OBSERVABILIDADE DE SISTEMAS: Detecção Preditiva de Falhas[1]

Guilherme A. Lima[2]
Danilo B. Seufitelli[3]

Submetido em: 19/06/2026 | Aprovado em: 19/06/2026

RESUMO

Este trabalho avalia métodos preditivos para detectar falhas em softwares, aplicando conceitos de observabilidade e aprendizado de máquina. A proposta parte da necessidade de se antecipar falhas em sistemas complexos, onde o volume e a diversidade de métricas dificultam a identificação manual. O estudo adota uma abordagem experimental em duas etapas: a primeira cria um ambiente de simulação, utilizando softwares open source para auxiliar na geração de dados de treino e teste. Na segunda, são avaliados diferentes modelos de aprendizado de máquina para predição, como LightGBM, CatBoost, XGBoost e LSTM. Por fim, são avaliados os resultados e a viabilidade do uso dos modelos gerados. Os resultados demonstram a viabilidade técnica da utilização de métricas de observabilidade para a detecção preditiva de falhas, com modelos capazes de identificar padrões associados à degradação e indisponibilidade dos sistemas. Além disso, este projeto fornece uma arquitetura reprodutível e de baixo custo, aplicável em ambientes de pesquisa e desenvolvimento, reforçando o papel da observabilidade como componente essencial na confiabilidade de sistemas modernos.

Palavras-chave: Observabilidade. Aprendizado de Máquina. Detecção de Falhas.

ABSTRACT

This work evaluates predictive methods for detecting software failures by applying concepts of observability and machine learning. The proposal stems from the growing need to anticipate failures in complex systems, where the volume and diversity of metrics make manual identification difficult. The study adopts an experimental approach in two stages: the first creates a simulation environment using open-source software to assist in generating training and test data. In the second stage, different machine learning models for prediction are evaluated, such as LightGBM, CatBoost, XGBoost, and LSTM. Finally, the results and feasibility of using the

[1] Artigo científico elaborado para o trabalho de conclusão do curso de Engenharia de Computação do campus Nova Gameleira do Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG.)

[2] Graduando em Engenharia de Computação no campus Nova Gameleira do Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG.). guilherme.lima@aluno.cefetmg.br

[3] Orientador do trabalho. Docente do Departamento de Computação do campus Nova Gameleira do Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG.). danilobs@cefetmg.br

generated models are assessed. The results demonstrate the technical feasibility of using observability metrics for predictive fault detection, with machine learning models achieving high predictive performance. Furthermore, this project provides a reproducible and low-cost architecture applicable to research and development environments, reinforcing the role of observability as an essential component in the reliability of modern systems.

Keywords: Observability. Machine Learning. Failure Detection.

1 INTRODUÇÃO

A crescente complexidade dos sistemas de computação, principalmente em sistemas que utilizam arquiteturas de microsserviços, torna sua manutenção e disponibilidade cada vez mais desafiadoras. Afinal, é indesejável que qualquer sistema fique indisponível. Um destaque especial são as aplicações críticas, como os sistemas bancários, que podem causar grandes prejuízos financeiros (chegando a milhões de dólares) caso fiquem indisponíveis por algumas horas (New Relic, 2025). Nesse cenário, a observabilidade ajuda a identificar muitos dos problemas ao atuar com um conjunto de práticas essenciais para compreender o comportamento interno das aplicações. Tal técnica permite diagnosticar problemas após sua ocorrência e ajuda a identificar os problemas em fase inicial, antes de impactar muitos usuários (IBM, 2025).

A maior parte das soluções atuais de observabilidade e monitoramento de sistemas foca na detecção reativa (Hallur, Jayanna, 2024; Vollem, Shekar, 2023), portanto, em alertar sobre falhas que já estão ocorrendo. No entanto, a abordagem preditiva ainda é pouco utilizada e de forma limitada. A detecção de falhas antes de sua ocorrência gera um impacto positivo, principalmente no setor financeiro. Falhas em serviços como PIX[4] ou cartões de crédito podem gerar prejuízos milionários em pouco tempo de indisponibilidade. Na área da saúde, a indisponibilidade de sistemas de gestão hospitalar pode comprometer diagnósticos e tratamentos médicos (Kim, Mi Ok *et al.*, 2017). Além disso, em qualquer outro tipo de sistema, a estabilidade garante uma boa experiência do usuário, um fator estratégico e importante para a satisfação do cliente (Homyamyen, Patipol *et al.*, 2024).

Neste contexto, é possível afirmar que falhas inesperadas podem causar fortes impactos econômicos e sociais. Logo, é desejável diminuir a má experiência dos usuários quando alguma aplicação fica indisponível, pois um sistema inoperante pode ser frustrante e atrapalhar o cotidiano das pessoas. Além disso, deseja-se evitar ou reduzir as perdas financeiras com as quais as instituições arcam atualmente. Assim, o objetivo deste trabalho é transformar dados de observabilidade em insumos preditivos para depois utilizá-los em diferentes algoritmos de aprendizado de máquina capazes de antecipar falhas dos sistemas. Especificamente foram formuladas as seguintes questões de pesquisa (RQs - *Research Questions*):

RQ1. É tecnicamente viável e reprodutível utilizar dados de observabilidade para prever falhas em sistemas por meio de aprendizado de máquina supervisionado com alta performance (F1-score > 0.9)?

RQ2. Qual modelo de aprendizado de máquina supervisionado (LightGBM, CatBoost, XGBoost, ou LSTM) apresenta o melhor desempenho preditivo (F1-score) para o

[4] PIX: <https://www.bcb.gov.br/estabilidadefinanceira/pix>

sistema WordPress/MariaDB?

RQ3. Modelos de Redes Neurais que consideram a dimensão temporal, como LSTM, apresentam melhor desempenho (F1-score) em comparação com modelos de *Gradient Boosting* para predição de falhas?

A metodologia adota uma abordagem experimental. Inicialmente, são coletados e organizados dados representativos do comportamento do sistema WordPress em diferentes condições de operação. Em seguida, esses dados são utilizados na construção e comparação de modelos de aprendizado de máquina, visando identificar as técnicas mais eficazes na antecipação de falhas iminentes.

O trabalho permitiu avaliar a viabilidade da abordagem proposta e responder às três questões de pesquisa definidas. Além disso, foi desenvolvida uma arquitetura reprodutível, abrangendo desde a geração dos dados de treino até a criação e avaliação dos modelos de aprendizado de máquina. A principal contribuição deste trabalho é oferecer uma solução prática e de código aberto para detecção preditiva de falhas, aplicável tanto em ambientes acadêmicos quanto comerciais. Adicionalmente, a proposta reforça o papel da observabilidade como ferramenta estratégica na manutenção proativa de sistemas críticos, com potencial para apoiar a redução do tempo médio de detecção (*Mean Time to Detect* - MTTD) e do tempo médio de recuperação (*Mean Time to Repair* - MTTR).

O restante do trabalho está organizado da seguinte forma. A Seção 2 apresenta os conceitos teóricos fundamentais relacionados à observabilidade, métricas de desempenho, falhas em sistemas distribuídos e aprendizado de máquina. A Seção 3 discute os trabalhos relacionados, contextualizando a proposta em relação ao estado da arte. Na Seção 4 é detalhada a metodologia adotada, incluindo a geração dos dados e a construção dos modelos preditivos. A Seção 5 descreve os resultados, e, por fim, a Seção 6 apresenta as conclusões do trabalho.

2 REFERENCIAL TEÓRICO

Nesta seção, são discutidos os conceitos necessários para o entendimento do trabalho. Especificamente, são abordados os conceitos de observabilidade em sistemas distribuídos, os tipos de métricas utilizadas para monitoramento e diagnóstico de falhas, bem como os fundamentos de aprendizado de máquina aplicados à detecção preditiva. Esses conceitos fornecem a base teórica necessária para compreender a metodologia proposta e os modelos utilizados neste trabalho.

2.1 Microsserviços e Sistemas Distribuídos

Sistemas distribuídos são compostos por múltiplos componentes que executam de forma independente e se comunicam por meio de uma rede para fornecer um serviço unificado aos usuários (Tanenbaum, Andrew S.; Van Steen, Maarten, 2017). Uma das arquiteturas mais utilizadas atualmente para construção desses sistemas é a arquitetura de microsserviços (Datadog, 2024), na qual uma aplicação é dividida em serviços menores e independentes, cada um responsável por uma funcionalidade específica.

Essa abordagem oferece benefícios como escalabilidade independente dos componentes, maior flexibilidade de desenvolvimento e implantação, além de facilitar a manutenção

de aplicações complexas. Por esse motivo, arquiteturas de microsserviços são amplamente adotadas em sistemas modernos de grande porte (Datadog, 2024).

Entretanto, a distribuição das funcionalidades entre diversos serviços aumenta a complexidade operacional do sistema (Dragoni, Nicola *et al.*, 2017). Falhas podem ocorrer em diferentes componentes e se propagar entre eles, dificultando a identificação da causa raiz dos problemas. Nesse contexto, práticas de observabilidade tornam-se fundamentais para compreender o comportamento do sistema e apoiar a detecção de falhas.

2.2 Observabilidade em Sistemas Distribuídos

A observabilidade é um conjunto de práticas que permite compreender o estado interno de um sistema a partir de informações externas, como métricas, logs e traces (Sridharan, Cindy, 2018). Em arquiteturas modernas baseadas em microsserviços, a complexidade aumenta significativamente, tornando insuficiente o monitoramento tradicional (IBM, 2025). Assim, a observabilidade fornece meios para detectar anomalias, diagnosticar causas e aumentar a resiliência operacional. As técnicas de observabilidade de sistemas são possíveis graças à análise de dados, principalmente por esses três tipos de dados que os sistemas possuem (Cloud Native Computing Foundation, 2025):

- **Métricas:** valores numéricos que permitem acompanhar a saúde do sistema (uso de CPU, latência, taxa de erros, etc.).
- **Traces:** rastreamento de requisições ao longo de múltiplos serviços, possibilitando entender fluxos complexos e gargalos
- **Logs:** registros detalhados de eventos que descrevem o que ocorreu em determinado momento. Um log pode ser um erro, algum perigo, ou qualquer outra informação relevante de alguma ocorrência.

Entre os três tipos de dados, as métricas, em especial, serão fundamentais para o desenvolvimento deste trabalho, pois fornecem uma visão temporal quantitativa sobre o comportamento do sistema. Indicadores como uso de CPU, tempo de resposta de requisições, taxa de erros, entre outras métricas, ajudam a identificar padrões, prever situações de risco e agir proativamente. Por exemplo, se a utilização de CPU aumentar abruptamente, próximo de 90% da capacidade total do servidor, é um sinal de que o sistema pode enfrentar falhas ou lentidão em um futuro próximo.

Métricas de Desempenho em Sistemas Web. As métricas são fundamentais para avaliar o comportamento de sistemas web e identificar situações de degradação. Elas podem ser classificadas em:

- **Métricas de infraestrutura:** uso de CPU do servidor, uso de memória, total de escritas e leituras no disco, entre outras
- **Métricas de aplicação:** tempo de resposta de uma requisição, quantidade de requisições por segundo, taxa de erros.
- **Métricas de banco de dados:** locks, conexões ativas, latência de queries.

Vale ressaltar que um sistema pode ficar inoperante se houver problemas em qualquer uma das três classes (infraestrutura, aplicação e banco de dados), no entanto, é necessário

coletar e utilizar todas as métricas para criação de um modelo bom. A tarefa de coletar métricas de diferentes classes e fontes não é simples. Ferramentas como o Prometheus (Linux Foundation, 2025), ajudam a coletar e armazenar essas métricas em bancos de dados de séries temporais, possibilitando análises futuras. O Prometheus é um dos softwares mais utilizados para coleta de métricas de diferentes classes e fontes.

Falhas em Sistemas Distribuídos. As falhas podem ocorrer por diversos motivos, por exemplo: sobrecarga, bugs, indisponibilidade de serviços externos, saturação de banco de dados ou problemas de rede. Em alguns casos, sinais prévios de falha surgem minutos antes da indisponibilidade completa, como:

- aumento gradual da latência.
- crescimento repentino da utilização de CPU/memória.
- oscilação na taxa de erros HTTP.
- queda ou aumento de requisições.

A literatura mostra que detectar esses sinais antecipadamente reduz o *Mean Time To Detect* (MTTD) e o *Mean Time To Recover* (MTTR), melhorando a confiabilidade dos sistemas (Cheng, Qian *et al.*, 2023).

2.3 Aprendizado de máquina para detecção preditiva

Os algoritmos de aprendizado de máquina podem ser divididos em três categorias: aprendizado supervisionado, não supervisionado e semi-supervisionado. No aprendizado supervisionado, são utilizados dados rotulados (com a "resposta"correta) para treinar modelos de classificação. Por exemplo, dado um conjunto de valores de métricas, é possível verificar se o sistema está operante ou não operante. Esta foi a categoria escolhida para o projeto, pois quando se tem dados rotulados, o aprendizado supervisionado geralmente apresenta classificações mais precisas (Mitchell, Tom M., 1997).

Quanto ao aprendizado não supervisionado, este é utilizado com dados não rotulados para descobrir padrões e estruturas ocultas, sendo ideal para agrupamento (*Clustering*) de dados semelhantes. Embora essa abordagem pudesse ser utilizada para identificar padrões de operação normal e detectar comportamentos divergentes, ela não foi adotada neste trabalho. Isso ocorre porque um sistema pode apresentar diferentes estados de funcionamento considerados normais, formando múltiplos agrupamentos válidos. Como consequência, desvios em relação a um desses grupos podem ser interpretados incorretamente como anomalias, aumentando a incidência de falsos positivos e reduzindo a eficácia da predição de falhas.

No aprendizado semi-supervisionado, é combinada uma pequena quantidade de dados rotulados com uma grande quantidade de não rotulados para treinar modelos de forma mais eficiente.

Algoritmos de Aprendizado de Máquina. Os algoritmos *gradient boosting* combinam várias árvores de decisão fracas em um modelo forte, ajustando cada árvore para corrigir erros das anteriores. São altamente eficazes em tarefas estruturadas e costumam apresentar alto desempenho com baixo custo computacional (Friedman, Jerome H., 2001). Entre eles, os principais são:

- **XGBoost:** otimizado em performance e paralelização (Chen, Tianqi; Guestrin, Carlos, 2016).
- **LightGBM:** utiliza *leaf-wise growth*, sendo mais rápido que o XGBoost em *datasets* grandes (Ke, Guolin *et al.*, 2017).
- **CatBoost:** lida melhor com dados categóricos e colabora para evitar *overfitting* (Dorogush, Leo *et al.*, 2018).

Os algoritmos de *gradient boosting* conseguem capturar padrões, por exemplo, se a utilização da CPU estiver acima de 90% é um indicativo de falha, não importando quanto estavam antes. Ou então, capturar vários padrões que em conjunto são indicativos de falha, por exemplo, se a utilização da CPU estiver em 70% e a quantidade de requisições por minuto estiver acima de 1000. Mas se para essa mesma quantidade de requisições, o CPU estiver em 50%, então o sistema não está prestes a falhar.

Apesar disso, para esses algoritmos, o passado não importa. Nesse sentido, não é capturado o padrão de evolução das métricas ao longo do tempo. Por exemplo, se a utilização de CPU aumentar abruptamente, por exemplo, de 10% para 50% em menos de 1 minuto, os algoritmos de *gradient boosting* podem não prever falha, visto que 50% é considerado normal. Mas como o aumento foi abrupto, isso pode ser sim um indicativo de falha.

Redes Neurais LSTM. As redes *Long Short-Term Memory* (LSTM) são adequadas para séries temporais, pois conseguem capturar dependências de longo e curto prazo (Hochreiter, Sepp; Schmidhuber, Jürgen, 1997). Em métricas de observabilidade, cada valor depende de estados anteriores, o que torna LSTM uma escolha natural para detecção preditiva de falhas baseadas em padrões temporais. Diferentemente dos modelos baseados em árvores, as LSTMs analisam a sequência completa de métricas, não apenas valores isolados. Para o exemplo de aumento abrupto da CPU, esse algoritmo conseguiria capturar este padrão, mesmo que 50% de utilização de CPU não seja tão alto.

Métricas de avaliação. Durante a criação dos modelos de aprendizado de máquina, precisamos avaliar se o modelo em questão é considerado bom ou ruim. Para avaliar, existem diferentes métricas de avaliação. Dependendo de como os dados são distribuídos, e de como se define um modelo bom, podemos escolher a métrica de avaliação ideal (Mitchell, Tom M., 1997). Algumas das mais comuns são:

- **Acurácia**: Mede a proporção total de previsões corretas do modelo (tanto positivos quanto negativos corretos) em relação ao total de exemplos. É útil quando o conjunto de dados é balanceado, mas enganoso se as classes estiverem desbalanceadas.
- **Precisão**: é útil para responder: dos exemplos que o modelo previu como positivos, quantos são realmente positivos? É crucial quando o custo de um falso positivo (classificar como positiva uma observação que pertence à classe negativa) é alto.
- **Recall**: o Recall responde a pergunta: dos exemplos que são realmente positivos, quantos o modelo conseguiu identificar corretamente? É crucial quando o custo de um falso negativo (não detectar algo importante) é alto.
- **F1-score**: É a média harmônica entre Precisão e Recall, oferecendo um equilíbrio entre as duas métricas. É um indicador adequado quando se deseja um bom balanço entre falsos positivos e falsos negativos, especialmente em conjuntos desbalanceados.

- **AUC-ROC**: Mede a capacidade do modelo de distinguir entre as classes, ou seja, quão bem o modelo ranqueia os exemplos positivos acima dos negativos. O valor varia de 0 a 1, onde 1 é um classificador perfeito e 0.5 é um classificador aleatório.

Neste trabalho, um falso negativo (ou seja, não prever uma falha) vai contra o objetivo principal da proposta, portanto, totalmente indesejado. Já um falso positivo (ou seja, prever uma falha e ela não acontecer), também é danoso, pois uma pessoa que está de plantão, pode ser acionada na madrugada para verificar a integridade do sistema. Não só isso, mas se o sistema gerar muitos falsos positivos, o modelo perde credibilidade, e quando ele realmente prever uma falha real, os administradores do sistema podem não acreditar, e portanto, não dando a devida atenção para a solução da falha. Nesse sentido, como um falso positivo e falso negativo são indesejáveis, foi escolhida a métrica F1-score, pois é desejado um balanço entre falsos positivos e falsos negativos, especialmente no conjunto de dados de teste deste trabalho, que é desbalanceado.

3 TRABALHOS RELACIONADOS

Ferramentas comerciais de observabilidade como Datadog e NewRelic, passaram a incorporar técnicas de inteligência artificial e aprendizado de máquina para detecção de falhas (Cheng, Qian *et al.*, 2023). No entanto, essas soluções utilizam modelos genéricos treinados para um amplo conjunto de sistemas, o que limita sua eficácia em contextos específicos. Além disso, por serem ferramentas proprietárias, não permitem customização ou adaptação dos modelos às características de cada sistema. Nesse sentido, este trabalho se diferencia por propor um modelo treinado especificamente para cada aplicação monitorada, permitindo previsões mais adequadas ao comportamento real do sistema.

No que diz respeito à predição baseada em logs, (Hadadi, Fatemeh *et al.*, 2024) realizaram uma avaliação sistemática de modelos de *deep learning* para previsão de falhas usando exclusivamente dados de log. O objetivo do estudo é comparar diferentes arquiteturas (LSTM, BiLSTM, CNN e Transformer) combinadas com múltiplas estratégias de *embedding* (BERT, Logkey2vec e FastText+TF-IDF), cobrindo lacunas de estudos anteriores. Metodologicamente, os autores propõem uma arquitetura modular que permite combinar livremente *embeddings* e modelos, além de gerar 360 conjuntos sintéticos de dados para avaliar como características como tamanho do *dataset* e proporção de falhas afetam o desempenho. Os resultados mostram que a combinação CNN + Logkey2vec apresenta o melhor desempenho geral, especialmente quando o *dataset* possui mais de 350 sequências ou mais de 7,5% de falhas. Apesar de apresentar o mesmo objetivo geral deste trabalho (prever falhas) a metodologia difere significativamente: enquanto (Hadadi, Fatemeh *et al.*, 2024) se baseiam exclusivamente em logs e avaliam arquiteturas profundas para classificação supervisionada, este trabalho se concentra principalmente em métricas, o que altera o tipo de entrada, os algoritmos de aprendizado de máquina e a forma de detecção.

Já no cenário de detecção não supervisionada, (Zhao, Chenyu *et al.*, 2023) propõem o AnoFusion, um método voltado para detectar falhas em microsserviços combinando três modalidades de dados: métricas, logs e traces. O objetivo do trabalho é lidar com limitações de métodos unimodais, que tendem a gerar falsos positivos ou não detectar falhas que se manifestam apenas em um subconjunto dos sinais observáveis. A metodologia integra dados multimodais em grafos heterogêneos, modelando correlações entre sistemas por meio de uma *Graph Transformer Network* (GTN). Em seguida, aplica uma *Graph Attention*

Network (GAT) para filtrar padrões relevantes e uma GRU para capturar dependências temporais. Como resultado, o AnoFusion alcança F1-score de 0.857 e 0.922 em dois *datasets* de microserviços, superando métodos de estado da arte baseados em uma única modalidade. Diferentemente deste trabalho, o AnoFusion adota um modelo não supervisionado, foca na detecção (não predição) e integra múltiplas modalidades simultâneas. Já este trabalho se concentra principalmente em métricas e em modelos supervisionados específicos para cada sistema, além de buscar uma solução mais simples e aplicável em ambiente real.

Além desses trabalhos, um estudo amplamente reconhecido na área é DeepLog (Du, Min *et al.*, 2017). O objetivo do DeepLog é detectar anomalias e prever falhas com base no comportamento sequencial de logs. A metodologia consiste em modelar logs como sequências de eventos e treinar uma rede LSTM para aprender os padrões normais. Durante a inferência, uma entrada é considerada anômala quando o próximo evento previsto não corresponde ao observado. Os resultados mostraram que o modelo alcança precisão elevada em *datasets* reais (como HDFS e BGL), introduzindo um paradigma que influenciou diversos trabalhos posteriores. Diferentemente deste TCC, o DeepLog utiliza somente logs estruturados e se fundamenta na previsão do próximo evento textual, enquanto o presente trabalho utiliza métricas numéricas e modelos supervisionados focados no comportamento quantitativo do sistema.

Além disso, (Yèche, Hugo; others, 2023) apresenta uma discussão relevante sobre a forma de rotulagem temporal em problemas de previsão de eventos críticos. Os autores argumentam que abordagens tradicionais frequentemente tratam todos os instantes anteriores ao evento como exemplos negativos, ignorando que os momentos imediatamente anteriores à falha já contêm sinais importantes de degradação do sistema. Nesse contexto, o artigo propõe uma estratégia de rotulagem em que amostras próximas ao evento crítico passam a ser consideradas positivas ou pertencentes a uma zona de transição, permitindo que os modelos aprendam padrões precursores de falha com maior precisão. Essa ideia é particularmente relevante para este trabalho, pois métricas de observabilidade frequentemente apresentam alterações graduais antes da indisponibilidade completa do sistema, como aumento de latência, crescimento do uso de CPU e oscilações na taxa de erros. Assim, a estratégia de rotulagem temporal discutida por Yeche et al. reforça a importância de considerar o comportamento anterior à falha no treinamento dos modelos preditivos.

Dessa forma, embora os trabalhos analisados compartilhem o objetivo de identificar ou prever falhas automaticamente, eles diferem deste trabalho, tanto nas modalidades de dados utilizadas quanto na abordagem de aprendizado. Enquanto as abordagens correlacionadas exploram logs ou combinações complexas de dados em arquiteturas avançadas, este trabalho propõe uma solução centrada nas métricas e orientada à customização por sistema, visando melhorar a precisão da predição em cenários específicos.

4 METODOLOGIA

A metodologia do trabalho está dividida em duas partes. A primeira consiste na geração dos dados de treino e teste, por meio da criação de um ambiente de simulação de serviços, onde são induzidas falhas controladas para coleta de métricas. A segunda parte utiliza os dados gerados na primeira, para realizar a criação e comparação de diferentes modelos de aprendizado de máquina, prevendo se o sistema entrará em falha ou não.

4.1 Gerar Conjuntos de Dados

Nesta primeira parte, a metodologia consiste na geração dos dados de treino e teste. Para isso, é criado um ambiente de simulação controlado, capaz de reproduzir falhas e registrar métricas em tempo real. A Figura 1 apresenta a visão geral da metodologia proposta da primeira parte.

Monitorar o sistema WordPress e MariaDB. O sistema escolhido para o experimento é o WordPress. A escolha deve-se ao fato de ser um dos gerenciadores de conteúdo (CMS) mais populares do mundo, amplamente utilizado em sites e aplicações web. Além de ser *open source*, amplamente documentado e representativo de um sistema web real, que envolve tanto requisições HTTP quanto operações de banco de dados. O WordPress é composto essencialmente por duas partes:

- **Aplicação WordPress:** Responsável por responder às requisições HTTP e processar o conteúdo.
- **Banco de dados MariaDB:** Responsável por armazenar posts, configurações e dados do sistema.

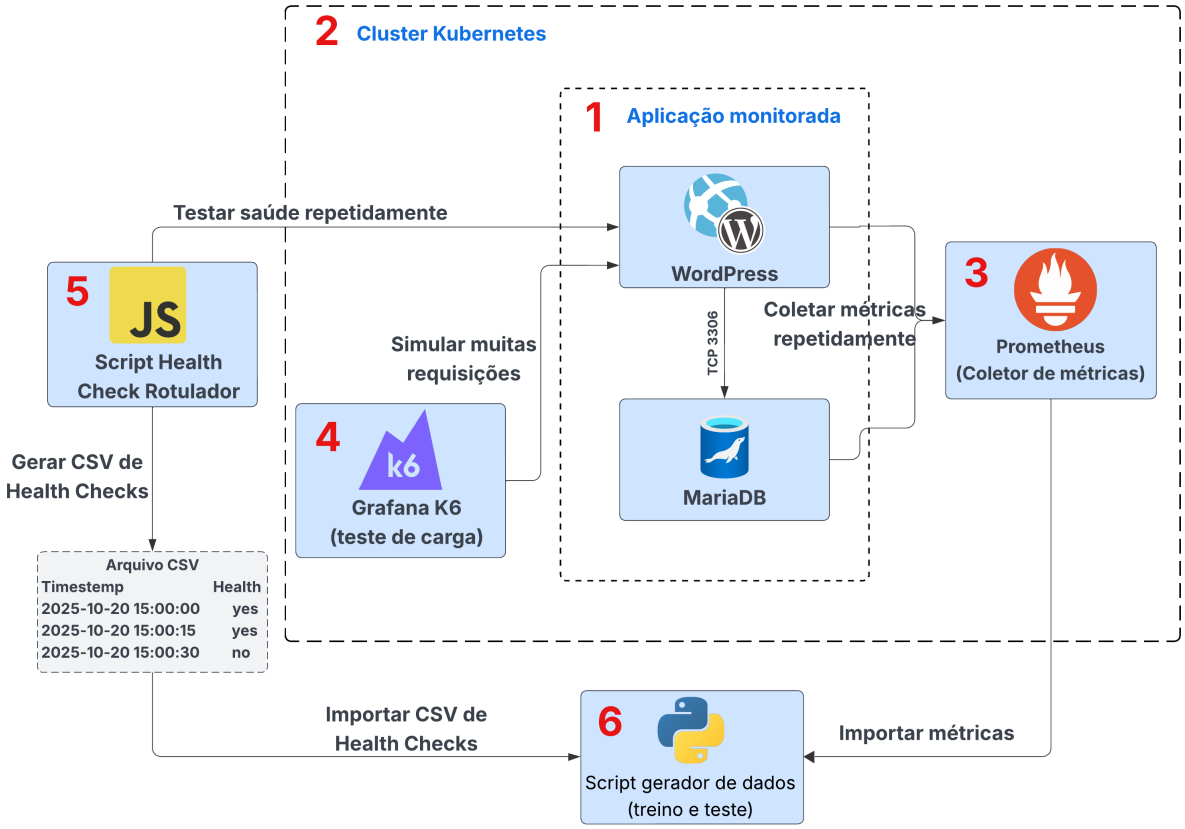
Apesar do foco do trabalho não ser o sistema em si, o WordPress foi escolhido por oferecer um comportamento realista e ainda ser muito utilizado em produção, tornando o estudo mais relevante do ponto de vista prático.

Executar o sistema e ferramentas com Kubernetes. Para facilitar a hospedagem do ambiente e coleta das métricas, foi criado um cluster Kubernetes no computador local. O Kubernetes[5] é uma plataforma de orquestração de contêineres, que permite gerenciar, escalar e monitorar aplicações distribuídas de forma automatizada. Ele facilita o gerenciamento de múltiplos serviços interconectados, como o WordPress, o MariaDB e o Prometheus. Para simplificar o processo de desenvolvimento e evitar a complexidade de uma infraestrutura em nuvem, foi utilizado o MicroK8s[6], que é uma distribuição simplificada do Kubernetes voltada para uso local e desenvolvimento. Ele permite criar um cluster completo de forma leve, com todos os componentes necessários para simulação e coleta de métricas.

Coletar métricas com Prometheus. Uma vez implantado o sistema WordPress e o banco MariaDB (bastando adicionar os manifestos deles no repositório Git), também foi adicionado o manifesto para instalar o Prometheus[7], uma ferramenta *open source* amplamente usada para monitoramento e observabilidade de sistemas distribuídos. O Prometheus funciona de forma ativa, realizando coletas periódicas de métricas publicadas pelos serviços monitorados, acessando seus endpoints HTTP (no caso, /metrics na porta 9090 do WordPress e do MariaDB). Essas métricas incluem informações como tempo de resposta, número de requisições, erros HTTP, entre outras. Além disso, o Prometheus também obtém as métricas das máquinas (servidores), a partir do Node Exporter que é instalado por padrão durante a instalação do Kubernetes. O Node Exporter permite que o Prometheus colete métricas de infraestrutura, como utilização de CPU, memória e disco. Por fim, todos os dados dessas métricas são salvos em um banco de dados de séries temporais do Prometheus, permitindo consultas precisas e correlacionadas com o tempo.

[5] Kubernetes: <https://kubernetes.io>
[6] MicroK8s: <https://canonical.com/microk8s>
[7] Prometheus: <https://prometheus.io/>

Figura 1 – Diagrama da arquitetura para geração de dados de treino e teste



Fonte: Produção do próprio autor.

Gerar carga com Grafana K6. Para simular o uso real do sistema e gerar situações de sobrecarga, foi utilizada a ferramenta Grafana K6. O Grafana K6[8] é uma ferramenta de teste de carga e desempenho, que executa *scripts* capazes de simular múltiplos usuários acessando a aplicação simultaneamente. O objetivo dessa ferramenta no trabalho é gerar estresse controlado para induzir falhas. Como resultado, são geradas métricas de séries temporais sob carga variável. No experimento, o K6 realiza requisições HTTP repetidas ao WordPress, acessando páginas, criando posts e realizando operações comuns. Esse processo é fundamental para observar o comportamento do sistema sob diferentes níveis de carga e gerar situações em que o desempenho se degrada ou o sistema falha.

Rotular a saúde do sistema com JavaScript. Paralelamente ao teste de carga, foi desenvolvido um *script* em JavaScript (executado com Node.js), cujo objetivo é monitorar a disponibilidade do sistema. Esse *script* realiza periodicamente requisições ao WordPress (acessando a página inicial e endpoints de teste) e mede o tempo de resposta. Caso a resposta seja algum erro, ou demore mais de 3 segundos (*timeout*), considera-se que o sistema está em estado de falha. Cada verificação gera um registro contendo o timestamp e o estado do sistema (health = yes/no). Esses dados são salvos em um arquivo CSV, formando o conjunto inicial de rótulos (labels) que indicam, em cada momento, se o sistema estava saudável ou não.

Gerar dados de treino e teste com Python. Por fim, foi desenvolvido um *script* em

[8] Grafana K6: <https://k6.io/>

Python responsável por unir os rótulos gerados pelo script rotulador às métricas coletadas pelo Prometheus. Esse *script* importa o arquivo CSV com os *timestamps* e estados do sistema, e para cada registro realiza uma consulta à API do Prometheus, recuperando as métricas. O alinhamento foi realizado por proximidade temporal. Como o Prometheus armazenava métricas a cada 10 segundos, cada rótulo foi associado às métricas coletadas no *timestamp* mais próximo disponível. O resultado é um conjunto de dados tabular completo, pronto para o treinamento dos modelos, no qual cada linha representa um momento específico, com todas as métricas numéricas disponíveis (CPU, memória, latência, etc.) e o respectivo rótulo (target) "funcionando" (target = 1) ou "falhando" (target = 0). Esse conjunto é então dividido em dados de treino (70%) e teste (30%), que são utilizados na segunda parte do trabalho para treinar e avaliar diferentes modelos de aprendizado de máquina voltados à predição de falhas em sistemas distribuídos.

Tabela 1 – Parte do Conjunto de dados A

t_f **(Timestamp)**	**UsoCPU**	**UsoMemoria**	**...**	**Target**
...	...	...	...	...
12/09/2025 12:00:00	2.0	3.0	...	1
12/09/2025 12:00:10	2.1	3.1	...	1
12/09/2025 12:00:20	2.6	3.1	...	1
12/09/2025 12:00:30	2.6	3.2	...	1
12/09/2025 12:00:40	2.7	3.3	...	1
12/09/2025 12:00:50	2.7	3.3	...	1
12/09/2025 12:01:00	2.8	3.4	...	1
12/09/2025 12:01:10	2.9	3.5	...	1
12/09/2025 12:01:20	3.0	3.6	...	0
...	...	...	...	...

Fonte: Produção do próprio autor.

A partir do conjunto de dados A (Tabela 1), foi criado um novo conjunto (Tabela 2), seguindo a ideia discutida em (Yèche, Hugo; others, 2023), de que instantes imediatamente anteriores a um evento crítico não devem ser tratados como negativos, foi aplicada uma rotulagem temporal de pré-falha. Em particular, para cada ocorrência de falha em t_f (identificada pelo target), todas as amostras no intervalo $[t_f - 60\,\text{s},\ t_f]$ foram rotuladas como 0. Assim, os modelos foram treinados para reconhecer padrões que antecedem falhas com uma antecedência de até 1 minuto. A Tabela 2 exemplifica esse novo conjunto de dados.

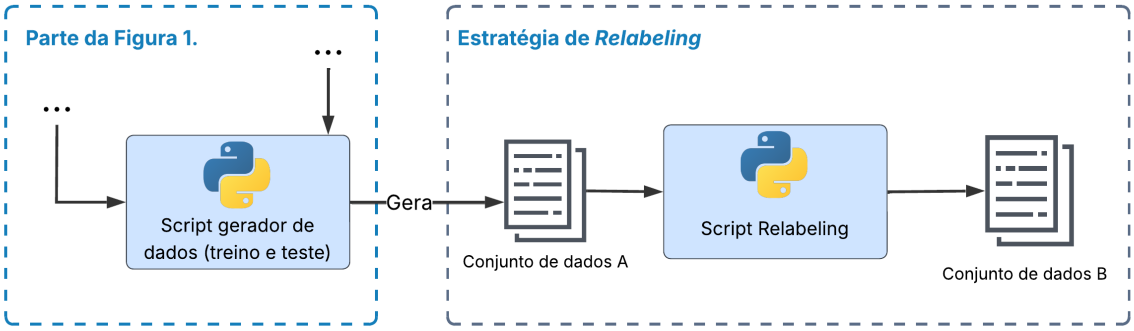
Essa estratégia é válida porque falhas em sistemas distribuídos raramente ocorrem de forma instantânea, sendo geralmente precedidas por sinais graduais de degradação, como aumento de latência, crescimento da taxa de erros etc. A Figura 2 apresenta o processo de geração do conjunto de dados B com *Relabeling*. Ao rotular também os instantes imediatamente anteriores à falha como 0 (falha), o modelo passa a aprender padrões precursores do problema, e não apenas o momento exato da indisponibilidade. Além disso, essa abordagem reduz ambiguidades na fronteira entre estados normais e estados críticos do sistema. A estratégia também está alinhada com o objetivo central do trabalho, que é realizar predição antecipada de falhas, permitindo ações preventivas antes da interrupção completa do serviço. Para realizar esse procedimento, um novo *script* em Python foi criado, que detecta o target 0, e altera as linhas com até 1 minuto anterior para target 0 também.

Tabela 2 – Parte do Conjunto de dados B (*Relabeling* 1 minuto antes da falha)

Timestamp	**UsoCPU**	**UsoMemoria**	**...**	**Target**
...	...	...	...	...
12/09/2025 12:00:00	2.0	3.0	...	1
12/09/2025 12:00:10	2.1	3.1	...	0
12/09/2025 12:00:20	2.6	3.1	...	0
12/09/2025 12:00:30	2.6	3.2	...	0
12/09/2025 12:00:40	2.7	3.3	...	0
12/09/2025 12:00:50	2.7	3.3	...	0
12/09/2025 12:01:00	2.8	3.4	...	0
12/09/2025 12:01:10	2.9	3.5	...	0
12/09/2025 12:01:20	3.0	3.6	...	0
...	...	...	...	...

Fonte: Produção do próprio autor.

Figura 2 – Diagrama da geração do conjunto de dados B



Fonte: Produção do próprio autor.

Qualidade do conjunto de dados. O conjunto de dados original (Conjunto de Dados A) foi composto por 1030 linhas coletadas a partir do ambiente de simulação. Esse *dataset* não é balanceado, pois contém 760 amostras com o sistema funcionando corretamente (target = 1) e 270 amostras em estado de falha (target = 0). Apesar do desbalanceamento, essa distribuição é coerente com cenários reais, nos quais falhas tendem a ocorrer com menor frequência em comparação aos períodos de funcionamento normal. A duração total de coleta foi de 2 horas e 51 minutos.

Durante a geração do *dataset*, houve uma preocupação em induzir diferentes tipos de falhas no sistema, uma vez que falhas distintas podem produzir comportamentos e métricas diferentes. Dessa forma, buscou-se garantir maior diversidade nos dados coletados, permitindo que os modelos de aprendizado de máquina aprendessem padrões variados de degradação e indisponibilidade do sistema a partir de diferentes combinações de métricas de observabilidade. O conjunto de dados foi gerado a partir de 15 eventos de falha distintos (falhas únicas) induzidos no ambiente experimental. Cada evento de falha produziu múltiplas amostras consecutivas rotuladas como target = 0, resultando em 270 registros classificados como falha no conjunto de dados A.

Para isso, foram desenvolvidos ao todo cinco *scripts* de geração de carga e falhas.

Quatro deles foram implementados utilizando a ferramenta Grafana K6, cada um simulando comportamentos específicos no ambiente monitorado. Entre esses *scripts*, um induzia uso mais intensivo do banco de dados MariaDB, outro gerava maior carga sobre o WordPress por meio de requisições HTTP e operações da aplicação, enquanto os demais exploravam diferentes padrões de acesso e estresse no sistema. Além disso, foi utilizado um quinto *script* baseado na ferramenta stress-ng[9], cujo objetivo era afetar diretamente os recursos do sistema operacional hospedeiro da aplicação, gerando sobrecarga de CPU, memória e disco. Com isso, o conjunto de dados passou a representar diferentes cenários de degradação, tornando o treinamento dos modelos mais robusto e próximo de situações reais.

A Parte I do trabalho permitiu a criação de um ambiente reprodutível para geração de *datasets* de observabilidade com falhas controladas. A integração entre monitoramento, geração de carga e rotulagem automática possibilitou obter dados adequados para o treinamento dos modelos preditivos. Na parte II, os conjuntos de dados A e B serão utilizados para treinar diferentes modelos de aprendizado de máquina.

4.2 Criação e Avaliação dos Modelos de Aprendizado de Máquina

A segunda etapa do trabalho concentrou-se na criação e avaliação de modelos de aprendizado de máquina capazes de identificar falhas em sistemas distribuídos a partir de métricas de observabilidade. Todo o processamento foi desenvolvido em Python, utilizando bibliotecas como pandas, NumPy, scikit-learn e TensorFlow/Keras.

Foram avaliados quatro modelos distintos: XGBoost, LightGBM, CatBoost e LSTM. Os experimentos foram executados duas vezes, utilizando os mesmos algoritmos e o mesmo fluxo experimental, porém com conjuntos de dados diferentes (A e B), permitindo comparar a capacidade dos modelos em detectar falhas já ocorrendo e em antecipar falhas futuras.

O conjunto de dados A representa um cenário de detecção de falhas. Nesse conjunto, o rótulo indica diretamente se o sistema está funcionando corretamente ou se já se encontra em estado de falha no instante analisado. Assim, os modelos aprendem a identificar padrões associados à indisponibilidade do sistema. Já o conjunto de dados B representa um cenário de predição de falhas. Conforme descrito na Seção 4.1, foi aplicada uma estratégia de *relabeling* temporal, na qual as amostras localizadas até um minuto antes de cada falha também foram rotuladas como falha. Dessa forma, os modelos deixam de apenas identificar um sistema já indisponível e passam a aprender padrões que antecedem a ocorrência da falha, permitindo uma atuação preventiva.

A divisão dos conjuntos foi realizada preservando a ordem temporal dos dados, utilizando os primeiros 70% das observações para treinamento e os 30% finais para teste, evitando vazamento temporal. Essa ordem da divisão é especialmente importante no caso do LSTM, pois o modelo aprende dependências sequenciais. Os dados de treinamento foram utilizados para construção dos modelos, enquanto os dados de teste permaneceram isolados até a etapa final de avaliação, permitindo medir a capacidade de generalização em dados nunca vistos anteriormente. Todos os modelos utilizaram a mesma divisão temporal inicial dos dados (70% para treinamento e 30% para teste). Dessa forma, os quatro algoritmos foram avaliados exatamente sobre o mesmo conjunto de teste, garantindo uma comparação justa de desempenho. A diferença está apenas na estratégia de validação aplicada ao conjunto de treinamento: validação cruzada para os modelos de Gradient Boosting e conjunto de validação separado para a LSTM.

[9] stress-ng: <https://wiki.ubuntu.com/Kernel/Reference/stress-ng>

Antes do treinamento, foi realizado um pré-processamento dos dados. Valores ausentes foram tratados quando necessário, e as métricas numéricas foram organizadas em formato tabular para os modelos baseados em Gradient Boosting. No caso da LSTM, os dados foram preparados como séries temporais. Inicialmente, as métricas foram ordenadas cronologicamente e normalizadas utilizando o método *StandardScaler* (para os dados de treinamento). Em seguida, foram construídas sequências temporais (após a divisão dos dados) contendo os 6 instantes mais recentes de observação. Considerando que as métricas eram coletadas a cada 10 segundos pelo Prometheus, a janela de tamanho 6 representa aproximadamente 1 minuto de histórico permitindo que a rede neural utilizasse não apenas o estado atual do sistema, mas também a evolução recente das métricas para realizar suas previsões.

Os modelos XGBoost, LightGBM e CatBoost foram treinados utilizando classificação supervisionada binária. Esses algoritmos foram escolhidos por apresentarem elevado desempenho em problemas envolvendo dados tabulares e métricas numéricas. Já o modelo LSTM foi implementado utilizando TensorFlow/Keras e uma arquitetura composta por duas camadas LSTM empilhadas, contendo 64 e 32 neurônios, respectivamente, seguidas por camadas de *dropout* para reduzir *overfitting* e uma camada densa responsável pela classificação final. A saída da rede utiliza função de ativação *sigmoid* para estimar a probabilidade de ocorrência de falha.

Durante os experimentos, também foram realizados ajustes básicos de hiperparâmetros, buscando melhorar o desempenho e reduzir problemas de *overfitting*. Entre os parâmetros avaliados estavam profundidade das árvores, taxa de aprendizado e número de estimadores nos modelos de Gradient Boosting. Para a LSTM, foram avaliados parâmetros como quantidade de neurônios, número de épocas de treinamento, tamanho das sequências temporais e técnicas de regularização. Além disso, foram utilizadas estratégias de *Early Stopping* e *Reduce Learning Rate*, interrompendo automaticamente o treinamento quando não havia melhoria no conjunto de validação.

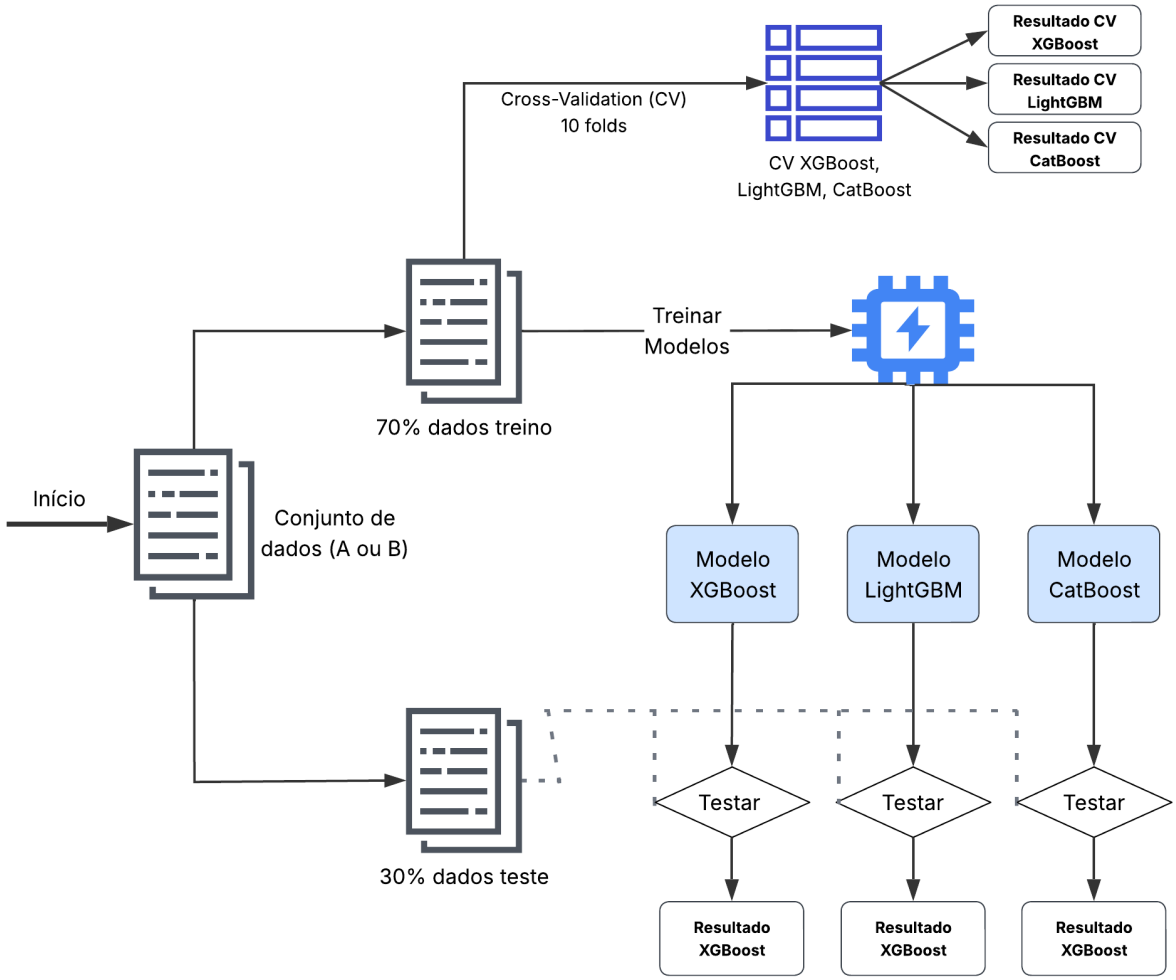
A Figura 3 apresenta o fluxo de avaliação adotado para os modelos de *Gradient Boosting*. Inicialmente, o conjunto de dados é dividido em treino (70%) e teste (30%). Em seguida, os dados de treino são submetidos a uma validação cruzada (cross-validation) com 10 folds. Nessa técnica, o conjunto de treinamento é dividido em dez subconjuntos de tamanho semelhante. Em cada iteração, nove subconjuntos são utilizados para treinamento e um para validação, repetindo-se o processo até que todos os subconjuntos tenham sido utilizados como validação uma vez. Ao final, calcula-se a média dos resultados obtidos pelos modelos XGBoost, LightGBM e CatBoost, permitindo avaliar sua estabilidade e reduzir a influência de uma divisão específica dos dados.

Cabe destacar que a validação cruzada tradicional com 10 folds não preserva a ordem temporal das observações. No Conjunto de Dados B, após o relabeling de 1 minuto antes da falha, amostras temporalmente próximas podem ser distribuídas entre treino e validação, gerando um possível vazamento temporal implícito e estimativas otimistas de desempenho. Por esse motivo, os resultados obtidos no conjunto de teste temporalmente separado são considerados a principal medida de generalização dos modelos.

Após a etapa de validação cruzada, os mesmos dados de treinamento são utilizados para gerar os modelos finais. Esses modelos são então avaliados utilizando exclusivamente o conjunto de teste, composto por dados que não participaram de nenhuma etapa de treinamento ou validação.

O modelo LSTM seguiu um fluxo diferente e não utilizou validação cruzada. Como

Figura 3 – Processo de avaliação dos modelos de *Gradient Boosting*



Fonte: Produção do próprio autor.

redes neurais demandam maior custo computacional de treinamento, optou-se por uma divisão temporal dos dados em treino (70%), validação (15%) e teste (15%). Nesse processo, a rede foi treinada utilizando apenas os dados de treino, enquanto o conjunto de validação foi utilizado para monitorar o aprendizado, ajustar o limiar de classificação e controlar o *overfitting*. Somente após a conclusão do treinamento o modelo foi avaliado no conjunto de teste final. Dessa forma, os resultados de validação cruzada apresentados neste trabalho referem-se exclusivamente aos modelos XGBoost, LightGBM e CatBoost.

Como resultado, o trabalho produz quatro grupos de resultados distintos. Para o conjunto de dados A, são obtidos os resultados da validação cruzada (10 folds) e os resultados finais dos modelos avaliados no conjunto de teste. Da mesma forma, para o conjunto de dados B, são obtidos os resultados da validação cruzada e os resultados finais em dados de teste. Essa organização permite analisar separadamente a capacidade dos algoritmos em detectar falhas já ocorrendo (conjunto A) e em prever falhas antes de sua ocorrência (conjunto B), além de comparar a estabilidade dos modelos durante o treinamento e seu desempenho final em dados inéditos.

5 RESULTADOS

Os resultados obtidos demonstram que as métricas de observabilidade coletadas foram capazes de capturar padrões associados ao comportamento operacional do sistema, permitindo a construção de modelos preditivos com elevado desempenho. Conforme descrito na metodologia, os experimentos foram realizados sobre dois conjuntos de dados distintos: o Conjunto A, voltado para detecção de falhas já ocorrendo, e o Conjunto B, voltado para predição antecipada de falhas por meio da estratégia de *relabeling* temporal.

Durante o treinamento, foi utilizada a validação cruzada com 10 folds para avaliar a estabilidade dos modelos. A Tabela 3 mostra a validação cruzada com 10 folds nos algoritmos baseados em Gradient Boosting no conjunto de dados A.

Tabela 3 – Resultados da validação cruzada (10 folds) - Conjunto de Dados A

Modelo	**F1-score médio**	**Desvio padrão**
XGBoost	0,9864	0,0078
LightGBM	0,9890	0,0085
CatBoost	0,9873	0,0078

Fonte: Produção do próprio autor.

Os resultados da validação cruzada indicam elevada estabilidade dos modelos. O LightGBM apresentou a melhor média de F1-score, seguido por CatBoost e XGBoost. A Tabela 4 apresenta os resultados finais obtidos no conjunto de teste para o Conjunto A.

Tabela 4 – Resultados finais em dados de teste (Conjunto de Dados A)

Modelo	**F1-score**
XGBoost	0,9191
LightGBM	0,9393
CatBoost	0,9390
LSTM	0,8946

Fonte: Produção do próprio autor.

Os resultados mostram uma redução do F1-score entre validação cruzada e teste final, indicando maior dificuldade de generalização em dados completamente inéditos. As causas podem ter sido *overfitting* (o modelo "decorou"os dados de treinamento), e também a diferença na distribuição dos dados de treino e teste, visto que foi respeitada a ordem temporal, sendo os primeiros 70% treino e os últimos 30% teste. Apesar disso, os modelos baseados em Gradient Boosting apresentaram desempenho superior ao modelo LSTM. Entre eles, o LightGBM obteve o melhor resultado geral, alcançando F1-score de 0,9393 no conjunto A. O CatBoost apresentou desempenho muito próximo ao LightGBM, enquanto o XGBoost também demonstrou resultados consistentes, porém ligeiramente inferiores.

Apesar da capacidade do LSTM de capturar dependências temporais, o desempenho inferior pode estar relacionado ao tamanho relativamente reduzido do *dataset*. Redes neurais profundas geralmente necessitam de grandes volumes de dados para atingir seu potencial máximo. Além disso, os padrões de falha presentes nas métricas puderam ser capturados de forma eficiente pelos modelos baseados em árvores.

A Tabela 5 apresenta os resultados da validação cruzada para o conjunto B, que representa o cenário de predição antecipada de falhas.

Tabela 5 – Resultados da validação cruzada (10 folds) - Conjunto de Dados B

Modelo	**F1-score médio**	**Desvio padrão**
XGBoost	0,9899	0,0077
LightGBM	0,9880	0,0093
CatBoost	0,9844	0,0115

Fonte: Produção do próprio autor.

De maneira semelhante ao observado no Conjunto A, os modelos apresentaram elevada estabilidade durante a validação cruzada. O XGBoost obteve a maior média de F1-score, embora as diferenças para LightGBM tenham sido pequenas. Entretanto, ao analisar os resultados em dados de teste, o LightGBM apresentou melhor capacidade de generalização, alcançando o maior desempenho final. Em seguida, na Tabela 6 são apresentados os resultados obtidos para o Conjunto de Dados B.

Tabela 6 – Resultados finais em dados de teste (Conjunto de Dados B)

Modelo	**f1-score**
XGBoost	0,8949
LightGBM	0,9037
CatBoost	0,9031
LSTM	0,8838

Fonte: Produção do próprio autor.

Os resultados mostram que o problema de predição antecipada de falhas é mais desafiador do que a simples detecção de falhas já ocorrendo. Ainda assim, os modelos LightGBM e CatBoost mantiveram desempenho superior a 0,90 de F1-score, demonstrando capacidade de identificar sinais precursores de degradação do sistema antes da ocorrência efetiva da indisponibilidade.

A diferença entre os resultados da validação cruzada e do conjunto de teste pode ser explicada, em parte, por um possível vazamento temporal, pois a validação cruzada tradicional não preserva a estrutura temporal dos dados. No Conjunto B, amostras próximas no tempo compartilham padrões associados ao mesmo evento de falha, o que pode ter tornado os resultados de validação mais otimistas do que aqueles observados em dados futuros e completamente inéditos.

Comparando os dois cenários experimentais, verifica-se que os resultados do Conjunto B foram ligeiramente inferiores aos do Conjunto A. Esse comportamento era esperado, pois a estratégia de *relabeling* temporal transforma o problema em uma tarefa de predição antecipada, exigindo que os modelos reconheçam sinais sutis de degradação antes da ocorrência efetiva da falha. Apesar desse aumento de complexidade, os modelos baseados em Gradient Boosting mantiveram desempenho elevado, reforçando a viabilidade da abordagem proposta.

Por fim, observa-se que os modelos LSTM apresentaram desempenho inferior aos algoritmos baseados em árvores em ambos os conjuntos de dados. Uma possível

explicação é o tamanho relativamente reduzido do *dataset*, que pode limitar a capacidade de generalização de redes neurais profundas. Os resultados sugerem que, para o contexto avaliado, modelos de Gradient Boosting são mais adequados para a utilização de métricas de observabilidade na detecção e predição de falhas.

5.1 Discussão das Questões de Pesquisa

Nesta seção, são discutidas as questões de pesquisa (RQs) definidas neste trabalho com base nos resultados experimentais obtidos. A análise permite avaliar a eficácia da abordagem proposta e responder aos objetivos estabelecidos inicialmente.

RQ1. É tecnicamente viável e reprodutível utilizar dados de observabilidade para prever falhas em sistemas por meio de aprendizado de máquina supervisionado com alta performance (F1-score > 0.9)?

Sim. Os resultados mostraram que os modelos LightGBM e CatBoost alcançaram F1-score superior a 0,9 nos conjuntos de teste, demonstrando que métricas de observabilidade podem ser utilizadas com sucesso para predição de falhas. Além disso, toda a arquitetura desenvolvida utilizou ferramentas *open source* e *scripts* reproduzíveis, permitindo replicação do experimento.

RQ2. Qual modelo de aprendizado de máquina supervisionado (LightGBM, CatBoost, XGBoost ou LSTM) apresenta o melhor desempenho preditivo para o sistema WordPress/MariaDB?

O modelo LightGBM apresentou o melhor desempenho geral. Ele obteve os maiores valores de F1-score tanto no conjunto de dados A (0,9393) quanto no conjunto B (0,9037), superando os demais modelos avaliados.

RQ3. Modelos temporais, como LSTM, apresentam ganho significativo de desempenho em comparação aos modelos de Gradient Boosting?

Apesar da capacidade das redes LSTM de capturar dependências temporais, os resultados mostraram que os modelos baseados em Gradient Boosting apresentaram desempenho superior em ambos os conjuntos de dados. Isso sugere que, para o contexto e volume de dados utilizados neste trabalho, modelos tradicionais baseados em árvores foram mais eficazes do que redes neurais profundas.

6 CONSIDERAÇÕES FINAIS

Este trabalho apresentou uma abordagem para detecção preditiva de falhas em sistemas distribuídos utilizando métricas de observabilidade e aprendizado de máquina supervisionado. Os resultados demonstraram que é tecnicamente viável utilizar dados de observabilidade para prever falhas com alto desempenho, alcançando F1-score superior a 0,9 em diferentes cenários experimentais.

Uma das principais contribuições do trabalho foi a construção de um ambiente reprodutível de geração de dados, utilizando exclusivamente ferramentas *open source*, como Kubernetes, Prometheus, Grafana K6 e *scripts* personalizados em Python e JavaScript. Essa arquitetura permitiu automatizar tanto a geração de carga quanto a coleta de métricas e rotulagem dos dados, tornando o processo replicável em ambientes acadêmicos e de pesquisa.

Outro ponto importante foi a utilização da estratégia de *relabeling* temporal para pré-falha no conjunto de dados B. Essa abordagem permitiu que os modelos aprendessem padrões anteriores à indisponibilidade completa do sistema, aproximando o problema de um cenário real de predição antecipada de falhas.

Os resultados também mostraram que os modelos tradicionais baseados em Gradient Boosting (LightGBM, CatBoost e XGBoost) apresentaram desempenho superior ao modelo temporal LSTM no contexto avaliado. Isso sugere que, para *datasets* menores e métricas estruturadas de observabilidade, modelos baseados em árvores podem ser mais eficientes e mais simples de treinar do que redes neurais profundas.

Durante o desenvolvimento do trabalho, observou-se que a etapa mais complexa não foi a criação dos modelos de aprendizado de máquina, mas sim a geração e preparação do *dataset*. A construção de um ambiente capaz de induzir diferentes tipos de falhas, coletar métricas corretamente e produzir dados consistentes exigiu maior esforço experimental. Além disso, o ajuste de hiperparâmetros dos modelos também se mostrou uma tarefa desafiadora, devido à necessidade de equilibrar desempenho e capacidade de generalização.

Por fim, é importante destacar que os modelos desenvolvidos não identificam exatamente onde está a falha no sistema, mas apenas indicam a probabilidade de que o sistema esteja próximo de falhar. Dessa forma, os modelos atuam como mecanismos de alerta preditivo, auxiliando administradores e equipes de operação a agir preventivamente antes da indisponibilidade completa do serviço.

Como trabalhos futuros, pretende-se realizar a análise de interpretabilidade, erros de classificação, explorar *datasets* maiores, incluir *logs* e *traces* como fontes adicionais de dados de observabilidade, além de investigar arquiteturas mais avançadas de *deep learning* e técnicas multimodais para melhorar ainda mais a capacidade preditiva dos modelos.

Referências

Chen, Tianqi; Guestrin, Carlos. Xgboost: A scalable tree boosting system. In: **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [S.l.: s.n.], 2016. p. 785–794. Citado na página 5.

Cheng, Qian; Sahoo, Doyen; Saha, Amrita; Yang, Wenzhuo; Liu, Chenghao; Woo, Gerald; Singh, Manpreet; Saverese, Silvio; Hoi, Steven C. H. Ai for it operations (aiops) on cloud platforms: Reviews, opportunities and challenges. **arXiv**, v. 2304.04661, 2023. Citado 2 vezes nas páginas 4 e 6.

Cloud Native Computing Foundation. **OpenTelemetry Documentation**. 2025. Documentação técnica. Disponível em: <https://opentelemetry.io/docs/>. Acesso em: 19 junho 2026. Citado na página 3.

Datadog. **How microservice architectures have shaped the usage of database technologies**. 2024. Relatório de pesquisa. Disponível em: <https://www.datadoghq.com/blog/datadog-database-research/>. Acesso em: 19 junho 2026. Citado 2 vezes nas páginas 2 e 3.

Dorogush, Leo; Ershov, Vasily; Gulin, Andrey. Catboost: Gradient boosting with categorical features support. **arXiv**, v. 1810.11363, 2018. Citado na página 5.

Dragoni, Nicola; Giallorenzo, Saverio; Lafuente, Alberto; Mazzara, Manuel; Montesi, Fabrizio; Mustafin, Ruslan; Safina, Larisa. Microservices: Yesterday, today, and tomorrow. **Present and Ulterior Software Engineering**, Springer, p. 195–216, 2017. Citado na página 3.

Du, Min; Li, Feifei; Zheng, Guineng; Srikumar, Vivek. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: **Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security**. Dallas, TX, USA: Association for Computing Machinery, 2017. p. 1285–1298. Citado na página 7.

Friedman, Jerome H. Greedy function approximation: A gradient boosting machine. **The Annals of Statistics**, v. 29, n. 5, p. 1189–1232, 2001. Citado na página 4.

Hadadi, Fatemeh; Dawes, Joshua H.; Shin, Donghwan; Bianculli, Domenico; Briand, Lionel. Systematic evaluation of deep learning models for log-based failure prediction. **Empirical Software Engineering**, Springer Nature, v. 29, n. 105, p. 1–53, 2024. Citado na página 6.

Hallur, Jayanna. From monitoring to observability: Enhancing system reliability and team productivity. **International Journal of Science and Research**, v. 13, n. 10, p. 602–606, 2024. Citado na página 1.

Hochreiter, Sepp; Schmidhuber, Jürgen. Long short-term memory. **Neural Computation**, v. 9, n. 8, p. 1735–1780, 1997. Citado na página 5.

Homyamyen, Patipol; Kulachai, Waiphot; Benchakhan, Khwanta; Wannarak, Junphen. Decoding digital loyalty: How service quality, platform performance, and menu diversity shape trust, satisfaction, and retention in online platforms. **Journal of Emerging Mobile and Internet Technologies**, v. 2, n. 4, p. 215–226, 2024. Citado na página 1.

IBM. O que é observabilidade? **IBM Think**, 2025. Disponível em: <https://www.ibm.com/br-pt/think/topics/observability>. Acesso em: 19 junho 2026. Citado 2 vezes nas páginas 1 e 3.

Ke, Guolin; Meng, Qi; Finley, Thomas; Wang, Taifeng; Chen, Wei; Ma, Weidong; Ye, Qiwei; Liu, Tie-Yan. Lightgbm: A highly efficient gradient boosting decision tree. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2017. v. 30. Citado na página 5.

Kim, Mi Ok; Coiera, Enrico; Magrabi, Farah. Problems with health information technology and their effects on care delivery and patient outcomes: a systematic review. **Journal of the American Medical Informatics Association**, Oxford University Press, v. 24, n. 2, p. 246–260, 2017. Citado na página 1.

Linux Foundation. **Prometheus: Monitoring system and time series database**. 2025. Documentação técnica. Disponível em: <https://prometheus.io/docs/introduction/overview/>. Acesso em: 19 junho 2026. Citado na página 4.

Mitchell, Tom M. **Machine Learning**. New York: McGraw-Hill, 1997. Citado 2 vezes nas páginas 4 e 5.

New Relic. **New Relic Report Reveals High-Business Impact Outages Cost Financial Services and Insurance Sectors $2.2 Million Per Hour**. 2025. Press Release. Disponível em: <https://newrelic.com/press-release/20250429-0>. Acesso em: 19 junho 2026. Citado na página 1.

Sridharan, Cindy. **Distributed Systems Observability**. 2018. Livro eletrônico. Disponível em: <https://www.oreilly.com/library/view/distributed-systems-observability/9781492033431/>. Acesso em: 19 junho 2026. Citado na página 3.

Tanenbaum, Andrew S.; Van Steen, Maarten. **Distributed Systems: Principles and Paradigms**. 3. ed. [S.l.]: CreateSpace Independent Publishing Platform, 2017. Citado na página 2.

Vollem, Shekar. From reactive alerts to predictive intelligence: Ai-assisted monitoring in modern cloud environments. **International Journal of Research and Applied Innovations**, v. 6, n. 1, p. 8337–8345, 2023. Citado na página 1.

Yèche, Hugo; others. Labeling strategies for event prediction in time series. In: **Proceedings of Machine Learning Research**. [S.l.]: PMLR, 2023. v. 202. Citado 2 vezes nas páginas 7 e 10.

Zhao, Chenyu; Ma, Minghua; Zhong, Zhenyu; Zhang, Shenglin; Tan, Zhiyuan; Xiong, Xiao; Yu, Lulu; Feng, Jiayi; Sun, Yongqian; Zhang, Yuzhi; Pei, Dan; Lin, Qingwei; Zhang, Dongmei. Robust multimodal failure detection for microservice systems. In: **Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining**. Long Beach, CA, USA: Association for Computing Machinery, 2023. p. 5639–5649. Citado na página 6.

MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO - NG

FOLHA DE APROVAÇÃO DE TCC Nº 10 / 2026 - DECOM (11.56.03)

Nº do Protocolo: 23062.031822/2026-51

Belo Horizonte-MG, 19 de junho de 2026.

GUILHERME DE ASSIS LIMA

OBSERVABILIDADE DE SISTEMAS: DETECÇÃO PREDITIVA DE FALHAS

Trabalho de Conclusão de Curso apresentado ao Curso de **Engenharia de Computação** do Centro Federal de Educação Tecnológica de Minas Gerais, do Campus **Campus Nova Gameleira**, como parte do requisito para obtenção do grau de Bacharel em **Engenheiro de Computação**.

Aprovado em 19 de junho de 2026.

Banca Examinadora:

Prof. Dr. Danilo Boechat Seufitelli - Orientador
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dra. Mariana de Oliveira Santos Silva
Pontifícia Universidade Católica de Minas Gerais

Prof. Dr. Fábio Rocha da Silva
Centro Federal de Educação Tecnológica de Minas Gerais

Belo Horizonte
2026

(Assinado digitalmente em 23/06/2026 10:48)
DANILO BOECHAT SEUFITELLI
PROFESSOR ENS BASICO TECN TECNOLOGICO
DECOM (11.56.03)
Matrícula: 1371545

(Assinado digitalmente em 19/06/2026 17:20)
FABIO ROCHA DA SILVA
PROFESSOR ENS BASICO TECN TECNOLOGICO
DECOM (11.56.03)
Matrícula: 1703246

(Assinado digitalmente em 23/06/2026 11:08)
MARIANA DE OLIVEIRA SANTOS SILVA
ASSINANTE EXTERNO
CPF: ###.###.716-##

Visualize o documento original em **https://sig.cefetmg.br/public/documentos/index.jsp** informando seu número: **10**, ano: **2026**, tipo: **FOLHA DE APROVAÇÃO DE TCC**, data de emissão: **19/06/2026** e o código de verificação: **e485679993**