UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE CRATEÚS

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

LAÍS CRISTINA GOMES MELO

TESTE DE SOFTWARE EM MLOPS: UM MAPEAMENTO SISTEMÁTICO DA LITERATURA

CRATEÚS

2025

LAÍS CRISTINA GOMES MELO

TESTE DE SOFTWARE EM MLOPS: UM MAPEAMENTO SISTEMÁTICO DA LITERATURA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de informação do Campus de Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de Bacharel em Sistemas de informação.

Orientador: Prof. Dr. Jose Wellington Franco da Silva.
Coorientadora: Profª. Drª. Amanda Drielly Pires Venceslau.

CRATEÚS

2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

M485t Melo, Laís Cristina Gomes.
Teste de software em MLOPs: um mapeamento sistemático da literatura / Laís Cristina Gomes Melo. – 2025.
70 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Crateús, Curso de Sistemas de Informação, Crateús, 2025.
Orientação: Prof. Dr. Jose Wellington Franco da Silva.
Coorientação: Prof. Dr. Amanda Drielly Pires Venceslau.

1. teste de software. 2. aprendizagem de máquina. 3. operações de aprendizado de máquina. 4. mapeamento sistemático da literatura. I. Título.

CDD 005

LAÍS CRISTINA GOMES MELO

TESTE DE SOFTWARE EM MLOPS: UM MAPEAMENTO SISTEMÁTICO DA LITERATURA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de informação do Campus de Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de Bacharel em Sistemas de informação.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Jose Wellington Franco da Silva (Orientador)
Universidade Federal do Ceará (UFC)

Profª. Drª. Amanda Drielly Pires Venceslau (Coorientadora)
Universidade Federal do Ceará (UFC))

Profa. Me. Lisieux Marie Marinho dos Santos Andrade
Universidade Federal do Ceará (UFC)

Prof. Me. Marciel Barros Pereira
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Em primeiro lugar, gostaria de expressar minha gratidão a mim mesma, por todas as vezes em que persisti, mesmo diante das dificuldades enfrentadas. Apesar de todas as variáveis que surgiram ao longo do processo, consegui encontrar motivação em meus próprios sonhos — sonhos que somente eu poderia realizar, e cuja concretização dependia exclusivamente de mim.

Em segundo lugar, rendo meus agradecimentos aos meus pais, Maria Nivanda Gomes Melo e Antônio Luiz Rodrigues Melo (in memoriam), por terem feito o melhor que puderam por mim, mesmo sem acesso à educação formal e sem compreender plenamente o que eu fazia ou os motivos que me impulsionavam. Reconheço que, com um pouco mais de conhecimento sobre a importância dos estudos, o apoio deles teria sido ainda maior. Agradeço, igualmente, aos meus amigos, que estiveram ao meu lado ao longo dos anos, dividindo alegrias e dificuldades, oferecendo apoio nos momentos mais desafiadores, palavras de incentivo quando mais precisei e, sobretudo, a amizade sincera que tornou essa caminhada mais leve e significativa. Por fim, agradeço também aos professores, tanto pela competência quanto, em alguns casos, pela gentileza. Meu sincero agradecimento a todos.

“Essas falhas e erros são o que eu sou, formando as estrelas mais brilhantes da constelação da minha vida.” (Kim Namjoon, Discurso na ONU, 2018)

RESUMO

Este trabalho teve como objetivo analisar práticas de teste de software no contexto de *Machine Learning Operations (MLOps)*, buscando compreender como garantir qualidade e confiabilidade em sistemas que integram aprendizado de máquina. A justificativa para a pesquisa está na crescente adoção de *MLOps* e na necessidade de métodos de teste adaptados a esse cenário. A metodologia empregada foi um mapeamento sistemático da literatura, conduzido com critérios de seleção bem definidos, visando identificar estudos relevantes sobre o tema. A análise possibilitou a categorização dos tipos de testes aplicados, além da sistematização das práticas relatadas. Os resultados revelaram que, embora haja iniciativas consistentes, ainda existem lacunas importantes, principalmente na definição de padrões e no suporte automatizado aos testes. Conclui-se que o campo apresenta oportunidades de avanço tanto na pesquisa quanto na prática, especialmente no desenvolvimento de ferramentas e diretrizes mais robustas para o teste de software em *pipelines* de *MLOps*.

Palavras-chave: teste de software; aprendizagem de máquina; operações de aprendizado de máquina; mapeamento sistemático da literatura.

ABSTRACT

This work aimed to analyze software testing practices in the context of *MLOps*, seeking to understand how to ensure quality and reliability in systems that integrate machine learning. The justification for this research lies in the growing adoption of *MLOps* and the need for testing methods adapted to this scenario. The methodology employed was a systematic mapping of the literature, conducted with well-defined selection criteria, in order to identify relevant studies on the topic. The analysis enabled the categorization of the types of tests applied, as well as the systematization of the reported practices. The results revealed that, although there are consistent initiatives, important gaps still remain, mainly in the definition of standards and in automated support for testing. It is concluded that the field presents opportunities for advancement both in research and in practice, especially in the development of more robust tools and guidelines for software testing in *MLOps* pipelines.

Keywords: software testing; machine learning; machine learning operations; systematic literature mapping.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CI/CD	Integração Contínua e Entrega Contínua
CNN	Rede Neural Convolucional
DevOps	Desenvolvimento e Operações
IA	Inteligência Artificial
IRR	Revisão Rápida Interativa
ML	Machine Learning
ML-BSA	Machine Learning Boundary Spanning Algorithm
MLOps	Machine Learning Operations
MR	Relações Metamórficas
MT	Metamorphic Testing
NLP	Processamento de Linguagem Natural
OPS	Operações
PICO	PICO (Population, Intervention, Comparison, Outcome)
PMC	Probabilistic Model Checking
RF	Random Forest
TI	Tecnologia da Informação

SUMÁRIO

1 INTRODUÇÃO

O teste de software é fundamental para assegurar a qualidade, confiabilidade e estabilidade do produto final, o que, por sua vez, contribui para a satisfação do usuário e a redução de custos com manutenção e correções após a entrega. Além disso, os testes oferecem uma oportunidade valiosa para identificar melhorias e otimizações que podem ser implementadas para melhorar o desempenho e a usabilidade do software, segundo Zhang *et al.* (2020).

A prática de integrar o desenvolvimento de modelos de aprendizagem de máquina com a operação de sistemas de software, também conhecida como Operações de Aprendizado de Máquina, do inglês *Machine Learning Operations* (*MLOps*), é, de acordo com Kreuzberger *et al.* (2023), uma abordagem que visa automatizar e gerenciar todo o ciclo de vida de desenvolvimento de modelos, desde a concepção e treinamento até a implantação e monitoramento em produção. Além disso, Zaharia *et al.* (2022) destaca que *MLOps* é fundamental para assegurar a reprodutibilidade e governança dos modelos, aspectos essenciais para aplicações em larga escala.

Em seu trabalho, Kreuzberger *et al.* (2023) ainda destaca a importância do *MLOps* para garantir a escalabilidade, confiabilidade e eficácia dos sistemas de Machine Learning *Machine Learning (ML)* em ambientes de produção, assim como o impacto positivo que essa prática pode ter no ciclo de vida completo de desenvolvimento de modelos. De forma complementar, Amershi *et al.* (2019) ressalta que a integração contínua e o monitoramento automatizado são cruciais para a manutenção da qualidade dos modelos em produção.

MLOps, segundo Kreuzberger *et al.* (2023), é uma abordagem interdisciplinar para a gestão eficiente de modelos de aprendizado de máquina em produção, integrando práticas de desenvolvimento de software com a implementação e manutenção de modelos. Essa prática combina áreas como *DevOps*, ciência de dados e especialistas em *ML*, buscando automação, monitoramento contínuo e escalabilidade, afirma Zaharia *et al.* (2022).

Inspirada no paradigma de *DevOps*, *MLOps* visa integrar o desenvolvimento, treinamento, implantação e operação de modelos, permitindo que organizações implementem modelos em produção de forma ágil, confiável e com manutenção eficaz ao longo de seu ciclo de vida, menciona Bass *et al.* (2015).

No ciclo de desenvolvimento de um modelo, a etapa de teste de software é crucial para garantir que o produto final atenda aos requisitos definidos. MAXIM e PRESSMAN (2021) explicam que teste de software consiste em uma série de atividades realizadas pela equipe de desenvolvimento, com o objetivo de verificar se o sistema funciona conforme esperado e está

livre de falhas. Esse processo envolve a verificação sistemática de diversos aspectos do software, como funcionalidades, desempenho e segurança, a fim de identificar e corrigir erros, falhas ou desvios das especificações originais. Conforme Amershi *et al.* (2019), a complexidade dos sistemas de *ML* torna os testes ainda mais desafiadores, demandando abordagens específicas para validar não apenas o código, mas também os dados e os modelos.

1.1 Justificativa

A temática dos testes de software no contexto do *MLOps* revela-se relevante devido ao avanço tecnológico e à crescente adoção da inteligência artificial, o que demanda a operacionalização de modelos de *ML* de forma mais eficiente e escalável. Nesse cenário, o *MLOps* emergiu como uma abordagem essencial, mas trouxe consigo um questionamento fundamental: como os testes são conduzidos nesse novo paradigma? Diferentemente do desenvolvimento tradicional de software, no qual já existem práticas consolidadas, o ciclo de vida dos modelos de *ML* apresenta desafios específicos, como a constante variação dos dados, a degradação do desempenho ao longo do tempo e a necessidade de re-treinamento contínuo. Assim, compreender como os testes se inserem nesse fluxo torna-se essencial para assegurar a qualidade e a confiabilidade das aplicações baseadas em inteligência artificial.

Ao decorrer deste presente trabalho, será abordado o conceito de *MLOps (Operação de aprendizado de máquina)*, destacando sua importância na gestão eficiente de modelos de aprendizagem de máquina em ambientes de produção. Diante disso, foi levantado uma questão de pesquisa, sendo ela:

- **Q1**: Quais são os tipos de testes de software aplicados no contexto de *MLOps* e como eles são realizados para garantir a qualidade e confiabilidade dos modelos de *Machine Learning*?

1.2 Objetivos

1.2.1 Objetivo geral

Investigar como são feitos os testes de software dentro do *MLOps*, destacando similaridades, diferenças, benefícios e lacunas entre as práticas de teste convencionais e aquelas voltadas para o contexto de operações de aprendizado de máquina.

1.2.2 Objetivos específicos

- Compreender os tipos de testes de software aplicados no contexto de *MLOps*, por meio do levantamento e análise de estudos existentes;
- Identificar tendências, lacunas e desafios atuais na área.

1.3 Estrutura do trabalho

O seguinte trabalho divide-se nos capítulos:

- Capítulo 2 - Fundamentação teórica: mostra os conceitos necessários para a base desta pesquisa. Sendo eles: introdução ao *ML*, *MLOps*, *DevOps* e Teste de software;
- Capítulo 3 - Trabalhos relacionados: análise dos trabalhos que se relacionam com a proposta desta pesquisa;
- Capítulo 4 - Metodologia: descrição da proposta do trabalho, as etapas e o protocolo de pesquisa;
- Capítulo 5 - Resultados esperados: descrição dos resultados esperados, análise de dados e cronograma de atividades.

2 FUNDAMENTAÇÃO TEÓRICA

A definição das subseções apresentadas neste capítulo foi orientada pela necessidade de contextualizar progressivamente os principais conceitos que sustentam este trabalho. O avanço tecnológico contínuo tem sido fundamental para a evolução de diversas disciplinas acadêmicas e áreas de pesquisa, esclarece Goodfellow *et al.* (2016). No contexto da *Inteligência Artificial (IA)*, o desenvolvimento de algoritmos e técnicas de aprendizagem de máquina tem desempenhado um papel essencial na automação de tarefas complexas e na análise de grandes volumes de dados, afirma Russell e Norvig (2021), Bishop (2006).

Dessa forma, a Seção 2.1 apresenta os fundamentos do *ML*, considerada a base de muitos sistemas inteligentes modernos. Em sequência, a Seção 2.2 descreve o conceito de *DevOps*, modelo que promove a integração entre desenvolvimento e operação, com impacto direto nos processos de entrega contínua. A Seção 2.3, por sua vez, discute o *MLOps*, uma adaptação dos princípios do *DevOps* ao ciclo de vida de projetos de *ML*. Por fim, a Seção 2.4 explora os testes de software, com ênfase naqueles aplicados no contexto do modelo *MLOps*.

A escolha dessas seções visa fornecer ao leitor uma base conceitual sólida, assegurando uma compreensão clara e abrangente do escopo desta pesquisa.

2.1 Machine Learning

O termo *ML* teve origem com Samuel (1959), que por sua vez é reconhecido por ter introduzido a ideia de aprendizado de máquina como uma área de estudo que permite que computadores aprendam sem necessidade de programação explícita.

Segundo Naqa *et al.* (2015), o *ML* é uma área em constante evolução da computação, focada no desenvolvimento de algoritmos que imitam a capacidade humana de aprendizado ao observar e interagir com o ambiente. Esses algoritmos são projetados para identificar padrões nos dados e utilizá-los para realizar previsões ou tomar decisões. Essa habilidade de aprender com os dados é essencial para uma ampla gama de aplicações, como reconhecimento de voz e imagem, recomendações de produtos e detecção de fraudes. O objetivo principal é permitir que os computadores atuem de forma mais inteligente e autônoma, adaptando-se continuamente às mudanças e novas informações.

Entretanto, a fim de esclarecer a distinção entre *IA* e *ML*, é pertinente a explicação proposta por Mitchell e Mitchell (1997). Eles afirmam que, na *IA* tradicional, o desenvolvedor

codifica explicitamente o conhecimento, fornecendo dados de entrada bem definidos e especificando a classificação que a máquina deve realizar durante a execução. Como exemplo, apresentam o cenário em que os dados de entrada consistem na observação de uma textura de pena com dois olhos e um bico, sendo, então, a tarefa da máquina classificar esses dados como pertencentes a um pássaro, com essa classificação sendo a resposta final do processo de execução.

Já sobre a *ML*, Mitchell e Mitchell (1997) afirma que é uma abordagem baseada em dados, que é fornecido uma série de observações de entrada e saída, e uma função, que pode ser instruções lógicas ou redes neurais, que irão mapear a entrada para a saída com mais precisão. Complementando, "o tipo de algoritmo empregado depende do tipo de problema que se deseja resolver, do número de variáveis, do tipo de modelo que melhor se adequa a ele, entre outros fatores", segundo Mahesh (2020). Portanto, "Aprendizado de máquina é uma área de *IA* cujo objetivo é o desenvolvimento de técnicas computacionais sobre o aprendizado, bem como a construção de sistemas capazes de adquirir conhecimento de forma automática", como afirma Monard e Baranauskas (2003).

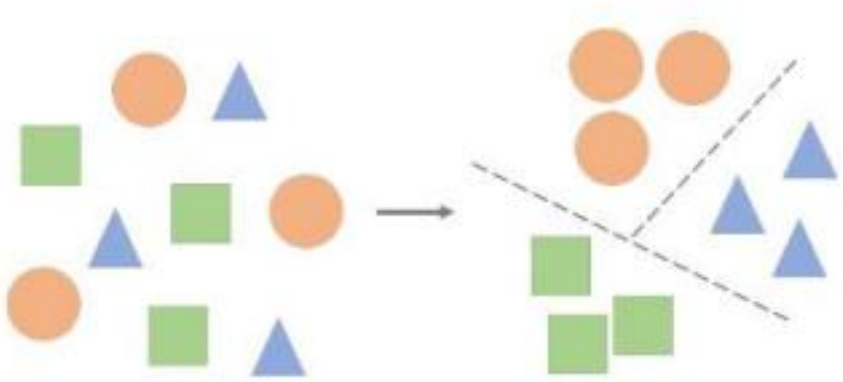
No campo de *ML*, há diversas abordagens ou tipos, cada uma projetada para lidar com diferentes tipos de problemas e características dos dados. A razão para essa variedade de métodos é que os desafios e conjuntos de dados podem ser bastante diversos. Portanto, diferentes tipos de *ML* foram desenvolvidos para atender a essas variações e necessidades específicas. Cada tipo de *ML* é mais adequado para certas situações, dependendo da natureza dos dados disponíveis e do problema que se deseja resolver. Assim, a existência desses diferentes tipos de aprendizado de máquina permite que os profissionais escolham a abordagem mais adequada com base nas características específicas do problema e dos dados que estão enfrentando. Os principais tipos são: supervisionado, não supervisionado, por reforço e semi supervisionado.

Autores como Monard e Baranauskas (2003), Sah (2020), Murphy (2012), Mitchell e Mitchell (1997), Mahesh (2020) e outros, discutem sobre aprendizado de máquina, assim como seus tipos. A seguir, serão abordados os principais tipos de aprendizado de máquina mencionados anteriormente, com base nas informações dos autores referidos.

- Supervisionado: Mahesh (2020) expõe em seu trabalho que o aprendizado supervisionado é uma abordagem de aprendizagem de máquina onde o objetivo é desenvolver uma função que relaciona uma entrada a uma saída, utilizando exemplos de pares de entrada e saída já conhecidos. Esse processo envolve a criação de uma função a partir de um conjunto de dados de treinamento que já possui rótulos. Os algoritmos utilizados nesse tipo de aprendizado requerem

dados externos para serem treinados e aprenderem a fazer previsões ou classificações baseadas nesses dados rotulados. Ademais, Sah (2020) explica em sua pesquisa que o aprendizado supervisionado é usado quando os dados estão organizados em variáveis de entrada e valores de saída desejados. O algoritmo desenvolve uma função que relaciona as entradas às saídas. No entanto, essa abordagem pode ser cara em situações onde há poucos dados rotulados disponíveis, pois exige um grande volume de amostras rotuladas para ser eficaz. Pode-se ter uma visão geral ao observar a Figura 1.

Figura 1 – Visão geral da aprendizagem supervisionada



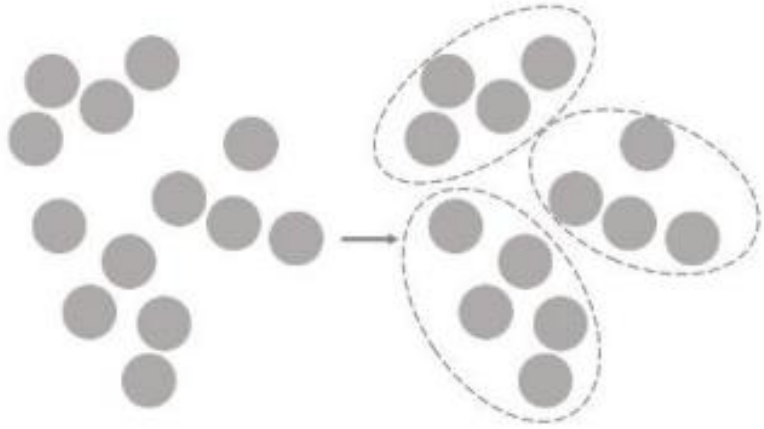
Fonte: Sah (2020)

- Não supervisionado: Mahesh (2020) afirma que essas técnicas são denominadas aprendizado não supervisionado porque, ao contrário do aprendizado supervisionado, não existem respostas corretas predefinidas nem um guia para orientar o processo. Os algoritmos operam de forma autônoma para identificar e expor padrões ou estruturas relevantes nos dados.

De acordo com os algoritmos de aprendizado não supervisionado extraem apenas um número limitado de características dos dados. Quando novos dados são apresentados, eles utilizam essas características previamente aprendidas para identificar a categoria ou classe dos novos dados. De acordo com Sah (2020), o aprendizado não supervisionado é utilizado quando os dados estão disponíveis apenas como entradas, sem uma variável de saída associada. Esses algoritmos são projetados para identificar padrões ocultos nos dados e aprender sobre suas características. Um dos principais métodos de aprendizado não supervisionado é o *clustering*, que organiza um conjunto de objetos em grupos de modo que os objetos dentro de cada grupo sejam mais semelhantes entre si do que com os objetos de outros grupos. Esta técnica é comum em aprendizado de máquina não supervisionado, onde não há rótulos para orientar o agrupamento. Com o *clustering*, os grupos naturais presentes nos dados são descobertos e podem ser utilizados

para prever a saída para novas entradas. A visão geral pode ser observada na Figura 2.

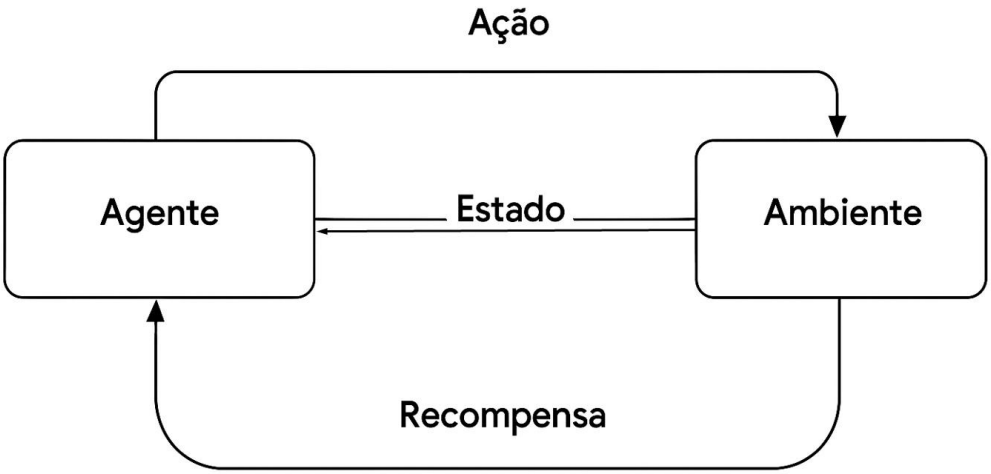
Figura 2 – Visão geral da aprendizagem não supervisionada



Fonte: Sah (2020)

- Por reforço: O aprendizado por reforço é uma área do aprendizado de máquina que se dedica a determinar como agentes de software devem agir em um ambiente para maximizar uma recompensa acumulada. Esse tipo de aprendizado é um dos três paradigmas principais do aprendizado de máquina, ao lado do aprendizado supervisionado e do aprendizado não supervisionado, segundo Mahesh (2020). A tarefa envolve tomar uma série de decisões para alcançar uma recompensa final. Durante o processo, um agente artificial recebe recompensas ou penalidades pelas ações que realiza, e seu objetivo é maximizar a recompensa total obtida. Exemplos de aplicação incluem agentes que aprendem a jogar *videogames* ou a realizar tarefas robóticas com um objetivo final em mente, de acordo com Sah (2020). Na Figura 3, é possível observar um exemplo em que um agente analisa o estado do ambiente e executa ações com o objetivo de maximizar a recompensa total.

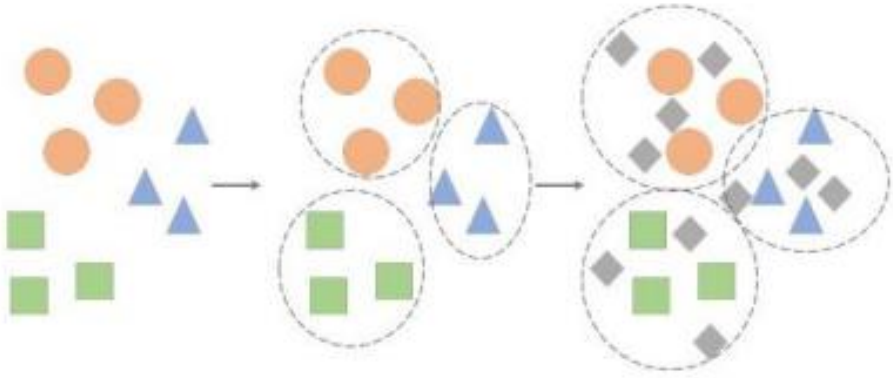
Figura 3 – Visão geral da aprendizagem por reforço



Fonte: Sah (2020)

- Semi supervisionado: Esse método é um intermediário entre as técnicas de aprendizado supervisionado e não supervisionado. Os algoritmos que utilizam essa abordagem são treinados com uma combinação de dados rotulados e não rotulados. Geralmente, há uma pequena quantidade de dados rotulados e uma grande quantidade de dados não rotulados. Essa abordagem pode ser útil em áreas de aprendizado de máquina e mineração de dados, onde já existe uma abundância de dados não rotulados e rotular esses dados pode ser um processo demorado. Nos métodos de aprendizado supervisionado mais comuns, um algoritmo é treinado com um conjunto de dados rotulado, onde cada registro contém as informações do resultado, segundo Mahesh (2020), como observado na Figura 4 a seguir.

Figura 4 – Visão geral da aprendizagem semi supervisionada



Fonte: Sah (2020)

2.2 DevOps

O *DevOps* é a união entre desenvolvimento e operações, e representa uma mudança significativa na cultura organizacional, focada em superar departamentos isolados e promover a cooperação entre diferentes equipes envolvidas no ciclo de vida do desenvolvimento de software, afirma Kim *et al.* (2016). Ao automatizar processos essenciais, como a integração e implantação contínuas, ele visa acelerar o tempo de entrega do software, garantindo ao mesmo tempo sua confiabilidade e qualidade, esclarece Bass *et al.* (2015). Essa abordagem não só melhora a eficiência operacional, mas também promove uma cultura de responsabilidade compartilhada e aprendizado contínuo, afirma Kim *et al.* (2016).

Além disso, o *DevOps* tem como objetivo criar uma mentalidade centrada no cliente, buscando atender às necessidades do usuário final de forma rápida e eficaz segundo Hüttermann (2012). Ao integrar práticas como a entrega contínua, ele capacita as organizações a responder rapidamente às mudanças no mercado e às demandas dos clientes, garantindo ao mesmo tempo a estabilidade e confiabilidade do software entregue, de acordo com Fitzgerald e Stol (2017). Em última análise, o *DevOps* representa uma abordagem holística para o desenvolvimento de software, que visa melhorar a colaboração, eficiência e qualidade dos produtos entregues, segundo Bass *et al.* (2015).

Entretanto, o *DevOps* foi cuidadosamente elaborado para incluir práticas que garantam testes contínuos, assegurando a qualidade do software, juntamente com monitoramento constante, registro de atividades e ciclos de *feedback*, menciona Kim *et al.* (2016). Esses elementos não apenas aceleram o ciclo de desenvolvimento, mas também melhoram a confiabilidade

e a estabilidade das aplicações em produção, de acordo com Bass *et al.* (2015). Dessa forma, o *DevOps* não apenas otimiza a entrega de software, mas também promove uma cultura de responsabilidade compartilhada e aprendizado contínuo entre as equipes envolvidas, segundo Kim *et al.* (2016).

Ademais, ao automatizar processos e promover práticas como *Integração Contínua e Entrega Contínua (CI/CD)*, testes contínuos e monitoramento constante, o *DevOps* capacita as organizações a lançarem software de alta qualidade de forma rápida, segura e confiável, enquanto promove uma cultura de melhoria contínua e responsabilidade compartilhada, argumenta Fitzgerald e Stol (2017).

Kreuzberger *et al.* (2023) argumenta que com a ampla aceitação do *DevOps* na indústria, temos observado o surgimento de uma variedade de ferramentas projetadas para suportar essa metodologia, as quais podem ser classificadas em seis categorias distintas, sendo elas:

- Colaboração e compartilhamento de conhecimento; Exemplo: *Slack*, *Trello*, etc.
- Gerenciamento de código fonte; Exemplo: *Github, Gitlab,* etc.
- Processo de construção; Exemplo: *Maven*.
- Integração contínua; Exemplo: *Jenkins, GitLab CI*, etc.
- Automação de implantação; Exemplo: *Kubernetes, Docker,* etc.
- Monitoramento e registro em *log*; Exemplo: *Prometheus, Logstash*.

2.3 MLOps: Machine Learning Operations

Combinando os conceitos apresentados nas duas seções anteriores sobre *ML* e *DevOps*, esta seção se propõe a explorar o papel do *MLOps*, detalhando o que é e como contribui para a gestão e a operacionalização de modelos de *ML*.

De acordo com o estudo **Machine Learning Operations (*MLOps*): Visão Geral, Definição e Arquitetura**, escrito por Kreuzberger *et al.* (2023), o *MLOps* é uma abordagem essencial para gerenciar eficazmente modelos de aprendizado de máquina em ambientes de produção, combinando princípios de desenvolvimento de software com a implementação e manutenção de modelos. Isso resulta em vantagens como automação de processos, monitoramento contínuo e capacidade de escalonamento. O *MLOps* é descrito como um esforço interdisciplinar, no qual a colaboração entre diferentes equipes é crucial para projetar, gerenciar, automatizar e operar sistemas de *ML* em ambientes de produção. No entanto, uma análise mais detalhada do

artigo mencionado revela que a comunidade de aprendizado de máquina tem se concentrado mais na criação de modelos do que em sua aplicabilidade no mundo real, o que pode gerar problemas ao longo de todo o ciclo de vida do modelo.

Segundo Kreuzberger *et al.* (2023), *MLOps (Machine Learning Operations)* é um paradigma, incluindo aspectos como melhores práticas, conjuntos de conceitos, bem como uma cultura de desenvolvimento quando se trata de conceituação, implementação, monitoramento, implantação e escalabilidade ponta a ponta de produtos de aprendizado de máquina. Acima de tudo, é uma prática de engenharia que aproveita três disciplinas contribuintes: aprendizado de máquina, engenharia de software e engenharia de dados. O *MLOps* visa produzir sistemas de aprendizado de máquina, preenchendo a lacuna entre desenvolvimento e operações (*DevOps*). Baseado nos estudos de Kreuzberger *et al.* (2023), cada princípio do *MLOps* pode ser explicado da seguinte forma:

- Automação de *CI/CD*: Refere-se à automação dos processos de integração contínua (CI) e entrega contínua (CD). A integração contínua envolve a automação da fusão de mudanças de código em um repositório compartilhado, enquanto a entrega contínua garante que essas mudanças sejam prontamente disponibilizadas em ambientes de produção. Esse princípio visa acelerar o ciclo de desenvolvimento, reduzindo erros manuais e melhorando a eficiência;
- Orquestração de fluxo de trabalho: A orquestração de fluxo de trabalho no contexto de *MLOps* envolve o gerenciamento e a coordenação de todas as etapas envolvidas no ciclo de vida de um modelo de aprendizado de máquina. Isso inclui desde a preparação dos dados, treinamento do modelo até a implantação e monitoramento. A orquestração facilita a automação e o controle do fluxo de tarefas, garantindo que o processo ocorra de forma eficiente e sem falhas;
- Reprodutibilidade: Este princípio assegura que os resultados de um modelo possam ser reproduzidos de maneira consistente, independentemente de quando ou onde ele for executado. Isso implica em garantir que todos os aspectos do ambiente de desenvolvimento, como versões de pacotes, parâmetros de modelo e dados, sejam registrados e versionados, permitindo que os experimentos sejam repetidos com os mesmos resultados;
- Versionamento de dados, modelo e código: O versionamento envolve o controle de alterações em todas as partes do sistema de aprendizado de máquina, incluindo

o código-fonte, os dados utilizados para treinar o modelo e o próprio modelo. Esse controle é essencial para manter a rastreabilidade e a consistência entre as versões e para possibilitar a colaboração eficiente entre diferentes equipes;

- Colaboração: A colaboração refere-se ao trabalho conjunto de diferentes equipes, como engenheiros de dados, cientistas de dados e desenvolvedores, para criar e implantar modelos de aprendizado de máquina. *MLOps* promove práticas que facilitam a comunicação e o compartilhamento de conhecimento entre essas equipes, melhorando a eficiência e a qualidade do trabalho;
- Treinamento e avaliação contínua de *ML*: Este princípio garante que os modelos de aprendizado de máquina sejam treinados e avaliados de forma contínua. À medida que novos dados são gerados ou as condições do ambiente mudam, os modelos precisam ser atualizados e avaliados novamente para garantir que continuem eficazes e precisos. Isso ajuda a evitar que os modelos se tornem obsoletos ou menos eficientes ao longo do tempo;
- Rastreamento e registro de metadados de *ML*: O rastreamento de metadados envolve a captura e o registro detalhado de informações sobre os experimentos de *ML*, como parâmetros do modelo, métricas de desempenho e características dos dados. Esses metadados são essenciais para a análise e auditoria dos resultados, bem como para a reprodução de experimentos passados;
- Monitoramento contínuo: O monitoramento contínuo envolve a observação constante do desempenho do modelo em produção, a fim de identificar quaisquer mudanças ou degradação no desempenho ao longo do tempo. Esse princípio é crucial para garantir que o modelo continue a fornecer resultados precisos e relevantes em ambientes de produção dinâmicos;
- Ciclos de *feedback*: Ciclos de *feedback* são processos nos quais o desempenho do modelo é avaliado, e com base nesse *feedback*, ajustes são feitos para melhorar a precisão e a eficácia do modelo. Isso pode envolver a coleta de novos dados, ajuste de parâmetros do modelo ou modificações no próprio código. O ciclo de *feedback* contínuo garante a evolução constante e a melhoria do modelo, promovendo a adaptação às mudanças nos dados ou no ambiente.

MLOps precisa ser um processo feito em equipe para ser implementado de forma eficaz. Na citação anterior foi dito que o *MLOps* é composto por três disciplinas contribuentes,

sendo elas: aprendizagem de máquina, engenharia de software e ciência dos dados. Porém, ainda nesse mesmo artigo, Kreuzberger *et al.* (2023) descreve as funções necessárias para a implementação do *MLOps*, pois "*MLOps* é um processo de grupo interdisciplinar, e a interação de diferentes funções é crucial para projetar, gerenciar, automatizar e operar um sistema de *ML* em produção". As funções são:

- **Gerente de projetos**: O gerente de projetos é responsável por coordenar e planejar as etapas do desenvolvimento de projetos, garantindo que as metas sejam alcançadas dentro dos prazos e orçamentos estabelecidos. No contexto de *MLOps*, esse papel envolve o gerenciamento da integração entre as equipes de diferentes disciplinas (dados, engenharia e operações) para garantir que todos os aspectos do projeto sejam bem executados;
- **Arquiteto de Tecnologia da Informação (TI)**: O arquiteto de TI é responsável por projetar e estruturar a infraestrutura tecnológica necessária para suportar as operações de aprendizado de máquina. Esse papel envolve a definição da arquitetura de sistemas e como diferentes componentes (banco de dados, serviços de computação, plataformas de aprendizado de máquina, etc.) se integram para garantir escalabilidade, segurança e eficiência;
- **Cientista de dados**: O cientista de dados é responsável pela análise dos dados e pelo desenvolvimento de modelos de aprendizado de máquina. Esse profissional coleta, prepara e analisa dados, além de aplicar técnicas de aprendizado de máquina para extrair *insights* ou construir modelos preditivos. O cientista de dados trabalha diretamente com dados e modelos, mas em *MLOps*, também é fundamental garantir que esses modelos sejam entregues de forma consistente e eficiente;
- **Engenheiro de dados**: O engenheiro de dados projeta, constrói e mantém as infraestruturas de dados necessárias para suportar os projetos de aprendizado de máquina. Eles são responsáveis por garantir que os dados estejam acessíveis, processados corretamente e disponíveis para o treinamento e avaliação de modelos. O trabalho do engenheiro de dados envolve principalmente a manipulação de grandes volumes de dados e a integração de fontes de dados diversas;
- **Engenheiro de software**: O engenheiro de software desenvolve as aplicações e sistemas que utilizam os modelos de aprendizado de máquina. Esse papel envolve

escrever o código que integra os modelos de *ML* com os sistemas de produção, garantindo que as soluções de aprendizado de máquina sejam implementadas de forma eficaz e escalável dentro do ambiente de *TI*;

- **Engenheiro *DevOps***: O engenheiro *DevOps* é responsável por automatizar e melhorar os processos de desenvolvimento e operações. No contexto de *MLOps*, esse papel envolve garantir a integração contínua (CI) e a entrega contínua (CD) de modelos de aprendizado de máquina, além de facilitar a colaboração entre as equipes de desenvolvimento e operações. Esse profissional também trabalha na infraestrutura de implantação e automação de *pipelines*;
- **Engenheiro *MLOps***: O engenheiro *MLOps* é responsável pela implementação das práticas de *MLOps* e pelo gerenciamento do ciclo de vida dos modelos de aprendizado de máquina. Ele trabalha na automação do treinamento, implantação, monitoramento e manutenção contínua dos modelos, além de garantir que os princípios de *MLOps*, como versionamento, rastreamento de metadados e monitoramento contínuo, sejam seguidos ao longo de todo o ciclo de vida do modelo.

2.4 Teste de software

Após abordar os conceitos fundamentais de *ML* e a operacionalização de modelos em *MLOps*, esta seção se concentra em testes de software, destacando sua importância para garantir a qualidade, confiabilidade e robustez de sistemas que integram aprendizado de máquina.

Segundo MAXIM e PRESSMAN (2021), teste de software consiste em uma sequência de atividades que precisam ser executadas dentro da equipe de desenvolvimento de software, com o objetivo de garantir que o produto está atendendo os requisitos solicitados. Partindo desse conceito, é certo dizer que este processo envolve a verificação sistemática de diferentes aspectos do software para identificar e corrigir erros, falhas ou desvios em relação às especificações originais. O teste de software é essencial para assegurar a qualidade e a confiabilidade do produto final, contribuindo para a satisfação do usuário e a redução de custos com manutenção e correções pós-entrega. Além disso, o processo de teste ajuda a identificar melhorias e otimizações que podem ser implementadas para aprimorar o desempenho e a usabilidade do software. Segundo Neto e Claudio (2007), o objetivo das atividades de teste é identificar falhas em um produto, permitindo que suas causas sejam corrigidas pela equipe de desenvolvimento antes da entrega

final. Essa característica torna o processo de teste destrutivo e não construtivo, uma vez que seu foco é aumentar a confiança no produto por meio da identificação de problemas, mas sempre antes de sua entrega ao usuário final.

Alguns conceitos básicos sobre teste de software deve ser abordado para um melhor entendimento. Sendo assim, Neto e Claudio (2007) destaca em seu trabalho as definições com relação a defeito, erro e falha. Dentro de um projeto de desenvolvimento de sistemas a equipe de qualidade deve ser capaz de identificar cada um destes e fazer o *report* adequado. Retornando aos conceitos básicos, Neto e Claudio (2007) expõe em sua pesquisa as seguintes definições:

- Erro: desvio da especificação, ou seja, incoerência entre o resultado obtido e o resultado esperado;
- Defeito: instrução ou comando incorreto;
- Falha: processamento incorreto e resultado inconsistente. "Uma falha pode ter sido causada por diversos erros e alguns erros podem nunca causar uma falha", de acordo com Neto e Claudio (2007)

Ainda sobre os conceitos básicos, é importante mencionar que os testes de software possuem elementos que são essenciais para a formalização da atividade, segundo Neto e Claudio (2007). Sendo eles:

- Caso de teste: De acordo com Neto e Claudio (2007) que referencia Craig e Jaskiel (2002), é composto por valores de entrada, resultado esperado e valores de saída em cima de uma condição específica a ser testada;
- Procedimento de teste: também conhecido como plano de teste, de acordo com Neto e Claudio (2007) que referencia Craig e Jaskiel (2002), é uma descrição dos passos necessários para executar um caso de teste ou um grupo de teste;
- Critério de teste: segundo Neto e Claudio (2007) que referencia Maldonado *et al.* (2001), é usado para avaliar e selecicionar casos de teste para aumentar a possibilidade de provocar falhas.

Na qualidade de software existem diversos tipos de testes que podem ser realizados. Em sua pesquisa, Braga (2019) destaca os tipos de testes que serão listados a seguir, e, adicionado a isso, as definições adaptadas do Aguiar (2023).

- ”Teste funcional: Este teste é caracterizado por possuir o objetivo de validar se as funções disponíveis pelo software estão de acordo com o especificado pelo cliente na documentação do produto, como por exemplo requisitos de negócios e

requisitos técnicos";

- "Teste de carga: é o teste que tem como objetivo simular a carga em condições de normais de uso.";
- "Teste de instalação: tipo de teste que verifica se o software é instalado corretamente em diferentes hardwares sob condições adversas como pouca memória,internet ruim e interrupções durante a instalação";
- "Teste de segurança: são testes focados em verificar se o sistema funciona de maneira segura e se os dados estão protegidos";
- Teste de volume: é o teste que visa simular o comportamento do sistema em períodos com uma maior quantidade de dados;
- Teste de stress: tem como objetivo obervar o comportamento do sistema em períodos que a quantidade de dados esteja no limite ou além do suportado;
- Teste de regressão: considerado um dos testes mais importantes e consiste em testar todos as funcionalidades já existentes para garantir que novas atualizações não causaram problemas;
- Teste de manutenção: teste que verifica se a mudança de ambiente gerou impacto no funcionamento do sistema;
- "Teste de usabilidade: teste que foca na utilização da aplicação pelo usuário, se o layout está correto, se a interface está de acordo com as necessidades do cliente, entre outros";
- "Teste de fumaça: teste que verifica os pontos mais críticos do software, é executado após a aplicação estar pronta no ciclo de desenvolvimento, para ser encaminhada para o ciclo de testes";

Na Subseção 2.4.1 será abordado os tipos de testes mais utilizados dentro do modelo *MLOps*.

2.4.1 Teste de software dentro das Operações de Aprendizado de Máquina

O teste de software em sistemas de *ML* é suscetível a uma série de desafios em comparação com o teste de sistemas de software tradicionais, afirma Marijan e Gotlieb (2020). Seguindo nessa linha de raciocínio, Marijan e Gotlieb (2020) ressalta que diversos pesquisadores tentaram aplicar testes convencionais, testes em sistemas que não utilizam *ML*, e que enfrentaram diversos desafios. Observaram que os testes convencionais precisam de adaptação para o contexto

de *ML*. Entretanto, ele frisa que diferentemente dos sistemas tradicionais que funcionam com instruções determinísticas pré-programadas, os sistemas de aprendizado de máquina operam com base em raciocínio estocástico. Esse tipo de raciocínio, fundamentado em probabilidades, introduz incerteza na resposta do sistema, resultando em um comportamento não determinístico, que pode ser imprevisível ou não especificado. Devido a essa característica, os sistemas de *ML* podem alterar seu comportamento conforme aprendem ao longo do tempo. Isso implica que as saídas do sistema podem variar para as mesmas entradas de teste ao longo do tempo, complicando bastante a especificação dos casos de teste.

Marijan e Gotlieb (2020) aponta que, geralmente, os casos de teste são especificados com entradas definidas para o sistema em teste e resultados esperadas para essas entradas, conhecidas como oráculos de teste. No entanto, devido ao raciocínio estocástico, a saída de um sistema de *ML* não pode ser antecipadamente especificada; isso significa que os sistemas de *ML* não têm valores esperados fixos contra os quais as saídas reais possam ser comparadas durante os testes. Marijan e Gotlieb (2020) faz referência ao Weyuker (1982) para afirmar que esse problema é reconhecido em sistemas tradicionais e denominados sistemas não testáveis. E para esses casos, Weyuker (1982) diz que uma abordagem que tem sido considerada para sistemas não testáveis são os pseudo-oráculos. Pseudo-oráculos são uma abordagem de teste diferencial que envolve a execução de vários sistemas que cumprem a mesma especificação do sistema original em teste. Esses sistemas recebem as mesmas entradas, e suas saídas são comparadas. Diferenças nas saídas são interpretadas como indicativos de possíveis erros no sistema sob teste (Marijan e Gotlieb (2020)).

Canuma (2021) destaca que o *MLOps* e o *DevOps* possuem semenlhanças diante do contexto de entrega contínua, testes de unidade, testes de integração e entrega contínua. Mas que o *MLOps* difere nos pontos a seguir:

- Integração Contínua (CI): responsável por testar e validar dados, esquemas de dados e modelos.
- Implantação Contínua (CD): "não se refere mais a um único pacote de software ou serviço, mas a um sistema (um pipeline de treinamento de *ML*) que deve implantar automaticamente outro serviço (serviço de previsão de modelo) ou reverter alterações de um modelo", afirma Canuma (2021);
- Teste Contínuo (TC): "é uma nova propriedade, exclusiva dos sistemas de *ML*, que se preocupa em treinar e atender automaticamente os modelos", afirma

Canuma (2021).

Riccio *et al.* (2020) evidencia em sua pesquisa que na engenharia de software clássica há níveis de teste, e dentre eles são incluídos os testes de unidade, teste de integração e teste de sistema. Ambos podem ser usados em testes em sistemas de aprendizado de máquina, mas como citado anteriormente, é necessário adaptações. No trabalho de Riccio *et al.* (2020) ele distingue os tipos de teste em quatro níveis: teste de entrada, teste de modelo, teste de integração e teste de sistema. Os tipos de teste serão explicados abaixo, sabendo que é baseado na pesquisa do Riccio *et al.* (2020).

- **Teste de entrada** são os testes de analisam os dados do treinamento que fora usados para treinar os componentes de *ML*. Os testes de nível de entrada não têm o objetivo de descobrir falhas diretamente, mas sim de identificar possíveis causas para um treinamento insatisfatório com os dados utilizados. Dessa forma, eles ajudam a reduzir o risco de falhas causadas por incertezas nas previsões.
- **Teste de modelo** são os testes que podem ser considerados equivalentes aos testes unitários para unidades. Os testes de nível de modelo analisam um modelo de *ML* de forma isolada, sem considerar os outros componentes.Esses testes buscam identificar falhas relacionadas a problemas como arquiteturas de modelo inadequadas, erros no processo de treinamento ou configurações incorretas dos hiperparâmetros. Para realizar esses testes, normalmente são usadas métricas clássicas como a precisão (para classificadores) ou o erro quadrático médio (para regressões), com base em um conjunto de dados de teste que já está rotulado.
- **Teste de integração** é o teste realizado para avaliar o comportamento do sistema quando integrado com outras partes já existentes do sistema. Teste em que se combinam e avaliam a interação entre componentes de software, *hardware*, ou ambos. O teste de integração se concentra em identificar problemas que aparecem quando vários modelos e componentes são integrados, mesmo que cada um funcione bem individualmente.
- **Teste de sistema** é realizado no sistema completo e integrado para verificar se ele atende aos requisitos estabelecidos. O objetivo desse teste é avaliar o modelo em sua totalidade, operando no ambiente em que será utilizado. Isso envolve verificar o desempenho do sistema em condições reais ou simuladas para garantir que ele funcione conforme o esperado e lide adequadamente com diferentes

cenários e desafios.

Portanto, sabe-se que os testes de software são de suma importância em qualquer tipo de sistemas, sejam eles os tradicionais ou um sistema de aprendizado de máquina. Desde que sejam adaptados se necessário, é possível testar modelos.

Diante do exposto, este capítulo apresentou os principais conceitos que sustentam o entendimento do modelo *MLOps*, partindo da definição de *Machine Learning* e avançando para os fundamentos do *DevOps*, até alcançar a integração entre esses dois domínios por meio do *MLOps*. Além disso, foram discutidos os testes de software no contexto do *MLOps*, com foco nas particularidades e desafios que esse modelo impõe às práticas tradicionais de garantia de qualidade. A explanação desses conceitos se mostrou essencial para estabelecer uma base sólida para as discussões que se seguirão, permitindo uma compreensão mais clara sobre o funcionamento do *MLOps* e o papel dos testes de software dentro desse ciclo de vida.

3 TRABALHOS RELACIONADOS

A seleção dos trabalhos relacionados descritos neste capítulo justifica-se pela abrangência e complementaridade dos temas abordados, os quais são fundamentais para sustentar a pesquisa em torno das operações de aprendizado de máquina (*MLOps*). O estudo de Lima *et al.* (2022) foi incluído por oferecer uma revisão sistemática que consolida o estado da arte em práticas, ferramentas, papéis e desafios do *MLOps*, apresentando uma visão estruturada de como práticas de *DevOps*, como *CI/CD*, são adaptadas para dar suporte à implantação e manutenção contínua de modelos de aprendizado de máquina. Essa base é essencial para entender as etapas do ciclo de vida e os papéis profissionais envolvidos.

Complementando essa perspectiva, o trabalho de Najafabadi *et al.* (2024) aprofunda a análise ao mapear detalhadamente os componentes arquiteturais que formam o ecossistema de *MLOps*, categorizando-os e avaliando o suporte de ferramentas específicas para cada um. Essa análise amplia o entendimento técnico ao destacar as interdependências entre os componentes e evidenciar lacunas de suporte, oferecendo subsídios para decisões técnicas e de pesquisa voltadas à melhoria de fluxos e integração de ferramentas.

Por fim, o estudo de Calefato *et al.* (2024) agrega uma dimensão crítica ao abordar os riscos de segurança associados à adoção de *MLOps* e as práticas recomendadas para mitigá-los. Ao tratar de ameaças como ataques cibernéticos e degradação de desempenho, além de propor medidas como criptografia, políticas de acesso e isolamento de ambientes, o trabalho reforça a importância de incorporar a cultura de segurança ao ciclo de vida dos modelos.

3.1 MLOps: Practices, Maturity Models, Roles, Tools, and Challenges – A Systematic Literature Review

O artigo apresenta uma análise do estado da arte sobre *MLOps* por meio de uma revisão sistemática da literatura. O trabalho parte do contexto de que o avanço das soluções baseadas em *machine learning* trouxe novos desafios para a operacionalização de modelos em ambientes de produção. Para superar essas barreiras, surgem práticas inspiradas em *DevOps*, como integração e entrega contínua, adaptadas para o ciclo de vida de modelos de *machine learning*.

Com o objetivo de entender como a comunidade científica vem abordando esse tema, os autores realizaram uma busca automatizada em bases de dados como ACM, IEEE Xplore,

ScienceDirect e SpringerLink. A pesquisa inicial identificou 1.905 artigos, mas, após rigorosos critérios de seleção, apenas 30 estudos foram incluídos na análise final. Os resultados mostram que o *MLOps* ainda é um campo em consolidação acadêmica, com abordagens fragmentadas e grande potencial para novos estudos.

A investigação foi guiada por cinco perguntas de pesquisa. A primeira delas buscou entender como os modelos de *machine learning* são implantados em produção. Foi constatado que a maioria dos artigos foca em descrever o ciclo de vida dos projetos de *machine learning*, mas poucos detalham as etapas práticas de operacionalização, como monitoramento contínuo, versionamento de dados e modelos ou automação de *pipelines*. Alguns trabalhos destacam práticas como o CD4ML (Continuous Delivery for *Machine Learning*), que propõe a automação de ponta a ponta.

Em relação aos modelos de maturidade, foram identificadas adaptações do modelo CMM (Capability Maturity Model) que organizam o desenvolvimento de *machine learning* em cinco níveis: inicial, repetível, definido, gerenciado e otimizado. Esses níveis avaliam capacidades como gestão de *pipelines* de dados, qualidade de modelo, transparência, justiça e confiabilidade. Apesar disso, a maioria dos estudos não trata a maturidade do *MLOps* de forma específica, revelando uma lacuna para futuras pesquisas.

O levantamento também mapeou as funções e papéis mais citados no contexto de *MLOps*. Entre eles estão o gestor de implantação, cientistas de dados, engenheiros de dados, desenvolvedores e especialistas de domínio. Esses profissionais atuam em diferentes fases do ciclo de vida, desde a coleta de dados e o treinamento de modelos até a avaliação, monitoramento e implantação em ambiente real.

Sobre as ferramentas, a revisão identificou algumas soluções amplamente citadas, como o *MLflow*, que possibilita rastrear experimentos, registrar modelos e controlar versões. Outras plataformas como *Kubeflow* também são mencionadas por oferecerem *pipelines* escaláveis para *machine learning*.

Por fim, os principais desafios destacados envolvem a falta de práticas padronizadas, a dificuldade de integração entre as equipes de desenvolvimento e operações, além da necessidade de monitoramento constante para evitar a degradação do desempenho do modelo ao longo do tempo. Problemas como o drift de dados, a gestão de novas versões e a governança robusta também são apontados como pontos críticos.

Em síntese, o estudo conclui que o *MLOps* é uma área promissora, mas que ainda

carece de referências consolidadas na literatura científica. Há uma grande oportunidade para pesquisas que desenvolvam modelos de maturidade específicos, guias de boas práticas aplicáveis a ambientes reais e estudos de caso que aproximem a teoria da prática nas organizações. Assim, espera-se que novos trabalhos ajudem a impulsionar a adoção de *MLOps* de forma mais estruturada, confiável e automatizada.

3.2 An Analysis of MLOps Architectures: A Systematic Mapping Study

O artigo analisa como arquiteturas de *MLOps* vêm sendo definidas na literatura científica recente, buscando preencher uma lacuna importante: apesar da rápida adoção do paradigma *MLOps* na indústria, ainda falta um conhecimento arquitetônico consolidado que ajude equipes técnicas a projetar sistemas robustos. Para isso, os autores realizaram um Estudo de Mapeamento Sistemático (SMS) com 43 estudos primários publicados entre 2020 e 2024, cobrindo um período de consolidação e amadurecimento da área.

O objetivo central foi entender quais são os componentes mais comuns em arquiteturas *MLOps*, como esses componentes se relacionam entre si e quais ferramentas são utilizadas para implementá-los. O método incluiu busca automatizada, seleção criteriosa, extração de dados em dupla e síntese por meio de técnicas de *card-sorting*. Essa abordagem resultou na identificação de 35 componentes arquitetônicos agrupados em seis grandes categorias, como curadoria de dados, *CI/CD*, treinamento de modelos, armazenamento e versionamento, inferência, infraestrutura e serviços de suporte.

Os resultados detalham, por exemplo, que muitos artigos propõem arquiteturas independentes de domínio, mas há casos específicos, como soluções voltadas para computação de ponta, manufatura e saúde. Além disso, o estudo mapeia variantes arquitetônicas, destacando quatro combinações principais de componentes, como a adição de módulos de treinamento online e mecanismos de inferência. O diagrama de componentes UML construído pelos autores ajuda a visualizar como esses elementos se conectam em um fluxo coerente, do armazenamento de dados brutos até a entrega de inferências em produção.

Em relação às ferramentas, o levantamento identificou 76 opções mencionadas, sendo que algumas, como *Jenkins*, *AWS SageMaker*, *MLflow* e *Kubeflow*, aparecem com frequência por suportarem múltiplos componentes, especialmente aqueles ligados a integração contínua, versionamento e implantação. Ainda assim, algumas lacunas foram detectadas: diversos componentes — como repositórios de conjuntos de dados ou bancos de dados de *feedback* — têm pouco suporte

de ferramentas específicas, o que evidencia oportunidades para pesquisa e desenvolvimento de soluções mais completas.

Por fim, o estudo destaca desafios como a falta de padronização na nomenclatura dos componentes, o uso de diagramas informais que misturam níveis de abstração e dificultam comparações, e a existência de arquiteturas específicas de domínio que nem sempre podem ser generalizadas. Como contribuição prática, os autores sugerem que o material sintetizado pode servir de base para profissionais que projetam sistemas *MLOps* e para pesquisadores interessados em consolidar melhores práticas e padrões arquitetônicos no campo.

3.3 Security Risks and Best Practices of MLOps: A Multivocal Literature Review

Este artigo realiza uma Revisão Multivocal da Literatura para investigar o estado atual do conhecimento sobre os riscos de segurança e as práticas recomendadas em sistemas habilitados por *machine learning* operados sob o paradigma de *MLOps*. Diante da expansão do uso de *machine learning* em ambientes críticos, os autores argumentam que os fluxos de trabalho de *MLOps* são vulneráveis a riscos complexos, que exigem abordagens específicas de proteção. Inspirado em conceitos da engenharia de software e no *DevOps*, o *MLOps* incorpora práticas de automação como *CI/CD*, mas ainda carece de definições consolidadas de segurança voltadas para suas particularidades.

O estudo foi guiado por três perguntas principais: (1) existe uma definição amplamente aceita de segurança em *MLOps*? (2) Quais são os riscos mais críticos que afetam os *pipelines* de *MLOps*? e (3) Quais práticas são recomendadas para mitigar esses riscos? Para responder, os autores analisaram tanto literatura revisada por pares quanto literatura cinzenta, como artigos técnicos e postagens de *blog*, alcançando uma visão mais próxima do estado real da prática profissional.

Como principal resultado, a revisão constatou que não há ainda uma definição única e aceita de segurança em *MLOps*. Para preencher essa lacuna, o artigo propõe uma definição consolidada: a segurança em *MLOps* envolve a integração explícita de controles de segurança ao longo de todo o ciclo de vida, visando proteger dados, modelos e infraestrutura contra ameaças específicas, garantindo privacidade, resiliência e conformidade regulatória.

Em relação aos riscos identificados, o estudo mapeou 15 ameaças distintas agrupadas em três fontes principais: ataques *cibernéticos*, degradação de desempenho de sistemas e modelos e a falta de uma cultura organizacional de segurança. Os ataques *cibernéticos* são apontados

como a ameaça mais incidente, abrangendo desde exploração de falhas na autenticação e na rede até adulteração da cadeia de suprimentos de software, com ênfase para a facilidade de reuso de imagens *Docker* ou dependências vulneráveis.

Como práticas de mitigação, foram identificadas 27 recomendações agrupadas em oito temas: autenticação e autorização, segurança da implantação, monitoramento contínuo, práticas de desenvolvimento seguro, segurança da cadeia de suprimentos, diretrizes éticas e de conformidade e fortalecimento da mentalidade de segurança, como treinamento de equipes e resposta a incidentes. Um destaque é o papel central de mecanismos de monitoramento constante para detectar degradação de modelos em produção e evitar comportamentos inesperados que podem afetar sistemas de missão crítica

Por fim, o estudo conclui que, embora muitos riscos sejam variações de problemas clássicos da segurança *cibernética*, o contexto de *MLOps* adiciona camadas novas de complexidade, principalmente ligadas à gestão dinâmica de dados e modelos. Assim, os autores reforçam a necessidade de pesquisas futuras que consolidem frameworks de segurança específicos para *pipelines* de *MLOps*, que combinem automação, governança robusta e conscientização cultural.

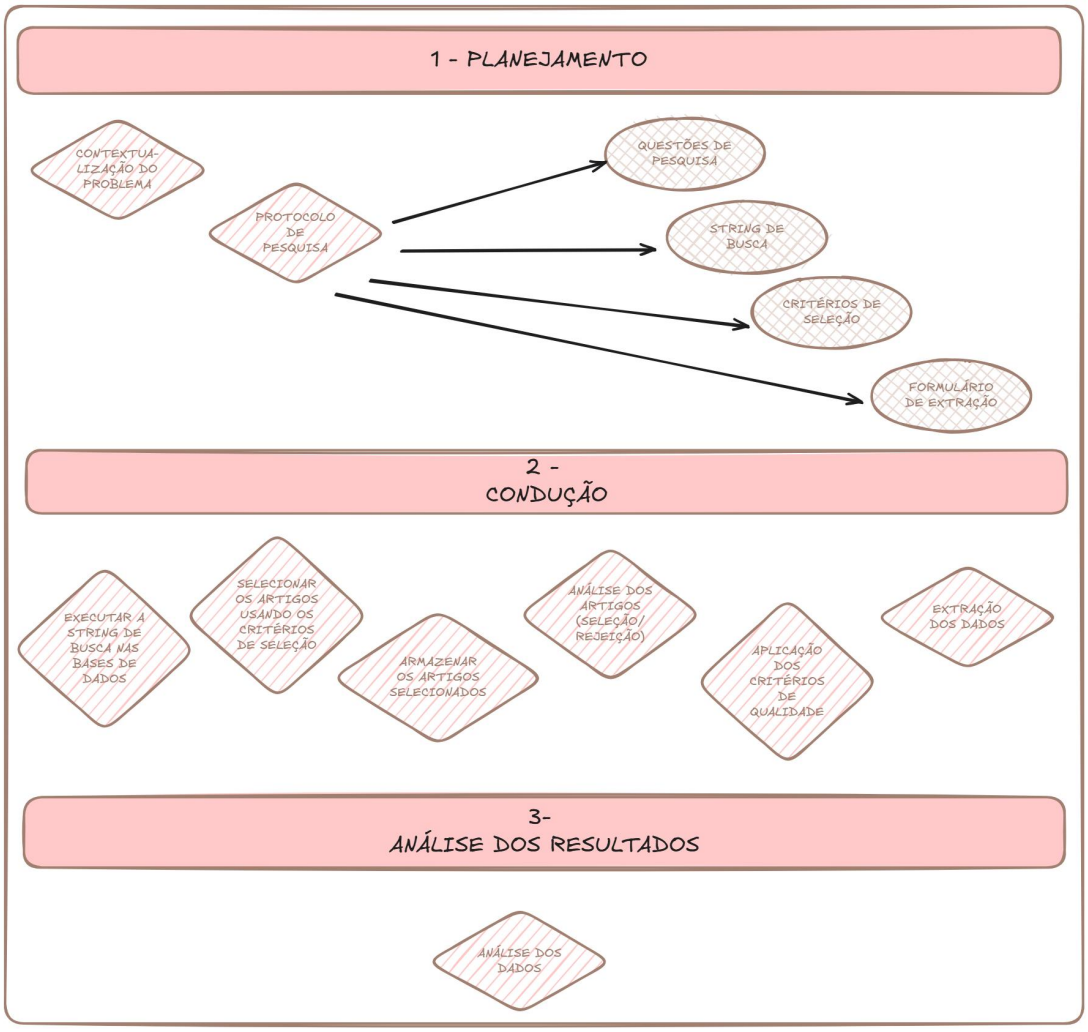
Assim, considerando os estudos revisados, o presente trabalho se posiciona como uma extensão necessária às contribuições de Lima *et al.* (2022), Najafabadi *et al.* (2024) e Calefato *et al.* (2024). Enquanto esses autores aprofundam aspectos relacionados às práticas, ferramentas, componentes arquiteturais e medidas de segurança no contexto do *MLOps*, o presente estudo avança ao direcionar o foco especificamente para a dimensão dos testes de software, etapa ainda pouco abordada na literatura consolidada sobre o tema. Por meio de uma revisão sistemática, esta pesquisa busca mapear e analisar as abordagens de testes aplicadas ao ciclo de vida de soluções baseadas em *machine learning* integradas a práticas de *DevOps* e *CI/CD*, contribuindo para fortalecer a qualidade, a confiabilidade e a governança. Dessa forma, o trabalho complementa os estudos existentes ao preencher lacunas metodológicas e fornecer subsídios para boas práticas voltadas especificamente ao contexto de verificação e validação dentro de arquiteturas *MLOps*.

4 METODOLOGIA

Conforme indicado por Kitchenham *et al.* (2007), a metodologia de mapeamento sistemático proporciona uma visão abrangente de um campo específico, tendo como objetivo identificar evidências relacionadas à pesquisa sem a necessidade de uma análise aprofundada das questões propostas. Essa abordagem foi selecionada pela sua capacidade de organizar e classificar os estudos existentes na área de interesse, facilitando investigações futuras que possam contribuir para o desenvolvimento e aprofundamento do tema.

A Figura 5 ilustra o processo de mapeamento sistemático conforme descrito por Kitchenham *et al.* (2007), dividindo-se em três etapas principais: planejamento, condução e análise dos resultados. Na fase de planejamento, são definidas as diretrizes do estudo, incluindo a formulação da pergunta de pesquisa, a estratégia de busca e os critérios de inclusão e exclusão dos estudos. Em seguida, na etapa de condução, a pesquisa é realizada de acordo com o protocolo estabelecido, abrangendo a busca nas bases de dados, a seleção dos estudos relevantes e a extração das informações. Por fim, na fase de análise dos resultados, os dados coletados são sintetizados e analisados, permitindo a identificação de padrões, lacunas e oportunidades para pesquisas futuras.

Figura 5 – Etapas do processo de mapeamento sistemático



Fonte: Elaborado pelo autor, 2025.

Para o auxílio da pesquisa, foi selecionado os elementos do protocolo *PICO* (População, Intervenção, Comparação e Resultado) para definir a *string* de busca do mapeamento sistemático. A justificativa pela escolha é sua capacidade de estruturar de forma clara e objetiva a formulação das questões de pesquisa. Esse modelo permite identificar os principais conceitos envolvidos no estudo, garantindo que a estratégia de busca seja abrangente e relevante. Ao definir os termos de busca com base nesses elementos, aumenta-se a precisão na recuperação de estudos pertinentes, reduzindo o risco de incluir artigos irrelevantes ou deixar de fora pesquisas essenciais para a análise. Dessa forma, o uso do *PICO* melhora a eficácia da busca e contribui para a obtenção de resultados mais consistentes e alinhados ao objetivo do estudo.

Para facilitar o entendimento sobre os elementos *PICO* dentro do contexto do tema abordado, será apresentado a seguir a Tabela 1. Esta tabela foi estruturada de maneira clara e objetiva, com o intuito de destacar os componentes essenciais do *PICO* de forma direta e

acessível. Ao consultar a tabela, será possível compreender como cada um dos elementos se relaciona entre si e como eles se aplicam ao desenvolvimento de uma questão de pesquisa.

Tabela 1 – Descrição dos elementos *PICO*

Elemento *PICO*	**Descrição**
Problema	Estudos que abordem práticas de teste em sistemas baseados em *MLOps*
Intervenção	Investigação dos tipos de testes aplicados no contexto do *MLOps*
Outcomes (resultados)	Compreender o funcionamento dos testes de software no contexto do *MLOps*

Fonte: Elaborado pelo autor, 2024.

4.1 Protocolo de pesquisa

O protocolo consiste em questão de pesquisa, string de busca, fontes de busca, critérios de seleção e formulário de extração. Utilizando o protocolo para buscas nas fontes específicas e aplicando os critérios de seleção, artigos relevantes sobre o presente tema serão encontrados.

O protocolo descrito é fundamental para garantir a organização e a consistência na busca e seleção dos artigos relevantes para a revisão sistemática. Ele compreende a definição da questão de pesquisa, a elaboração da *string* de busca, a escolha das fontes específicas e o estabelecimento dos critérios de seleção. Ao aplicar esse protocolo rigorosamente nas bases de dados indicadas, é possível filtrar e identificar artigos pertinentes ao tema.

4.1.1 Questão de pesquisa

A metodologia deste trabalho busca identificar e analisar publicações relevantes sobre testes de software no contexto do *MLOps*, considerando a dificuldade em encontrar estudos específicos nessa área. Diante dessa lacuna na literatura, foi definida uma questão de pesquisa essencial que orienta esta revisão sistemática. Essa questão foi elaborada para compreender **como as práticas de teste de software estão sendo aplicadas e adaptadas dentro do *MLOps***, além de permitir uma comparação com abordagens tradicionais. Assim, pretende-se responder **quais tipos de testes são mais utilizados e de que forma contribuem para garantir a eficiência e a**

qualidade do desempenho dos modelos de *Machine Learning*.

A questão de pesquisa é:

- **Q1**: Quais são os tipos de testes de software aplicados no contexto de *MLOps* e como eles são realizados para garantir a qualidade e confiabilidade dos modelos de *Machine Learning*?

Dessa forma, espera-se obter uma visão mais clara das práticas atuais, identificar lacunas e oportunidades de melhoria, além de fornecer uma base para futuras pesquisas sobre estratégias de teste em ambientes que combinam desenvolvimento de software e operações de *Machine Learning*.

4.1.2 String de busca

Nesta etapa, destaca-se a importância do uso de operadores lógicos do tipo *Booleano*, como *AND*, *OR* e *NOT*, para estruturar a pesquisa de forma mais eficaz. Esses operadores permitem combinar termos de busca para refinar ou ampliar os resultados.

O operador ***AND*** cria uma relação de interseção, retornando apenas documentos que atendam simultaneamente a todas as condições especificadas. Já o operador ***OR*** expande a busca, permitindo que qualquer um dos termos indicados seja suficiente para a recuperação do resultado. Por sua vez, o operador ***NOT*** atua como restrição, excluindo resultados que contenham um termo indesejado, mantendo o foco apenas no conteúdo relevante.

Para ilustrar o funcionamento desses conectivos, a seguir são apresentados exemplos de *strings* de busca criadas apenas para fins explicativos, demonstrando como os operadores podem ser combinados na prática:

Exemplos com o operador *NOT*:

- *Software testing in MLOps **NOT** testing in traditional operations*
- *Software testing in MLOps **NOT** conventional software development*
- *Software testing in MLOps **NOT** system analysis without machine learning*
- *Software testing in MLOps **NOT** software validation without ML integration*

Nesses exemplos, o objetivo é ilustrar como o *NOT* pode excluir resultados relacionados a práticas tradicionais ou contextos que não envolvam *MLOps* ou *ML.*

Exemplos com o operador *AND*:

- *Software testing in MLOps **AND** evaluation of machine learning models*
- *Software testing in MLOps **AND** validation of ML algorithms*

- *Software testing in MLOps **AND** quality assurance in machine learning operations*
- *Software testing in MLOps **AND** verification of ML model integrity*

Aqui, o *AND* demonstra como restringir a busca a documentos que abordem mais de um tema em conjunto.

Exemplos com o operador *OR*:

- *Test of software in MLOps **OR** testing in machine learning operations*
- *Software testing in MLOps **OR** software validation in ML operations*
- *MLOps software testing **OR** ML operations testing*
- *Testing software in MLOps **OR** machine learning operations testing*

Nesses casos, o *OR* mostra como incluir resultados que contenham pelo menos um dos termos listados, ampliando o alcance da pesquisa.

Por fim, é importante lembrar que os operadores lógicos seguem uma ordem de execução: *NOT*, depois *AND* e, por último, *OR*. Assim, o uso de parênteses é essencial para garantir que a lógica definida pelo pesquisador seja interpretada corretamente pelos motores de busca.

Exemplo de combinação:

– *Software testing in MLOps **NOT** traditional operations testing **AND** software validation in ML operations*

– Tradução: Teste de software em *MLOps **NOT*** teste de operações tradicionais ***AND*** validação de software em operações de *ML.*

Este exemplo mostra como é possível combinar mais de um operador para criar buscas mais específicas e alinhadas aos objetivos de investigação.

Com base na análise dos principais descritores, sinônimos e operadores lógicos exemplificados anteriormente, definiu-se a *string* de busca que foi utilizada para prosseguir com a pesquisa em bases de dados acadêmicas. A *string* final foi elaborada de forma a garantir a recuperação de estudos que abordem testes de software especificamente no contexto de *MLOps* ou de operações de aprendizado de máquina, contemplando também discussões sobre os tipos de testes empregados e suas adaptações. Além disso, inclui-se a possibilidade de recuperar publicações que relacionem testes de software a operações tradicionais, possibilitando comparações relevantes entre abordagens convencionais e práticas voltadas para ambientes que integram *Machine Learning* e operações. Essa estrutura visa assegurar uma busca abrangente,

porém direcionada, alinhada aos objetivos da revisão sistemática conduzida neste trabalho. A *string* base de busca definida se encontra a seguir:

((("SOFTWARE TESTING"OR "TESTING") AND ("MLOps"OR "MACHINE LEARNING OPERATIONS")) AND (("TYPES OF SOFTWARE TESTING") OR ("ADAPTATION OF SOFTWARE TESTING"))) OR ((("SOFTWARE TESTING"OR "TESTING") AND ("TRADITIONAL OPERATIONS")))

4.1.3 Fontes de busca

As fontes de pesquisa utilizadas para a elaboração deste trabalho incluem bases de dados reconhecidas e de alta qualidade no campo da ciência da computação e engenharia de software. Abaixo estão listadas as principais fontes utilizadas, juntamente com seus links oficiais:

- **IEEE Xplore**:
  https://ieeexplore.ieee.org
- **ScienceDirect**:
  https://www.sciencedirect.com
- **ACM Digital Library**:
  https://dl.acm.org
- **Scopus**:
  https://www.scopus.com/home.uri

Estas bases de dados foram escolhidas devido à sua ampla aceitação e credibilidade na comunidade acadêmica. Cada uma delas oferece um vasto acervo de artigos, conferências e outros recursos acadêmicos que são essenciais para a realização de pesquisas de alta qualidade.

Para atender às especificidades de cada base de dados, a *string* de busca precisou passar por pequenas adaptações, preservando, no entanto, sua estrutura conceitual principal. A *string* base definida foi:

```
((("SOFTWARE TESTING"OR "TESTING") AND ("MLOps"OR "MACHINE LEARNING
OPERATIONS")) AND (("TYPES OF SOFTWARE TESTING") OR ("ADAPTATION OF
SOFTWARE TESTING"))) OR (("SOFTWARE TESTING"OR "TESTING") AND ("TRADITIONAL
OPERATIONS"))
```

Esta *string* foi aplicada integralmente nas bases *IEEE Xplore* e *ScienceDirect*. Para a

ACM Digital Library, a *string* foi ajustada para:

```
((SOFTWARE TESTING OR TESTING) AND ("MLOps"OR "MACHINE LEARNING OPERATIONS")
AND ("TYPES OF SOFTWARE TESTING"OR ADAPTATION OF SOFTWARE TESTING))
```

Por fim, para a base *Scopus*, optou-se por uma versão simplificada, visando ampliar a abrangência dos resultados:

```
(("SOFTWARE TESTING"OR TESTING) AND ("MLOps"OR "MACHINE LEARNING OPERATIONS"))
```

Essas adaptações foram necessárias para contornar restrições de operadores e limitações específicas de cada plataforma de busca.

4.1.4 Critérios de seleção

Para garantir a relevância, consistência e qualidade dos estudos incluídos nesta revisão sistemática, foram definidos critérios claros de inclusão, exclusão e avaliação de qualidade, alinhados aos elementos *PICO* descritos anteriormente e à string de busca empregada.

4.1.4.1 Critérios de inclusão

Os critérios de inclusão foram elaborados para assegurar que apenas estudos diretamente relacionados ao objetivo desta pesquisa sejam considerados. São eles:

- Publicações que abordem práticas de testes de software no contexto de *MLOps*, *Machine Learning Operations* ou operações de aprendizado de máquina;
- Estudos que descrevam tipos de testes de software aplicados especificamente ao contexto do *MLOps* ou adaptação de práticas de teste para cenários que integrem desenvolvimento de software e aprendizado de máquina;
- Estudos publicados em inglês ou português;
- Publicações com acesso ao texto completo.

A justificativa para esses critérios está diretamente relacionada à definição do **Problema** e **Intervenção** da pesquisa: identificar de forma específica como os testes de software são realizados em sistemas baseados em *MLOps* e quais tipos ou adaptações são empregados.

4.1.4.2 Critérios de exclusão

Para evitar a inclusão de materiais irrelevantes ou fora do escopo, definiram-se os seguintes critérios de exclusão:

- Estudos que tratem exclusivamente de operações tradicionais de desenvolvimento de software, sem considerar contextos envolvendo *MLOps* ou aprendizado de máquina;
- Publicações que mencionem testes de software apenas de forma superficial, sem detalhar práticas, tipos, métodos ou adaptações;
- Materiais duplicados, versões preliminares, resumos sem o texto completo ou capítulos de livros;
- Não disponível na web;
- Estudos publicados em idiomas diferentes do inglês ou português;
- Estudos que não mencionem "*Machine Learning*", "*ML*"ou "*MLOps*"no título.

Esses critérios se justificam por manter o foco exclusivamente em evidências sólidas, práticas ou teóricas, que possibilitem entender de forma aprofundada o funcionamento dos testes de software no contexto do *MLOps* — evitando diluir a análise com abordagens genéricas ou não aplicáveis.

No entanto, para garantir que a análise contemplasse exclusivamente pesquisas diretamente pertinentes ao tema deste estudo, adotou-se um critério de exclusão rigoroso. Nesse sentido, foram desconsiderados todos os trabalhos que não apresentassem, em seus títulos, as palavras-chave "*Machine Learning*", "*ML*"ou "*MLOps*". Essa abordagem assegurou a seleção de literatura altamente relevante, promovendo a precisão e a consistência dos resultados obtidos.

4.1.4.3 Critérios de qualidade

Além dos critérios de inclusão e exclusão, foram estabelecidos critérios de qualidade para assegurar a confiabilidade dos estudos selecionados. Serão priorizados estudos que atendam aos seguintes requisitos:

- Clareza metodológica: descrição explícita dos métodos de teste aplicados, contexto de aplicação e resultados obtidos;
- Detalhamento das práticas de teste de software utilizadas em ambientes de *MLOps* ou em contextos de aprendizado de máquina;
- Presença de discussões críticas sobre desafios, limitações ou necessidades de adaptação

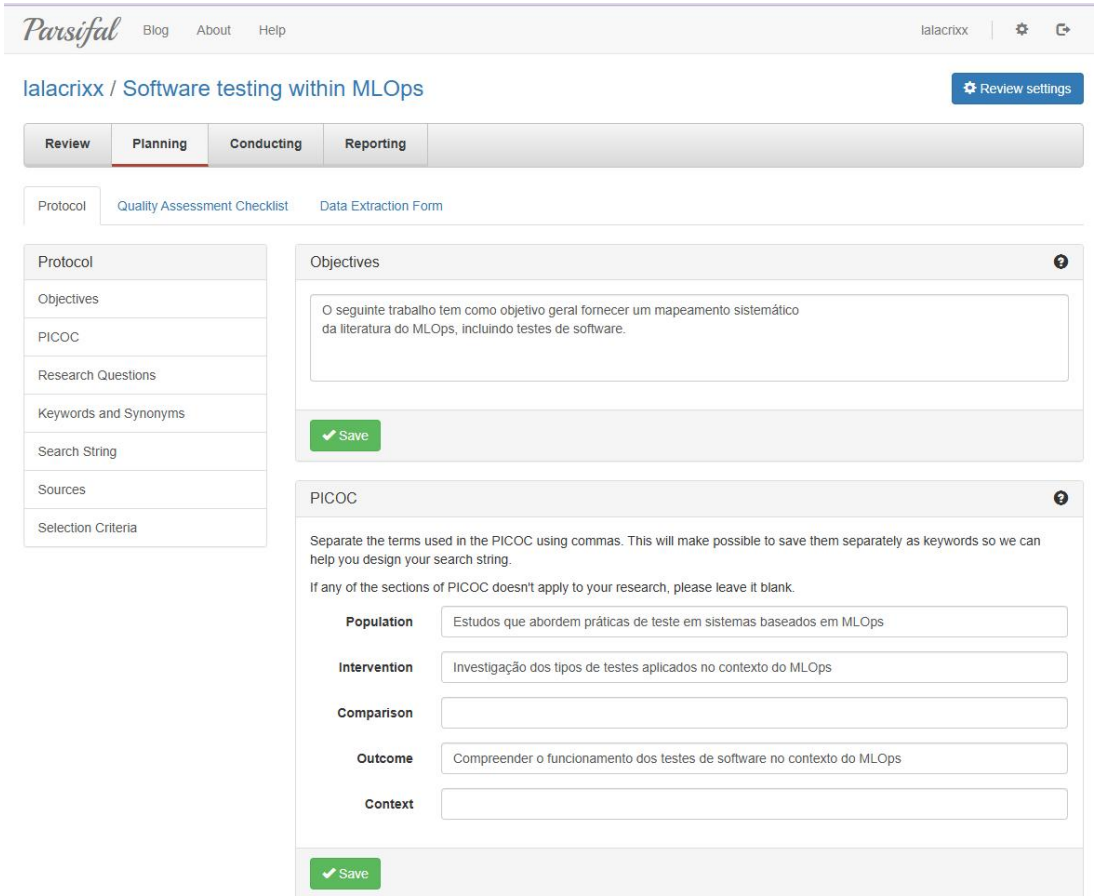
das práticas de teste em cenários de *MLOps*;

– Evidências empíricas ou estudos de caso que comprovem a aplicação prática das práticas de teste descritas.

A adoção desses critérios de qualidade visa garantir que os estudos incluídos contribuam de fato para responder à questão de pesquisa, possibilitando a compreensão detalhada dos tipos de testes de software utilizados, suas adaptações e impactos no desempenho de sistemas que integram *MLOps*.

A ferramenta *Parsifal* foi utilizada como suporte na condução da revisão sistemática. Trata-se de uma plataforma *online* projetada especificamente para auxiliar pesquisadores na execução de revisões sistemáticas, permitindo gerenciar cada etapa do processo de forma integrada e colaborativa. Por meio do *Parsifal*, foi possível estruturar o protocolo da revisão, incluindo a definição das perguntas de pesquisa, critérios de inclusão e exclusão, estratégias de busca, e métodos de extração e análise dos dados. Consulte a Figura 6, que apresenta o *layout* da ferramenta. [1]

Figura 6 – Layout do Parsifal

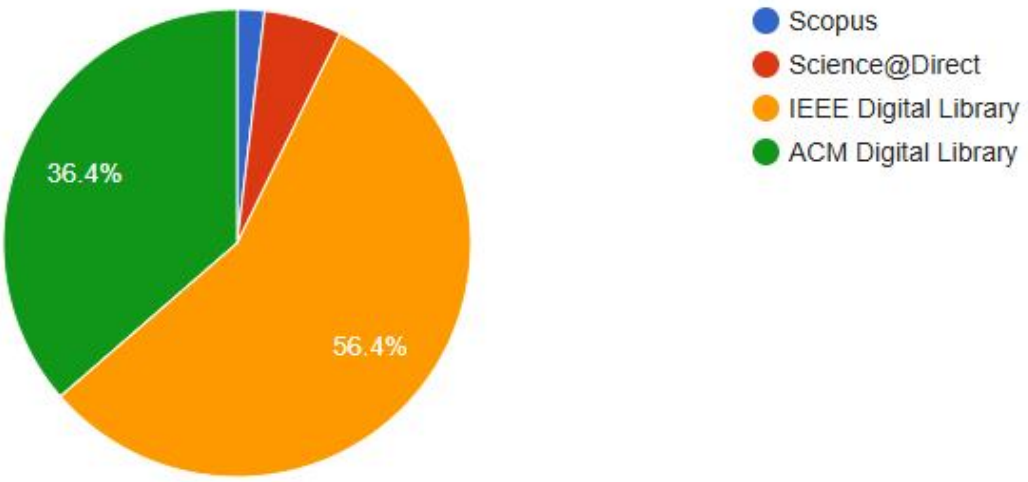


Fonte: Elaborado pelo autor, 2025.

[1] https://parsif.al/lalacrixx/software-testing-within-mlops/

A condução da pesquisa foi realizada por meio da execução da *string* de busca nas fontes de pesquisa previamente definidas, resultando em um total de 690 trabalhos identificados. Abaixo, na Figura 7, será apresentada a quantidade de trabalhos encontrados em cada fonte de pesquisa.

Figura 7 – Artigos por fonte de pesquisa



Fonte: Elaborado pelo autor, 2025.

Utilizando a *string* de busca definida na Subseção 4.1.2 e aplicada aos motores de busca, descritos na Subseção 4.1.3, foi obtido um total de 690 artigos a serem analisados, sendo 251 da *ACM Digital Library*, 389 da *IEEE Digital Library*, 37 do *ScienceDirect* e 13 do *Scopus*. Esses artigos foram devidamente armazenados na plataforma *Parsifal*, com o objetivo de realizar a filtragem necessária e, posteriormente, apresentar os resultados obtidos de forma organizada e sistemática.

Por fim, o *Parsifal* facilitará a síntese dos resultados ao fornecer um ambiente onde os dados extraídos possam ser revisados e validados de maneira eficiente, assegurando a transparência da revisão sistemática.

4.1.5 Formulário de extração

Para garantir a padronização e consistência na coleta de informações relevantes, foi elaborado um formulário de extração de dados, apresentado na Tabela 2. Este formulário orientou a coleta de informações essenciais de cada estudo incluído na revisão, assegurando

que os elementos PICO fossem contemplados e que os resultados pudessem ser comparados de forma sistemática.

Tabela 2 – Formulário de extração de dados

Identificação do Estudo	
Título do estudo	Nome do artigo ou publicação selecionada.
Autores	Nome(s) do(s) autor(es).
Ano de publicação	Ano em que o estudo foi publicado.
Detalhamento do Conteúdo	
Comparação com abordagens tradicionais	Informações sobre comparação com práticas de teste em operações tradicionais, se houver.
Métodos e abordagem	Descrição da metodologia aplicada no estudo.
Resultados principais	Principais achados e contribuições do estudo.
Desafios e limitações	Principais dificuldades ou limitações reportadas.
Evidências empíricas	Indicação se o estudo apresenta estudos de caso, experimentos ou dados práticos.

Fonte: Elaborado pelo autor, 2025.

O detalhamento do conteúdo incluído no formulário de extração de dados justifica-se pela necessidade de responder de forma precisa à questão de pesquisa proposta, que busca compreender como os tipos de testes de software aplicados em ambientes de *MLOps* se comparam às abordagens tradicionais, considerando sua eficiência e o impacto na qualidade do desempenho dos modelos de *Machine Learning*. Assim, categorias como comparação com abordagens tradicionais, métodos e abordagem, resultados principais, desafios e limitações, e evidências empíricas foram definidas para possibilitar a coleta de informações relevantes e organizadas. Essa estruturação garante que os dados extraídos permitam uma análise crítica, destacando similaridades, diferenças, benefícios e lacunas entre as práticas de teste convencionais e aquelas voltadas para o contexto de operações de aprendizado de máquina.

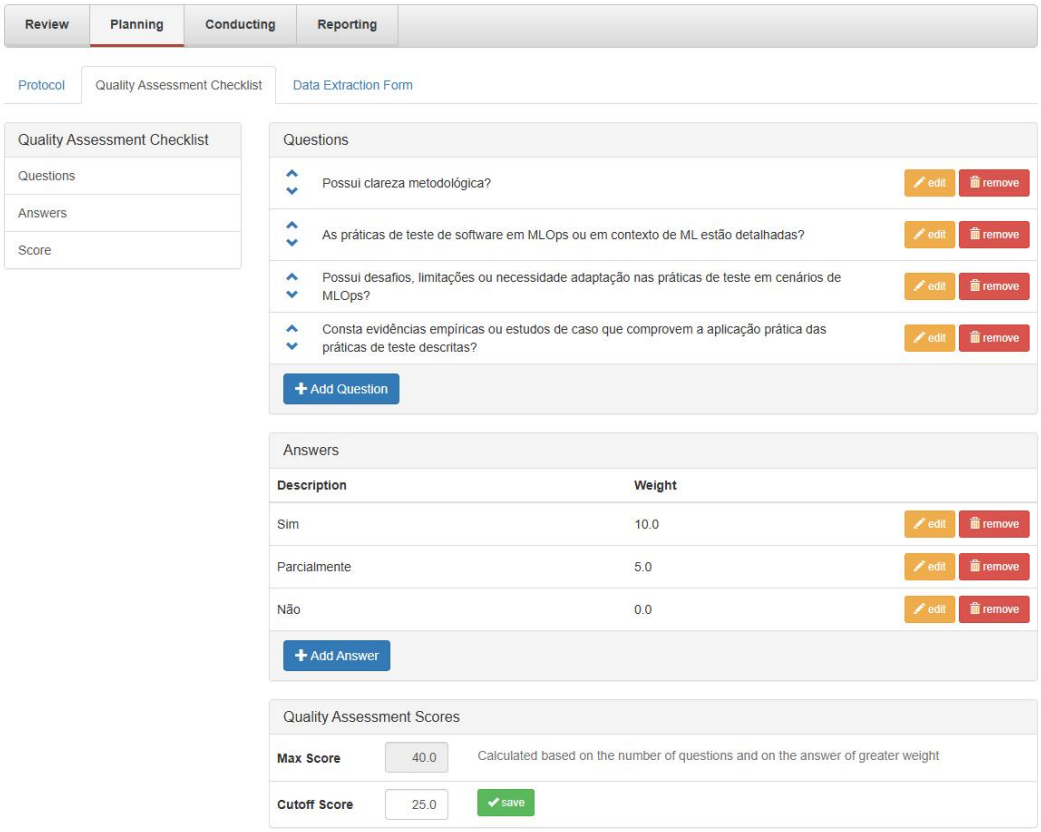
5 SELEÇÃO E RESULTADOS

Neste capítulo, apresenta-se de forma detalhada o processo de seleção realizado, desde a identificação inicial dos estudos até a definição do conjunto final de trabalhos qualificados. Além disso, serão expostos os principais resultados obtidos a partir da aplicação dos critérios de inclusão, exclusão e qualidade, evidenciando as etapas que fundamentaram a composição da amostra final desta pesquisa.

5.1 Seleção dos estudos

Na condução deste trabalho, todos os estudos identificados foram importados na ferramenta *Parsifal*, que foi utilizada para gerenciar as etapas de planejamento e execução da revisão sistemática. No *Parsifal*, a etapa de planejamento disponibiliza uma guia específica para o *checklist* dos critérios de qualidade, como visto na Figura 8. Cada critério de qualidade foi ponderado, por padrão, em 10 pontos, totalizando 40 pontos possíveis, uma prática alinhada com as diretrizes propostas por Kitchenham *et al.* (2007). Para garantir a qualidade metodológica dos estudos incluídos, estabeleceu-se uma nota de corte de 25 pontos. Assim, artigos que obtiveram nota igual ou inferior a 25 foram excluídos da análise final, enquanto aqueles com pontuação superior a este limite foram aprovados para compor a amostra definitiva.

Figura 8 – *Checklist* dos critérios de qualidade



Fonte: Elaborado pelo autor, 2025.

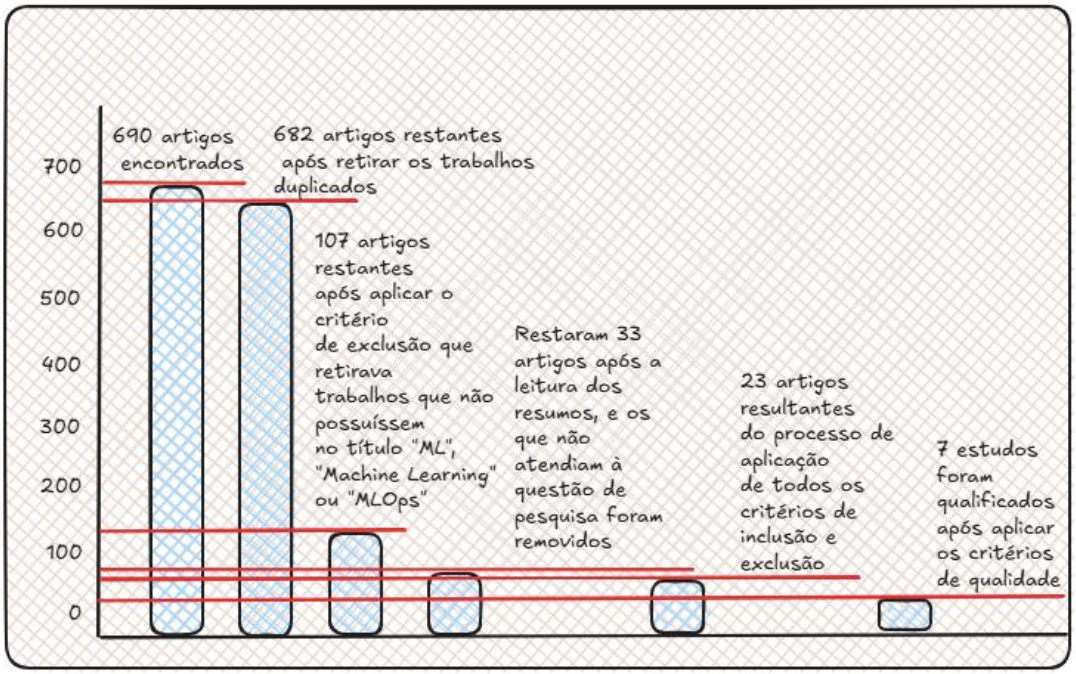
O processo de seleção inicial resultou em um total de 690 trabalhos identificados. Primeiramente, foram removidos 8 trabalhos duplicados, restando, assim, 682 registros. Em seguida, aplicou-se o primeiro critério de exclusão, que consistiu em rejeitar artigos cujo título não continha os termos *MLOps*, *Machine Learning* ou *ML*. Após esta etapa, permaneceram 107 trabalhos.

Posteriormente, procedeu-se à leitura dos resumos, buscando identificar trabalhos que atendiam à questão de pesquisa, o que resultou na seleção de 33 artigos. Sobre este conjunto, realizou-se uma filtragem mais criteriosa, aplicando-se detalhadamente todos os critérios de inclusão e exclusão definidos na Subseção 4.1.4, de forma que restaram 23 artigos, podendo ser observado nas Figuras 10, 11 e 12.

Na etapa final, referente aos critérios de qualidade, apenas 7 artigos atingiram pontuação superior a 25, valor estipulado como nota de corte para inclusão na análise. Os estudos qualificados, podem ser visualizados na Tabela 3.

Para simplificar como foi realizado o processo de extração dos estudos, foi adicionado a Figura 9 para uma visualização mais clara e direta.

Figura 9 – Explicação simplificada do processo de extração dos estudos



Fonte: Elaborado pelo autor, 2025.

Figura 10 – *T*rabalhos selecionados

A Drift Handling Approach for Self-Adaptive ML Software in Scalable Industrial Processes	Bayram, Firas and Ahmed, Bestoun S. and Hallin, Erik and Engman, Anton		2023	lalacrixx	13 Jul 2025 01:31:43	Accepted
An Exploratory Study of V-Model in Building ML-Enabled Software: A Systems Engineering Perspective	Wu, Jie JW		2024	lalacrixx	13 Jul 2025 01:31:43	Accepted
"Project smells": experiences in analysing the software quality of ML projects with mllint	van Oort, Bart and Cruz, Lu\'{\i}s and Loni, Babak and van Deursen, Arie		2022	lalacrixx	13 Jul 2025 01:31:43	Accepted
Collaboration challenges in building ML-enabled systems: communication, documentation, engineering, and process	Nahar, Nadia and Zhou, Shurui and Lewis, Grace and K\"{a}stner, Christian		2022	lalacrixx	13 Jul 2025 01:31:43	Accepted
Quality trade-offs in ML-enabled systems: a multiple-case study	Indykov, Vladislav and Wohlrab, Rebekka and Str\"{u}ber, Daniel		2025	lalacrixx	13 Jul 2025 01:31:43	Accepted
Exploring ML testing in practice: lessons learned from an interactive rapid review with axis communications	Song, Qunying and Borg, Markus and Engstr\"{o}m, Emelie and Ard\"{o}, H\r{a}kan and Rico, Sergio		2022	lalacrixx	13 Jul 2025 01:31:43	Accepted
Automated Boundary Identification for Machine Learning Classifiers	Dobslaw, Felix and Feldt, Robert		2024	lalacrixx	13 Jul 2025 00:02:23	Accepted
Formal Verification of a Hybrid Machine Learning-Based Fault Prediction Model in Internet of Things Applications	Souri, Alireza and Mohammed, Amin Salih and Yousif Potrus, Moayad and Malik, Mazhar Hussain and Safara, Fatemeh and Hosseinzadeh, Mehdi	IEEE Access	2020	lalacrixx	13 Jul 2025 00:02:23	Accepted
Testing of machine learning models with limited samples: an industrial vacuum pumping application	Chatterjee, Ayan and Ahmed, Bestoun S. and Hallin, Erik and Engman, Anton		2022	lalacrixx	13 Jul 2025 01:31:43	Accepted
Deepchecks: a library for testing and validating machine learning models and data	Chorev, Shir and Tannor, Philip and Israel, Dan Ben and Bressler, Noam and Gabbay, Itay and Hutnik, Nir and Liberman, Jonatan and Perlmutter, Matan and Romanyshyn, Yurii and Rokach, Lior	J. Mach. Learn. Res.	2022	lalacrixx	13 Jul 2025 01:31:43	Accepted
Adoption and Effects of Software Engineering Best Practices in Machine Learning	Serban, Alex and van der Blom, Koen and Hoos, Holger and Visser, Joost		2020	lalacrixx	13 Jul 2025 01:31:43	Accepted

Fonte: Elaborado pelo autor, 2025.

Figura 11 – *T*rabalhos selecionados

Self-adapting Machine Learning-based Systems via a Probabilistic Model Checking Framework	Casimiro, Maria and Soares, Diogo and Garlan, David and Rodrigues, Lu\'{\i}s and Romano, Paolo	ACM Trans. Auton. Adapt. Syst.	2024	lalacrixx	13 Jul 2025 01:31:43	Accepted
"We Have No Idea How Models will Behave in Production until Production": How Engineers Operationalize Machine Learning	Shankar, Shreya and Garcia, Rolando and Hellerstein, Joseph M. and Parameswaran, Aditya G.	Proc. ACM Hum.-Comput. Interact.	2024	lalacrixx	13 Jul 2025 01:31:43	Accepted
Mithridates: Auditing and Boosting Backdoor Resistance of Machine Learning Pipelines	Bagdasarian, Eugene and Shmatikov, Vitaly		2024	lalacrixx	13 Jul 2025 01:31:43	Accepted
MLOps: A Taxonomy and a Methodology	Testi, Matteo and Ballabio, Matteo and Frontoni, Emanuele and Iannello, Giulio and Moccia, Sara and Soda, Paolo and Vessio, Gennaro	IEEE Access	2022	lalacrixx	13 Jul 2025 00:02:10	Accepted
Amazon SageMaker Model Monitor: A System for Real-Time Insights into Deployed Machine Learning Models	Nigenda, David and Karnin, Zohar and Zafar, Muhammad Bilal and Ramesha, Raghu and Tan, Alan and Donini, Michele and Kenthapadi, Krishnaram		2022	lalacrixx	13 Jul 2025 01:31:43	Accepted
Operationalizing machine learning models: a systematic literature review	Kolltveit, Ask Berstad and Li, Jingyue		2023	lalacrixx	13 Jul 2025 01:31:43	Accepted
MLOps Pipeline Development: The OSSARA Use Case	Moreschini, Sergio and H\"{a}stbacka, David and Taibi, Davide		2023	lalacrixx	13 Jul 2025 01:31:43	Accepted
State-of-the-Art and Challenges of Engineering ML- Enabled Software Systems in the Deep Learning Era	Assres, Gebremariam and Bhandari, Guru and Shalaginov, Andrii and Gronli, Tor-Morten and Ghinea, Gheorghita	ACM Comput. Surv.	2025	lalacrixx	13 Jul 2025 01:31:43	Accepted
Combining Metamorphic Testing and Machine Learning to Enhance OpenStreetMap	Méndez, Manuel and Becerra-Terón, Antonio and Almendros-Jiménez, Jesús M. and Merayo, Mercedes G. and Núñez, Manuel	IEEE Transactions on Reliability	2024	lalacrixx	13 Jul 2025 00:02:29	Accepted
A monitoring framework for deployed machine learning models with supply chain examples	Eck, Bradley and Kabakci-Zorlu, Duygu and Chen, Yan and Savard, France and Bao, Xiaowei	Proceedings - 2022 IEEE International Conference on Big Data, Big Data 2022	2022	lalacrixx	13 Jul 2025 02:12:19	Accepted

Fonte: Elaborado pelo autor, 2025.

Figura 12 – *T*rabalhos selecionados

Test automation with grad-CAM Heatmaps - A future pipe segment in MLOps for Vision AI?	Borg, Markus and Jabangwe, Ronald and Aberg, Simon and Ekblom, Arvid and Hedlund, Ludwig and Lidfeldt, August	Proceedings - 2021 IEEE 14th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2021	2021	lalacrixx	13 Jul 2025 02:12:19	Accepted
Professional Insights into Benefits and Limitations of Implementing MLOps Principles	Araujo, Gabriel and Kalinowski, Marcos and Endler, Markus and Calefato, Fabio	International Conference on Enterprise Information Systems, ICEIS - Proceedings	2024	lalacrixx	13 Jul 2025 02:12:19	Accepted

Fonte: Elaborado pelo autor, 2025.

Tabela 3 – Estudos qualificados

Título	**Autor(es)**	**Ano**
Automated Boundary Identification for Machine Learning Classifiers	Dobslaw, Felix; Feldt, Robert	2024
Combining Metamorphic Testing and Machine Learning to Enhance OpenStreetMap	Méndez, Manuel; Becerra-Terón, Antonio; Almendros-Jiménez, Jesús M.; Merayo, Mercedes G.; Núñez, Manuel	2024
Deepchecks: a library for testing and validating machine learning models and data	Chorev, Shir; Tannor, Philip; Israel, Dan Ben; Bressler, Noam; Gabbay, Itay; Hutnik, Nir; Liberman, Jonatan; Perlmutter, Matan; Romanyshyn, Yurii; Rokach, Lior	2022
Exploring ML testing in practice: lessons learned from an interactive rapid review with Axis Communications	Song, Qunying; Borg, Markus; Engström, Emelie; Ardö, Håkan; Rico, Sergio	2022
Mithridates: Auditing and Boosting Backdoor Resistance of Machine Learning Pipelines	Bagdasarian, Eugene; Shmatikov, Vitaly	2024
Self-adapting Machine Learning-based Systems via a Probabilistic Model Checking Framework	Casimiro, Maria; Soares, Diogo; Garlan, David; Rodrigues, Luís; Romano, Paolo	2024
Test automation with Grad-CAM Heatmaps - A future pipe segment in MLOps for Vision AI?	Borg, Markus; Jabangwe, Ronald; Åberg, Simon; Ekblom, Arvid; Hedlund, Ludwig; Lidfeldt, August	2021

Fonte: Elaborado pelo autor, 2025.

Dessa forma, conclui-se que o cuidado aplicado em cada etapa da seleção garantiu uma amostra consistente e alinhada aos objetivos da pesquisa. O conjunto final de estudos qualificados representa uma base sólida para responder à questão de pesquisa proposta, oferecendo evidências relevantes que serão analisadas na próxima seção.

5.2 Síntese dos estudos qualificados

Com o crescimento exponencial do uso de *ML* em domínios críticos, os processos de teste, validação e auditoria tornaram-se componentes essenciais para garantir que os modelos sejam confiáveis, robustos e transparentes. Diversos pesquisadores têm investigado soluções inovadoras para problemas práticos, como a identificação de limites de decisão, a redução de

falsos positivos, a detecção de *bias*, a resistência a ataques de *backdoor*, além da validação contínua de dados e modelos em ambientes reais. Este texto apresenta uma síntese de diferentes propostas acadêmicas que exemplificam a diversidade de abordagens voltadas a assegurar a qualidade de sistemas de *ML* em produção.

- Automated Boundary Identification for Machine Learning Classifiers

No trabalho de Dobslaw e Feldt (2024), é apresentado o Algoritmo de Extensão de Limites de *ML* (*Machine Learning Boundary Spanning Algorithm (ML-BSA)*), que se destaca por propor uma forma automatizada de identificar limites de decisão em classificadores de *ML*. O diferencial dessa abordagem é um objetivo de diversidade, que espalha de forma mais ampla e uniforme os pares candidatos a limites, possibilitando que *testers* e desenvolvedores tenham uma compreensão mais clara de onde o limite de classificação realmente se encontra. Assim, é possível compará-lo com as expectativas, além de concentrar os testes e os refinamentos nas regiões mais críticas. O método é agnóstico ao modelo e funciona como uma abordagem de caixa preta, o que o torna versátil para diferentes tipos de algoritmos. Diferentemente de técnicas como explicações contrafactuais ou geração de exemplos adversariais, que se limitam a casos específicos próximos dos dados de treino, o *ML-BSA* visa mapear toda a fronteira de decisão.

- Combining Metamorphic Testing and Machine Learning to Enhance OpenStreetMap

No trabalho de Méndez *et al.* (2024), a preocupação recai sobre sistemas sem oráculo, em que os *Metamorphic Testing (MT)* são uma solução valiosa, mas que, na prática, costumam gerar muitos falsos positivos. Para lidar com esse desafio, os autores propõem um novo *framework* que combina *MT* com *ML*, permitindo descartar falsos positivos de forma automática e precisa. O *framework* utiliza modelos *Random Forest (RF)* para classificar as inconsistências. Os resultados foram expressivos: a precisão geral ficou acima de 0,90, frequentemente atingindo 1,0, superando largamente a eficácia de usar *MT* ou *ML* de forma isolada. Outro ponto positivo é que o *MT* atua como um filtro, permitindo que o *ML* se concentre nos elementos mais problemáticos. O *framework* também mostrou boa capacidade de generalização para cidades não incluídas no treinamento, independentemente de idioma ou localização.

- Deepchecks: a library for testing and validating machine learning models and data

Chorev *et al.* (2022) apresentam o *Deepchecks*, uma biblioteca *Python* de código aberto voltada à validação abrangente de modelos e dados de *ML*. Essa biblioteca foi projetada

para lidar com problemas de integridade dos dados, desajustes de distribuição e a degradação do desempenho preditivo em cenários reais. O *Deepchecks* organiza suas funcionalidades em três níveis: *checks* (verificações específicas), *conditions* (validação de resultados com base em limites definidos) e *suítes* (coleções de *checks* adaptados a cenários comuns, como novos dados, divisão de dados e treinamento de modelos). É uma solução extensível que já suporta dados tabulares e de imagem, com planos de expansão para séries temporais e *Processamento de Linguagem Natural (NLP)*).

- Exploring ML testing in practice: lessons learned from an interactive rapid review with Axis Communications

No estudo conduzido por AN (2023), uma *Revisão Rápida Interativa (IRR)* com a *Axis Communications* explorou desafios práticos enfrentados em projetos de visão computacional. Um dos principais resultados foi o desenvolvimento da *SERP-taxonomy*, uma taxonomia de testes de *ML* para facilitar o diálogo entre pesquisadores e profissionais da indústria. O trabalho identificou doze desafios prioritários para a empresa, sendo o principal: "Como testar o conjunto de dados?". Essa questão inclui identificar dados mal rotulados, verificar a cobertura e diversidade do conjunto e detectar *bias*. Embora não tenham encontrado soluções prontas que atendessem integralmente às necessidades, os autores extraíram diretrizes tecnológicas de estudos existentes, como o uso de testes de mutação para avaliar a qualidade dos dados de teste e a adequação da "surpresa" como critério para orientar amostragens de dados. O estudo destaca ainda a lacuna na pesquisa em técnicas para identificar rótulos errados.

- Mithridates: Auditing and Boosting Backdoor Resistance of Machine Learning pipelines

O trabalho de Bagdasarian e Shmatikov (2024) aborda uma questão crítica: a vulnerabilidade de modelos de *ML* a ataques de *backdoor*, nos quais dados de treino maliciosos ensinam comportamentos ocultos ao modelo. Para contornar a complexidade e as mudanças disruptivas exigidas pelas defesas atuais, os autores propõem o *Mithridates*, um método em múltiplas etapas que integra a resistência a *backdoors* na fase de pesquisa de configurações de treino, por meio da otimização de hiperparâmetros. Um conceito central é o ponto de resistência, que quantifica a percentagem mínima de envenenamento necessária para um *backdoor* ter sucesso. Na prática, o *Mithridates* demonstrou ser capaz de aumentar a resistência a ataques de *backdoor* em 3 a 5 vezes, com impacto mínimo na precisão. Além disso, os resultados indicam que configurações resistentes a tarefas primitivas de teste também se estendem a cenários de ataques

mais realistas, sendo uma abordagem promissora para ambientes *AutoML* e *federated learning*.

- Test automation with Grad-CAM Heatmaps - A future pipe segment in MLOps for Vision AI?

No trabalho de Borg *et al.* (2021), a proposta é explorar mapas de calor *Grad-CAM* para aumentar a explicabilidade de modelos de reconhecimento de imagem baseados em *Rede Neural Convolucional (CNN)*. Esses mapas ajudam a visualizar as regiões de uma imagem que mais ativam os neurônios responsáveis por uma classificação específica, possibilitando identificar *bias* em modelos — por exemplo, quando *pixels* irrelevantes são ativados em imagens de teste. O artigo defende que a análise automatizada de mapas de calor *Grad-CAM* tem potencial para se tornar um segmento futuro em *pipelines* de *MLOps*, contribuindo para detecção automática de viés e cumprimento de requisitos regulatórios de *IA* Confiável na União Europeia, incluindo robustez técnica, agência humana, transparência, diversidade e prestação de contas.

- Self-adapting Machine Learning-based Systems via a Probabilistic Model Checking Framework

Por fim, o estudo de Casimiro *et al.* (2024) propõe um *framework* que utiliza verificação de modelos probabilísticos para raciocinar sobre custos e benefícios da adaptação em sistemas baseados em *ML*, especialmente em cenários onde há erros de previsão e atrasos na obtenção dos rótulos verdadeiros. O *framework* permite a síntese de estratégias de adaptação ótimas, como decidir quando realizar o retreino de modelos. Além disso, apresenta o *Class-Based ATC (CB-ATC)*, uma nova técnica de estimativa da qualidade preditiva, que se mostrou mais precisa que abordagens de ponta, mesmo quando há atrasos significativos na disponibilidade dos rótulos.

Em conjunto, estes trabalhos ilustram o dinamismo e a complexidade que envolvem o teste e a validação de sistemas de *machine learning*. A qualidade dos dados, a explicabilidade dos modelos, a resistência a ameaças intencionais e a capacidade de adaptação a ambientes mutáveis são aspectos cada vez mais críticos para garantir uma *IA* confiável (*Trustworthy AI*). As soluções analisadas mostram que abordagens híbridas, que combinam técnicas tradicionais de teste, algoritmos de *ML* e metodologias baseadas em pesquisa, representam um caminho promissor para superar os desafios práticos enfrentados por organizações que dependem de sistemas de *ML* em escala real.

5.3 Resultados obtidos para a questão de pesquisa

Os testes de software tradicionais enfrentam limitações significativas de eficiência quando aplicados a sistemas baseados em *ML*. No paradigma clássico, o testador define entradas e compara as saídas com um oráculo — ou seja, uma referência confiável de comportamento correto — para decidir se o sistema funciona como esperado, segundo Myers *et al.* (2011). No entanto, em modelos de *ML*, muitas vezes não existe oráculo viável, já que os algoritmos aprendem padrões complexos a partir de dados e podem produzir saídas válidas variadas para a mesma entrada, afirma Méndez *et al.* (2024).

Além disso, a natureza opaca (*black-box*) dos modelos de *ML* dificulta o uso de ferramentas tradicionais de teste, como cobertura de código e testes unitários, pois não há uma lógica determinística totalmente acessível ao testador, de acordo com Dobslaw e Feldt (2024). Essa lacuna torna o processo manual e custoso em termos de tempo e recursos humanos.

Em contraste, as abordagens de *MLOps* — práticas de engenharia que integram *ML* a operações (*Operações (OPS)*) para automação e monitoramento em produção — incorporam técnicas adaptadas e automatizadas, elevando drasticamente a eficiência, argumenta Borg *et al.* (2021).

Entre as principais técnicas e ferramentas investigadas neste trabalho, estão:

5.3.1 Metamorphic Testing (MT)

Metamorphic Testing (MT), conceito que testa sistemas sem oráculo verificando relações de transformação conhecidas, quando combinado com classificadores de *ML*, permite automatizar a detecção de falsos positivos. No trabalho de Méndez *et al.* (2024), a combinação *MT + RF* reduziu verificações manuais de 4193 para apenas 91 tags no *OpenStreetMap* (sistema voluntário de informação geográfica), economizando semanas de esforço especializado. Essa abordagem resolve o problema clássico da inviabilidade de verificação humana em grandes volumes de dados de mapas.

Os testes metamórficos são uma técnica valiosa para avaliar sistemas nos quais o testador não dispõe de um oráculo de teste. A principal diferença entre o *MT* e o método clássico de teste é que, enquanto este último analisa isoladamente a saída gerada por uma entrada específica, o *MT* examina conjuntamente as respostas obtidas para múltiplas entradas. Em seguida, verifica-se se determinadas propriedades esperadas, denominadas relações metamór-

ficas (*Relações Metamórficas (MR)*), são preservadas entre as entradas aplicadas e as saídas observadas.

Uma estrutura de *MT* basicamente procede da seguinte forma. Um especialista precisa definir *MR* e elas devem ser implementadas. Em seguida, o testador seleciona a *MR* que deseja verificar e um conjunto de entradas de origem. As entradas são aplicadas ao sistema, as saídas correspondentes são observadas e a *MR* é invocada para verificar se foram violadas. Se tal violação for observada, será reportado. Caso contrário, o processo é iteirado com a aplicação de mais entradas. A geração de entradas subsequentes deve levar em consideração a entrada da iteração anterior e a *MR* específica que está sendo utilizada.

Na abordagem apresentada, existem dois tipos distintos de entradas: as entradas de origem, fornecidas pelo testador, que devem cobrir as funcionalidades principais do sistema; e as entradas de acompanhamento, utilizadas durante as iterações do processo. Uma boa estrutura de *MT* deve ser capaz de gerar essas entradas automaticamente, sempre que possível.

5.3.2 ML Boundary Spanning Algorithm (ML-BSA)

O *ML-BSA*, de Dobslaw e Feldt (2024), é um algoritmo para identificação automatizada de limites de decisão em classificadores de *ML*, com o objetivo de identificar automaticamente os limites de decisão de modelos de classificação de *ML*. Limites de decisão são as fronteiras que separam diferentes classes aprendidas pelo modelo. O *ML-BSA* realiza buscas e gera candidatos de fronteira em poucos segundos, classificando-os quase instantaneamente, o que viabiliza a exploração de regiões de incerteza com baixa sobrecarga computacional, mesmo tratando o modelo como caixa preta.

O *ML-BSA* baseia-se no conceito de "programa derivado". Este conceito ajuda a encontrar pares de entradas muito próximas no espaço de entrada, mas com saídas o mais diferentes possível. Tais pares são chamados de "candidatos a limite". Visa detetar um conjunto de candidatos a limite que capturam uma vasta gama do limite de decisão de um modelo de classificação. Ou seja:

- Entradas: Recebe um classificador treinado e os dados nos quais foi treinado, que consistem em tuplas.

- Saída: Devolve um conjunto de pares de candidatos a limite, onde cada par tem saídas diferentes e as suas entradas são muito próximas.

Passos do algoritmo

1. **Inicialização:** Começa com o conjunto de limites (*BC*) vazio e extrai todas as classes possíveis dos dados de treino (*O*).
2. **Procura:** O algoritmo continua testando novos limites até cumprir um critério de parada.
3. **Seleção:** Em cada ciclo, escolhe duas saídas diferentes de *O* e seleciona um ponto de treino para cada uma, formando um par (i_1, i_2).
4. **Par Candidato:** Cria o par candidato $c = (i_1, i_2)$.
5. **Otimização (*Squeeze*):** O par é ajustado por uma função de otimização que tenta aproximar os pontos no espaço de entrada, mas mantendo saídas diferentes.
6. **Programa Derivado:** Mede o quanto os pontos estão próximos na entrada e diferentes na saída.
7. Se os pontos ficarem muito próximos (menores que δ em todas as dimensões) mas com saídas distintas, formam um novo limite.
8. **Atualização:** O novo limite é adicionado ao conjunto *BC*, garantindo diversidade.

5.3.3 Deepchecks Library

A **biblioteca *Deepchecks*** de Chorev *et al.* (2022) oferece um *framework* automatizado para validar dados e modelos em múltiplas etapas do ciclo de vida de *ML*. Essa biblioteca executa verificações para integridade de dados, desvio de dados (*data drift*) e mudança de conceito (*concept drift*).

Blocos principais do Deepchecks

- **Checks (Verificações)**
  - Inspecionam aspectos específicos dos dados ou do modelo, como fuga de dados e deriva de conceito (*concept drift*).
  - Atualmente, há 62 verificações: 42 para modelos de classificação e regressão em dados tabulares e o restante para visão computacional.
  - Organizam-se em três módulos principais:
    - **Integridade dos dados**: Verifica duplicatas, tipos de dados inconsistentes, colunas com um único valor, entre outros.
    - **Validação treino-teste**: Analisa se treino e teste têm distribuições semelhantes,

detecta fuga de dados e amostras repetidas entre os conjuntos.

    * **Avaliação do modelo**: Avalia métricas de desempenho, compara com *baseline*, verifica *overfitting* e identifica padrões de erro.

- **Conditions (Condições)**
  - Funções que avaliam o resultado de uma verificação segundo um limite ou lógica definida.
  - Retornam estados: se passou, falhou ou retorna um aviso.
- **suítes (Conjuntos de verificações)**
  - Agrupam verificações e condições em uma execução única.
  - Geram relatórios resumidos e detalhados com os resultados.
  - Facilitam rodar múltiplas verificações de uma só vez.

Fluxo de uso

1. **Preparar dados e modelo**
   - Fornecer dados brutos, dados de treino/teste e o modelo.
2. **Escolher o cenário de validação**
   - **Novos Dados**: Checar qualidade e consistência.
   - **Após Divisão**: Verificar equilíbrio entre treino e teste.
   - **Após Treino**: Avaliar métricas, *overfitting* e subgrupos problemáticos.
   - ***On-Demand***: Executar verificações específicas para investigar casos pontuais.
3. **Executar verificações ou suítes**
   - Usar suítes pré-definidas ou criar combinações personalizadas.
4. **Analisar resultados**
   - Cada verificação gera visualizações e valores de retorno.
   - As condições indicam se o teste passou, falhou ou precisa de atenção.

Personalização

- Verificações e suítes podem ser modificadas, salvas e reaproveitadas.
- Atualmente, o *Deepchecks* suporta tarefas de classificação/regressão tabular e visão computacional.
- Expansões para séries temporais e tarefas de *NLP* estão em desenvolvimento.

5.3.4 Mithridates

O ***Mithridates*** de Bagdasarian e Shmatikov (2024) introduz uma abordagem para auditar e fortalecer modelos contra ataques de *backdoor* — técnica em que atacantes inserem dados maliciosos de treino para ensinar ao modelo um comportamento oculto e indesejado. *Mithridates* explora hiperparâmetros — configurações ajustáveis do processo de treino, como taxa de aprendizado ou regularização — para equilibrar precisão e resistência a ataques. Essa defesa é integrada na etapa de otimização de hiperparâmetros, prática já padrão em *MLOps*, o que elimina a necessidade de alterações estruturais no *pipeline*. Assim, obtém-se segurança adicional com custo computacional único, sem comprometer o desempenho em produção.

O *Mithridates* é uma ferramenta prática para engenheiros de *ML* testarem a vulnerabilidade dos seus *pipelines* de treino contra ataques do tipo *backdoor* e ajustarem os hiperparâmetros para aumentar a resistência, sem precisar modificar toda a estrutura do projeto.

A ideia principal é medir a resistência do modelo. Para isso, o *Mithridates* usa o conceito de ponto de resistência, que indica a menor percentagem de dados de treino que precisa ser corrompida para que um *backdoor* funcione de forma eficaz. Quanto maior esse ponto, mais difícil é explorar o modelo.

Para estimar esse ponto de forma geral, o *Mithridates* aplica uma sub-tarefa primitiva: adiciona um padrão grande e fácil de aprender em parte dos dados e observa se o modelo é enganado. Se o modelo resistir a essa tarefa simples, é provável que também resista melhor a *backdoors* reais, que costumam usar padrões mais discretos.

O processo funciona em etapas:

1. Encontrar uma configuração de hiperparâmetros que ofereça boa precisão na tarefa principal.
2. Realizar uma auditoria: envenenar parte dos dados de treino com a sub-tarefa primitiva em diferentes níveis e medir o ponto de resistência atual.
3. Fazer uma nova busca por configurações que mantenham a precisão, mas aumentem a resistência, ou seja, dificultar o sucesso de um *backdoor*. Isso pode ser feito combinando os objetivos de precisão e segurança.
4. Verificar se o novo ponto de resistência realmente melhorou e se a precisão do modelo continua aceitável.

Na prática, o *Mithridates* pode aumentar a resistência a diferentes tipos de *backdoors* em até 3 a 5 vezes, com impacto mínimo no desempenho. É compatível com qualquer modelo

ou tarefa, pode ser combinado com outras técnicas como regularização, e se integra facilmente a processos já usados, como pesquisa de hiperparâmetros e *AutoML*.

5.3.5 Framework Self-Adaptive Systems

O ***framework Self-Adaptive Systems*** de Casimiro *et al.* (2024) combina verificação de modelos probabilísticos para gerar estratégias de retreinamento de modelos. A geração dessas estratégias, que define quando e como atualizar o modelo, leva apenas 3,5 segundos, valor considerado aceitável para uso online, afirma Casimiro *et al.* (2024). Isso garante a eficiência operacional, já que o tempo para definir a estratégia é mínimo em comparação ao custo real do retreinamento.

Aplicação prática do framework

No artigo, os autores apresentam um *framework* para auto-adaptação de sistemas baseados em *ML*, utilizando *Probabilistic Model Checking* (*Probabilistic Model Checking (PMC)*), ou Verificação de Modelos Probabilística. O objetivo é permitir que o sistema avalie, de forma automática, o momento mais apropriado para realizar adaptações, como o re-treinamento do modelo, equilibrando o custo de adaptação com o ganho em desempenho.

A cada nova janela de dados, o sistema:

- Monitora a qualidade do modelo, mesmo quando os rótulos verdadeiros são recebidos com atraso.
- Estima a perda de desempenho esperada.
- Calcula o ganho potencial de um novo re-treinamento.
- Avalia o custo computacional envolvido na adaptação.

Utilizando o *PMC*, o *framework* simula diferentes estratégias de adaptação, por exemplo, re-treinar imediatamente, re-treinar futuramente ou não fazer nada (*No Operation*, NOP), e estima o impacto dessas decisões em termos de precisão do modelo e custo total.

Em termos práticos, a aplicação real do *framework* consiste em integrá-lo a um pipeline de *MLOps*, permitindo que modelos de *ML* em produção sejam monitorados continuamente. Quando sinais de degradação de desempenho (*concept drift*, *data shift* ou amostras fora de distribuição) são detectados, o sistema decide de forma autônoma se e quando a adaptação deve ocorrer, garantindo equilíbrio entre qualidade do serviço e custo operacional.

5.3.6 Test Automation with Grad-CAM Heatmaps

Por fim, a **automação de *Grad-CAM*** de Borg *et al.* (2021). São mapas de calor que destacam quais partes de uma imagem mais influenciam a decisão de uma *CNN* — representa outro ganho de eficiência. Analisar esses mapas manualmente para encontrar viés seria impraticável em larga escala. Integrando essa verificação como um segmento automatizado do *pipeline*, é possível identificar modelos enviesados de forma contínua, apoiando requisitos de *IA* Confiável como robustez, explicabilidade e mitigação de discriminação.

O *Grad-CAM* é útil para entender por que o modelo fez uma determinada previsão, verificando se ele está usando pistas corretas na imagem — por exemplo, se ao classificar um ciclista, realmente analisou a bicicleta e não o fundo da foto. Além disso, ajuda a identificar possíveis viéses, mostrando se o modelo depende de padrões irrelevantes ou enviesados. Por fim, o *Grad-CAM* pode ser integrado ao *pipeline* de *MLOps* como uma etapa de teste automatizado para detectar problemas antes de colocar o modelo em produção.

Como usar?

1. Treinar o modelo *CNN* normalmente;
2. Escolher uma imagem de teste e a classe-alvo que deseja analisar;
3. Usar uma implementação de *Grad-CAM*;
4. O algoritmo calcula os gradientes da classe-alvo em relação à última camada convolucional;
5. O *Grad-CAM* gera um mapa de ativação destacando as regiões que mais influenciaram a predição;
6. Sobrepor o *heatmap* na imagem original;
7. Analise: o modelo olhou para o objeto certo? Se não, isso pode indicar viés, dados inconsistentes ou problemas no treino.

Exemplo de uso prático

No estudo analisado, o *Grad-CAM* foi utilizado para:

– Verificar se o modelo detecta corretamente pedestres, ciclistas e cães em imagens de um túnel.
– Identificar erros de classificação. Por exemplo, quando o modelo confunde um cão com um ciclista devido a sombras ou reflexos.

– Testar o comportamento do modelo em condições fora do treinamento, como imagens noturnas.

Diante de diversas limitações e adaptações, foram apresentados os principais tipos de testes de software realizados no contexto de *ML* e *MLOps*, destacando que esta é uma área em constante evolução e que novas abordagens certamente continuarão a surgir. Assim, este capítulo reuniu os principais resultados relacionados à questão de pesquisa, evidenciando lacunas, práticas recomendadas e exemplos de aplicações, como o uso de técnicas de explicabilidade, verificação formal, monitoramento automatizado, entre outros. Espera-se que estas contribuições sirvam de base para trabalhos futuros, apoiando o desenvolvimento de estratégias de teste cada vez mais robustas, seguras e adequadas às exigências de sistemas de *ML* em ambientes produtivos.

6 CONCLUSÃO

6.1 Discussão

O teste de software é uma etapa fundamental no ciclo de vida de qualquer projeto de software, com o objetivo de garantir que o produto final atenda aos requisitos especificados e esteja livre de falhas. No entanto, no contexto de *ML*, o processo de teste apresenta desafios específicos e mais complexos em comparação com sistemas tradicionais. A natureza estocástica dos modelos de *ML*, fundamentada em cálculos probabilísticos, introduz incertezas e comportamentos não determinísticos, tornando a especificação de casos de teste e a previsão de saídas uma tarefa desafiadora. Além disso, a ausência de um oráculo de teste confiável limita a aplicação de técnicas clássicas de teste. Soma-se a isso a natureza opaca (*black-box*) dos modelos, que dificulta o uso de ferramentas tradicionais, como testes unitários e cobertura de código, pois a lógica interna do modelo nem sempre é interpretável.

Para superar essas limitações, o *MLOps* surge como uma abordagem que integra práticas de desenvolvimento, implantação e monitoramento contínuo, incorporando técnicas de teste adaptadas e automatizadas. Esta integração contribui para elevar significativamente a eficiência e a qualidade do desempenho dos modelos de *ML* em produção.

A revisão realizada evidencia um conjunto de técnicas e ferramentas emergentes que fortalecem o ciclo de vida de teste em *MLOps*. Dentre elas, destaca-se o *Metamorphic Testing MT*, que viabiliza a verificação de sistemas sem oráculo ao explorar relações de transformação conhecidas, como demonstrado por Méndez *et al.* (2024), reduzindo verificações manuais em tarefas de grande escala, como a validação de *tags* no *OpenStreetMap*. O *ML Boundary Spanning Algorithm ML-BSA*, proposto por Dobslaw e Feldt (2024), automatiza a identificação de limites de decisão em classificadores de *ML*, permitindo uma análise mais precisa de regiões de incerteza com baixa sobrecarga computacional. A biblioteca *Deepchecks*, desenvolvida por Chorev *et al.* (2022), organiza verificações de integridade de dados, detecção de *concept drift* e validação de resultados em um *framework* automatizado e reutilizável. O *Mithridates*, de Bagdasarian e Shmatikov (2024), contribui para reforçar a resistência de modelos a ataques de *backdoor*, integrando a avaliação de vulnerabilidades e a otimização de hiperparâmetros sem alterações estruturais no *pipeline*. O *Framework* de Sistemas Auto-Adaptativos, proposto por Casimiro *et al.* (2024), faz uso de *Probabilistic Model Checking PMC* para planejar retreinamentos de forma automática e eficiente, equilibrando custo e ganho de desempenho. Por fim, a automação de testes

com mapas de calor *Grad-CAM*, discutida por Borg *et al.* (2021), amplia a explicabilidade em modelos de visão computacional, permitindo identificar *bias* ou padrões irrelevantes e agregando robustez e transparência às decisões do modelo.

Em síntese, a transição para práticas de *MLOps* é motivada pela necessidade de mitigar as limitações dos testes tradicionais frente à complexidade dos sistemas de *ML*. As abordagens discutidas demonstram que a combinação de técnicas híbridas, automação, explicabilidade e adaptação dinâmica é essencial para garantir a qualidade, a confiabilidade e a escalabilidade dos modelos, superando restrições como o não-determinismo, a ausência de oráculos e a opacidade dos modelos. Assim, os resultados desta pesquisa reforçam o entendimento de que o avanço contínuo de soluções específicas para teste em *MLOps* é um caminho promissor para responder às demandas reais de sistemas de *ML* em ambientes produtivos.

6.2 Considerações finais

Este trabalho teve como objetivo principal investigar como são realizados os testes de software dentro do contexto de *MLOps*. Por meio de um mapeamento sistemático da literatura, foi possível compreender os tipos de testes aplicados, sistematizar o conhecimento disponível e analisar criticamente os resultados encontrados.

Para garantir que a análise contemplasse exclusivamente pesquisas diretamente pertinentes ao tema deste estudo, adotou-se um critério de exclusão rigoroso. Nesse sentido, foram desconsiderados todos os trabalhos que não apresentassem, em seus títulos, as palavras-chave "*Machine Learning*", "*ML*"ou "*MLOps*". Essa abordagem assegurou a seleção de literatura altamente relevante, promovendo a precisão e a consistência dos resultados obtidos. No entanto, tal rigor também impôs limitações, uma vez que estudos potencialmente relevantes que não utilizavam essas palavras-chave em seus títulos podem ter sido excluídos, reduzindo a abrangência da revisão e limitando a identificação de contribuições indiretas ou emergentes relacionadas ao tema.

A pesquisa revelou que, embora o teste de software seja essencial para a qualidade e confiabilidade em sistemas tradicionais, ele apresenta desafios distintos e complexos no domínio de *ML*, devido à natureza estocástica dos modelos, à ausência de oráculos de teste e à sua característica de funcionamento como "caixa preta". Tais fatores exigem a adaptação e evolução das práticas convencionais de teste, impulsionando o surgimento de novas estratégias, ferramentas e *frameworks*.

Os resultados levantados para a questão de pesquisa - Q1: Quais são os tipos de testes de software aplicados no contexto de *MLOps* e como eles são realizados para garantir a qualidade e confiabilidade dos modelos de *Machine Learning*? - evidenciaram que as práticas de teste em *MLOps* não apenas se comparam, mas podem superar as abordagens tradicionais ao incorporar técnicas adaptadas e automatizadas, capazes de elevar a eficiência e a qualidade dos modelos em produção. Entre os exemplos identificados destacam-se: *MT*, *ML-BSA*, biblioteca *Deepchecks*, *Mithridates*, *frameworks* de sistemas auto-adaptativos e o uso de *heatmaps* com *Grad-CAM* — todos voltados a enfrentar problemas como a falta de oráculos, detecção de limites de decisão, validação contínua, resistência a ataques e garantia de explicabilidade.

Assim, conclui-se que o *MLOps* desempenha um papel central na garantia da qualidade, confiabilidade e governança do ciclo de vida dos modelos de *ML*, adaptando-se constantemente às exigências de variação de dados, degradação de desempenho e necessidade de retreinamento. Ressalta-se que esta é uma área em plena evolução, com forte potencial de expansão e consolidação.

As lacunas identificadas na literatura incluem a ausência de práticas padronizadas, a dificuldade de integração entre equipes de desenvolvimento e operações e a necessidade de monitoramento contínuo para evitar a degradação de modelos em produção. Assim, espera-se que esta pesquisa contribua como base para trabalhos futuros, incentivando o desenvolvimento de modelos de maturidade específicos para *MLOps*, guias de boas práticas alinhados a cenários reais e estudos de caso que aproximem teoria e prática. Dessa forma, as contribuições aqui apresentadas reforçam a importância de estratégias de teste cada vez mais robustas, seguras e adequadas às demandas reais de sistemas de *ML* em ambientes produtivos.

REFERÊNCIAS

AGUIAR, I. V. G. Uma análise comparativa entre testes de software manual e automatizado. 2023.

AMERSHI, S.; BEGEL, A.; BIRD, C.; DELINE, R.; GALL, H.; KAMAR, E.; NAGAPPAN, N.; NUSHI, B.; ZIMMERMANN, T. Software engineering for machine learning: A case study. In: **Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice**. [*S. l.*: *s. n.*], 2019. p. 291–300.

AN, L. F. Exploring ml testing in practice–lessons. **Building Stronger Bridges: Strategies for Improving Communication and Collaboration Between Industry and Academia in Software Engineering**, p. 135, 2023.

BAGDASARIAN, E.; SHMATIKOV, V. Mithridates: Auditing and boosting backdoor resistance of machine learning pipelines. In: **Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security**. [*S. l.*: *s. n.*], 2024. p. 4480–4494.

BASS, L.; WEBER, I.; ZHU, L. **DevOps: A Software Architect's Perspective**. [*S. l.*]: Addison-Wesley Professional, 2015.

BISHOP, C. M. **Pattern Recognition and Machine Learning**. [*S. l.*]: Springer, 2006.

BORG, M.; JABANGWE, R.; ÅBERG, S.; EKBLOM, A.; HEDLUND, L.; LIDFELDT, A. Test automation with grad-cam heatmaps-a future pipe segment in mlops for vision ai? In: IEEE. **2021 IEEE international conference on software testing, verification and validation workshops (ICSTW)**. [*S. l.*], 2021. p. 175–181.

BRAGA, F. A. Qualidade de software: Proposta de automação de testes e de um processo ágil para uma empresa de software. **Sistemas de Informação-Florianópolis**, 2019.

CALEFATO, F.; LANUBILE, F.; QUARANTA, L. Security risks and best practices of mlops: A multivocal literature review. 2024.

CANUMA, P. Mlops: What it is, why it matters, and how to implement it (from a data scientist perspective). **Blog post at Neptune**, 2021.

CASIMIRO, M.; SOARES, D.; GARLAN, D.; RODRIGUES, L.; ROMANO, P. Self-adapting machine learning-based systems via a probabilistic model checking framework. **ACM Transactions on Autonomous and Adaptive Systems**, ACM New York, NY, v. 19, n. 3, p. 1–30, 2024.

CHOREV, S.; TANNOR, P.; ISRAEL, D. B.; BRESSLER, N.; GABBAY, I.; HUTNIK, N.; LIBERMAN, J.; PERLMUTTER, M.; ROMANYSHYN, Y.; ROKACH, L. Deepchecks: A library for testing and validating machine learning models and data. **Journal of Machine Learning Research**, v. 23, n. 285, p. 1–6, 2022.

CRAIG, R. D.; JASKIEL, S. P. **Systematic software testing**. [*S. l.*]: Artech House, 2002.

DOBSLAW, F.; FELDT, R. Automated boundary identification for machine learning classifiers. In: **Proceedings of the 17th ACM/IEEE International Workshop on Search-Based and Fuzz Testing**. [*S. l.*: *s. n.*], 2024. p. 1–8.

FITZGERALD, B.; STOL, K.-J. Continuous software engineering: A roadmap and agenda. **Journal of Systems and Software**, Elsevier, v. 123, p. 176–189, 2017.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [*S. l.*]: MIT Press, 2016.

HüTTERMANN, M. **DevOps for Developers**. [*S. l.*]: Apress, 2012.

KIM, G.; HUMBLE, J.; DEBOIS, P.; WILLIS, J. **The DevOps Handbook: How to Create World-Class Agility, Reliability, Security in Technology Organizations**. [*S. l.*]: IT Revolution, 2016.

KITCHENHAM, B.; CHARTERS, S. *et al.* Guidelines for performing systematic literature reviews in software engineering version 2.3. **Engineering**, v. 45, n. 4ve, p. 1051, 2007.

KREUZBERGER, D.; KÜHL, N.; HIRSCHL, S. Machine learning operations (mlops): Overview, definition, and architecture. **IEEE access**, IEEE, 2023.

LIMA, A.; MONTEIRO, L.; FURTADO, A. P. Mlops: Practices, maturity models, roles, tools, and challenges-a systematic literature review. **ICEIS (1)**, p. 308–320, 2022.

MAHESH, B. Machine learning algorithms-a review. **International Journal of Science and Research (IJSR).[Internet]**, v. 9, n. 1, p. 381–386, 2020.

MALDONADO, J. C.; ROCHA, A. R. C. d.; WEBER, K. C. Qualidade de software: teoria e prática. **São Paulo**, 2001.

MARIJAN, D.; GOTLIEB, A. Software testing for machine learning. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [*S. l.*: *s. n.*], 2020. v. 34, n. 09, p. 13576–13582.

MAXIM, B. R.; PRESSMAN, R. S. **Engenharia de Software: uma abordagem profissional**. [*S. l.*]: Porto Alegre: AMGH, 2021.

MÉNDEZ, M.; BECERRA-TERÓN, A.; ALMENDROS-JIMÉNEZ, J. M.; MERAYO, M. G.; NÚÑEZ, M. Combining metamorphic testing and machine learning to enhance openstreetmap. **IEEE Transactions on Reliability**, IEEE, v. 73, n. 4, p. 1834–1848, 2024.

MITCHELL, T. M.; MITCHELL, T. M. **Machine learning**. [*S. l.*]: McGraw-hill New York, 1997. v. 1.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. **Sistemas inteligentes-Fundamentos e aplicações**, v. 1, n. 1, p. 32, 2003.

MURPHY, K. P. **Machine learning: a probabilistic perspective**. [*S. l.*]: MIT press, 2012.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The Art of Software Testing**. 3rd. ed. [*S. l.*]: John Wiley & Sons, 2011. ISBN 978-0-470-91179-1.

NAJAFABADI, F. A.; BOGNER, J.; GEROSTATHOPOULOS, I.; LAGO, P. An analysis of mlops architectures: A systematic mapping study. **arXiv preprint arXiv:2406.19847**, 2024.

NAQA, I.; LI, R.; MURPHY, M. J. **Machine learning in radiation oncology: theory and Applications**. [*S. l.*]: Springer International Publishing, 2015.

NETO, A.; CLAUDIO, D. Introdução a teste de software. **Engenharia de Software Magazine**, v. 1, p. 22, 2007.

RICCIO, V.; JAHANGIROVA, G.; STOCCO, A.; HUMBATOVA, N.; WEISS, M.; TONELLA, P. Testing machine learning based systems: a systematic mapping. **Empirical Software Engineering**, Springer, v. 25, p. 5193–5254, 2020.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 4. ed. [*S. l.*]: Pearson, 2021.

SAH, S. Machine learning: a review of learning types. Preprints, 2020.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. **IBM Journal of Research and Development**, IBM, v. 3, n. 3, p. 210–229, 1959.

WEYUKER, E. J. On testing non-testable programs. **The Computer Journal**, The British Computer Society, v. 25, n. 4, p. 465–470, 1982.

ZAHARIA, M.; CHEN, A.; DAVIDSON, A.; GHODSI, A.; HONG, H.; KONWINSKI, A.; MIAO, M.; PERRY, P.; ROBINSON, M.; WITT, X. Machine learning operations: Overview, definition, and architecture. **arXiv preprint arXiv:2202.04837**, 2022.

ZHANG, Y.; WU, X.; YANG, P.; XU, B.; LI, S. Machine learning testing: Survey, landscapes and horizons. **IEEE Transactions on Emerging Topics in Computing**, IEEE, v. 9, n. 3, p. 1486–1506, 2020.