Feature Model Engineering in Machine Learning for Pipeline Variability Management and Plant Disease Recognition in Brazilian Flora

Arthur Henrique Andrade Farias[1], Ricardo Theis Geraldi (Orientador)[1]
[1]Faculdade de Computação (FACOM) – Universidade Federal de Mato Grosso do Sul (UFMS)
Caixa Postal 549 – 79070-900 – Campo Grande – MS, Brasil
arthur.h.a.farias@ufms.br, ricardo.geraldi@ufms.br

***Abstract**: This work presents the engineering and empirical evaluation of a feature model for variability management in Artificial Intelligence (AI) pipelines applied to plant disease recognition in Brazilian flora. The proposed artifact, named PAIES (**P**ipeline **A**rtificial **I**ntelligence **E**ngineering **S**ystem) feature model, systematically organizes the configurable dimensions of an AI pipeline, data source, preprocessing, model training, evaluation and deployment, allowing the derivation of distinct configurations oriented by functional and non-functional quality attributes. Two configurations were derived and evaluated on the PlantVillage dataset [4]: Pipeline A (performance-oriented, using SwinTransformer and cloud deployment) and Pipeline B (efficiency-oriented, using MobileNet and edge deployment). The results indicate that Pipeline A achieves F1-Score of 0.9931 and accuracy of 0.9923, while Pipeline B achieves F1-Score of 0.9593 and accuracy of 0.9607, representing a loss of only ∼3.4 percentage points. However, the non-functional difference is substantial: Pipeline A presents latency 3.2× higher, model size 6.4× larger and energy consumption 4.7× greater. The results allow rejecting H0 in the context of this empirical study, indicating that the feature model may enable the prediction of significant non-functional trade-offs before model training*.

Resumo: Este trabalho apresenta a engenharia e a avaliação empírica de um feature model para gerenciamento de variabilidade em *pipelines* de Inteligência Artificial (IA) aplicados ao reconhecimento de doenças em plantas da flora brasileira. O artefato proposto, denominado PAIES (*Pipeline Artificial Intelligence Engineering System*) feature model, organiza sistematicamente as dimensões configuráveis de um pipeline de IA, fonte de dados, pré-processamento, treinamento de modelo, avaliação e implantação, permitindo a derivação de configurações distintas orientadas por atributos de qualidade funcionais e não funcionais. Duas configurações foram derivadas e avaliadas no dataset PlantVillage [2]: Pipeline A (orientada a desempenho, usando SwinTransformer e implantação em nuvem) e Pipeline B (orientada a eficiência, usando MobileNet e implantação em dispositivo de borda). Os resultados indicam que a Pipeline A atinge F1-Score de 0,9931 e acurácia de 0,9923, enquanto a Pipeline B atinge F1-Score de 0,9593

e acurácia de 0,9607, representando perda de apenas ~3,4 pontos percentuais. Contudo, a diferença não funcional é substancial: a Pipeline A apresenta latência 3,2 vezes maior, tamanho de modelo 6,4 vezes maior e consumo de energia 4,7 vezes superior. Os resultados permitem rejeitar H0 no contexto deste estudo empírico, indicando que o feature model pode possibilitar a previsão de trade-offs não funcionais significativos antes do treinamento do modelo.

1. Introdução

O agronegócio brasileiro responde por parcela expressiva do Produto Interno Bruto (PIB) nacional e enfrenta, entre seus principais desafios, as perdas causadas por doenças foliares em culturas de alto valor econômico. As perdas agrícolas causadas por doenças em plantas ultrapassam 20% da produção mundial de alimentos [1], comprometendo a segurança alimentar e a rentabilidade de produtores de diferentes portes.

Nos últimos anos, modelos de Aprendizado de Máquina (AM) baseados em redes neurais convolucionais profundas têm demonstrado desempenho relevante nessa tarefa. CNNs (Redes Neurais Convolucionais) profundas treinadas no dataset PlantVillage [2] alcançaram taxas de acurácia superiores a 99% em cenários controlados [3]. No entanto, a seleção da arquitetura de AM, da infraestrutura de implantação e das estratégias de pré-processamento configura um espaço de variabilidade amplo. Cada escolha impacta de forma distinta os requisitos funcionais (acurácia, F1-Score) e os requisitos não funcionais (latência, tamanho do modelo, consumo de energia), conforme definido pela norma ISO/IEC 25010 [4].

No cenário atual, a escolha de um pipeline costuma ser feita de forma *ad hoc*, por tentativa e erro ou por meio de scripts específicos que não se reutilizam em outros projetos. Essa abordagem dificulta a reprodutibilidade, eleva custos de experimentação e torna opaca a justificativa de decisões técnicas, aspectos críticos quando se pretende transferir tecnologia para o setor agrícola ou obter certificações de boas práticas.

Diante desse cenário, este trabalho propõe a engenharia de um *feature model*, denominado **P**ipeline **A**rtificial **I**ntelligence **E**ngineering **S**ystem (PAIES) *feature model*, para o gerenciamento de variabilidade em pipelines de IA agrícola. O *feature model* é o artefato central deste trabalho: ele não apenas lista opções de configuração, mas conecta explicitamente decisões de projeto a impactos quantificáveis em atributos de qualidade, permitindo que o engenheiro preveja trade-offs antes de executar o treinamento.

O sistema alvo para o qual o *feature model* foi projetado é o PAIES (**P**ipeline **A**rtificial **I**ntelligence **E**ngineering **S**ystem), definido neste trabalho como um sistema

baseado em AM destinado ao suporte à decisão em contextos agrícolas, integrando aquisição de imagens de folhas, inferência via modelos de AM e entrega de diagnósticos sobre o estado fitossanitário das plantas.

2. Estruturação da Pesquisa

2.1. Delineamento da Pesquisa

Esta pesquisa é de natureza aplicada e exploratória. Adota-se como referência alguns passos do método de pesquisa *Design Science Research Methodology* (DSRM) [5]. O DSRM se concentra na resolução de problemas práticos por meio da concepção, desenvolvimento e avaliação de artefatos. As fases I, II e III descritas a seguir correspondem, respectivamente, às etapas *Problem Identification*, *Design & Development* e *Demonstration & Evaluation* da DSRM.

2.2. Questões de Pesquisa e Hipóteses

As seguintes questões de pesquisa orientam este estudo empírico:

QP1: Em que medida um *feature model* pode representar a variabilidade de pipelines de IA agrícola de forma que *features* funcionais, como CNNBackbone, DataSource, Preprocessing, ModelTraining e ModelEvaluation, bem como *features* não funcionais, como latência, consumo de energia e tamanho do modelo, sejam explicitamente contempladas?

QP2: Configurações derivadas do *feature model* com objetivos de qualidade distintos produzem diferenças em requisitos não funcionais para níveis semelhantes de desempenho funcional?

As hipóteses formuladas são:

H0: O uso de *feature models* para configuração de pipelines de IA agrícola não produz diferenças relevantes nos requisitos não funcionais das configurações derivadas, para níveis semelhantes de desempenho funcional.

H1: O uso de *feature models* para configuração de pipelines de IA agrícola produz diferenças relevantes nos requisitos não funcionais das configurações derivadas, para níveis semelhantes de desempenho funcional.

No contexto deste estudo empírico, a decisão será sempre a de rejeitar ou não rejeitar H0. A aceitação de H1 como verdade absoluta não é afirmada diretamente.

2.3. Objetivos GQM

O objetivo desta avaliação foi formulado seguindo o template Goal-Question Metric (GQM) [6]:

- **Objeto de Análise**: Pipelines de AM derivados de um *feature model* para o PAIES.
- **Propósito**: Avaliar.
- **Foco de Qualidade**: Diferenças em requisitos funcionais (F1-Score, Acurácia) e em requisitos não funcionais (latência de inferência, consumo de energia, tamanho do modelo).
- **Ponto de Vista**: Pesquisador e engenheiro de software.
- **Contexto**: Dataset PlantVillage [2], tarefa de classificação multiclasse de doenças foliares (38 classes), ambiente de avaliação controlado com hardware de CPU convencional.

2.4. Fases de Execução da Pesquisa

As etapas do método adotado nesta pesquisa foram executadas em três fases sequenciais, conforme ilustrado na Figura 1.

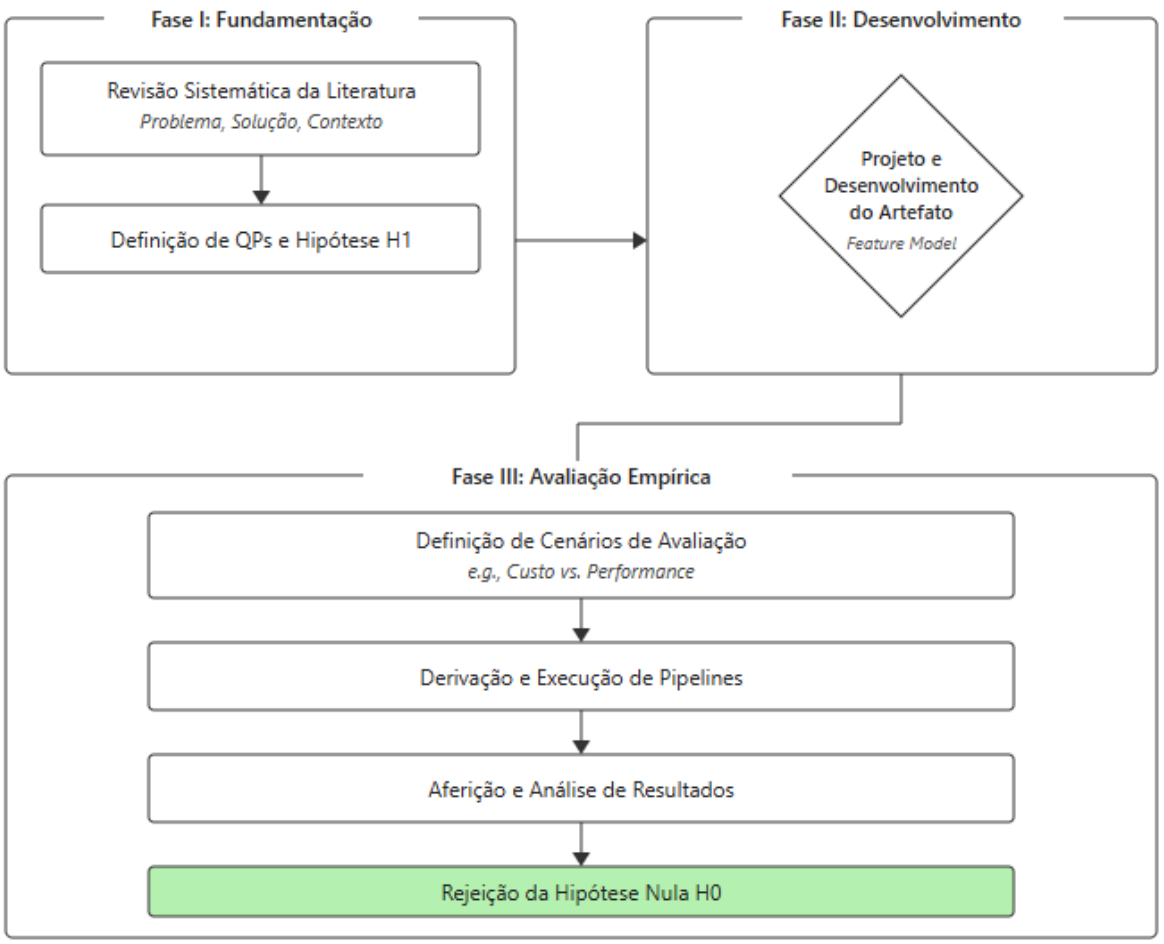

Figura 1: Etapas do método de pesquisa baseado na DSRM (Adaptado e traduzido de Peffers et al. [5])

Fase I - Identificação do Problema e Revisão da Literatura. A primeira fase consistiu em uma revisão da literatura para estabelecer o estado da arte e identificar a lacuna de pesquisa. A investigação abrangeu três pilares: (i) análise da origem e formalismo dos *feature models* como ferramenta para gestão de variabilidade em sistemas complexos [7, 8]; (ii) Operações de AM (MLOps): estudo dos desafios no ciclo de vida de sistemas de AM, incluindo a complexidade na construção de pipelines e a necessidade de incorporar Requisitos Não Funcionais (RNFs) [27, 28]; (iii) IA na Agricultura (AgriTech): investigação dos desafios específicos do domínio, como a variabilidade de fontes de dados e a necessidade de modelos robustos para detecção de doenças em plantas [11].

Fase II - Projeto e Desenvolvimento do Artefato. Nesta fase, o artefato central da pesquisa foi projetado e desenvolvido por meio das seguintes etapas: (1) identificação dos pontos de variabilidade do pipeline; (2) estruturação hierárquica do *feature model*; (3) especificação de atributos de qualidade (RNFs) às *features*; (4) formalização das restrições cross-tree (requires e excludes).

Fase III - Demonstração e Avaliação Empírica. A fase final visa apresentar a utilidade do artefato e coletar evidências para os testes das hipóteses: (1) definição dos cenários de avaliação A e B; (2) derivação e execução das configurações de pipeline; (3) aferição de métricas funcionais e não funcionais; e (4) análise comparativa.

3. Revisão da Literatura

3.1. *Feature Models* e Variabilidade

O gerenciamento de variabilidades em sistemas de software é complexo, sendo particularmente relevante no desenvolvimento de múltiplas variantes. Uma variante, é uma unidade específica do sistema, configurada para atender a um conjunto particular de requisitos ou a um segmento de mercado [7], tanto a nível de um produto ou em domínios específicos. Neste contexto, os *Feature Models* emergem como uma técnica fundamental para a representação e o gerenciamento sistemático desses sistemas.

3.1.1. Fundamentos dos *Feature Models*

Feature Models são representações hierárquicas, nos quais elementos são organizados em diferentes níveis de abstração, do mais geral para o mais específico, formando uma ordem de inclusão ou composição [12]. Os *feature models* representam graficamente todas as funcionalidades e atributos que um sistema de software pode possuir em um domínio específico. Essa representação foi introduzida na técnica *Feature-Oriented Domain Analysis* (FODA).

O FODA é uma técnica sistemática para a análise de um domínio de aplicação, com o objetivo de identificar características comuns e variáveis entre os sistemas

pertencentes a esse domínio, visando o desenvolvimento e a reutilização de ativos de software [7]. Dessa maneira, sua estrutura hierárquica permite organizar as características desde as mais genéricas até as mais específicas, facilitando a visualização e a compreensão da complexidade do domínio.

A essência dos *Feature Models* distingue explicitamente dois tipos de atributos [7]:

- **Características Comuns (Commonality)**: Representam as funcionalidades ou atributos em todas as variantes de um sistema em uma família de produtos.
- **Características Variáveis (Variability)**: Correspondem às funcionalidades ou atributos que podem ser incluídos, excluídos ou configurados de diferentes maneiras nas diversas variantes do sistema. São os elementos que diferenciam os produtos dentro da mesma família.

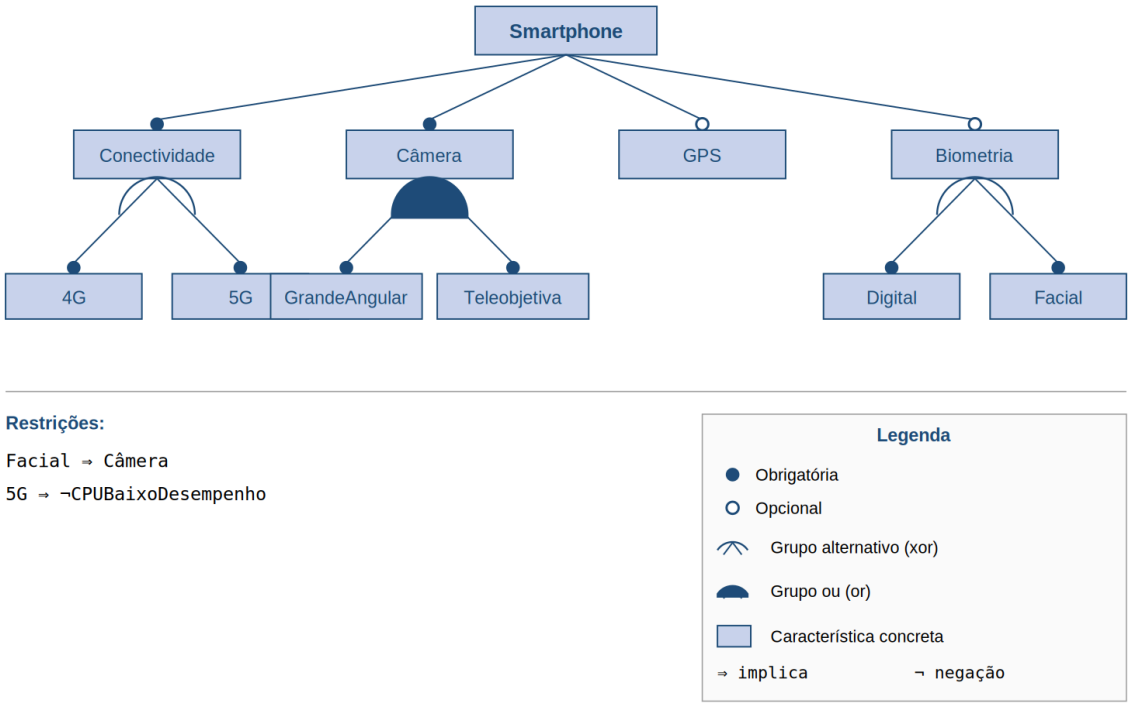

Figura 2: Exemplo de feature model no domínio de smartphones, ilustrando os quatro relacionamentos clássicos da notação introduzida por Kang et al. [7] (Adaptado e traduzido de Kang et al. [7]).

Para tornar os conceitos da notação clássica tangíveis, a Figura 2 apresenta um exemplo didático no domínio de smartphones. Esse exemplo é especialmente útil porque sintetiza, em uma única representação, os quatro tipos de relacionamento descritos a seguir, além de exemplificar como restrições cross-tree (FaceID requires Camera) e o rationale associado tornam explícitas decisões que, de outra forma, ficariam implícitas no projeto.

A representação desses conceitos em *feature models* permite uma análise rigorosa e uma gestão eficaz da variabilidade em sistemas de software. Essa capacidade de modelar a variação de forma estruturada logo revelou um grande potencial para

guiar a automação do desenvolvimento, servindo como um catalisador para a adoção da técnica em novas áreas.

3.1.2. Popularização e Programação Generativa

A relevância dos Modelos de Características foi ampliada com a popularização da Programação Generativa. Czarnecki e Eisenecker [8] demonstraram como a combinação da variabilidade explícita (representada por *Feature Models*), com técnicas de geração de código, pode otimizar significativamente o ciclo de desenvolvimento de software.

A Programação Generativa é um paradigma que objetiva automatizar a criação de software a partir de especificações de alto nível. Ao integrar Modelos de Características nesse processo, os benefícios incluem:

- **Redução do esforço de desenvolvimento**: A geração automatizada de código e componentes elimina a necessidade de codificação manual repetitiva para cada variante.
- **Garantia de consistência**: A derivação de produtos a partir de um modelo único e regras bem definidas assegura que as características e funcionalidades sejam implementadas de forma padronizada e consistente em todas as variantes.

A concretização desses benefícios de otimização e consistência, essenciais para a engenharia de sistemas de software modernos, depende diretamente da capacidade dos *Feature Models* de representar a variabilidade de forma precisa. Essa que é alcançada por meio de sua notação clássica, definindo um conjunto específico de relacionamentos e restrições para modelar as interdependências entre as características.

3.1.3. Notação Clássica e Relacionamentos

A representação e expressividade dos Modelos de Características é crucial para sua aplicabilidade em cenários complexos, sendo concretizada por meio de sua notação clássica, que permite representar a *feature* de forma inequívoca e estruturada. Essa notação foi introduzida inicialmente no método FODA [7] e subsequentemente consolidada e padronizada na literatura de Engenharia de Linhas de Produto [12].

A notação clássica estabelece quatro tipos de relacionamentos fundamentais entre uma característica pai e suas sub-características, organizando a hierarquia de funcionalidades [7]:

- **Required (Requerida)**: Define uma relação de dependência de inclusão incondicional entre uma característica pai e sua sub-característica. Semanticamente, se a característica pai está presente em uma configuração de produto, a sub-característica requerida deve obrigatoriamente ser incluída. Este relacionamento constitui a forma padrão de composição hierárquica em

um *Feature Model*, onde a ausência de um marcador explícito de feature (como opcional) denota essa obrigatoriedade [7].

- **Optional (Opcional)**: Denota uma sub-característica cuja inclusão em uma configuração de produto é contingente, mas não imposta, pela seleção de sua característica pai. A sua presença ou ausência representa um grau de liberdade no espaço de configuração do sistema, sendo um dos principais mecanismos para diferenciar produtos dentro de uma mesma família [7].
- **Alternative (Alternativa)**: Impõe uma restrição de exclusividade mútua sobre um conjunto de sub-características vinculadas a uma característica pai. A seleção de qualquer membro deste grupo é condicional à seleção do pai e obrigatoriamente exclui a possibilidade de selecionar qualquer outro membro do mesmo grupo na mesma instância de produto, garantindo assim a integridade de uma decisão de especialização única [7].

A flexibilidade destes elementos de notação é particularmente útil para modelar e gerenciar a complexidade em sistemas com alto grau de variação. Um exemplo proeminente é a configuração de pipelines de Inteligência Artificial (IA), nos quais diferentes algoritmos, etapas de pré-processamento e fontes de dados podem ser selecionados com base em requisitos específicos. Essa necessidade de estruturação evidencia uma importante interseção entre as áreas de IA e Engenharia de Software (ES), onde técnicas de gestão de *feature* têm sido aplicadas para domar a complexidade dos sistemas inteligentes. Pesquisas recentes demonstram que essa combinação é promissora. D'Aloisio *et al*. [13], por exemplo, propõem extensões ao meta-modelo de *feature models* para representar atributos de qualidade em pipelines de ML, evidenciando o crescente interesse da comunidade de Engenharia de Software nessa interseção. Dentro do vasto campo da inteligência artificial, uma área em particular se destaca pela necessidade de processos sistemáticos.

3.2. Pipelines de Aprendizado de Máquina

O desenvolvimento e a manutenção de sistemas de Aprendizado de Máquina (campo da IA que capacita sistemas a aprender e aprimorar seu desempenho em tarefas específicas diretamente a partir de dados, sem serem explicitamente programados para cada ação, Mitchell *et al*., 1997 em ambientes de produção exigem uma abordagem sistemática e padronizada. Neste contexto, os pipelines de ML representam um componente arquitetural fundamental para gerenciar o ciclo de vida dos modelos.

3.2.1. Definição e Estágios do Pipeline de ML

Um pipeline de Aprendizado de Máquina (do inglês, Machine Learning (ML)) pode ser definido como uma cadeia padronizada e orquestrada de estágios (ou componentes) que automatiza e rastreia o ciclo de vida completo de um modelo de ML. Tal ciclo abrange desde a aquisição de dados até a sua implantação em produção. O principal objetivo de um pipeline é garantir a consistência, a automação e a reprodutibilidade dos fluxos de trabalho de ML [14].

A gestão de pipelines é um pilar da MLOps (Machine Learning Operations), disciplina que aplica a cultura e as práticas de DevOps (construção de Pipelines e suas devidas estruturações) ao ciclo de vida de Machine Learning [10]. O objetivo do MLOps é industrializar o processo de ML, automatizando a implantação, o monitoramento e a manutenção de modelos em produção, e engloba práticas como integração e entrega contínua (CI/CD), versionamento e governança [15].

Os estágios típicos de um pipeline de ML incluem [14]:

- **Coleta de Dados**: Estágio inicial que envolve a aquisição de dados de diversas fontes.
- **Pré-processamento**: Tratamento e limpeza dos dados brutos para torná-los adequados para o modelo.
- **Extração de Atributos (*Feature Engineering*)**: Criação de novas variáveis (*features*) a partir dos dados existentes para melhorar o desempenho do modelo.
- **Treinamento**: Utilização do algoritmo de ML e dos dados pré-processados/atributos extraídos para construir o modelo.
- **Avaliação**: Medição do desempenho do modelo em dados não vistos para verificar sua eficácia e robustez.
- **Implantação (*Deployment*)**: Disponibilização do modelo treinado para uso em um ambiente de produção, onde ele pode fazer previsões ou tomar decisões.

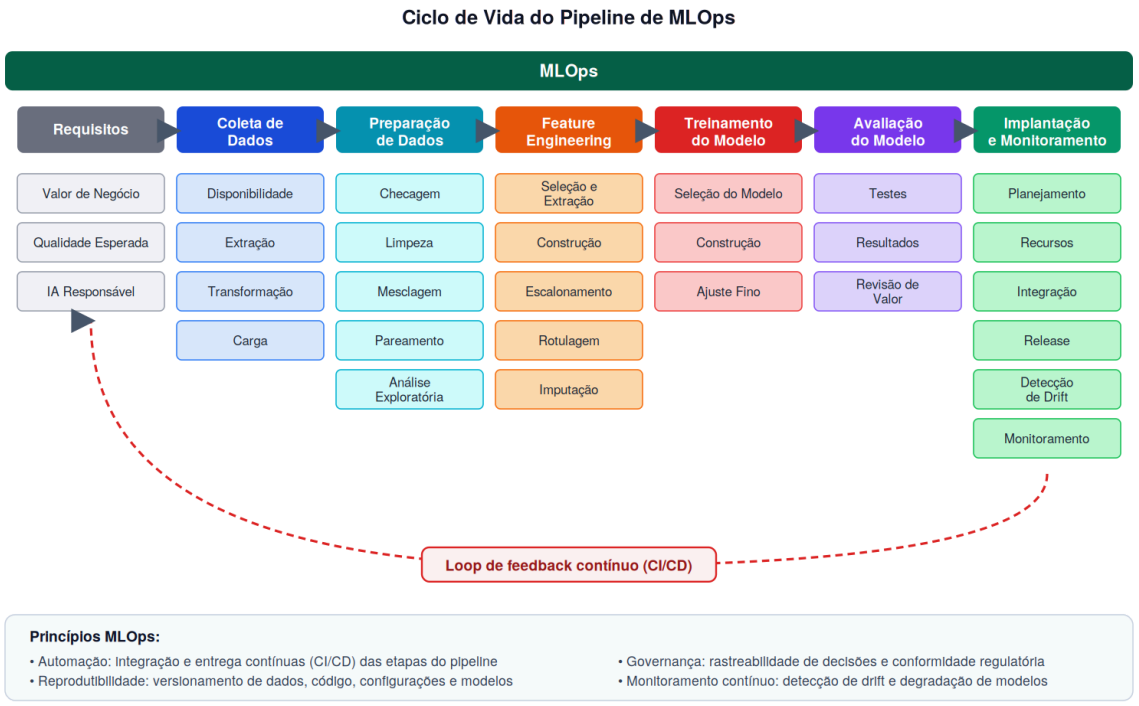

Figura 3: Estrutura conceitual do ciclo de vida de um pipeline de MLOps, organizado em sete etapas sequenciais, desde os requisitos de negócio até a implantação e o

monitoramento contínuo do modelo (Adaptado e traduzido de Reis et al.,(2020) [14]).

A Figura 3 organiza visualmente os estágios típicos de um pipeline de MLOps, evidenciando três propriedades importantes para esta pesquisa: (i) o caráter sequencial e dependente das etapas, em que decisões iniciais (como coleta e preparação de dados) restringem o espaço de escolhas das etapas seguintes; (ii) a presença de um ciclo de feedback contínuo, característico da industrialização do AM; e (iii) a multiplicidade de subatividades em cada etapa, que evidencia o espaço de variabilidade que motivará o uso de feature models na Seção 3.4.

A automação e o rastreamento desses estágios são cruciais para a gestão eficiente do ciclo de vida dos modelos, promovendo a consistência e a reprodutibilidade dos resultados [14]. Contudo, essa automação e a complexidade inerente aos pipelines não estão isentas de desafios e introduzem riscos significativos como a dívida técnica.

3.2.3. Ênfase em Requisitos Não-Funcionais na Modelagem de Pipelines

Guias de engenharia de ML recentes, a exemplo do livro Designing Machine Learning Systems [9], têm enfatizado a necessidade de considerar e integrar requisitos não-funcionais. Esses requisitos são atributos de qualidade que definem "como" um sistema se comporta (incluindo características como desempenho, segurança e usabilidade) em vez de apenas sua funcionalidade ("o que" ele faz), conforme modelado pela norma ISO/IEC 25010:2011 [4]. São atributos definidos desde as fases iniciais da modelagem e desenvolvimento de pipelines, em vez de tratá-los como preocupações secundárias ou tardias [16]. Esses atributos podem ser consideradas características (*features*) e são cruciais para a confiabilidade, a ética e a conformidade de sistemas de ML em produção:

- **Privacidade (*Privacy*)**: Refere-se à garantia de que informações sensíveis sobre indivíduos ou entidades sejam protegidas contra acesso, uso ou divulgação não autorizados ao longo de todo o ciclo de vida do pipeline de ML. Essa garantia abrange desde a coleta inicial dos dados até o armazenamento, processamento, treinamento e implantação do modelo [4].
- **Equidade (*Fairness*)**: Concerne à capacidade de um modelo de ML de operar de forma imparcial sem vieses que possam levar a tratamentos ou resultados desiguais e discriminatórios para diferentes grupos ou indivíduos [17]. O objetivo é garantir que as decisões do sistema não prejudiquem sistematicamente minorias ou grupos vulneráveis [4].
- **Interpretabilidade (*Explainability*)**: É definida como a extensão em que um humano pode compreender a causa e o efeito de uma decisão tomada por um modelo de ML, bem como a transparência de seu funcionamento interno. Modelos mais interpretáveis permitem que os stakeholders (usuários, reguladores, desenvolvedores) compreendam "por que" uma previsão ou decisão específica foi gerada [4].

A incorporação precoce dessas características ou atributos no design do pipeline permite a criação de sistemas de ML mais robustos, éticos e alinhados às expectativas sociais e regulatórias, conforme defendido por abordagens modernas de MLOps. A robustez e a integridade promovidas pelo MLOps são essenciais para a aplicação bem-sucedida de IA em domínios específicos, como o reconhecimento de doenças em plantas.

3.3. Inteligência Artificial (IA) para Reconhecimento de Doenças em Plantas

A aplicação de Inteligência Artificial (IA) no reconhecimento de doenças em plantas representa uma área de pesquisa e desenvolvimento de crescente importância, com o potencial de revolucionar a agricultura por meio da detecção precoce de patologias e da otimização de intervenções. A capacidade de detectar e diagnosticar essas patologias de forma precoce e precisa é crucial para a implementação de medidas de controle eficazes e para a minimização de perdas, como destacado por [18], que ressalta a importância da detecção acurada para a segurança alimentar.

Neste cenário, a IA, impulsionada pelos avanços em visão computacional e aprendizado de máquina, emerge para o reconhecimento automatizado de doenças em plantas. Essa promessa é substancialmente concretizada por meio da aplicação de Modelos de Visão Computacional. Segundo [11], esses modelos têm demonstrado um desempenho superior em comparação com métodos tradicionais de identificação. Contudo, é preciso considerar os desafios inerentes à generalização desses modelos em ambientes agrícolas reais, uma preocupação amplamente discutida na literatura, incluindo a *feature* de condições de iluminação e a complexidade do fundo das imagens [3].

3.3.1. Modelos de Visão Computacional e seus Desafios

Modelos de visão computacional permitem aos computadores “ver”, processar e interpretar imagens e vídeos do mundo real, emulando a percepção visual humana [19], têm demonstrado capacidades notáveis no reconhecimento de doenças em plantas. A Figura 4 apresenta alguns exemplos de uso relacionados à Visão Computacional e sua importância social, profissional e acadêmica.

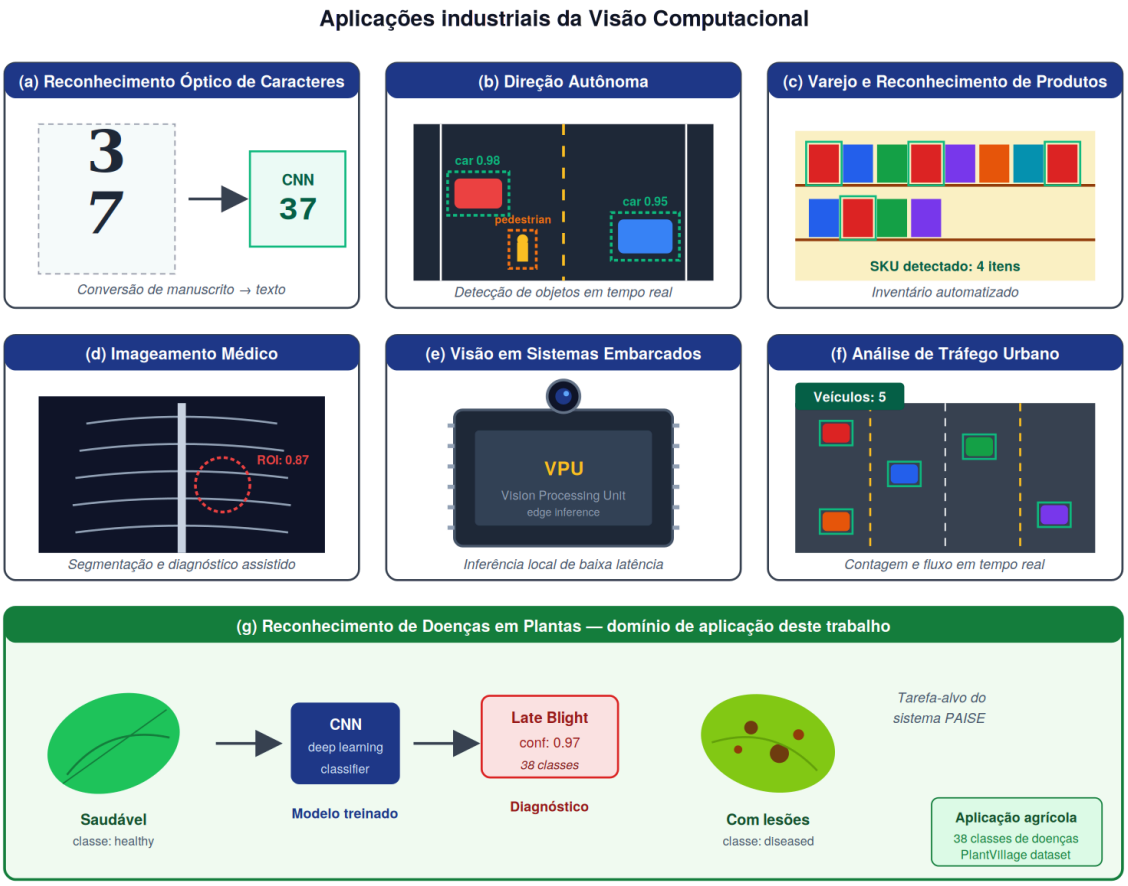

Figura 4: Painel de aplicações representativas da visão computacional em diferentes domínios industriais e científicos (Adaptado e traduzido de Szeliski (2010) [19]*).*

A Figura 4 contextualiza o reconhecimento de doenças em plantas dentro de um conjunto mais amplo de aplicações da visão computacional, evidenciando duas características fundamentais para esta pesquisa. Primeiro, todos os domínios ilustrados compartilham um pipeline conceitual semelhante (aquisição → pré-processamento → modelo → inferência → decisão), o que reforça a relevância de gerenciar essa variabilidade de forma sistemática. Segundo, cada domínio impõe restrições não funcionais distintas, latência crítica em (b) e (e), confiabilidade em (d), escalabilidade em (c) e (f), que precisam ser explicitadas desde a fase de projeto. O painel (g) destaca o domínio-alvo deste trabalho, no qual essas mesmas tensões entre desempenho funcional e restrições não funcionais (custo, energia, latência) serão tratadas pelo PAIES feature model.

Especificamente, as Redes Neurais Convolucionais (CNNs) profundas, são modelos de aprendizado profundo que se destacam na análise de dados visuais ao aprenderem hierarquias de características diretamente de imagens, de forma hierárquica e automática [20], as quais alcançam consistentemente acurácias superiores.

A acurácia é uma métrica que representa a proporção de previsões corretas (verdadeiros positivos e verdadeiros negativos) em relação ao total de previsões realizadas pelo modelo [21] em bases de dados de referência, como a PlantVillage (uma coleção pública de imagens de plantas com diversas doenças, amplamente utilizada para pesquisa em diagnóstico automatizado [2]). Tem ultrapassado 99% na identificação de lesões foliares, manifestações visíveis de danos ou anomalias nas folhas das plantas, indicativas de doenças ou estresses [3].

A transição desses modelos de ambientes controlados de laboratório para cenários reais em campo enfrenta problemas significativos que comprometem a generalização, a capacidade de um modelo de manter seu desempenho e robustez em dados não vistos que diferem ligeiramente dos dados de treinamento, refletindo a variabilidade do mundo real [21]. Os principais desafios incluem: (i) a variabilidade de espécies (diferenças morfológicas entre cultivares); (ii) as condições de iluminação (luz solar variável e sombras); (iii) e a escassez de rótulos (a dificuldade e o custo de obter grandes volumes de imagens anotadas por especialistas para cada nova doença ou espécie) [22].

A persistência desses problemas, especialmente a dependência de grandes volumes de dados rotulados e a sensibilidade a variações ambientais, impede a ampla adoção de modelos de visão computacional em contextos agrícolas dinâmicos. Para superar essas limitações e aprimorar a capacidade de generalização dos modelos, a pesquisa em aprendizado de máquina tem explorado estratégias que permitem o aprendizado eficaz com dados limitados e a adaptação a novos domínios. Tais abordagens, incluindo o Few-Shot Learning e as técnicas de Domain Adaptation.

3.3.3. A Importância da Curadoria de Datasets Agrícolas

Apesar dos avanços nos modelos e algoritmos, a qualidade e a representatividade dos dados continuam sendo fatores determinantes para o sucesso da IA no reconhecimento de doenças em plantas. Trabalhos dedicados à curadoria de datasets agrícolas, o processo sistemático de coleta, limpeza, organização, anotação e validação de conjuntos de dados específicos para aplicações agrícolas, garantindo sua qualidade, integridade e adequação para o treinamento de modelos de ML [23], têm ressaltado desafios persistentes, como as lacunas de balanceamento de classes e a escassez de metadados.

- **Balanceamento de Classes**: Refere-se à distribuição de amostras entre as diferentes categorias ou classes em um conjunto de dados. Lacunas de balanceamento de classes ocorrem quando algumas classes (ex: doenças raras) são representadas por um número significativamente menor de amostras do que outras (ex: plantas saudáveis ou doenças comuns), o que pode levar modelos a serem viesados e a apresentarem desempenho inferior nas classes minoritárias [24].

- **Metadados**: São “dados sobre dados”, ou seja, informações descritivas e contextuais que acompanham as imagens ou amostras (como tipo de câmera, condições de iluminação, localização geográfica, severidade da doença). A escassez de metadados limita a capacidade dos modelos de aprender padrões mais robustos e de generalizar para diferentes cenários, uma vez que informações cruciais sobre o contexto da coleta de dados estão ausentes [23].

Essas questões de dados reforçam a necessidade crítica de mecanismos formais para selecionar e configurar adequadamente as etapas de pré-processamento, as arquiteturas de redes neurais e as métricas de avaliação. A qualidade da curadoria dos dados é, portanto, um pré-requisito para o desenvolvimento de sistemas de IA confiáveis e eficazes na detecção de doenças em plantas. Nesse cenário, a aplicação de *Feature Models* emerge como uma abordagem promissora, alinhada a pesquisas que investigam o uso de conceitos da Engenharia de Linhas de Produto para gerenciar a *feature* e permitir a reutilização de módulos de Redes Neurais Profundas [13].

3.4. *Feature Models* para Seleção de Pipelines em IA Agrícola

A crescente complexidade e a diversidade de requisitos em aplicações de IA, especialmente em domínios críticos como a agricultura, demandam mecanismos formais para a gestão e a seleção sistemática de componentes de pipelines. Pesquisas recentes propõem estender a representação de *feature models* com o objetivo de representar não apenas as etapas funcionais de um pipeline de ML, mas também seus atributos de qualidade [13].

Esses atributos de qualidade (F1-Score, acurácia, consumo energético, dentre outros) expandem a capacidade de configuração para além da funcionalidade básica, incluindo critérios como fairness (equidade, a imparcialidade do modelo, evitando viéses discriminatórios [17]) robustez, custo energético (consumo de energia associado à execução e inferência de modelos de ML e pipelines [25]), garantia de integração com aplicações (capacidade de um pipeline de se conectar e operar de forma transparente com outros sistemas, plataformas e fluxos de trabalho existentes no ecossistema de software, assegurando um fluxo de dados e controle eficiente [26]) e usabilidade demonstram como tais extensões no meta-modelo de *feature models* permitem derivar configurações de pipeline, que satisfazem múltiplas restrições simultaneamente.

Aplicando *Feature Models* para Seleção de Pipelines em IA Agrícola ao reconhecimento de doenças em plantas, pode-se modelar *features* intrínsecas ao domínio agrícola e ao processo de desenvolvimento do pipeline, tais como:

- **Aquisição de imagem**: Diversas fontes e tecnologias para captura de dados (ex: drone RGB, câmera hiperespectral e smartphone).

- **Estratégia de aumento de dados (Data Augmentation)**: Técnicas para expandir o conjunto de dados de treinamento (ex: rotação, GAN, síntese espectral).
- **Backbone da CNN**: Diferentes arquiteturas de redes neurais convolucionais (ex: EfficientNet, Swin-Transformer, MobileNet), cada uma com suas características de desempenho e requisitos computacionais.
- **Estratégia de balanceamento**: Métodos para lidar com desequilíbrio de classes em datasets (ex: focal loss, reamostrage, aprendizado custo-sensível).
- **Métrica de avaliação**: Métricas de desempenho utilizadas para avaliar o modelo (ex: F1-score, mean Average Precision (mAP), Kappa de Cohen (κ)).

A formalização dessas opções e suas interdependências dentro de um Feature Model permite que dependências (ex.: imagens hiperespectrais requerem operadores de normalização específicos) e restrições (ex.: arquitetura MobileNet + Edge TPU para inferência embarcada exclui o uso de GPUs de alta performance) tornem-se verificáveis por um selecionador automático. Podendo assim minimizar o retrabalho experimental, que é oneroso em termos de tempo e recursos, além de facilitar a rastreabilidade de requisitos regulatórios.

4. PAIES *Feature Model*

Este capítulo apresenta o resultado central como artefato principal deste trabalho, alinhado à metodologia DSRM. O artefato desenvolvido é um feature model cujo objetivo é apoiar, de forma sistemática, a tomada de decisão na criação de pipelines de IA para o reconhecimento de doenças em plantas da flora brasileira.

4.1. Estrutura e Componentes do Modelo

O PAIES *feature* model foi projetado para representar a variabilidade de pipelines de AM no contexto do Intelligent Agricultural Expert System (IAES). O IAES () é definido neste trabalho como um sistema baseado em AM destinado ao suporte à decisão em contextos agrícolas, integrando aquisição de imagens de folhas, inferência via modelo de visão computacional e entrega de diagnóstico fitossanitário.

O *feature model* é organizado em cinco *features* principais, alinhadas ao ciclo de vida clássico de modelos de AM [14]: as *features* *DataSource*, *Preprocessing*, *ModelTraining* e *ModelEvaluation* são mandatórias, pois constituem o fluxo mínimo de qualquer pipeline de AM. A *feature Deployment* é opcional, sendo adequada para cenários de prototipagem nos quais a implantação formal não é o objetivo imediato.

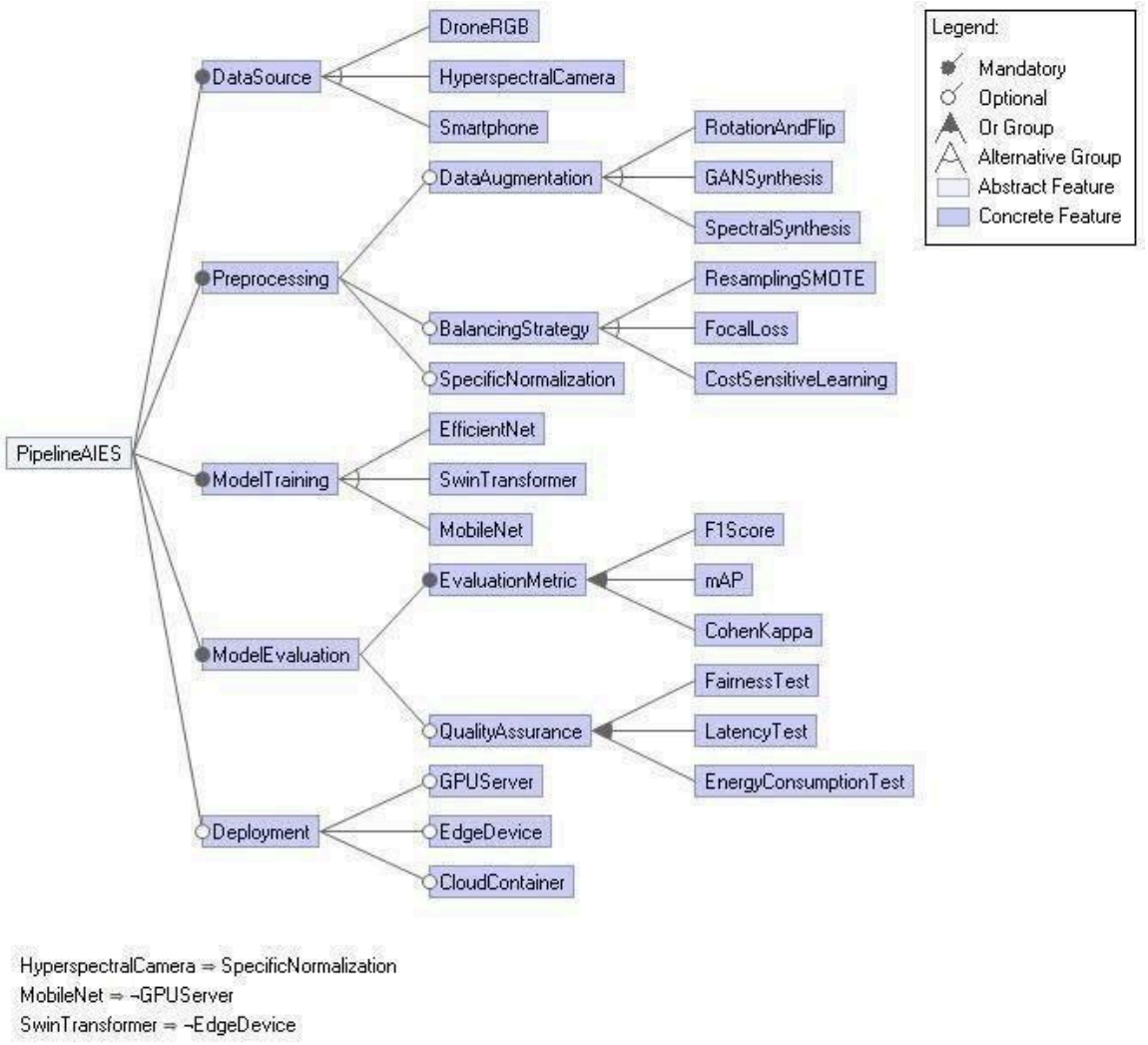

Figura 5 — Feature model para a configuração de pipelines de IA agrícola, modelado na ferramenta FeatureIDE. A figura ilustra a decomposição hierárquica do pipeline em suas etapas principais e respectivas features. Ferramenta disponível em: https://featureide.github.io/

A Figura 5, ilustra a decomposição hierárquica do pipeline em suas cinco *features* principais. A seguir, detalham-se os subgrupos de cada *feature*.

4.2. Detalhamento da Variabilidade e Restrições

A principal contribuição do *Feature* Model está na forma como ele torna explícita a variabilidade do IAES, articulando alternativas como *features* de implementação e suas implicações em termos funcionais e não funcionais. A seguir, detalha-se cada grupo de *features*, com exemplos de *features* de base.

4.2.1. *DataSource*

A etapa *DataSource* representa as tecnologias de captura de imagem disponíveis para o sistema IAES. Ela é modelada como um grupo de escolha exclusiva (*Alternative Group*), incluindo:

- **DroneRGB** – imagens de alta resolução capturadas por drones, típicas de cenários de agricultura de precisão;
- **HyperspectralCamera** – sensores hiperespectrais, adequados para análises mais finas de estresse e doenças;
- **Smartphone** – câmera de dispositivos móveis, cenário de menor custo e maior acessibilidade para produtores.

Essa *feature* como variabilidade reflete diferentes custos de aquisição, resoluções e condições de uso, permitindo ao stakeholder selecionar a fonte de dados que melhor se alinha ao contexto de aplicação.

4.2.2. *Preprocessing*

A etapa *Preprocessing* agrupa transformações aplicadas às imagens antes do treinamento e inferência. Ela é dividida em dois subgrupos principais:

- **DataAugmentation** (grupo Or):
  - *RotationAndFlip* (rotações, espelhamentos);
  - *ColorJitter* (variação de brilho/contraste);
  - *GANSynthesis* (geração de imagens sintéticas via GANs).
- **BalancingStrategy** (grupo de escolha exclusiva):
  - *ClassWeighting* (pesos de classe na função de perda);
  - *Oversampling* (ex.: SMOTE);
  - *FocalLoss* (perda focada em exemplos difíceis).

Essas *features* permitem configurar pipelines que vão desde cenários mais leves, com aumento de dados simples, até configurações mais sofisticadas com síntese de imagens e estratégias avançadas de balanceamento, como as utilizadas na Configuração A (Seção 5.2).

4.2.3. *ModelTraining*

A etapa *ModelTraining* concentra a *feature* mais crítica em termos de impacto nas métricas de qualidade. O grupo *CNNBackbone* é definido como *Alternative Group*, incluindo:

- **EfficientNet** – otimizada para acurácia versus FLOPs;
- **SwinTransformer** – arquitetura baseada em Transformers para visão, com alto poder de representação;
- **MobileNet** – projetada para execução leve em dispositivos móveis e de borda.

Cada arquitetura carrega atributos de qualidade distintos (e.g., acurácia esperada, tamanho do modelo, consumo de energia), o que permite ao *Feature Model* representar trade-offs entre performance e custo computacional já na fase de engenharia de requisitos de qualidade.

4.2.4. *ModelEvaluation*

Na etapa *ModelEvaluation*, o modelo permite a seleção de múltiplas *EvaluationMetric* (grupo Or):

- Accuracy, F1-Score, Precision, Recall e, opcionalmente, métricas mais robustas como MCC ou AUROC;
- *QualityAssurance* (*feature* opcional), cobrindo testes adicionais, como robustez a ruído, variação de iluminação ou generalização entre cultivares.

Essa *feature* abstrata, com variantes concretas distintas, reflete a necessidade de avaliar o modelo sob perspectivas diferentes, conforme recomendado em estudos empíricos em Engenharia de Software e Aprendizado de Máquina.

4.2.5. *Deployment* e Restrições *Cross-Tree*

A etapa *Deployment* inclui plataformas de implantação, tais como:

- **GPUServer** – servidores com GPU para inferência em nuvem;
- **CloudContainer** – ambientes conteinerizados em provedores de nuvem;
- **EdgeDevice** – dispositivos de borda, como smartphones ou gateways IoT.

Para garantir a consistência das configurações, o *Feature* Model incorpora restrições cross-tree, por exemplo:

- *Hyperspectral Camera requires SpecificNormalization* (uso de sensores hiperespectrais exige normalização específica).
- *MobileNet excludes GPUServer* (MobileNet foi projetada para dispositivos de borda; sua combinação com GPUServer é redundante e contrária ao objetivo de baixo custo).

Além disso, cada feature folha foi anotada com atributos não funcionais, como Custo Energético, Latência, Uso de Memória e Complexidade Computacional, com valores derivados da literatura e de medições empíricas [X, Y]. Esses atributos são utilizados no Capítulo 5 para embasar a análise comparativa das configurações derivadas, seguindo um protocolo sistemático de estudo empírico inspirado em [27] e [28].

4.3. Configurações derivadas

Das possíveis configurações válidas do PipelineAIES *feature model*, duas foram selecionadas para este estudo empírico, orientadas por objetivos de qualidade opostos.

4.3.1. Pipeline A - Configuração Orientada a Desempenho

A Pipeline A foi derivada selecionando as *features* que maximizam o desempenho funcional, assumindo recursos computacionais abundantes. As seleções estão registradas em *config_A_performance.yaml*: DataSource = DroneRGB; Augmentation = [RotationAndFlip, GANSynthesis]; Balancing = FocalLoss; CNNBackbone = SwinTransformer; Deployment = CloudContainer.

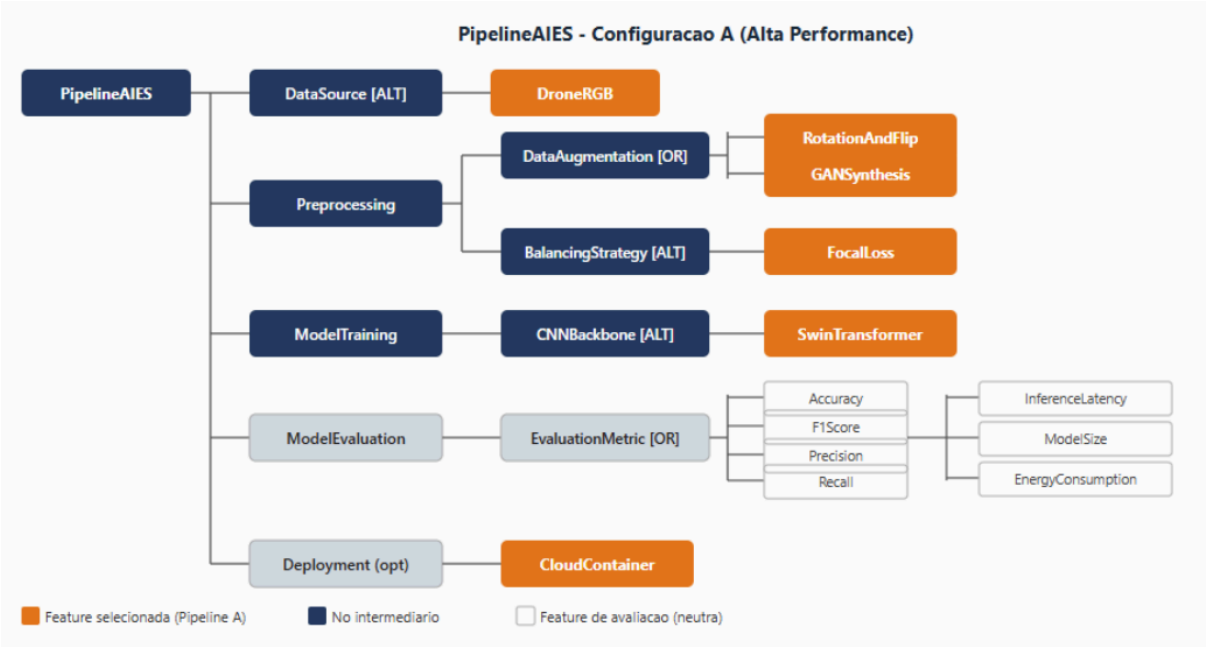

Figura 6: Configuração derivada — Pipeline A (Alta Performance). Features selecionadas destacadas no PipelineAIES feature model: DroneRGB, GANSynthesis, FocalLoss, SwinTransformer e CloudContainer.

4.3.2. Pipeline B - Configuração Orientada a Desempenho

A Pipeline B foi derivada para minimizar o custo computacional, visando execução em dispositivos com recursos limitados. As seleções estão registradas em *config_B_custo.yaml*: DataSource = Smartphone; Augmentation = [RotationAndFlip]; CNNBackbone = MobileNet; Deployment = EdgeDevice.

Percebe-se como a combinação de *features* orientada pelo *feature model* incorpora, desde a fase de projeto, uma preocupação explícita com restrições de borda e custo, alinhada aos atributos não funcionais associados às *features*.

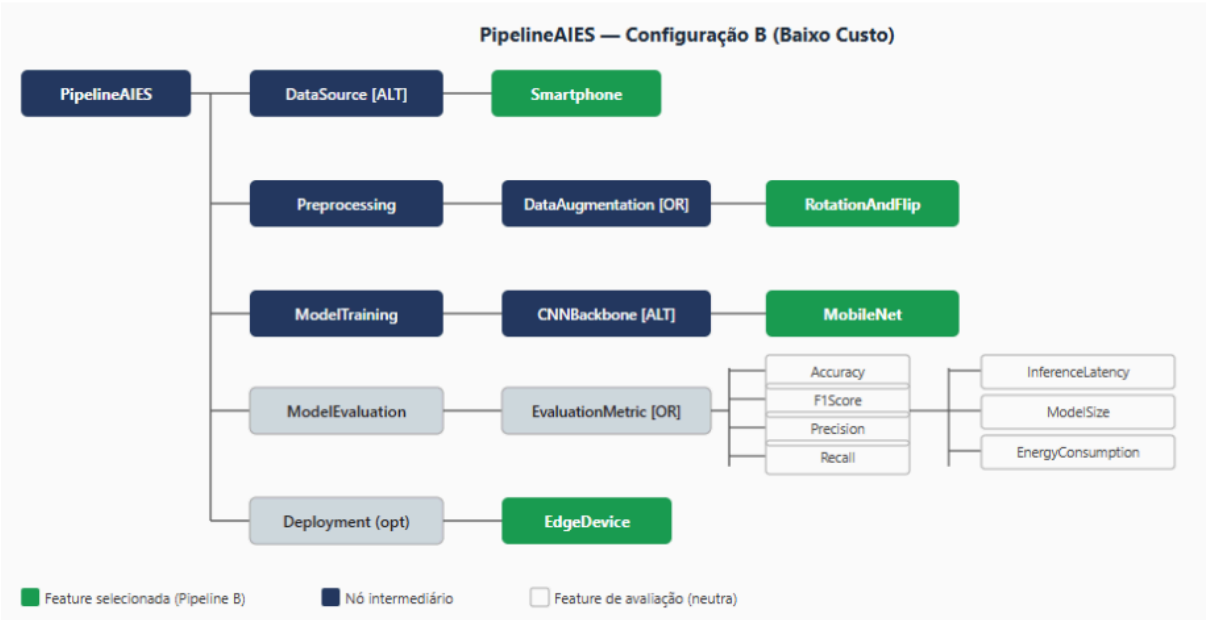

Figura 7: Configuração derivada, Pipeline B (Baixo Custo). Features selecionadas destacadas no PipelineAIES feature model: Smartphone, RotationAndFlip, MobileNet e EdgeDevice.

5. Resultados e Discussão

Esta seção apresenta e discute os resultados obtidos com as configurações derivadas do PAIES *feature model*. A avaliação segue um protocolo de estudo empírico baseado nas referências de [27] e [28].

5.1. Protocolo do Estudo Empírico

O estudo empírico foi estruturado conforme descrito a seguir.

Objetivo (GQM). Conforme definido na Seção 2: avaliar se o uso do *feature model* permite derivar configurações com perfis de desempenho distintos e previsíveis, alinhados a requisitos funcionais e não funcionais.

Hipóteses. Conforme definidas na Seção 2: H0 (equivalência) e H1 (diferença relevante nos requisitos não funcionais). Ao longo desta seção, busca-se rejeitar ou não rejeitar H0 no contexto deste estudo empírico.

Contexto e Objetos de Estudo.

- Dataset: PlantVillage [2], com imagens rotuladas de folhas saudáveis e doentes em 38 classes.
- Particionamento: conjuntos de treino (70%), validação (15%) e teste (15%), fixados via *code/prepare_data.py* (random seed = 42, estratificado por classe).
- Métricas funcionais: F1-Score macro e Acurácia macro.
- Métricas não funcionais: latência média de inferência (ms), tamanho do modelo (MB) e consumo de energia (J).

Ambiente Experimental

- Ambiente de software isolado em virtualenv Python, com versões registradas em *requirements.txt*;
- Controle de versão de código e configurações via Git;
- Execuções registradas com logs de hiperparâmetros, seeds e métricas via PyTorch Lightning e WandB.

Procedimento. Cada configuração foi treinada e avaliada seguindo o mesmo protocolo de treinamento, com conjunto de teste idêntico, de forma a controlar variáveis externas, conforme o paradigma de estudos empíricos em Engenharia de

Software [28]. O treinamento foi realizado via *code/main.py* e a avaliação via *code/evaluate.py* [31].

Esse desenho não configura um estudo empírico controlado completo no sentido estrito de [28], mas um estudo empírico quantitativo sistemático, com objetivo de observar e comparar o comportamento das configurações derivadas pelo feature model em um contexto realista de uso.

5.2. Resultados Quantitativos

A Tabela 1 apresenta os resultados consolidados das métricas funcionais e não funcionais coletadas para as duas configurações derivadas.

Métricas	Pipeline A (Alta Performance)	Pipeline B (Baixo Custo)	Variação
F1-Score	0.9931	0.9593	3.5%
Acurácia	0.9923	0.9607	3.3%
Latência Média	59.46 ms	18.48 ms	320%
Tamanho Modelo	315.69 MB	48.98 MB	640%
Consumo Energia	1.46 Joules	0.31 Joules	470%

Tabela 1: Resultados comparativos das configurações derivadas do PAIES feature model, avaliadas no dataset PlantVillage [2].

5.3 Análise Visual dos Resultados

As figuras a seguir apresentam os resultados de forma visual, facilitando a interpretação dos trade-offs entre as duas configurações.

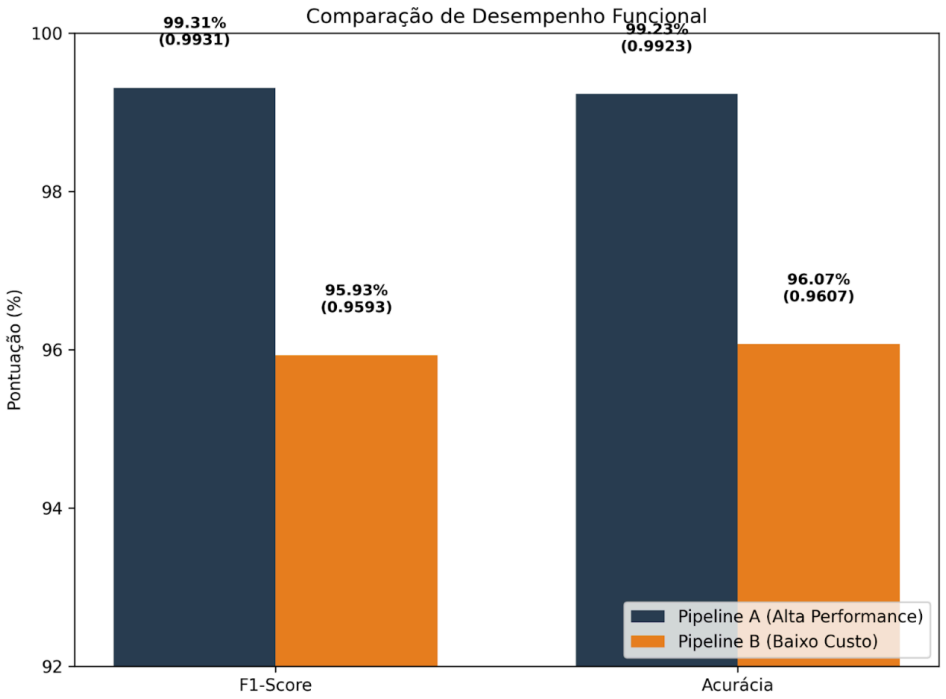

Figura 8: Comparação do desempenho funcional (F1-Score Macro e Acurácia Macro) entre as configurações derivadas do PAIES feature model. Pipeline A: SwinTransformer + CloudContainer. Pipeline B: MobileNet + EdgeDevice. Dataset: PlantVillage [2].

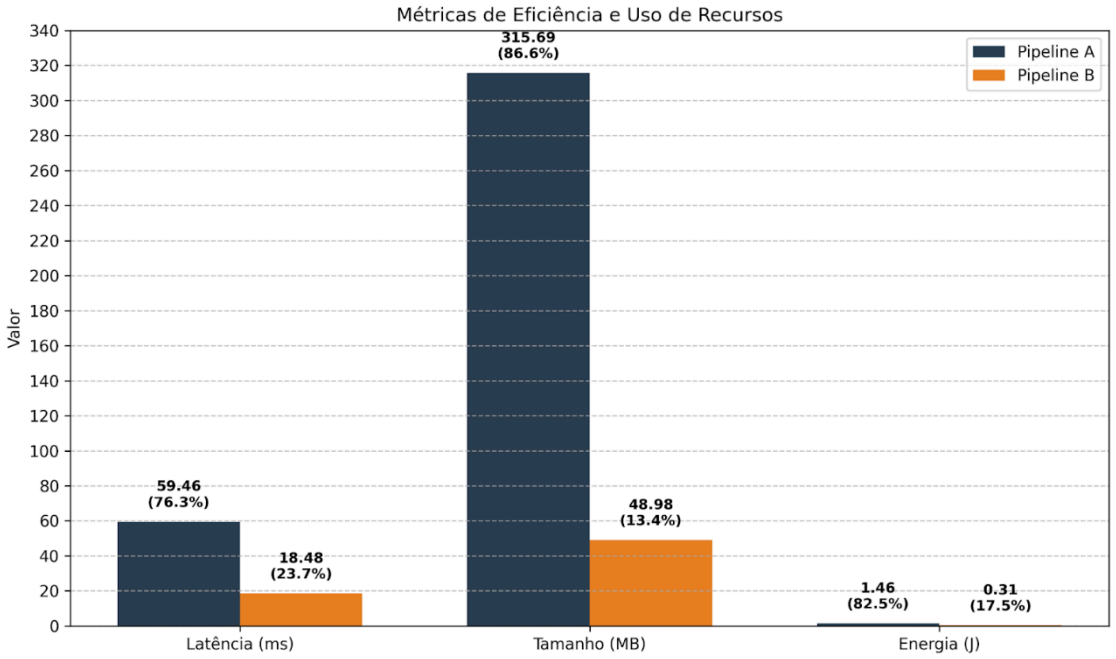

Figura 9: Comparação das métricas não funcionais (Latência Média em ms, Tamanho do Modelo em MB e Consumo de Energia em J) entre as configurações derivadas do PAIES feature model. Pipeline A apresenta latência 3,2 vezes, tamanho 6,4 vezes e energia 4,7 vezes maiores em relação à Pipeline B.

5.4. Análise Comparativa e Discussão

Os resultados da Tabela 1 e das Figuras 8–9 fornecem evidências empíricas para a análise das hipóteses. A seguir, discute-se cada dimensão de trade-off, buscando explicar as causas das diferenças a partir das decisões de *features* capturadas no *feature model*.

5.4.1 Latência

A Pipeline A utiliza o SwinTransformer [29], com aproximadamente 28 milhões de parâmetros e mecanismo de atenção em janelas deslizantes. A Pipeline B utiliza o MobileNet [30], com aproximadamente 5,4 milhões de parâmetros e convoluções separáveis por profundidade, projetadas especificamente para inferência eficiente.

A diferença de latência (59,46 ms vs. 18,48 ms) reflete a diferença fundamental no número de operações por inferência: o SwinTransformer realiza operações de atenção com complexidade quadrática em relação ao tamanho da janela, enquanto o MobileNet utiliza convoluções separáveis que reduzem drasticamente o número de multiplicações necessárias. Essa diferença decorre de decisões arquiteturais refletidas diretamente na *feature CNNBackbone* do *feature model*.

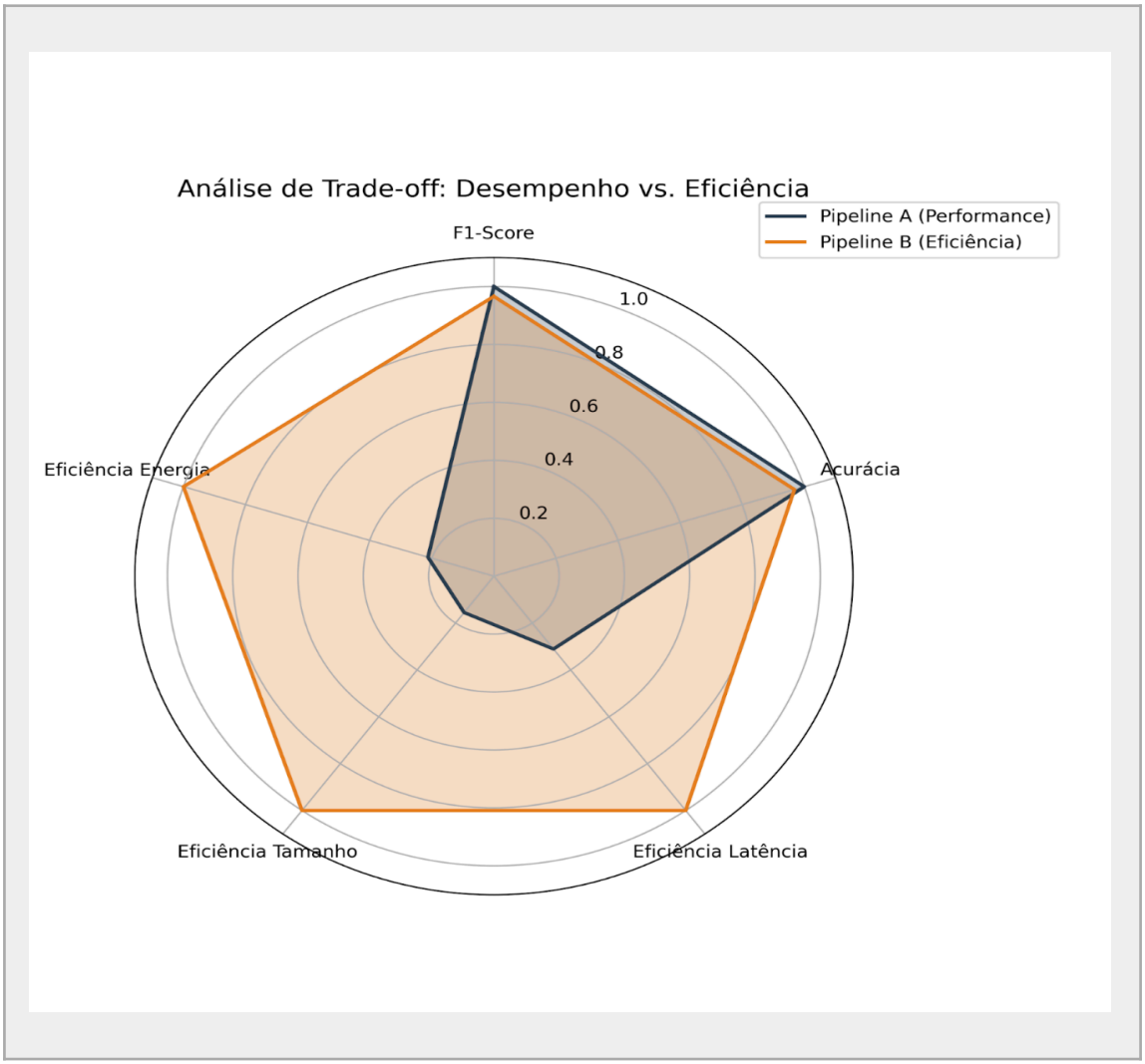

Figura 10: Análise de trade-off entre desempenho e eficiência das configurações derivadas (gráfico radar). Eixos normalizados em [0, 1], em que 1,0 representa o melhor resultado em cada dimensão. Pipeline A domina nos eixos funcionais; Pipeline B domina nos eixos não funcionais.

5.4.2 Tamanho do Modelo

O SwinTransformer gera um checkpoint de 315,69 MB, enquanto o MobileNet gera 48,98 MB. Essa diferença tem implicações diretas para o cenário de implantação: modelos superiores a 300 MB são impraticáveis para instalação em dispositivos de borda com armazenamento limitado, como smartphones de entrada. É precisamente por essa razão que a restrição R1 do *feature model* impede a combinação de SwinTransformer com EdgeDevice, tornando a inconsistência verificável antes da execução do treinamento.

5.4.3. Consumo de Energia

A Pipeline A consumiu 1,46 J por inferência, contra 0,31 J da Pipeline B. Acredita-se que essa diferença resulte do maior volume de operações de ponto flutuante do SwinTransformer, em especial as operações de atenção de dimensões elevadas.

Em cenários de aplicação móvel em campo, nos quais o dispositivo opera com bateria, essa diferença acredita-se que possa inviabilizar o uso contínuo da Pipeline A ao longo de uma jornada de trabalho.

5.4.4. Perda Funcional Modesta

A diferença funcional entre as configurações (F1-Score de 0,9931 vs. 0,9593) é de apenas ∼3,4 pontos percentuais. Acredita-se que isso se deva ao fato de o MobileNet, com aprendizado por transferência a partir de pesos pré-treinados no ImageNet, já capturar features discriminativas suficientes para o dataset PlantVillage. As doenças foliares nesse dataset apresentam padrões visuais relativamente discriminativos (manchas, descolorações, texturas características) que não exigem o poder representacional máximo de um Vision Transformer.

5.4.5 Implicações Práticas

- **Pipeline B para aplicações móveis em campo**: a latência de 18,48 ms é inferior ao limiar de 100 ms considerado adequado para experiências responsivas. Isso acredita-se tornar a Pipeline B viável para uso em campo por agricultores com smartphones convencionais, operando sem conexão com servidores remotos.
- **Pipeline A para sistemas em nuvem**: adequada para cenários em que a precisão máxima é crítica e a infraestrutura de nuvem está disponível, como sistemas de monitoramento automatizado em larga escala ou contextos de pesquisa.
- **Trade-off explicitado pelo *feature model*** : ao associar atributos não funcionais às *features* *CNNBackbone* e *Deployment*, o engenheiro pode prever os impactos de qualidade antes de executar o treinamento, economizando tempo e recursos.

5.4.6. Contribuição do *Feature Model*

O PAIES *feature model* não apenas lista as opções de configuração disponíveis, mas ele conecta explicitamente decisões de projeto a impactos quantificáveis em atributos de qualidade variáveis. O resultado observado não é trivial: um engenheiro sem o *feature model* poderia assumir que uma arquitetura mais complexa trará ganhos proporcionais em todas as dimensões. Entretanto, é necessário sistematizar tal seleção para resultados pertinentes. Os resultados mostram que, no contexto deste estudo empírico, a diferença funcional de ∼3,4 p.p. não justifica, para o cenário de campo, uma penalidade de 4,7× em energia e 6,4× em tamanho de modelo.

Desse modo, a análise comparativa indica que o uso do *feature model*: (i) estrutura a seleção de configurações, afastando o processo de decisões ad hoc; (ii) explicita os trade-offs entre requisitos funcionais e não funcionais; e (iii) acredita-se que permita justificar para os stakeholders por que determinada combinação de *features* atende melhor a um cenário específico.

5.4.7. Resposta às Questões de Pesquisa

Em relação a QP1, o PAIES *feature model* representa de forma estruturada a variabilidade de pipelines de AM agrícola, contemplando *features* funcionais (CNNBackbone, DataSource, Preprocessing, ModelTraining, ModelEvaluation) e *features* não funcionais (latência, energia, tamanho).

Em relação a QP2, os resultados indicam que configurações derivadas com objetivos de qualidade distintos produzem diferenças mensuráveis e substanciais em requisitos não funcionais (3,2× a 6,4×), para diferença funcional inferior a 4 p.p. Com base nos resultados, rejeita-se H0 no contexto deste estudo empírico: o uso de *feature* models para configuração de pipelines de AM agrícola produz diferenças relevantes nos requisitos não funcionais das configurações derivadas, mesmo para níveis semelhantes de desempenho funcional.

6. Conclusões

Este trabalho apresentou a engenharia e a avaliação empírica do PAIES *feature* model, um artefato para gerenciamento de *features* em pipelines de AM aplicados ao reconhecimento de doenças foliares em plantas da flora brasileira. O *feature* model foi estruturado em cinco *features* principais com restrições de compatibilidade que guiam a derivação de configurações válidas.

Duas configurações foram derivadas sistematicamente e avaliadas no dataset PlantVillage [2]: (i) Pipeline A, orientada a alto desempenho funcional; e (ii) Pipeline B, orientada a baixo custo computacional. Os resultados indicam diferença funcional modesta (~3,4 p.p. de F1-Score), com diferenças não funcionais substanciais (latência 3,2×, tamanho 6,4×, energia 4,7× maiores na Pipeline A). Os resultados permitem rejeitar H0 no contexto deste estudo empírico.

6.1. Contribuições

As principais contribuições deste trabalho são:

- A definição do PAIES *feature* model para pipelines de IA agrícola, cobrindo desde a captura de dados até a implantação, com *feature* alinhada à literatura e ao contexto de uso.
- A incorporação de atributos/requisitos não funcionais às *features*, permitindo raciocínio sobre qualidade ainda na fase de projeto (design).
- A condução de um estudo empírico sistemático quantitativo, em linha com protocolos de Engenharia de Software experimental [27, 28], apresentando os impactos das decisões derivadas do *feature model*.

6.3. Limitações e Trabalhos Futuros

Entre as limitações, destacam-se: (i) o uso de um único dataset (PlantVillage), que, embora amplamente utilizado, não captura toda a diversidade de condições de campo; (ii) a avaliação em apenas duas configurações representativas de cenários extremos; e (iii) a ausência de um estudo empírico controlado formal com maior número de repetições e análise estatística aprofundada.

Como trabalhos futuros, acredita-se que seja relevante: (i) replicar o estudo em outros datasets e culturas, bem como em condições reais de campo; (ii) ampliar o *feature model* para incluir novas técnicas de pré-processamento, modelos de última geração e estratégias avançadas de implantação; e (iii) realizar estudos empíricos controlados em larga escala, seguindo os protocolos de Engenharia de Software Experimental, de forma a aprofundar a análise de causa e efeito entre decisões de *features* e métricas de qualidade.

Referências

[1] Oerke, E.-C. (2006). Crop losses to pests. The Journal of Agricultural Science, 144(1):31–43.

[2] Hughes, D. P. e Salathé, M. (2015). An open access repository of images on plant health to enable the development of mobile disease diagnostics. arXiv preprint arXiv:1511.08060.

[3] Mohanty, S. P., Hughes, D. P. e Salathé, M. (2016). Using deep learning for image-based plant disease detection. Frontiers in Plant Science, 7:1419.

[4] ISO/IEC (2011). ISO/IEC 25010:2011 – Systems and software Quality Requirements and Evaluation (SQuaRE). International Organization for Standardization.

[5] Peffers, K., Tuunanen, T., Rothenberger, M. A. e Chatterjee, S. (2007). A design science research methodology for information systems research. Journal of Management Information Systems, 24(3):45–77.

[6] Basili, V. R., Caldiera, G. e Rombach, H. D. (1994). The Goal Question Metric Approach. In Encyclopedia of Software Engineering, Wiley.

[7] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E. e Peterson, A. S. (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, Carnegie Mellon University.

[8] Czarnecki, K. e Eisenecker, U. (2000). Generative Programming: Methods, Tools, and Applications. Addison-Wesley, EUA.

[9] Huyen, C. (2022). Designing Machine Learning Systems. O'Reilly Media.

[10] Treveil, M. et al. (2020). Introducing MLOps. O'Reilly Media.

[11] Kamilaris, A. e Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: a survey. Computers and Electronics in Agriculture, 147:70–90.

[12] Apel, S., Batory, D., Kästner, C. e Saake, G. (2013). Feature-Oriented Software Product Lines: Concepts and Implementation. Springer, Berlin.

[13] D'Aloisio, M., Alves, V., de Almeida, E. S. e Neto, F. M. (2022). Modeling Non-Functional Attributes in Machine Learning Pipelines with Extended Feature Models. In Proceedings of the 26th SPLC, pp. 157–167.

[14] Reis, H., Carvalho, M. e Souza, L. (2020). Hands-On Machine Learning for Algorithmic Trading. Packt Publishing.

[15] Kreuzberger, D., Kühl, N. e Hirschl, S. (2023). Machine learning operations (MLOps): Overview, definition, and architecture. IEEE Access, 11:31866–31879.

[16] Schelter, S. et al. (2018). Challenges in production environments. In Workshop on ML Systems at NeurIPS.

[17] Mehrabi, N. et al. (2021). A survey on bias and fairness in machine learning. ACM Computing Surveys, 54(6):1–35.

[18] Barbedo, J. G. A. (2016). A focus on the challenges and limitations of plant disease detection. Tropical Plant Pathology, 41(4):245–249.

[19] Szeliski, R. (2010). Computer Vision: Algorithms and Applications. Springer, New York.

[20] LeCun, Y., Bengio, Y. e Hinton, G. (2015). Deep learning. Nature, 521(7553):436–444.

[21] Goodfellow, I., Bengio, Y. e Courville, A. (2016). Deep Learning. MIT Press.

[22] Barbedo, J. G. A. (2021). Data fusion in agriculture: resolving ambiguities and closing data gaps. Sensors, 21(16):5309.

[23] Barbedo, J. G. A. (2018). Factors influencing the use of deep learning for plant disease recognition. Biosystems Engineering, 172:84–95.

[24] He, H. e Garcia, E. A. (2009). Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering, 21(9):1263–1284.

[25] Schwartz, R. et al. (2020). Green AI. Communications of the ACM, 63(12):54–63.

[26] Bass, L., Clements, P. e Kazman, R. (2012). Software Architecture in Practice. 3. ed., Addison-Wesley.

[27] Jedlitschka, A. e Pfahl, D. (2005). Reporting guidelines for controlled experiments in software engineering. In Proceedings of the 2005 International Symposium on Empirical Software Engineering (ISESE), pp. 95–104.

[28] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B. e Wesslén, A. (2012). Experimentation in Software Engineering. Springer, Berlin.

[29] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S. e Guo, B. (2021). Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. In Proceedings of the IEEE/CVF ICCV, pp. 10012–10022.

[30] Howard, A. et al. (2019). Searching for MobileNetV3. In Proceedings of the IEEE/CVF ICCV, pp. 1314–1324.

[31] Zenodo – Repositório de código e configurações do PAIES. Disponível em: https://zenodo.org/records/19639277