessário).

Para software e utilitários:

* Raspbian / Raspberry Pi OS
* Python 3 com librarias: pyserial, adafruit-circuitpython-bme680, torch (runtime), torchvision, numpy, pandas, paho-mqtt, flask/fastapi (opcional).
* Arduino IDE
* Servidor de treino (PC com GPU) com PyTorch completo, scikit-learn, Optuna, MLflow/W&B, DVC (para dataset versioning).

Extras (opcionais):

* Coral Edge TPU / Hailo / Jetson — acelerar inferência se necessário.
* Broker MQTT / InfluxDB / TimescaleDB / S3 — para escalabilidade.
  1. Ligações físicas e esquema de construção

As ligações dos equipamentos são descritas através das especificidades de cada conexão.

Raspberry Pi 4 (conector de 40 pinos — usamos números físicos):

Pino 1 (3V3) → VCC BME680 (SEN0248)

Pino 3 (GPIO2 SDA1) → SDA BME680

Pino 5 (GPIO3 SCL1) → SCL BME680

Pino 6 (GND) → GND BME680 (partilhar com GND comum)

Pino 2/4 (5V) → VCC SEN0233 (PM2.5) (se módulo exigir 5V)

Pino 9/14/10/8 (GND/TX/RX) → GPIO14(TX0)/GPIO15(RX0) do Pi para RX/TX SEN0233:

SEN0233 TX → Pi GPIO15 (pino 10, RX)

SEN0233 RX → Pi GPIO14 (pino 8, TX) (geralmente não é necessário enviar dados para sensor, apenas ouvir)

USB A (porta) → Arduino Uno (USB B) — comunicação serial (ttyACM0 ou ttyUSB0)

Arduino Uno:

A0 → MG811 AO (SEN0159)

A1 → MQ7 OUT (SEN0132)

GND → GND comum (vincular com GND do Pi)

5V → alimentar sensores MQ/MG — mas MG811 recomienda fonte externa 7–12V no jack do Arduino devido ao aquecimento.

USB → Raspberry Pi USB (dados + alimentação opcional)

SEN0233 (PM2.5):

VCC → 5V do Pi (pino 2/4) ou fonte 5V regulada

GND → GND comum

TX → Pi GPIO15 (RX)

RX → Pi GPIO14 (TX) — geralmente não usado

SET/RESET → deixar soltos (normal operation) ou usar para controle

BME680 (SEN0248):

VCC → 3V3 (pino 1)

GND → GND (pino 6)

SDA → GPIO2 (pino 3)

SCL → GPIO3 (pino 5)

Observações:

Se preferir isolamento, usar conversores de nível lógico para TTL serial; no entanto, a SEN0233 indica que os níveis de dados são 3.3V compatíveis.

Antes de qualquer ligação, verificar datasheets e confirmar pinos.

* 1. Descrição do código implementado

Fornecerei três blocos principais de código (resumo + explicação detalhada): (A) Arduino firmware; (B) Raspberry Pi — aquisição e logging / inferência (Python); (C) Scripts de treino (PyTorch multimodal). Também descreverei utilitários auxiliares e arquivos de configuração.

3.3.1 (A) Arduino Uno — Firmware (C/C++)

Objetivo: ler analógicos (A0 MG811, A1 MQ7), aplicar pequena filtragem, enviar leituras por Serial (USB) com formato simples e timestamp relativo.

Explicação e notas:

analogRead retorna 0–1023 (10-bit) no UNO. Estas leituras são "raw" e necessitam escala/calibração.

Média de 5 leituras reduz ruído. Poderia aplicar mediana ou filtro IIR.

Timestamp é millis() relativo ao boot; no Pi será convertido para timestamp absoluto ao receber via USB.

Para MG811 recomenda-se aquecimento e referência; ler documentação para ciclo de aquecimento.

3.3.2 (B) Raspberry Pi — Script principal (Python) para aquisição, logging e inferência

Arquivos principais:

acquire\_and\_infer.py — captura imagem, lê Arduino serial, lê BME680 via I²C, lê SEN0233 via UART, salva imagem e log, executa inferência (TorchScript) e opcionalmente publica alerta via MQTT/HTTP.

Explicações detalhadas:

capture\_image: usa OpenCV (0) para webcam; alternativa: picamera para CSI.

read\_arduino\_line: lê uma linha CSV do Arduino; ao receber, converte timestamp relativo para timestamp absoluto (usar NTP).

read\_pm\_sensor: lê frame UART do SEN0233 e extrai PM1/PM2.5/PM10. Frame parsing inclui checksum validação (implementado no código completo).

read\_bme: usa adafruit\_bme680 para obter T/RH/Press/Gas.

Logs: gravar linha no CSV com timestamp ISO8601 e todos os sensores. Preferível também gravar em SQLite ou enviar para InfluxDB se for necessário.

Inferência: o script carrega multimodal\_traced.pt (TorchScript) e aplica transformações: resize, normalize, transformar vetor sensores (StandardScaler carregado). O TorchScript espera dois tensores (image\_tensor, sensor\_tensor).

Inferência (detalhes do preprocess):

Imagem: PIL.Image.open(img).convert('RGB'), transforms.Resize((224,224)), ToTensor, Normalize(ImageNet stats), unsqueeze(0).

Sensores: preencher NaNs, criar vetor float32 (ordem igual scaler.pkl), scaler.transform([vector]).

Resultado: softmax para probabilidades; output: {'label': 'smoke', 'prob':0.87}.

Alerting e thresholds:

Thresholds configuráveis: ALERT\_PROB = 0.7, ALERT\_PM2\_5 = 100 μg/m3 — lógica: alert if image\_prob>ALERT\_PROB and pm2\_5>ALERT\_PM2\_5 (reduces false positives).

3.3.3 (C) Scripts de treino e exportação (PyTorch — multimodal)

Estrutura:

train\_multimodal.py — já fornecido anteriormente; resumo com foco nos passos:

Carregar CSV com pandas e image\_path colunas.

Dividir stratified train/val/test (70/15/15).

StandardScaler para sensores; salvar scaler.pkl.

Dataset multimodal: \_\_getitem\_\_ retorna (image\_tensor, sensor\_tensor, label\_idx).

Modelo: MobileNetV2 backbone (classifier=Identity) + MLP sensores + final classifier.

Treino: congela backbone, treina cabeça e MLP; usar Adam, LR=1e-3, early stopping.

Export: salvar multimodal\_best.pth e gerar TorchScript via torch.jit.trace(model\_cpu, (dummy\_img, dummy\_sensors)) → multimodal\_traced.pt.

Salvar label\_map.json e scaler.pkl.

Hiperparâmetros padrão:

batch\_size=16, epochs=12, LR=1e-3, weight\_decay=1e-5, dropout=0.5.

Validação e tracking:

Usar MLflow/W&B para logging de métricas; guardar confusion matrix e curvas ROC.

* 1. Técnicas usadas e explicação detalhada

3.4.1 Por que usar Arduino como ADC e não ADS1115?

Motivo prático: não há ADS1115 disponível e o Arduino já tem ADC integrado com entradas A0/A1. Usar o Arduino poupa custos e integrações adicionais.

Contras: ADC do Arduino é 10-bit (0–1023) e refere-se ao Vcc (5V) que limita resolução; ADS1115 daria 16-bit e melhor precisão. Compensação: média, filtro e calibração mitigam limitações.

3.4.2 Escolha do BME680 (SEN0248)

Vantagens: fornece temp, hum, press, VOC/gas — útil para contexto (p.ex. condições meteorológicas), que melhora robustez do modelo visual/regressão. Comunicação I²C direta com Pi é simples e confiável.

3.4.3 SEN0233 para PM2.5 via UART

Fornece PM1/PM2.5/PM10 e tem firmware com framing e checksum — robustez. UART evita usar ADC e poupa canais do Arduino. É natural ler diretamente no Pi pelo serial.

3.4.4 Arquitetura multimodal — MobileNetV2 + MLP (fusão tardia)

MobileNetV2: projetado para edge — poucos parâmetros, boa acurácia transfer learning. Ideal para o Raspberry Pi.

MLP para sensores: sensores entregam vetores numéricos; MLP modela não-linearidades e interações entre sensores.

Fusão tardia (concatenação dos vetores de features) é simples, interpretável e permite treinar as duas torres separadamente. Literatura e artigos analisados indicam que late fusion é robusta para casos com poucos dados multimodais.

Alternativas (e por que não escolhemos primeiro):

YOLO (detecção) seria melhor para localizar plumas mas exige maior anotação (bboxes) e inferência mais pesada.

Segmentação (U-Net) daria área/opacidade, mas requer máscaras custosas e recursos computacionais maiores.

Modelos vídeo (AQP-Mamba) são superiores em dinâmicas temporais; contudo, têm custo computacional elevado e complexidade de dataset.

3.3. Pré-processamento de Dados

O pré-processamento de dados é uma etapa crítica para o sucesso de qualquer modelo de machine learning, assegurando que os dados de entrada estão num formato adequado, são consistentes e relevantes para o modelo aprender padrões significativos. A nossa abordagem multimodal exigiu um pipeline distinto para cada tipo de dado.

Para Imagens: Técnicas de Aumento de Dados, Redimensionamento, Normalização

O pré-processamento do fluxo de imagens foi desenhado para extrair o máximo potencial do Transfer Learning com a MobileNetV2, ao mesmo tempo que introduz robustez para as condições variáveis do ambiente real de operação.

Redimensionamento e Recorte (Center Crop): As imagens capturadas pela câmara foram redimensionadas para 256x256 pixels para depois ser feito um recorte central de 224x224 pixels. Estas dimensões são a input size standard para a MobileNetV2 pré-treinada na ImageNet, permitindo-nos aproveitar integralmente os pesos já aprendidos.

Normalização: Os valores de pixel (originalmente entre 0 e 255) foram normalizados para uma distribuição com média [0.485, 0.456, 0.406] e desvio padrão [0.229, 0.224, 0.225]. Estes valores são os da ImageNet, e a sua utilização é crucial quando se usa Transfer Learning, pois alinha a distribuição dos nossos dados com a distribuição dos dados que o modelo já "conhece".

Aumento de Dados (Data Augmentation): Para combater o overfitting e tornar o modelo invariante a alterações comuns no ambiente, aplicámos técnicas de aumento de dados apenas durante o treino. Estas incluíram:

Variações de Cor (Jitter): Pequenas alterações no brilho, contraste e saturação da imagem para simular diferentes condições de iluminação (sol, nuvens, diferentes horas do dia).

Desfocagem (Blur): Aplicação de um ligeiro desfoque gaussiano para tornar o modelo mais tolerante a imagens menos nítidas ou com movimento residual.

Variações Geométricas: Pequenas rotações aleatórias e flips horizontais. Esta última é particularmente útil, pois não introduz viros na interpretação de fenómenos físicos como o fumo.

Para Dados dos Sensores: Filtragem, Normalização, Sincronização

Os dados dos sensores, sendo séries temporais numéricas, requerem um tratamento diferente para garantir precisão e estabilidade.

Filtragem: Para mitigar o ruído inerente aos sensores de baixo custo, implementámos um filtro de média móvel numa janela temporal curta (ex.: 5 leituras). Isto suaviza picos abruptos causados por interferências momentâneas, resultando num sinal mais limpo e fiável.

Normalização: Os dados dos sensores (CO₂, CO, PM2.5, temperatura, humidade) têm escalas e unidades completamente diferentes (ppm, µg/m³, °C, %). Para evitar que o modelo atribua maior importância a variáveis simplesmente por terem uma escala numérica maior, aplicámos a técnica StandardScaler (normalização Z-score). Esta técnica transforma os dados para ter uma média de 0 e um desvio padrão de 1, colocando todas as características na mesma escala. O scaler foi treinado apenas com os dados de training para evitar data leakage.

Sincronização: Garantir a temporalidade correta entre a imagem e as leituras dos sensores é vital para a fusão multimodal. Cada imagem capturada é carimbada com um timestamp preciso. As leituras dos sensores, que são amostradas a uma frequência superior, são depois interpoladas ou é selecionada a leitura mais próxima desse mesmo timestamp, criando assim um par sincronizado (imagem, vetor de sensores) para cada instante de amostragem.

3.4. Arquitetura do Modelo

A arquitetura do modelo foi meticulosamente escolhida para equilibrar a precisão da deteção com a eficiência computacional, permitindo a sua execução em hardware de baixo custo e limitado, como o Raspberry Pi.

Por que MobileNetV2?

A escolha da MobileNetV2 como backbone para a extração de características (features) visuais deve-se a um conjunto de vantagens decisivas para este projeto:

Eficiência Computacional vs. Precisão: A MobileNetV2 utiliza inverted residual blocks e depthwise separable convolutions, que reduzem drasticamente o número de parâmetros e operações computacionais necessárias, quando comparada com arquiteturas como a ResNet ou VGG. Isto traduz-se em tempos de inferência muito mais rápidos, com uma perda mínima de precisão, tornando-a ideal para aplicações em tempo real.

Capacidade de Transfer Learning a partir da ImageNet: Ao ser inicializada com pesos pré-treinados na ImageNet (um grande dataset de classificação de imagens genéricas), a rede já aprendeu a reconhecer padrões visuais fundamentais (bordas, texturas, formas). Isto permite-nos obter um desempenho excelente mesmo com um dataset relativamente pequeno de imagens de poluição, fine-tunando apenas as camadas finais da rede para a nossa tarefa específica.

Adequação para Dispositivos com Recursos Limitados: A sua designação "Mobile" não é acidental. Esta arquitetura foi especificamente concebida para ser executada em dispositivos edge com restrições de CPU e memória, como é o caso do Raspberry Pi, alinhando-se perfeitamente com os constraints do nosso projeto.

Por que MLP para os Sensores?

Para processar os dados numéricos dos sensores, optou-se por uma rede Multilayer Perceptron (MLP), uma escolha pragmaticamente sólida:

Capacidade de Capturar Relações Não-Lineares: Ao contrário de um modelo linear, uma MLP com uma ou mais camadas ocultas e funções de ativação não-lineares (ex.: ReLU) pode aprender interações complexas entre as diferentes leituras de sensores. Por exemplo, pode aprender que uma combinação específica de alta humidade e moderado PM2.5 tem um significado diferente de PM2.5 alto com humidade baixa.

Simplicidade e Eficiência Computacional: As MLPs são arquiteturas simples, mas poderosas, para dados tabulares. Têm um custo computacional muito baixo, especialmente quando comparadas com arquiteturas complexas como as LSTM, que não se justificavam para este contexto.

Facilidade de Implementação e Treino: A integração de uma MLP no fluxo existente em PyTorch é trivial. A sua natureza feed-forward torna-a estável e fácil de treinar com o algoritmo de backpropagation standard.

Por que Fusão Tardia (Late Fusion)?

A estratégia de fusion escolhida foi a fusão tardia, onde as features de cada modalidade (imagem e sensores) são processadas independentemente e combinadas apenas nas camadas finais da rede.

Permite Treinar as Modalidades Separadamente: Esta abordagem oferece flexibilidade. Podemos, por exemplo, congelar o backbone de imagem e treinar apenas a MLP dos sensores inicialmente, ou vice-versa, o que é útil para debugging e estabilização do treino.

Mais Interpretável que a Fusão Inicial: É mais fácil inspecionar o que cada "perna" da rede está a aprender individualmente. Técnicas como Grad-CAM podem ser aplicadas na perna visual para ver que regiões da imagem estão a influenciar a decisão, enquanto podemos analisar os pesos da MLP para entender a importância relativa de cada sensor.

Robustez a Missing Data numa Modalidade: Se, em produção, um sensor falhar e os seus dados estiverem em falta, a arquitetura pode ser adaptada para realizar inferência usando apenas a modalidade visual (e vice-versa), ainda que com potencial perda de desempenho. Isto é muito mais complexo numa fusão inicial.

Menos Propenso a Overfitting com Datasets Pequenos: A fusão tardia atrasa a interação entre as modalidades para um estágio mais alto e abstrato do modelo, o que tende a ser mais eficiente com conjuntos de dados de tamanho moderado, reduzindo o risco de o modelo aprender correlações espúrias entre pixels específicos e leituras de sensores.

Alternativas Consideradas e Rejeitadas:

Fusão Inicial (Early Fusion): Envolveria a concatenação dos dados dos sensores diretamente com o tensor da imagem numa fase inicial (ex.: nos pixels de entrada). Esta abordagem foi rejeitada porque força a rede a aprender desde o início como combinar dados de naturezas radicalmente diferentes (espaciais vs. numéricos), o que é uma tarefa complexa e geralmente leva a uma perda de informação espacial e estrutural da imagem.

Fusão Intermédia (Intermediate Fusion): Utilizar mecanismos de atenção complexos para fundir features de várias camadas. Embora possa ser mais performante, foi considerada excessivamente complexa para o scope deste projeto, aumentando o risco de overfitting e o custo computacional, sem garantias de benefício significativo.

Abordagens Temporais (LSTM/Transformers): A introdução de camadas recorrentes (LSTM) ou de atenção (Transformers) para modelar a sequência temporal dos dados foi considerada. No entanto, foi rejeitada porque a dinâmica temporal da poluição (em comparação com, por exemplo, a fala ou vídeo) é relativamente lenta e os nossos intervalos de amostragem são curtos. A informação contida numa única frame e no instante correspondente dos sensores foi considerada suficientemente informativa, tornando a complexidade adicional desnecessária para o problema em questão.

3.5. Estratégia de Treino do Modelo

O treino do modelo multimodal seguiu um protocolo rigoroso para garantir generalização e robustez.

Divisão do Dataset (Train/Validation/Test): O conjunto de dados completo foi dividido numa proporção de 70/15/15 para treino, validação e teste, respetivamente. A divisão foi feita de forma estratificada, garantindo que a distribuição das classes (poluído/não-poluído) é mantida em todos os conjuntos. É crucial que o conjunto de teste tenha ficado completamente isolado durante todo o processo de desenvolvimento e ajuste de hiperparâmetros, sendo usado apenas para a avaliação final imparcial do modelo.

Hiperparâmetros: Após iterações experimentais, os seguintes hiperparâmetros demonstraram os melhores resultados:

Batch Size: 32 (um compromisso entre estabilidade do gradiente e eficiência de memória).

Learning Rate: 1e-4 (taxa de aprendizagem baixa para fine-tuning estável, utilizando o scheduler ReduceLROnPlateau para reduzir a taxa pela metade se a loss de validação estagnar por 5 epochs).

Optimizer: Adam (escolhido pela sua eficiência computacional e boa performance em problemas de computer vision).

Número de Épocas: O treino foi parado antecipadamente (early stopping) com paciência de 10 epochs para monitorizar a loss de validação e prevenir overfitting.

Estratégia de Otimização (Congelamento/Descongelamento de Camadas): Adotou-se uma estratégia de fine-tuning em duas fases:

Fase 1: Congelaram-se todas as camadas do backbone da MobileNetV2 e treinaram-se apenas as camadas de classificação da MLP e o head de fusão. Isto permite estabilizar o treino inicialmente.

Fase 2: Descongelaram-se as últimas 5-10 camadas convolucionais da MobileNetV2 para permitir um ajuste mais específico das features visuais para a nossa tarefa, enquanto as camadas iniciais (que detetam características genéricas) permanecem largely unchanged.

Função de Perda e Métricas de Avaliação:

Função de Perda (Loss Function): nn.CrossEntropyLoss() - a função standard e apropriada para problemas de classificação multiclasse.

Métricas de Avaliação: Para uma avaliação abrangente, monitorizámos:

Acurácia (Accuracy): Percentagem geral de previsões corretas.

Precisão (Precision): Dos casos previstos como poluídos, quantos efetivamente estão.

Recall (Sensibilidade): De todos os casos reais de poluição, quantos foram detetados.

F1-Score: Média harmónica entre Precisão e Recall, dando uma medida única do equilíbrio do modelo.

Matriz de Confusão: Para uma análise detalhada dos tipos de erros (falsos positivos vs. falsos negativos).

Análise da Implementação do Servidor de Inferência no Raspberry Pi

O código fornecido implementa um servidor HTTP leve (FastAPI) que corre no Raspberry Pi, servindo como interface entre o hardware (câmara, sensores) e o utilizador. A sua arquitetura segue um princípio de microserviço especializado, onde o Pi funciona como um dispositivo de edge computing que processa dados localmente, reduzindo a dependência de conectividade de rede e garantindo baixa latência.

1. Arquitetura Geral e Fluxo de Dados

O sistema é composto por três camadas principais:

Camada de Aquisição: Funções capture\_image\_from\_camera() e read\_sensors\_from\_hardware() (a implementar) que interagem diretamente com o hardware.

Camada de Inferência: O módulo model\_utils (importado) que carrega o modelo TorchScript e executa a previsão (predict\_from\_image\_and\_sensors).

Camada de Serviço e API: O servidor FastAPI que expõe endpoints REST para controlo e monitorização, servindo uma interface web (UI).

O fluxo operacional é o seguinte:

Um thread em segundo plano (capture\_loop) é iniciado via API.

Este thread ciclicamente:

Captura uma imagem da câmara.

(Opcional) Lê os valores dos sensores.

Executa o modelo multimodal com os dados capturados.

Guarda a imagem e o relatório resultante num local partilhado.

Aguarda um intervalo definido (ex: 5 segundos).

Uma UI web (/static/index.html) permite ao utilizador visualizar a imagem mais recente e o relatório em tempo real, além de controlar o modo de operação.

2. Decisões de Implementação Chave e Justificações

Por que FastAPI?

Alto Desempenho e Concorrência: FastAPI, sendo assíncrono, lida eficientemente com múltiplos pedidos (como a UI a pedir a imagem e o relatório ao mesmo tempo) sem bloquear o loop de captura, algo crucial para um sistema em tempo real.

Geração Automática de Documentação: Gera automaticamente uma documentação interativa da API (/docs), o que é invaluable para debugging e para outros sistemas se integrarem com o Pi.

Validação de Dados Integrada: Usa Pydantic para validar automaticamente os dados de entrada (como o payload JSON dos sensores), aumentando a robustez.

Simplicidade: O código da API é extremamente legível e conciso, reduzindo a probabilidade de erros.

Por que Threading para a Captura?

Operação Não-Bloqueante: Colocar o loop contínuo de captura e inferência numa thread separada impede que esta operação bloqueie o servidor web. Isto significa que a UI responde instantaneamente aos comandos do utilizador (Start/Stop) mesmo enquanto a inferência está a decorrer.

Controlo de Estado: O uso de um Event (capture\_flag) e de um Lock (state\_lock) para gerir o estado da thread e o acesso ao latest\_report é uma prática segura de programação concorrente, prevenindo condições de corrida (race conditions).

Por que uma UI Web Simples?

Acesso Universal: Qualquer dispositivo (portátil, telemóvel, tablet) na mesma rede local pode aceder e monitorizar o sistema através de um browser, sem necessidade de instalar software adicional.

Baixo Custo Computacional: A UI é estática (HTML/JS) e é servida diretamente pelo Pi. O frontend faz polling simples ao backend para atualizar os dados, uma estratégia leve e eficiente.

Prototipagem Rápida: Permite testar e demonstrar rapidamente a funcionalidade do sistema antes do desenvolvimento de uma UI mais complexa.

Abordagem para Hardware (Placeholders)

As funções capture\_image\_from\_camera e read\_sensors\_from\_hardware são implementadas como placeholders. Esta é uma melhor prática de desenvolvimento. Permite desenvolver e testar a totalidade do fluxo de dados e da lógica da aplicação antes da integração final com o hardware específico, que pode ser propenso a problemas de drivers e configuração.

3. Comparação com Possíveis Alternativas

Abordagem Alternativa Vantagens Desvantagens Por que foi Rejeitada?

Script Python Monolítico (sem API) Mais simples de escrever inicialmente. Sem interface remota; difícil de monitorizar ou controlar. Sem concorrência. Inflexível e não escalável. Impedia o controlo remoto e a visualização de dados em tempo real, que são requisitos fundamentais.

Inferência na Cloud (Enviar dados para servidor) Maior poder computacional para modelos complexos. Latência elevada, dependência de internet, custos de transmissão de dados (imagens), problemas de privacidade. Contraria o princípio de edge computing. A latência para enviar uma imagem, processá-la e receber a resposta tornaria o sistema incapaz de alertas em tempo real.

Framework mais pesado (ex: Django) Mais funcionalidades out-of-the-box (auth, ORM). Muito pesado para os recursos limitados do Raspberry Pi. Consumo excessivo de memória e CPU. Ineficiente. O FastAPI oferece o equilíbrio perfeito entre funcionalidade e leveza para esta aplicação específica.

UI mais complexa (ex: React) Experiência de utilizador mais rica e interativa. Requer mais recursos do Pi para servir ou um servidor frontend separado. Maior complexidade de desenvolvimento. Complexidade desnecessária. A UI simples de polling é suficiente para a função de monitorização e controlo, mantendo o sistema leve.

4. Integração com o Modelo Multimodal

O endpoint /predict\_multimodal é onde a arquitetura do teu modelo se materializa:

Ele aceita simultaneamente um ficheiro de imagem e um payload JSON com leituras de sensores.

Encaminha estes dados para a função predict\_from\_image\_and\_sensors, que deve estar construída para:

Pré-processar a imagem exatamente como foi feito durante o treino (redimensionar, normalizar).

Pré-processar o vetor de sensores (aplicar o StandardScaler salvo).

Executar o modelo TorchScript.

Interpretar o resultado (e.g., usar decode\_label para obter o nome da classe).

A flexibilidade dos endpoints permite operar o sistema em dois modos: apenas visual (image\_only) ou multimodal completo, tornando-o adaptável a situações onde um sensor possa falhar.

3.4.6 Estratégias para combate a falsos positivos

Usar negativos difíceis no dataset (nuvens, vapor, poeira).

Implementar política de dupla confirmação antes de alertar: imagem+sensor (por exemplo pm2\_5 acima do limiar OU prob imagem alta por X segundos).

Use Grad-CAM para inspecionar decisões e identificar padrões falsos.

3.4.7 Export e inferência

TorchScript é portátil e razoavelmente rápido no Pi CPU; alternativo: ONNX + ONNXRuntime, ou TFLite (se usar Coral). TorchScript adequa-se ao fluxo PyTorch e é simples de usar.

3.5 Fluxo operacional: passo a passo (runtime)

Boot & startup: Pi e Arduino iniciam; Pi inicializa serviços (logger, NTP, serial).

Aquisição periódica (ex.: a cada 2 s):

Captura imagem (webcam/CSI) → salva YYYY-mm-ddTHH:MM:SS.jpg.

Ler linhas serial do Arduino (MG811/MQ7 raw) → parse → timestamp absoluto.

Ler PM frame da SEN0233 via UART → parse.

Ler BME680 via I²C.

Compor registo e gravar CSV/SQLite (timestamp,image,co2,co,pm2\_5,...).

Preprocess & inferência local:

Carregar imagem, aplicar transforms (resize, normalize).

Construir vetor sensores na ordem esperada, aplicar scaler.

Chamar model(image\_tensor, sensor\_tensor) no TorchScript.

Receber probabilidades; aplicar limiar/esquema de confirmação.

Alerta e logging:

Se condição de alerta for satisfeita, enviar notificação (MQTT publish) e gravar evento com imagem e metadados.

Envio/Backup (opcional):

Periodicamente enviar agregados para servidor (CSV + imagens) via HTTPS/MQTT.

Operação offline:

Se sem conexão, armazenar localmente e sincronizar quando online.

Rotina de manutenção:

Recalibração periódica, limpeza física do sensor de particulado, substituição de ficheiros antigos.

3.6 Pipeline completo de implementação (textual e visual)

Etapas (resumidas)

Montagem hardware: ligar sensores, Arduino, Pi conforme esquema; testar leituras individuais.

Firmware Arduino: compilar e gravar; verificar serial output.

Software Edge: instalar dependências (Python, libs), configurar systemd para autostart do acquire\_and\_infer.py.

Dataset:

Executar aquisição para recolher pares (imagem + sensores).

Rotular manualmente (smoke/clean) e anotar masks se necessário com SAM.

Pré-processamento:

Limpeza, imputação, scaler fit, organização train/val/test.

Treino:

Treino inicial com backbone congelado; avaliar validação; ajustar LR e congelar/descongelar camadas se tiver dados.

Export:

Salvar multimodal\_best.pth e multimodal\_traced.pt.

Salvar scaler.pkl, label\_map.json.

Deploy:

Transferir artefactos para Pi; configurar inferência e thresholds.

Validação em campo:

A/B tests, comparar com leituras de referência; ajustar limiares e re-treinar.

Operação contínua:

Logs, monitorização, retraining programado com novos dados.

A poluição atmosférica constitui uma das principais ameaças ambientais à saúde humana, sendo responsável por milhões de mortes prematuras anualmente em todo o mundo. A exposição a poluentes como partículas inaláveis (PM2.5 e PM10), monóxido de carbono (CO) e dióxido de carbono (CO₂) está associada ao agravamento de doenças respiratórias (asma, DPOC), cardiovasculares, e ao desenvolvimento de condições neurológicas e oncológicas. Para além dos impactos na saúde, a poluição do ar tem consequências económicas significativas, incluindo custos com cuidados de saúde, perda de produtividade laboral e danos em ecossistemas. A monitorização contínua e precisa é, portanto, fundamental para informar políticas públicas, implementar alertas precoces à população e fiscalizar fontes emissoras, mas os sistemas de monitorização tradicionais são muitas vezes caros, fixos e escassos, deixando lacunas de cobertura significativas.

Análise de Stakeholders

Governo e Autoridades Reguladoras: Beneficiam com uma ferramenta de fiscalização ambiental acessível, permitindo a monitorização de zonas industriais específicas e a verificação do cumprimento de normas de emissões, fundamentais para a definição de políticas públicas informadas.

Indústria e Setor Energético: Podem utilizar a solução para monitorização interna de emissões, gestão proativa do impacto ambiental e demonstração de conformidade com regulamentos, potencialmente reduzindo multas e melhorando a imagem corporativa.

Cidadãos e Comunidades: Ganham acesso a informação em tempo real sobre a qualidade do ar na sua proximidade, permitindo decisões informadas sobre atividades ao ar livre e maior transparência sobre impactos ambientais locais. Organizações não-governamentais podem usar os dados para advocacy ambiental.

Comunidade Científica: Beneficia com a disponibilização de conjuntos de dados abertos e de alto detalhe espacial e temporal para investigação em qualidade do ar e epidemiologia.

Requisitos Funcionais

Deteção em Tempo Real de Poluentes: O sistema deve ser capaz de adquirir dados dos sensores e processar imagens, executando a inferência do modelo multimodal em menos de 5 segundos por amostra, permitindo uma resposta quase imediata a eventos de poluição.

Interface de Monitorização Remota: Deve ser disponibilizada uma interface web acessível via browser, atualizada automaticamente, que apresente a imagem mais recente, as leituras dos sensores, o resultado da inferência e o histórico de eventos.

Alertas Automáticos: O sistema deve ser capaz de gerar e publicar automaticamente alertas (ex: via MQTT, email ou API) quando os níveis de poluição detetados excederem limiares pré-definidos configuráveis pelo utilizador.

Operação Offline: A solução deve funcionar de forma autónoma e completa sem dependência de uma ligação à Internet contínua, armazenando dados localmente e sincronizando quando a conectividade for restabelecida.

Requisitos Não Funcionais

Baixo Consumo Energético: O sistema completo (Raspberry Pi, sensores e periféricos) deve ser optimizado para um consumo energético mínimo, permitindo a sua operação contínua 24/7, potencialmente alimentado por fontes renováveis (ex: painel solar).

Baixo Custo de Implementação: A solução deve ser construída com componentes de hardware off-the-shelf e de baixo custo, tornando-a acessível para implementação widespread por municípios, comunidades ou pequenas indústrias. O custo alvo por unidade é inferior a 300€.

Tolerância a Falhas: O sistema deve ser resiliente a falhas intermitentes de sensores, erros de leitura ou reinícios inesperados, retomando a operação normal automaticamente sem perda crítica de dados ou funcionalidade.

Escalabilidade: A arquitetura de software e comunicação (ex: MQTT) deve permitir a integração futura de múltiplos nós de monitorização numa rede federada, com gestão centralizada de dados.

Restrições Técnicas

Limitações de Hardware do Raspberry Pi: O design do sistema deve respeitar os constraints computacionais do Raspberry Pi 4 (CPU ARM de 4 núcleos, 4-8GB RAM, sem GPU dedicada potente), otimizando a inferência para operar eficientemente na CPU.

Latência Máxima Permitida para Inferência: O tempo total de aquisição de dados, pré-processamento, inferência do modelo e publicação de resultados não deve exceder 10 segundos para garantir a utilidade dos alertas em tempo real.

Precisão Mínima Exigida: O modelo de classificação multimodal deve atingir uma precisão (precision) mínima de 85% na classe "poluído" para minimizar falsos positivos que possam levar a alertas desnecessários e desgastar a confiança no sistema.

3.2. Análise de Alternativas e Trade-offs

Comparação de Arquiteturas de ML

Abordagem Unimodal vs Multimodal:

Unimodal (Apenas Imagem):

Vantagens: Maior simplicidade arquitetural, mais fácil de treinar e implementar, menor custo computacional por inferência.

Desvantagens: Elevada propensão para falsos positivos (e.g., confundir nuvens, nevoeiro ou sombras com fumo/poluição). A precisão é intrinsecamente limitada pela informação disponível.

Multimodal (Imagem + Sensores):

Vantagens: Robustez significativamente maior e precisão superior. Os sensores fornecem dados físicos quantitativos que corroboram ou refutam a hipótese visual, reduzindo drasticamente falsos positivos.

Desvantagens: Complexidade aumentada no desenvolvimento (sincronização de dados, arquitetura de fusão), treino (necessidade de dataset multimodal) e deployment (mais componentes de hardware). Custo computacional ligeiramente superior.

Trade-off e Decisão: Optou-se pela abordagem multimodal, priorizando a robustez e fiabilidade da deteção sobre a simplicidade, uma vez que falsos positivos tornariam o sistema impraticável.

Modelos Leves para Edge Computing:

MobileNetV2: Excelente equilíbrio entre precisão, número de parâmetros (~3.4M) e velocidade. Opera eficientemente na CPU do Pi. A arquitetura com inverted residuals e linear bottlenecks é altamente optimizada para dispositivos móveis/embebidos.

EfficientNet: Família de modelos que otimiza simultaneamente profundidade, largura e resolução. Oferece melhor precisão que a MobileNet para o mesmo nível de complexidade, mas é ligeiramente mais pesada computacionalmente e menos popular em edge sem aceleradores.

SqueezeNet: Extremamente leve (~1.2M parâmetros) e rápido. No entanto, a sua precisão é notavelmente inferior à da MobileNetV2, o que é crítico para esta aplicação.

Trade-off e Decisão: A MobileNetV2 foi selecionada por oferecer o melhor compromisso global para o hardware alvo (Raspberry Pi CPU), sendo suficientemente precisa, eficiente e amplamente suportada no ecossistema PyTorch/TorchScript.

Alternativas de Hardware

Raspberry Pi 4 vs Jetson Nano vs Google Coral:

Raspberry Pi 4: CPU ARM competente, amplamente disponível, vasto suporte comunitário, baixo custo. Desvantagem: falta de acelerador de IA dedicado, executando modelos na CPU.

NVIDIA Jetson Nano: Possui GPU CUDA dedicada, excelente para inferência de modelos de ML. Mais potente para ML, mas também mais caro, com maior consumo energético e mais complexo de configurar.

Google Coral Dev Board: Inclui um TPU (Tensor Processing Unit) de borda, oferecendo aceleração de inferência extremamente rápida e eficiente para modelos compilados para TFLite. Custo intermédio, mas com um ecossistema mais fechado (TFLite).

Trade-off e Decisão: O Raspberry Pi 4 foi escolhido devido ao seu custo radicalmente inferior, simplicidade, adequação à inferência CPU de modelos leves como a MobileNetV2, e à vastidão de recursos e tutorials disponíveis, que agilizam o desenvolvimento.

Sensores de Baixo Custo vs Sensores Profissionais:

Sensores de Baixo Custo (MQ-7, MG-811, PMS5003): Custam uma fração dos sensores profissionais (dezenas vs milhares de euros). São "suficientemente bons" para detetar a presença e variações significativas de poluentes. Requerem calibração e são propensos a deriva.

Sensores Profissionais (ex: da SPEC, Alphasense): Fornecem medições de alta precisão e são muito estáveis. São essenciais para aplicações de referência legal. O seu custo proibitivo impossibilita a criação de uma rede de sensores densa.

Trade-off e Decisão: A opção por sensores de baixo custo é fundamental para cumprir o requisito de baixo custo e escalabilidade. A calibração contra um sensor de referência e a fusão com o modelo visual são as estratégias para mitigar a sua menor precisão.

Alternativas de Implementação

FastAPI vs Flask vs Django:

Flask: Microframework muito simples e flexível. No entanto, é síncrono por padrão, o que pode ser um gargalo para aplicações de IO-bound como uma API que serve inferências.

Django: Framework "baterias incluídas", muito poderoso para aplicações web complexas. É excessivamente pesado e complexo para uma aplicação de borda focada numa API REST simples.

FastAPI: Framework moderno, assíncrono nativo, oferece desempenho muito alto. Geração automática de documentação OpenAPI é um grande benefício para desenvolvimento e teste. Perfeito para construir APIs leves e eficientes.

Trade-off e Decisão: FastAPI foi a escolha clara, otimizando para desempenho, concorrência e experiência de desenvolvimento (com a documentação interativa) num pacote leve.

Threading vs Multiprocessing vs AsyncIO:

Threading: Adequado para tarefas de IO-bound (como servir pedidos HTTP ou ler da serial). Partilha de memória simplifica a comunicação entre threads (e.g., para o estado latest\_report). O GIL do Python não é um grande obstáculo aqui porque a inferência (CPU-bound) está confinada a uma thread principal.

Multrocessing: Melhor para tarefas CPU-bound, pois contorna o GIL usando processos separados. Introduz complexidade significativa na partilha de dados entre processos (requerendo queues, pipes, ou memória partilhada).

AsyncIO: Paradigma assíncrono, excelente para IO-bound de alto escalabilidade. Pode ser complexo de integrar com código bloqueante (como a inferência do modelo PyTorch).

Trade-off e Decisão: A abordagem híbrida Threading foi selecionada. Uma thread de worker dedicada trata do loop bloqueante de aquisição e inferência (CPU-bound), enquanto o servidor FastAPI (assíncrono) lida com os pedidos HTTP (IO-bound) de forma não bloqueante. Esta arquitetura é robusta e eficiente para a carga esperada.

O estudante deve questionar se o relatório descreve o trabalho de forma suficientemente detalhada para que possa ser compreendido e reproduzido, se necessário, no futuro por outras pessoas da organização. Todas as boas práticas abordadas no curso deverão ser utilizadas neste capítulo e no relatório em geral. Note-se que este capítulo não deve conter exclusivamente a explicação da forma como métodos, técnicas, algoritmos, tecnologias, etc. foram usados, mas também o processo de compreensão do problema, a análise nos seus vários níveis, a identificação e especificação de requisitos, a modelação, a descrição dos componentes da solução, etc. Recomenda-se que a descrição técnica siga uma abordagem que parta do “geral” (descrição inicial do problema) para o “particular” (descrição técnica completa e coerente da solução), sem saltar etapas.

Nota: Pode justificar-se este capítulo ser divido em dois (Análise do Problema e Desenho da Solução).

**Atenção: A proposta de estrutura das subsecções seguintes adequa-se a projetos/estágios de desenvolvimento de produto ou sistema. O estudante deve, conjuntamente com o orientador, definir a estrutura de secções mais adequada ao seu projeto.**

* 1. Domínio do problema

Devem ser especificados os conceitos de domínio do problema através de artefactos adequados (e.g. glossário, modelo de domínio).

* 1. Requisitos funcionais e não funcionais

Especificar os principais requisitos funcionais e não funcionais do sistema. O levantamento de requisitos pode ser obtido dialogando com o cliente, de forma a identificar as funcionalidades de alto nível pretendidas no sistema para cada perfil de utilizador, recorrendo a diagramas de casos de utilização e/ou diagramas de domínio. De um modo geral, pretende-se a documentação final dos requisitos (e não a sua evolução no tempo).

* 1. Desenho

Dependendo do volume, o desenho da solução pode ser incluída neste capítulo (uma secção) ou pode constituir um capítulo separado.

O estudante deve especificar a arquitetura global da solução (diagramas de componentes e outros se necessário, como por exemplo, diagramas de implantação caso existam instalações) e deve apresentar uma justificação para a arquitetura que faça referência aos requisitos. Se foram estudadas alternativas arquiteturais, estas devem ser apresentadas.

Apresentar uma especificação global do sistema com base em um ou mais diagramas de Classes e apresentar especificações adicionais em casos que o justifiquem (e.g. diagramas de estados para classes com comportamento relativamente complexo).

Especificar o modelo de dados, caso a solução contemple esse aspeto, e justificar o modelo de dados com base nos artefactos anteriores (particularmente o diagrama de classes).

A aplicação de padrões e boas práticas é recomendada.

1. **Diagrama conceptual do sistema** - Visão de alto nível mostrando os componentes principais (sensores, Raspberry Pi, cloud/server, utilizador)
2. **Fotos do protótipo implementado** - Mostrando a montagem física do sistema (Raspberry Pi, câmara, sensores, ligações)
3. **Diagrama de arquitetura do sistema** - Detalhando todos os componentes de hardware e software
4. **Diagrama de blocos da solução multimodal** - Mostrando o fluxo de dados desde a aquisição até à inferência
5. **Diagrama de fusão multimodal** - Ilustrando a arquitetura late fusion (MobileNetV2 + MLP)
6. **Esquema de ligações eletrónicas** - Diagrama de ligação dos sensores ao Arduino/Raspberry Pi
7. **Diagrama de sequência** - Mostrando o fluxo temporal de aquisição, processamento e inferência
8. **Diagrama de classes/componentes** - Representando a estrutura do software (opcional mas útil)
9. **Capturas de ecrã da interface web** - Mostrando a dashboard de monitorização
10. **Exemplos de imagens do dataset** - Mostrando amostras positivas (poluição) e negativas (falsos positivos)
11. **Visualizações de data augmentation** - Mostrando as transformações aplicadas às imagens
12. **Gráficos de pré-processamento de dados dos sensores** - Mostrando dados brutos vs. filtrados
13. **Código em snippet** - Exemplos de partes críticas do código (API, inferência, fusão)
14. **Matriz de confusão** - Para os modelos unimodal e multimodal
15. **Gráficos de métricas de performance** - Comparando accuracy, precision, recall, F1-score
16. **Curvas de aprendizagem** - Mostrando loss e accuracy durante o treino/validação
17. **Grad-CAM ou heatmaps** - Mostrando as regiões da imagem que influenciaram a decisão
18. **Gráficos temporais de leituras de sensores** - Correlacionando com eventos de poluição detetados
19. **Tabelas de comparação de desempenho** - Modelo proposto vs. baseline vs. estado da arte
20. **Gráficos de latency e throughput** - Desempenho computacional no Raspberry Pi
21. Implementação da Solução

4.1 Descrição da Implementação

A solução proposta foi implementada tendo como base um modelo de deep learning multimodal, capaz de integrar simultaneamente informação proveniente de imagens captadas por câmara e dados recolhidos por sensores ambientais. O modelo combina uma rede neuronal convolucional (ResNet-18, pré-treinada em ImageNet) para extração de características visuais e uma rede densa dedicada ao processamento dos valores dos sensores (PM2.5, CO, CO₂). As representações obtidas em cada ramo são posteriormente concatenadas e processadas por uma camada totalmente ligada, resultando na classificação final da qualidade do ar.

O processo de treino foi realizado em PyTorch, utilizando como conjunto de dados um ficheiro CSV com as leituras dos sensores e uma pasta de imagens associadas. O pré-processamento incluiu normalização dos dados dos sensores com StandardScaler e codificação das classes de qualidade do ar com LabelEncoder. Para as imagens, aplicou-se redimensionamento e conversão para tensores. O treino decorreu durante 10 épocas com o otimizador Adam, função de perda cross-entropy e batch size de 16.

No que respeita ao ambiente de desenvolvimento, a solução foi concebida e testada em Python 3.10, suportada por bibliotecas como torchvision, scikit-learn, Pandas, PIL e FastAPI. O sistema foi preparado para correr tanto em ambientes de alto desempenho (GPU) como em dispositivos mais modestos, como o Raspberry Pi, que desempenhará o papel de coletor de imagens e dados de sensores.

Para garantir a integração da solução com uma interface acessível, foi desenvolvido um servidor FastAPI, responsável por disponibilizar uma interface web (via HTML e JavaScript) acessível através de qualquer navegador. Esta interface permite ao utilizador selecionar o modo de operação — apenas imagem ou imagem e sensores — e iniciar/parar a captura. Uma vez iniciada, a imagem mais recente e o relatório com a previsão do modelo são apresentados em tempo real no browser.

A arquitetura final encontra-se organizada em módulos distintos:

dataset\_and\_model.py: definição do modelo multimodal e preparação dos dados.

model\_utils.py: funções auxiliares para carregar o modelo e realizar inferência.

evaluate.py: avaliação do desempenho do modelo treinado.

main.py: servidor FastAPI e lógica de integração com a UI.

static/index.html: interface web que permite interação com o utilizador.

Esta organização modular reflete diretamente o desenho previamente definido, garantindo separação de responsabilidades e facilitando a manutenção futura.

4.2 Testes

Foram realizados vários níveis de testes para validar a solução:

Testes Unitários

Teste da função PollutionDataset.\_\_getitem\_\_ para garantir leitura correta de imagens e dados de sensores.

Teste da função predict em model\_utils.py para verificar a coerência entre entrada e saída (classe prevista e probabilidades).

Testes Funcionais

Verificação do correto funcionamento da interface web: seleção de modo de captura, início e paragem da aquisição, exibição de imagem e relatório.

Teste do servidor FastAPI para garantir resposta adequada aos endpoints REST criados.

Testes de Integração

Execução de inferência completa, desde a captura de imagem e dados até à geração do relatório no navegador.

Simulação de cenários distintos: apenas imagem, apenas sensores e combinação de ambos.

Testes de Sistema

Avaliação da solução em ambiente próximo ao real, utilizando o Raspberry Pi para captura de dados e interface de acesso remoto via browser.

Verificação da estabilidade do sistema durante períodos de execução contínua.

4.3 Avaliação da Solução

A avaliação do modelo foi conduzida com base no conjunto de teste reservado (30% dos dados originais, estratificado por classes). Os resultados obtidos foram:

Acurácia Global: 97%

Relatório de Classificação: valores de precision, recall e f1-score superiores a 0.92 em todas as classes, destacando-se a classe Hazardous com desempenho perfeito (F1=1.0).

Matriz de Confusão: revelou uma taxa muito reduzida de erros de classificação, com pequenas confusões apenas entre classes vizinhas como Good e Moderate, ou Moderate e Unhealthy for Sensitive Groups.

Adicionalmente, foram realizados testes complementares:

Análise de Erros: identificou os exemplos mal classificados, permitindo constatar que a maioria dos erros ocorrem em casos limítrofes de qualidade do ar, onde os valores dos sensores se aproximam dos limites entre categorias.

Curvas ROC/AUC: calculadas para cada classe com base nas probabilidades de saída do modelo, resultando em valores de AUC superiores a 0.95, evidenciando uma elevada capacidade discriminatória.

Perspetiva Crítica

A solução demonstrou ser adequada aos objetivos propostos, apresentando elevada fiabilidade tanto no uso de dados de sensores como na fusão com imagens. Contudo, verificou-se que a utilização exclusiva de imagens apresenta limitações — algumas condições visuais podem indicar poluição (ex.: fumo visível) que não é detetada pelos sensores, mas a previsão nestes casos pode ser menos consistente devido à subjetividade visual. Ainda assim, a integração multimodal mostrou-se eficaz para compensar estas limitações.

Do ponto de vista de desempenho, a solução é executável em hardware modesto, embora o processamento em tempo real dependa da otimização do modelo e da capacidade do dispositivo utilizado. Quanto à usabilidade, a interface web provou ser intuitiva e prática, permitindo que o utilizador não necessite de conhecimentos técnicos para operar o sistema.

A implementação da solução consistiu na construção de um sistema multimodal capaz de inferir a qualidade do ar a partir da fusão de imagens captadas por câmara e dados recolhidos por sensores ambientais. O sistema foi desenvolvido de forma modular, seguindo uma arquitetura clara que permite a sua execução em diferentes plataformas, desde ambientes de desenvolvimento em computador até ao dispositivo final composto por um Raspberry Pi com câmara e um Arduino ligado a sensores.

Definição do Conjunto de Dados e Estrutura do Modelo

A primeira etapa consistiu na preparação de um conjunto de dados multimodal. Para tal, criou-se um módulo PollutionDataset, responsável por:

Ler os caminhos das imagens e os valores dos sensores a partir de um ficheiro CSV.

Normalizar os valores dos sensores utilizando StandardScaler da biblioteca scikit-learn.

Codificar as classes de qualidade do ar (Good, Moderate, Unhealthy, Hazardous, etc.) com LabelEncoder.

Aplicar transformações às imagens (redimensionamento para 224x224 e conversão em tensores).

Esta estrutura permitiu que cada amostra fosse composta por duas entradas: uma imagem processada e um vetor de três sensores (PM2.5, CO, CO₂).

No núcleo da solução encontra-se o modelo PollutionModel, uma rede multimodal desenvolvida em PyTorch, cuja função forward integra dois fluxos distintos:

Um ramo visual, baseado na rede ResNet-18 pré-treinada em ImageNet, utilizada para extração de características da imagem. A última camada de classificação da ResNet foi removida e substituída por um head adaptado.

Um ramo numérico, constituído por uma rede totalmente ligada com camada oculta de 32 neurónios e ativação ReLU, responsável por processar os valores normalizados dos sensores.

A fusão ocorre através da concatenação das representações extraídas, que são posteriormente processadas por uma camada final linear com saída correspondente ao número de classes de qualidade do ar.

Um excerto crítico do código que define esta integração é:

def forward(self, image, sensors):

img\_features = self.cnn(image) # Extração visual (ResNet)

sensor\_features = self.sensor\_net(sensors) # Extração dos sensores

combined = torch.cat((img\_features, sensor\_features), dim=1)

output = self.fc(combined)

return output

Treino do Modelo

O modelo foi treinado durante 10 épocas com o otimizador Adam (taxa de aprendizagem inicial de 0.001) e função de perda CrossEntropyLoss, adequada a problemas de classificação multiclasse.

Durante o treino, observou-se uma redução significativa da perda (loss) de 0.86 para 0.10 e uma melhoria constante na acurácia, alcançando 96.59% no conjunto de teste.

Foram ainda implementados scripts auxiliares:

train.py — responsável pelo treino e armazenamento do modelo (pollution\_model.pth).

evaluate.py — avalia o desempenho do modelo no conjunto de teste, produzindo relatório de classificação, matriz de confusão e métricas complementares como curvas ROC/AUC.

model\_utils.py — módulo reutilizável para carregar o modelo treinado e realizar inferência, integrando normalização e mapeamento de classes.

Servidor e Interface de Utilizador

Para permitir a utilização prática do sistema, foi desenvolvido um servidor FastAPI, implementado em main.py. Este servidor desempenha duas funções:

Disponibilizar endpoints REST para receber imagens, dados de sensores e devolver previsões.

Servir a interface web localizada em static/index.html.

A interface web foi construída em HTML e JavaScript, permitindo:

Seleção do modo de operação (apenas imagem ou imagem + sensores).

Início e paragem da captura em tempo real.

Visualização da imagem captada.

Exibição do relatório gerado pelo modelo, incluindo classe prevista e probabilidades associadas.

Assim, qualquer dispositivo com um navegador pode aceder ao sistema, bastando estar na mesma rede do Raspberry Pi.

Integração em Dispositivo Físico (Raspberry Pi + Arduino)

A fase final da implementação consistiu na adaptação do sistema ao ambiente real:

O Raspberry Pi foi configurado como unidade central, responsável por correr o servidor FastAPI, gerir a câmara e servir a interface web.

O Arduino, ligado ao Raspberry Pi por USB/serial, foi encarregado de recolher os valores dos sensores ambientais (PM2.5, CO e CO₂). A comunicação é feita em tempo real, permitindo ao servidor FastAPI obter os valores e encaminhá-los para o modelo.

A câmara do Raspberry Pi foi utilizada para captura de imagens periódicas do ambiente, integradas no processo de inferência.

Este desenho garante a complementaridade: mesmo que os sensores não detetem alterações de qualidade do ar (por exemplo, se a poluição estiver afastada do ponto de medição), a câmara pode registar fenómenos visuais como colunas de fumo. O inverso também é verdadeiro — valores críticos nos sensores podem não ser visíveis na imagem. A fusão multimodal permite, assim, maior robustez.

Este capítulo deve ser dedicado à apresentação de detalhes relacionados com o enquadramento e implementação das soluções preconizadas no capítulo anterior. Note-se, no entanto, que detalhes desnecessários à compreensão do trabalho devem ser remetidos para anexos. Dependendo do volume, a avaliação do trabalho pode ser incluída neste capítulo ou pode constituir um capítulo separado.

* 1. Descrição da implementação

Descrever a implementação da solução proposta no capítulo anterior, podendo ser dadas explicações e evidências de soluções intercalares. Devem também ser descritas as tecnologias e metodologias utilizadas (software, sistemas de operação, linguagens, dispositivos ou outras ferramentas) e perspetiva crítica sobre as mesmas.

Esta secção descreve a implementação da solução proposta no capítulo anterior. Alguns dos diagramas referidos na secção anterior podem aparecer nesta secção. Por exemplo, diagramas de classes ou diagramas de módulos, sendo detalhadas as operações de cada classe ou as funções de cada módulo (diagramas de atividades). Devem também ser descritas as especificidades de implementação de acordo com o ambiente de desenvolvimento, plataforma e linguagem escolhida para o desenvolvimento e deve ser claro que o desenho apresentado anteriormente foi, de facto, adotado na implementação

* 1. Testes

A descrição dos testes efetuados (e.g. unitários, funcionais, de integração, de sistema) pode ser feita nesta secção ou, caso não se justifique, na secção anterior.

* 1. Avaliação da solução

Nesta secção deve ser descrita a abordagem seguida para avaliar a solução, ou parte da solução, nomeadamente um ou mais requisitos de qualidade (e.g. desempenho, usabilidade). São descritas experiências efetuadas e apresentados os dados/modelos utilizados, bem como os resultados obtidos. Devem ser descritos e avaliados os resultados obtidos. Deve ser feita uma discussão sobre a adequação dos resultados obtidos relativamente aos planeados anteriormente.

Esta secção poderá não existir em alguns relatórios de projeto/estágio, mas nesse caso deverá ser dada uma justificação para tal.

1. Conclusões

O capítulo de conclusões é um dos mais importantes do relatório, devendo ser apresentado um resumo dos resultados do trabalho efetivamente desenvolvido. ***As conclusões devem basear-se nos resultados realmente obtidos***. Devem enquadrar‑se os resultados obtidos com os objetivos enunciados e procurar extrair conclusões mais gerais, eventualmente sugeridas pelos resultados. Podem acompanhar as conclusões incluindo recomendações apropriadas resultantes do trabalho, nomeadamente sugerindo e justificando eventuais extensões e modificações futuras.

* 1. Objetivos concretizados

Nesta secção devem ser repetidos os objetivos apresentados no capítulo de introdução e para cada um deles deve ser descrito o seu grau de realização. Recomenda-se o uso de uma lista ou tabela, dado que facilita a leitura e compreensão.

* 1. Limitações e trabalho futuro

Nesta secção devem ser identificadas as limitações do trabalho realizado, fazendo uma análise autocrítica do trabalho realizado, bem como extrapolar eventuais direções de desenvolvimento futuro.

* 1. Apreciação final

Esta secção deve fornecer uma opinião pessoal sobre o trabalho desenvolvido.

1. Referências

|  |  |
| --- | --- |
| [1] | E. E. Agency, “Europe's air quality status 2023,” *EEA Report No 10/2023,* 2023. |
| [2] | P. M. L. M. C. e. a. Kumar, “The rise of low-cost sensing for managing air pollution in cities,” *Environment International,* vol. 75, pp. 199-205, 2015. |
| [3] | P. a. D. W. Wohlschlegel, *Guideline for Development of a Quality Assurance Program: Volume IX—Visual Determination of Opacity Emissions from Stationary Sources,* November 1975. |
| [4] | M. R. a. A. Stein, “Guidelines for Evaluation of Visible Emissions Certification, Field Procedures, Legal Aspects, and Background Material,” *EPA Publication No. EPA-340/1-75-007,* April 1975. |
| [5] | Q. X. J. X. L. &. G. H. Zhang, “Deep Convolutional Neural Networks for Forest Fire Detection,” *Actas do Fórum Internacional sobre Gestão, Educação e Aplicação de Tecnologias de Informação.,* 2016. |
| [6] | D. e. a. Rolnick, “Tackling Climate Change with Machine Learning,” *ACM Computing Surveys (CSUR),* p. 55, 2022. |
| [7] | X. S. W. L. L. &. W. X. Li, “A Computer Vision-Based Method for Traffic-Related Air Pollution Estimation,” *Science of The Total Environment,* vol. 781, nº 146748, 2021. |
| [8] | S. S. &. R. D. P. Kumar, “Global Operational Land Imager Landsat-8 Reflectance-Based Active Fire Detection Algorithm,” *International Journal of Digital Earth,* vol. 13, nº 8, pp. 1-24, 2020. |
| [9] | R. N. F. R. W. J. S. C. D. P. D. S. G. S. M. M. M. d. C. E. C. B. F.-F. J. O. M. B. L. d. S. &. C. C. L. Vasconcelos, “Fire Detection with Deep Learning: A Comprehensive Review,” *Land,* vol. 13, nº 10, 2024. |
| [10] | P. T. P. A. B. T. K. a. M. K. T. Bunpleng, “Smoke Segmentation Improvement Based on Fast Segment Anything,” *n Proceedings of the 10th International Conference on Internet of Things, Big Data and Security (IoTBDS 2025),* pp. 129-140, 2025. |
| [11] | ,. J. C. F. D. P. S. Z. C. L. Y. Q. T. K. X. S. C. Z. C. Q. F. R. L.-H. L. S.-H. B. C. S. H. Chaoning Zhang, “A Survey on Segment Anything Model (SAM): Vision Foundation Model Meets Prompt Engineering,” *arXiv:2306.06211v4 [cs.CV],* 19 Outubro 2024. |
| [12] | A. a. N. M. a. C. G. Dewangan, “FIgLib & SmokeyNet: Dataset and Deep Learning Model for Real-Time Wildland Fire Smoke Detection,” *NeurIPS 2021 Workshop on Tackling Climate Change with Machine Learning,* 2021. |
| [13] | M. &. C. M. &. M. L. &. K. H. &. R. A. &. J. D. Hardini, “Image-based Air Quality Prediction using Convolutional Neural Networks and Machine Learning,” *Aptisi Transactions on Technopreneurship (ATT),* vol. 5, pp. 109-123, 2023. |
| [14] | L. M. Laura Deveer, “Real-time air quality prediction using traffic videos and machine learning,” *Transportation research part D: Transport and environment,* vol. 142, nº 104688, 2025. |
| [15] | F. F. R. T. Qiang Zhang, “A deep learning and image-based model for air quality estimation,” *Science of the total environment,* vol. 724, nº 138178, 2020. |
| [16] | M. W. X. L. Y. M. Y. C. S. D. Xiaochu Wang, “Surveillance-image-based outdoor air quality monitoring,” *Environmental Science and ecotechnology,* vol. 18, nº 100319, 2024. |
| [17] | S. a. X. T. a. W. C. Q. a. L. G. a. Y. J. a. L. K. Xia, “AQMon: A Fine-grained Air Quality Monitoring System Based on UAV Images for Smart Cities,” *Association for Computing Machinery,* vol. 20, nº 2, 2024. |
| [18] | L. L. C. R. &. N. A. Bogdanffy, “Development of a Low-Cost Traffic and Air Quality Monitoring Internet of Things (IoT) System for Sustainable Urban and Environmental Management,” *Sustainability,* vol. 17, nº 11, 2025. |
| [19] | Y. Y. S. H. a. J. X. Liang Zhao, “Design and implementation of a low-cost and multi-parameter indoor air quality detector based on IoT,” *International journal of computer applications in technology,* vol. 72, nº 4, 2023. |
| [20] | F. F. R. T. Qiang Zhang, “A deep learning and image-based model for air quality estimation,” *Science of The Total Environment,* vol. 724, nº 138178, 2020. |
| [21] | X. Z. Y. S. T. A. S. A. A. A. A. G. W.-H. N. N. C. Maqsood Ahmed, “Low-cost video-based air quality estimation system using structured deep learning with selective state space modeling,” *Environment International,* vol. 199, nº 109496, 2025. |

**Nota:** Ver no Anexo B (secção B.4) indicações sobre como elaborar a lista de Referências.

1. Conteúdo em anexos

Esta parte do relatório deve conter informação adicional organizada por capítulos que, embora seja interessante, não faz parte do material estritamente necessário ao relatório. Documentos importantes produzidos ou utilizados durante o estágio que, pela sua dimensão, não sejam colocáveis no corpo principal do relatório podem ser incluídos em anexos.

1. Regras de Conteúdo e Estrutura

Dependendo de cada projeto, a proposta de estrutura apresentada neste documento pode ser alterada, acrescentando novos capítulos, subdividindo capítulos ou juntando capítulos. ***O aluno deve discutir com o orientador qual a melhor abordagem para o seu caso, seguindo as linhas orientadoras apresentadas*.**

A dimensão dos capítulos deve ser equilibrada de forma a não haver muita diferença no número de páginas entre capítulos. Consideram-se exceção os capítulos de introdução e de conclusão. Cada um destes capítulos deve ter uma dimensão à volta de 10% do total de páginas do relatório.

A estrutura dos capítulos deve ser tal que contenha secções e subsecções de forma ponderada, cada uma contendo partes relativamente separadas do trabalho. A primeira secção deve começar no princípio do capítulo. Não deverão existir secções ou subsecções com menos de uma página e não deverá haver apenas uma secção (subsecção) dentro de um capítulo (secção). Devem ser evitadas subsecções com demasiados níveis, devendo-se usar, como regra geral, até ao 3º nível.

As primeiras páginas, até à notação inclusive, identificam-se com numeração romana em letras minúsculas. A numeração de capítulo/secção é efetuada em sequência. Cada novo capítulo deve iniciar-se no topo de uma página. A numeração de páginas dos anexos é feita continuando a numeração do texto principal. Para separar os capítulos devem inserir quebras de página do documento.

* 1. Linguagem

A linguagem de um relatório deve ser rigorosa, clara e com caráter técnico. Devem ser evitadas frases na primeira pessoa: por exemplo, a frase “desenvolvi o módulo de faturação” deve ser rescrita da seguinte forma: “foi desenvolvido o módulo de faturação” ou “desenvolveu-se o módulo de faturação”.

Também se deve evitar o uso de expressões “coloquiais”, de opiniões pessoais não fundamentadas e o excesso de adjetivos/advérbios. Excecionalmente poderão ser dadas opiniões pessoais nas conclusões, tendo sempre em atenção a clareza e a boa educação. As siglas e os acrónimos devem ser sempre definidos da primeira vez que são usados no relatório.

É essencial não esquecer de rever ortograficamente o texto.

* 1. Formatação

A formatação do relatório (tipo de letra, tamanho, estilos utilizados, etc.) é da responsabilidade do autor. No entanto devem seguir algumas regras de bom senso e boas práticas (Sousa, 2002) (DEI-ISEP, 2002):

* Reduzir ao mínimo o número de tipos de letra utilizados (dois ou três no máximo);
* Usar um tipo de letra com tamanho de fácil leitura (por exemplo, Calibri 11pts);
* Ser consistente na utilização dos tipos de letra (usar sempre o mesmo tipo para o texto, usar sempre o mesmo tipo para os cabeçalhos, etc.);
* Utilizar tamanhos de letra lógicos (por exemplo, se o tamanho da letra se modificar de acordo com o nível de cabeçalho, não deverá ser usado um tamanho de letra maior para um cabeçalho de nível inferior);
* Evitar “decorações” nos tipos de letra (sombras, rebordos, etc.);
* Usar judiciosamente o negrito e o itálico nos parágrafos de texto, devendo a sua utilização ficar restrita a pequenos excertos de texto que *realmente* importa realçar;
* Usar espaçamento um e meio ou duplo entre as linhas para facilitar a leitura;
* Usar parágrafos “justificados” (simultaneamente à esquerda e à direita);
* Usar numeração correta e lógica de páginas, capítulos, secções, etc.

Deverão utilizar as funcionalidades do processador de texto escolhido para a definição de estilos, de modo a garantir um aspeto homogéneo do relatório.

É necessário ter em atenção as margens das páginas: deverão ser evitadas margens inferiores a 2 cm.

* 1. Imagens e tabelas

As imagens só devem ser colocadas no texto quando auxiliem a interpretação do assunto que se está a abordar. Na preparação das imagens deve haver cuidado para evitar má legibilidade, prestando atenção ao número de elementos existentes na imagem, ao tamanho dos elementos e ao tamanho do texto. Adicionalmente, devem evitar demasiadas cores e “floreados” nos diagramas técnicos a apresentar.

Cada imagem deve ser apresentada com um título curto que a identifique claramente, colocado por baixo da imagem, **assim como a origem da figura (obrigatório)**. A Figura 1 mostra dois diagramas que traduzem o mesmo conteúdo, no entanto são diferentes em termos visuais e de facilidade de leitura.



Figura 1 - Exemplo de imagens a) difícil leitura; b) fácil leitura (Sousa 2002)

As tabelas devem ser usadas para apresentar dados/informação que se queira cruzar em várias dimensões ou que se queira analisar segundo vários atributos. Cada tabela deve ser apresentada com um título curto que a identifique claramente, colocado por cima da tabela. A formatação de tabelas deve respeitar as regras apresentadas anteriormente, evitando “floreados” e garantindo que a tabela não fica dividida entre páginas. Deve facilitar-se a leitura e identificar corretamente a linha e/ou coluna de cabeçalho. A Tabela 1 é um exemplo possível de utilização e formatação de tabelas.

Tabela 2 - Exemplo de tabela

|  |  |  |
| --- | --- | --- |
| Modelo | Fator Preço | Fator Desempenho |
| Pentium III 800 Mhz | Muito barato | Já não é aceitável |
| Pentium IV 2.7 GHz | Caro | Um dos melhores no mercado |

As figuras e tabelas devem ter sempre um título e número. Verifiquem as funcionalidades do processador de texto para criação automática de título de figuras e tabelas, pois isso facilita a sua numeração e posterior criação de índices.

Os índices de tabelas e de figuras devem ser gerados automaticamente pelo processador de texto.

Quando se finalizar o relatório devem atualizar as tabelas de índices e outras existentes, gerando em seguida a versão em formato PDF, a qual deverá ser cuidadosamente revista de modo a verificar-se a sua correção em relação ao documento original.

* 1. Referências

O capítulo de Referências apresenta a lista de Referências consultada para a execução do projeto/estágio. A lista de Referências deve estar ordenada por autor (último nome) e em seguida por ano.

No texto, sempre que utilizem dados ou afirmações de terceiros devem indicar a fonte desses dados ou a afirmação, colocando entre parêntesis curvos o nome da referência, por exemplo (Sousa, 2002).

A formatação de cada entrada bibliográfica é diferente consoante o tipo de documento em questão:

* *Para um livro:* nome(s) do(s) autor(es), ano da edição entre parêntesis, título do livro em itálico, nome da editora, local da edição, país da edição;
* *Para um artigo em revista:* nome(s) do(s) autor(es), ano da edição entre parêntesis, título do artigo em itálico, nome da revista, volume da edição a negrito, número da edição, páginas;
* *Para uma comunicação em conferência:* nome(s) do(s) autor(es), ano da conferência entre parêntesis, título da comunicação entre aspas, nome da conferência em itálico, local da conferência, país da conferência, mês da conferência;
* *Para uma tese:* nome do autor, ano da tese entre parêntesis, título da tese, tipo de tese, universidade da tese, local da universidade, país da universidade;
* *Para um relatório interno:* nome(s) do(s) autor(es), ano do relatório entre parêntesis, título do relatório, origem do relatório, referência do relatório, instituição de acesso ao relatório, local da instituição, país da instituição, mês do relatório (abreviado com 3 letras, exceção aos meses com 4 letras);
* *Para um documento extraído da Internet:* o endereço entre parênteses, incluindo a data da última visita ao documento;
* *Para um endereço da Internet:* o endereço e a data da última visita ao endereço.

Na Figura 2 é apresentada uma lista de referências com um exemplo de cada tipo referido.

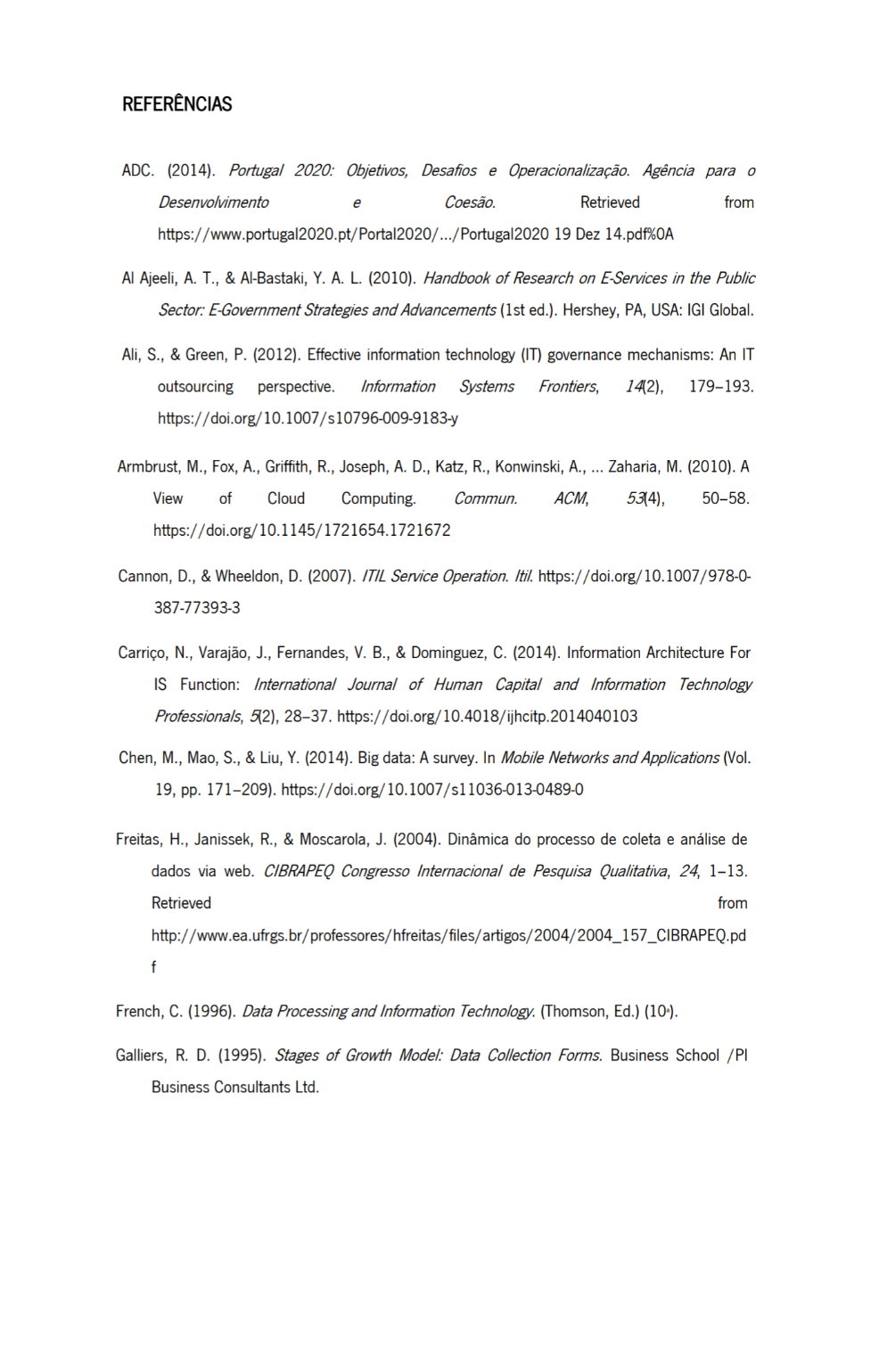


Figura 2 - Exemplo de lista de Referências.