Universidade Federal Rural do Semiárido – UFERSA
Centro de Engenharias – CE
Departamento de Engenharia e Tecnologia – DET
Curso de Engenharia Mecânica

Artigo

Detecção De Anomalias Em Rolamentos De Eixo Rápido Utilizando Transformadas Wavelet Para Manutenção Preditiva

Antonio Lucas Dos Santos Carlos [1], Rômulo Pierre Batista dos Reis [2]

[1] Universidade, Bacharelado em engenharia mêcanica; Antonio Lucas Dos Santos Carlos; lucas-_-7@outlook.com

[2] Universidade, curso; Dr. Rômulo Pierre Batista dos Reis; romulopierre@ufersa.edu.br

Data da defesa: 27/03/2025;

Resumo: Este estudo propõe uma metodologia para a detecção de falhas em rolamentos utilizando sinais de vibração coletados em um sistema mecânico real. Em virtude da natureza não rotulada dos dados, foi empregado um modelo baseado em autoencoders LSTM (Long Short Term Memory), combinado à transformada wavelet scattering, técnica que possibilita a extração de representações robustas dos sinais vibratórios. A implementação da metodologia foi realizada em Python, adaptando-se uma referência originalmente desenvolvida em MATLAB. Os sinais foram segmentados em janelas temporais de 1,04 segundo com sobreposição, normalizados e subsequentemente processados pelo modelo. Para fins de treinamento, utilizaram-se dados coletados após a reparação de um dano na pista interna do rolamento, enquanto os dados pré-reparo foram reservados para teste. A métrica adotada para a identificação de anomalias foi o erro de reconstrução (Mean Absolute Error – MAE), com um limiar de detecção estabelecido com base no terceiro quartil acrescido de 1,5 vezes o intervalo interquartil (Interquartile Range – IQR) dos erros de validação. Os resultados obtidos demonstraram a eficácia do modelo na identificação de anomalias, bem como na definição de um limiar adequado para sinalização de alarmes de falha. O método mostrou-se capaz de discriminar entre o estado de normalidade e o estado de falha do componente analisado, oferecendo assim uma solução acessível e eficiente para aplicações em manutenção preditiva.

Palavras-chave: Wavelet; LSTM autoencoder; manutenção preditiva; detecção de falhas; Python

---

.

1. INTRODUÇÃO

A manutenção preditiva é essencial em sistemas rotativos críticos, e a análise de vibração é uma técnica eficaz para identificar falhas antes que causem paradas inesperadas, o que a torna uma das principais ferramentas para análise de manutenção preventiva [1]. Em muitos casos, os dados coletados não têm rótulos que indiquem falhas, o que dificulta o uso de métodos supervisionados.

Modelos não supervisionados, como autoencoders LSTM (Long Short Term Memory), têm se mostrado úteis por aprenderem o comportamento normal do sistema e detectarem desvios. A combinação com a transformada wavelet scattering melhora ainda mais a representação dos sinais durante sua etapa de pré-processamento, tornando o processo mais robusto [2]. A literatura já apresenta o uso e combinação de transformadas wavelet com redes neurais para analisar sinais de máquinas em funcionamento. Essa combinação melhorou muito a capacidade de identificar problemas antes que eles aconteçam, deixando as máquinas mais seguras e confiáveis [3].

Este trabalho adapta uma metodologia originalmente desenvolvida em MATLAB [4] para Python, utilizando ferramentas e bibliotecas open-source, e aplica esse modelo ao diagnóstico de falhas na pista interna do rolamento do eixo rápido de uma caixa de engrenagens. Usando dados reais de 100 dias, buscamos mostrar que o método identifica anomalias mesmo sem rótulos prévios.

A estrutura deste artigo é: Seção 2 descreve os dados e métodos; Seção 3 apresenta os resultados; Seção 4 discute as conclusões.

2. MATERIAIS E MÉTODOS

2.1. Dados de vibração e contextualização

Os dados utilizados neste estudo consistem em sinais de vibração adquiridos por um sistema de monitoramento SMP12C, com sensor de aceleração instalado radialmente em um rolamento localizado no eixo rápido, na saída, de uma caixa de engrenagens com dois estágios planetários conforme Figura 1. A coleta foi realizada diariamente, sempre que a máquina atingia sua velocidade nominal de operação. Nesse momento, registrava-se o sinal de aceleração por um período de 2,08 segundos, com uma taxa de amostragem de 25.600 Hz, o que resultava em um total de 53.248 amostras por dia.

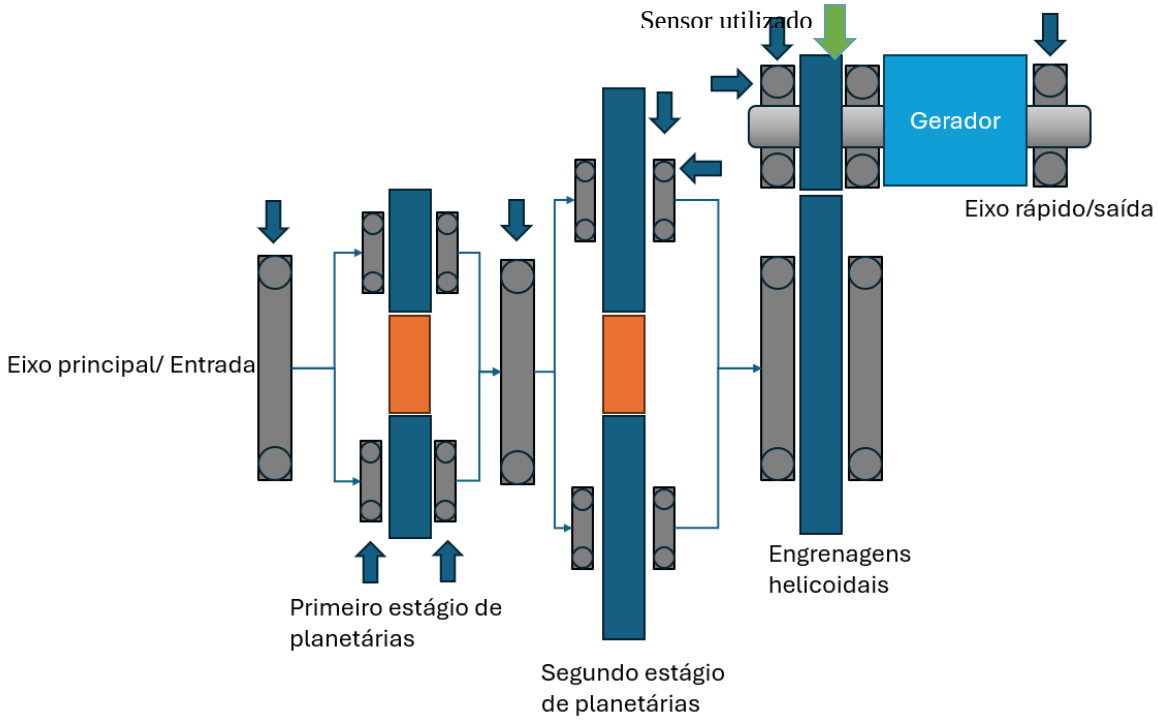

Figura 1 Esquema de posições dos sensores (Autoria própria)

Ao longo de um período total de 100 dias consecutivos, a máquina foi monitorada diariamente, totalizando 100 séries temporais distintas de dados de vibração. Essas séries foram utilizadas com o objetivo de detectar anomalias que pudessem indicar uma deterioração no estado operacional do equipamento, e ajudar na criação de um alarme ou limite que ajude na identificação precoce de falhas no rolamento do eixo rápido. Tal abordagem visa contribuir para a manutenção preditiva, prevenindo falhas catastróficas e otimizando a disponibilidade do sistema.

2.2 Bibliotecas utilizadas

- NumPy: Biblioteca fundamental para computação numérica com arrays e operações vetoriais. Usada para manipulação de vetores e arrays (np.array, np.stack), estatísticas (np.quantile, np.mean, np.sum), cálculos de limiar. [5]
- Pandas: Biblioteca para manipulação e análise de dados tabulares (tipo Excel/CSV). Foi utilizada para manipulação de dados e leitura de arquivos .csv (pd.read_csv), conversão para arrays NumPy (.values).[6]
- PyTorch: Estrutura de aprendizado profundo para construção e treinamento de redes neurais. Foi utilizada para definição de modelos (nn.Module, nn.LSTM), funções de perda (nn.L1Loss), otimização (optim.Adam), manipulação de tensores (torch.stack, torch.tensor, torch.save/load).[7]
- Scikit-learn: Biblioteca de aprendizado de máquina com ferramentas para pré-processamento, classificação, regressão e mais. Serviu para fazer normalização z-score dos dados com a sua função StandardScaler.[8]
- Matplotlib: Biblioteca de visualização para criação de gráficos e plots. Foi utilizada para criar os gráficos ev isualização do erro MAE (plt.plot, plt.axhline, plt.axvspan), destaque de faixas (teste/validação/treino), legendas e layout. [9]
- Kymatio: Biblioteca para aplicar scattering transform (baseado em wavelets) em 1D, 2D e 3D, usada para extração de características robustas. Foi usada no pré-processamento de dados para construção da rede Scattering1D e extração dos coeficientes de scattering para cada janela de sinal.[10]

2.3. Pré-processamento dos dados

2.3.1 Análise espectral com MODWPT

Esta etapa visa verificar se a energia dos sinais de vibração está majoritariamente concentrada em faixas de baixa frequência, o que possibilitaria a aplicação de técnicas de downsampling, reamostragem tentando reduzir a quantidade total de dados, com mínima perda de informação [4]. Para isso, foi utilizada a MODWPT (Transformada Wavelet discreta de sobreposição máxima), que realiza a decomposição do sinal em bandas de frequência de largura constante ($Fs/2^5$), mantendo o alinhamento temporal, onde Fs é frequência de aquisição de dados.

O procedimento seguiu os seguintes passos:

- Leitura dos sinais de um arquivo .csv;
- Aplicação da MODWPT com a wavelet sym4;
- Cálculo da energia relativa por faixa de frequência;
- Determinação da porcentagem de energia abaixo de Fs/4 (6.400 Hz);
- Identificação dos valores mínimo e máximo dessa métrica entre todos os registros.

Caso fosse comprovado que mais de 95% da energia está abaixo de Fs/4, a redução da taxa de amostragem torna-se justificável, otimizando as etapas subsequentes de processamento e modelagem [11].

A wavelet Symlet foi selecionada por sua simetria superior, que minimiza distorções e aumenta a precisão da análise espectral. A função MODWPT (transformação wavelet em pacotes discretos com sobreposição máxima), foi utilizada com a família de wavelet Symlet que fornece resolução de frequência que facilita a extração de características significativas e essenciais para a identificação precoce de falhas [12]. Essa capacidade foi evidenciada em estudos comparativos conduzidos por Strömbergsson et al (2011), que explorou cenários de falhas em rolamentos de eixo de alta velocidade, comparando sistematicamente diferentes famílias de wavelets com técnicas de transformada rápida de Fourier (FFT) [13].

2.3.2 Segmentação temporal e extração de características via wavelet scattering

A etapa seguinte consistiu na segmentação dos sinais em janelas de 1,04 segundo com sobreposição de 50% (0,52 s), totalizando 200 janelas por dia. Essa chamada de janela móvel (sliding window) é aplicada ao sinal de vibração para segmentá-lo em quadros sobrepostos. Cada quadro é assumido como aproximadamente estacionário

e é processado individualmente para extrair características ou realizar diagnósticos. A sobreposição entre as janelas garante que eventos transitórios não sejam perdidos nas fronteiras entre os quadros [14].

Cada janela foi submetida à transformada wavelet scattering utilizando a biblioteca Kymatio. Essa técnica gera representações invariantes a pequenas deformações temporais e robustas a ruído. Os parâmetros empregados foram:

- Frequência de amostragem: 25.600 Hz;
- Escala de invariância: 0,2 s;
- Filtros por oitava: Q = [4,1];
- Ordem máxima dos coeficientes: 2;

A função Scattering1D, desta biblioteca, produziu coeficientes multidimensionais que descrevem modulações de energia em diferentes escalas e ordens. O coeficiente de ordem zero foi descartado por conter apenas a média do sinal. O resultado foi um conjunto de vetores de características capazes de diferenciar entre condições normais e condições em estado de falha.

A separação dos conjuntos de dados foi feita tendo conhecimento prévio de que a máquina em questão passou por um processo de reparo. Sabendo que o equipamento estava operando em boas condições após essa intervenção, os dados dos últimos dias do período de monitoramento foram considerados representativos do estado saudável da máquina.

Assim, a divisão foi feita de forma retroativa "voltando no tempo", com o objetivo de demonstrar o funcionamento do método a partir de exemplos conhecidos de normalidade:

- Treinamento: dias 71–100 (60 janelas): período pós-reparo com condição estável;
- Validação: dias 51–70 (20 janelas): 10 dias anteriores ao treinamento, também em condição estável;
- Teste: dias 1–50 (100 janelas): primeiros 50 dias, período em que a máquina apresentava falha progressiva, especificamente uma degradação na pista interna do rolamento do eixo rápido.

Essa abordagem permite treinar o modelo com exemplos saudáveis e testá-lo em um cenário conhecido de falha, permitindo validar sua capacidade de identificar anomalias com base em desvios em relação ao comportamento aprendido. Essa divisão pode ser observada conforme mostrado na Figura 2, que os primeiros dias refletem um estado de degradação e os últimos incorporam sinais de saudáveis após uma manutenção corretiva e troca do componente,

2.4. Construção e treinamento da rede LSTM

Após a extração dos vetores via wavelet scattering, os dados foram preparados para treinar um autoencoder LSTM, adequado para séries temporais como sinais de vibração.

Os conjuntos foram normalizados com Z-score por canal, garantindo média zero e desvio padrão um, o que melhora a estabilidade do treinamento.

Os dados foram organizados em batches de 16 amostras para otimizar a eficiência computacional e a generalização do modelo.

Então foi implementado um modelo de autoencoder LSTM, composto por:

- Um codificador LSTM que reduz a dimensionalidade dos dados capturando suas dinâmicas principais;
- Um decodificador LSTM que reconstrói os sinais a partir do vetor latente;
- A função de perda usada foi o erro absoluto médio (L1Loss), ideal para dados com ruídos e anomalias.

A rede foi treinada por 1000 iterações (*epochs*), onde cada uma representa uma passada completa por todos os dados de treinamento. A cada iteração, o modelo é ajustado para minimizar a perda entre a entrada e a saída reconstruída.

2.5. Conceitos teóricos relevantes

- Autoencoder: Um autocodificador é uma rede neural que aprende a reduzir a quantidade de informação dos dados e depois reconstruir esses dados novamente. Essa rede possui duas partes principais: o codificador, que

reduz as informações em um espaço menor e mais simples, e o decodificador, que tenta recriar as informações originais a partir dessa versão reduzida. Isso ajuda a simplificar os dados, identificar características importantes e até reconstruir dados perdidos ou incompletos. Os autocodificadores podem ser ajustados para diferentes tipos de dados e necessidades específicas. [15]

- LSTM: As redes Long Short-Term Memory (LSTM) são um tipo especial de rede neural que consegue aprender bem com dados organizados em sequências. Essas redes são criadas para lembrar informações importantes por mais tempo, sem esquecer rapidamente ou ter problemas para aprender. Isso acontece porque elas têm uma espécie de "memória" interna com portões que controlam quais informações devem ser guardadas ou esquecidas. [16].
- Z-score normalization: Técnica que padroniza os dados para distribuição normal, centrando e escalando por canal [16].
- Batch size: Define quantos exemplos são processados por vez. Tamanhos pequenos aumentam a variabilidade do gradiente e podem melhorar generalização. Tamanhos grandes tornam o treinamento mais estável, porém mais lento e com risco de sobreajuste (*overfitting*)[16].
- Épocas (Epochs): Quantidade de ciclos completos de aprendizado sobre os dados. É formada por vários "batches" Mais épocas permitem melhor aprendizado, mas em excesso podem levar ao sobreajuste [16].
- Otimizador Adam: É um método usado para treinar redes neurais, especialmente em aplicações de aprendizado profundo. Ele ajuda a rede neural a aprender mais rápido e de forma mais estável. O ADAM combina duas estratégias importantes: uma que ajusta automaticamente a velocidade de aprendizado com base no histórico das mudanças anteriores, e outra que mantém a direção geral das mudanças mais recentes. Isso torna mais fácil e rápido ensinar a rede neural a realizar tarefas, como reconhecer imagens ou prever valores numéricos [17].
- L1Loss: Também chamada de Erro Médio Absoluto (MAE) mede o erro absoluto médio entre entrada e saída, sendo mais robusto a outliers do que o erro quadrático. A fórmula matemática é:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{1}$$

É uma função muito usada em modelos de aprendizado de máquina para medir a precisão das previsões. Essa métrica calcula a média das diferenças absolutas entre os valores previstos e os valores reais, tornando fácil entender a quão boa foi a previsão [18].

O treinamento do modelo é validado continuamente com o conjunto de validação, e os pesos finais são salvos ao final do processo. A saída é um modelo treinado capaz de reconstruir padrões normais, e cuja diferença (erro de reconstrução) servirá como métrica para detecção de anomalias em novas medições.

2.6. Arquitetura do Autoencoder LSTM

- A estrutura do modelo, feita usando o PyTorch, tem as seguintes partes:
- Camada de entrada: Os dados entram aqui após serem normalizados por um método chamado Z-score, deixando cada canal com média zero e variância um. Isso facilita o aprendizado do modelo.
- Codificador:

Uma primeira camada LSTM com 179 unidades, que lê e mantém a estrutura inicial dos dados;

Uma segunda camada LSTM com 90 unidades, que reduz os dados pela metade, mantendo as informações essenciais.

- Camada de repetição: O resultado da segunda camada é repetido várias vezes, para que o decodificador consiga reconstruir os dados originais completos.
- Decodificador:

Uma primeira camada LSTM com 90 unidades, que começa a reconstruir os dados;

Uma segunda camada LSTM com 179 unidades, que recupera totalmente a forma original dos dados.

- Camadas auxiliares: Utilizam uma função chamada ReLU para adicionar não-linearidade e também usam uma técnica chamada dropout (20%) para evitar que o modelo aprenda detalhes irrelevantes ou ruídos (sobreajuste).
- Camada de saída: Finalmente, gera a reconstrução completa dos dados originais, realizando uma regressão.
- Com essa estrutura, o modelo consegue aprender e identificar padrões complicados em sinais de vibração. O erro na reconstrução dos sinais é usado como uma medida prática para descobrir desvios ou anomalias no comportamento normal.

2.7. Treinamento

O modelo LSTM Autoencoder foi treinado por 1000 épocas com batches de 16 amostras, buscando minimizar o erro de reconstrução entre entrada e saída. Esse número de épocas permite aprendizado suficiente, desde que o erro de validação seja monitorado para evitar sobreajuste.

O tamanho do batch favorece equilíbrio entre eficiência computacional e generalização, sem sobrecarregar a GPU. A perda adotada foi o erro absoluto médio (L1Loss), mais robusto a ruídos e outliers.

Usou-se o otimizador Adam, que ajusta automaticamente a taxa de aprendizado. Os pesos finais foram salvos para uso na detecção de anomalias.

A métrica MAE foi usada para avaliar o desempenho, por sua eficácia em identificar desvios sutis entre o padrão normal e o comportamento observado.

2.8. Avaliação do modelo e limiar de detecção

Após o treinamento, o modelo foi aplicado ao conjunto completo de dados (teste, validação e treinamento concatenados) para calcular o erro de reconstrução (MAE – erro absoluto médio) de cada janela de 1,04 segundo. O objetivo foi quantificar o desvio entre o sinal de entrada e sua reconstrução feita pelo modelo, assumindo que reconstruções ruins indicam comportamentos anômalos.

Para definir um limiar de anomalia de forma não supervisionada e estatisticamente robusta, foi utilizada a técnica baseada no intervalo interquartil (IQR), um método comum de detecção de valores atípicos, isto é, outliers. Esse método considera os seguintes elementos:

Q1: Primeiro quartil (25% dos dados);

Q3: Terceiro quartil (75% dos dados);

$$IQR = Q_3 - Q_1 \tag{2}$$

De tal maneira o limiar do que é considerado anomalia foi definido como sendo qualquer janela com erro MAE acima do valor IQR, mostrado na Equação 2. Esse método é amplamente aceito em análise de dados por não assumir uma distribuição específica e por sua resistência a distorções causadas por valores extremos [19].

3. RESULTADOS

3.1 Erro de reconstrução

Seguindo os passos indicados para análise espectral com MODWPT, a Figura 2 mostra que foi comprovado que para os dados usados neste trabalho as percentagens variaram entre 93,78% e 98,97%, sendo assim optamos por prosseguir com a utilização dos dados como todo sem realizar reamostragem conforme feito por [4].

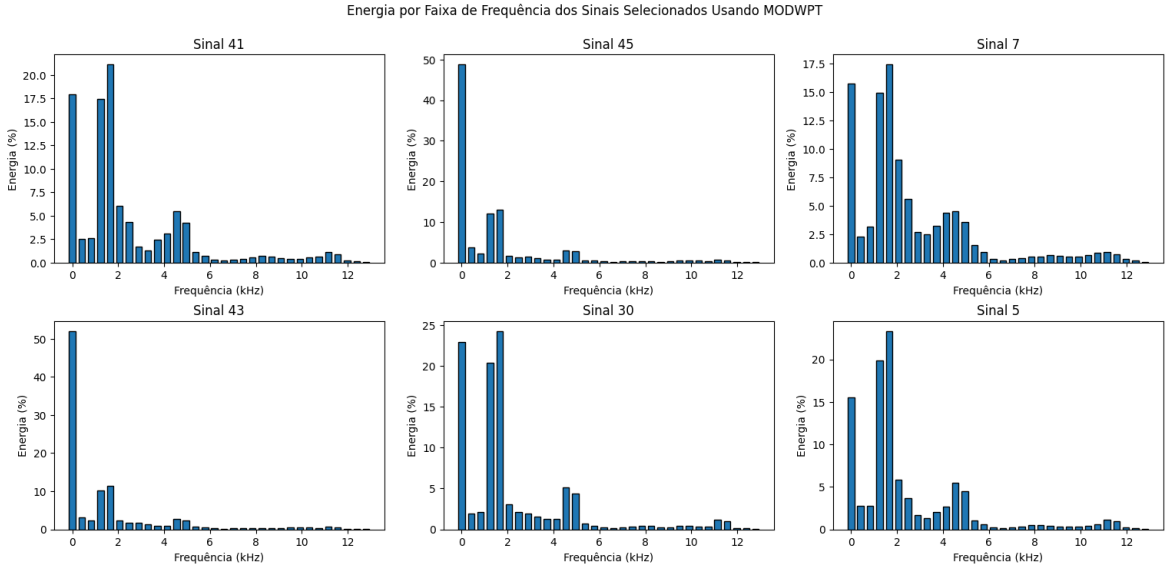

Figura 2 - Separação das bandas de energia para alguns dias aleatórios (Autoria própria)

Além disso, foi gerada uma visualização cumulativa dos sinais ao longo dos 100 dias de medição, com as janelas diárias concatenadas graficamente (2,08 segundos cada). Essa representação contínua permite observar tendências globais, como aumentos de amplitude e padrões anormais que se ressaltem a vista ao olhar o gráfico, funcionando como ferramenta de análise visual essencial antes da análise estatística ou modelagem preditiva.

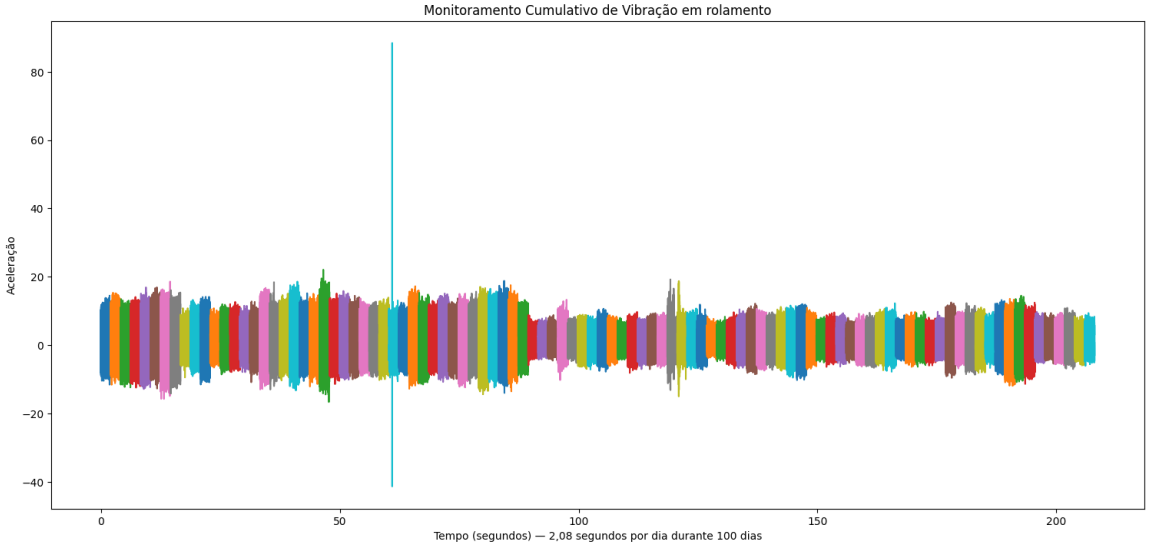

Figura 3 – Histograma de dados de aceleração (Autoria Própria)

Abaixo nas figuras 4 e 5 são mostrados os valores de erro médio absoluto durante todo o treinamento da rede neural. Onde para meios de comparação o treino A utilizou uma janela deslizante com de intervalos menores que a B. No modelo A foi utilizada uma janela deslizante com quadros de 0.52 segundos e sobreposição de 0.13 segundos, enquanto que no B quadros de 1.04 segundos e sobreposição de 0.52 segundos Nelas vemos que ambos os conjuntos de treinamento demostraram diminuição do MAE durante a criação da rede neural, porém ao avaliarmos o conjunto de validação, é possível perceber que o uso de uma maior quantidade de janelas no A resultou em um crescimento no erro absoluto a partir de um certo ponto, o que indica a perda na sua capacidade de

generalizar os resultados para novos dados, isto é, este modelo estava sofrendo um sobreajuste, Desta maneira seguimos com o modelo de treino B.

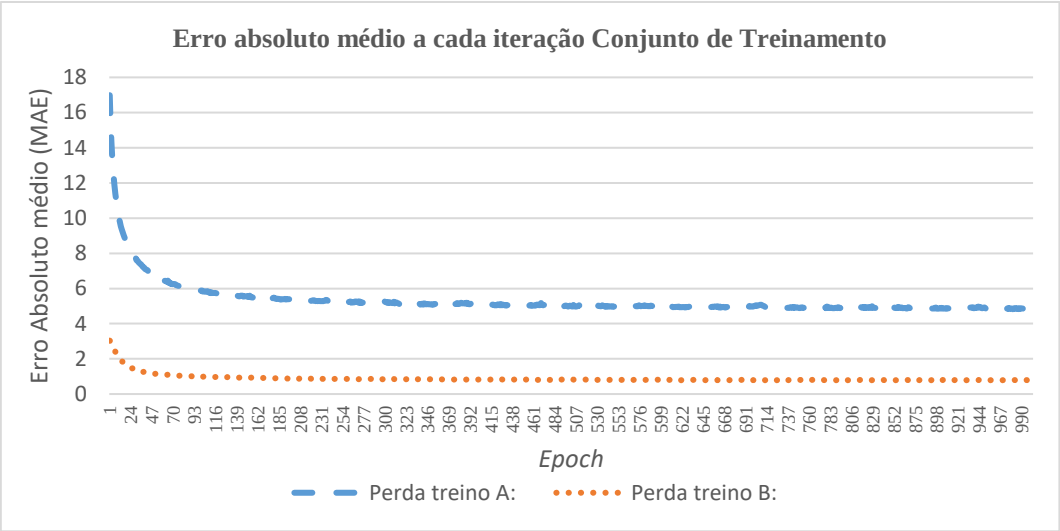

Figura 4- Evolução do erro absoluto para dados de treinamento

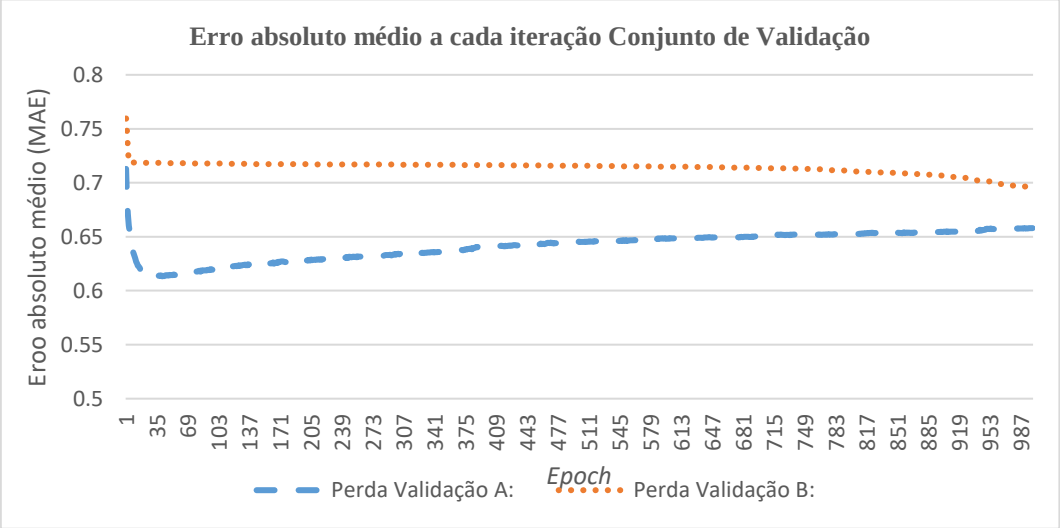

Figura 5- Evolução do erro absoluto para dados de validação

O erro de reconstrução foi calculado para cada uma das janelas de 1 segundo em todos os conjuntos (treinamento, validação e teste), utilizando a métrica MAE (Mean Absolute Error). Observou-se que os erros nos dados de treinamento e validação mantiveram-se baixos e estáveis, como esperado, uma vez que representam o funcionamento normal da máquina.

3.2. Determinação do Limite de Anomalia

O limiar de detecção de anomalias foi definido a partir dos erros no conjunto de validação, com base no valor do terceiro quartil (Q3) acrescido de 1,5 vezes o intervalo interquartil (IQR). Esse critério é robusto e adequado para cenários onde não há rótulos, permitindo isolar janelas com erro anormalmente elevado.

3.3. Visualização

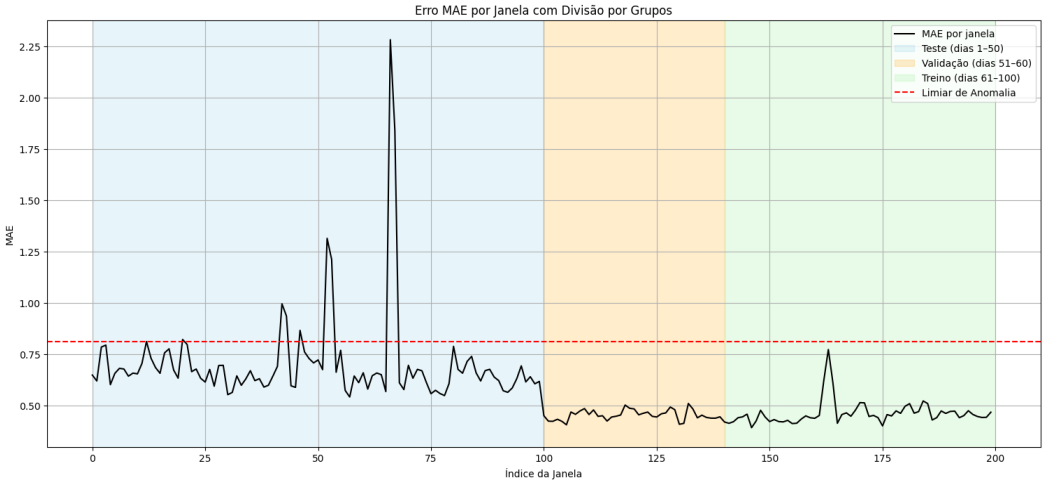

Figura 6 - Avaliação de todo conjunto de dados e limiar de anomalias. O fundo colorido segmenta o conjunto de dados de acordo com os períodos: Azul claro: período de teste (dias 1–50, pré-reparo); Laranja: período de validação (dias 51–70, pós-reparo); Verde claro: período de treinamento (dias 71–100, pós-reparo).

A figura 6 apresenta o MAE de cada janela ao longo do tempo (eixo x), com a linha de limiar de anomalia representada em vermelho tracejado.

A visualização permite verificar em quais regiões o erro ultrapassa o limiar que serve como alarme, indicando possíveis anomalias. Como esperado, a maior concentração de anomalias ocorre no período de teste, pois corresponde aos dias em que o sistema operava com falha na pista interna do rolamento do eixo rápido.

3.4. Discussão dos Resultados

Os resultados mostram que o modelo foi capaz de detectar corretamente os padrões anômalos no período de teste. A partir dos dias 11–12 já é possível notar elevações no erro, sugerindo o início de deterioração. Após o dia 30, as anomalias tornam-se frequentes e intensas, o que está de acordo com o histórico conhecido de falha na pista interna do rolamento do eixo rápido.

O método demonstrou capacidade de separar com clareza o funcionamento normal do comportamento falho, mesmo sem o uso de rótulos, o que válida a eficácia da abordagem proposta com wavelet scattering e autoencoders LSTM.

4. CONCLUSÃO

Este trabalho apresentou a aplicação de autoencoders baseados em LSTM combinados com a transformada wavelet scattering para a detecção de falhas em sinais de vibração provenientes de um sistema mecânico. O modelo foi capaz de aprender padrões normais de operação e detectar, com base no erro de reconstrução, desvios característicos de anomalias, mesmo sem o uso de dados rotulados.

A definição do limiar de detecção foi realizada por meio de uma abordagem estatística baseada no intervalo interquartil (IQR), o que conferiu robustez ao processo de identificação de anomalias. Os resultados demonstraram que a metodologia permite detectar falhas de forma precoce, o que é fundamental em estratégias de manutenção preditiva.

Uma contribuição adicional relevante foi a tradução e adaptação da metodologia originalmente proposta em MATLAB para a linguagem Python, utilizando bibliotecas amplamente acessíveis. Essa tradução torna o método mais acessível e reproduzível, contribuindo diretamente com a literatura técnica e com a comunidade que utiliza ferramentas open-source para análise de dados industriais.

5. AGRADECIMENTOS

Gostaria de agradecer primeiramente ao meu orientador, Prof. Dr. Rômulo Pierre, por toda a paciência e dedicação fundamentais ao longo deste trabalho.

6. REFERÊNCIAS

[1] KHAYAT, Mohammed Mouath AL; SHORBAJY, Mohammed Rashid AL; NAJJAR, Basim AL. Vibration Measurment And Analysis of Two Reciprocating Machines. **FES Journal of Engineering Sciences**, v. 3, n. 1, p. 14-14, 2008.

[2] RAIZER, Klaus et al. Análise de Sinais de ECG com o uso de Wavelets e Redes Neurais em FPGA. **Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica. Campinas**, 2010.

[3] CHENG, Yiwei et al. Intelligent fault diagnosis of rotating machinery based on continuous wavelet transform-local binary convolutional neural network. **Knowledge-Based Systems**, v. 216, p. 106796, 2021.

[4] MISITI, Michel et al. Wavelet toolbox user's guide. **The Math Works Ins**, p. 228-244, 2023.

[5] HARRIS, Charles R. et al. Array programming with NumPy. **Nature**, v. 585, n. 7825, p. 357-362, 2020.

[6] MCKINNEY, Wes et al. Data structures for statistical computing in Python. **SciPy**, v. 445, n. 1, p. 51-56, 2010.

[7] PASZKE, A. Pytorch: An imperative style, high-performance deep learning library. **arXiv preprint arXiv:1912.01703**, 2019.

[8] PEDREGOSA, Fabian et al. Scikit-learn: Machine learning in Python. **the Journal of machine Learning research**, v. 12, p. 2825-2830, 2011.

[9] HUNTER, John D. Matplotlib: A 2D graphics environment. **Computing in science & engineering**, v. 9, n. 03, p. 90-95, 2007.

[10] ANDREUX, Mathieu et al. Kymatio: Scattering transforms in python. **Journal of Machine Learning Research**, v. 21, n. 60, p. 1-6, 2020.

[11] PROAKIS, John G. **Digital signal processing: principles algorithms and applications**. Pearson Education India, 2001.

[12] MALLAT, Stéphane. **A wavelet tour of signal processing**. Elsevier, 1999.

[13] STRÖMBERGSSON, Daniel et al. Bearing monitoring in the wind turbine drivetrain: A comparative study of the FFT and wavelet transforms. **Wind Energy**, v. 23, n. 6, p. 1381-1393, 2020.

[14] RANDALL, Robert Bond. **Vibration-based condition monitoring: industrial, automotive and aerospace applications**. John Wiley & Sons, 2021.

[15] RALTE, Zonunfeli; KAR, Indrajit. **Learn Python Generative AI: Journey from autoencoders to transformers to large language models (English Edition)**. BPB Publications, 2024;p. 61-62

[16] HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long short-term memory. **Neural computation**, v. 9, n. 8, p. 1735-1780, 1997.

[17] KHARE, Y. What is Adam Optimizer and How to Tune its Parameters in PyTorch. Disponível em: <https://www.analyticsvidhya.com/blog/2023/12/adam-optimizer/> (acesso em 16/02/2025).

[18] SOM. Understanding L1 and SmoothL1Loss - Som - Medium. Disponível em: <https://someshfengde.medium.com/understanding-l1-and-smoothl1loss-f5af0f801c71> (acesso em 16/02/2025).

[19] TUKEY, John Wilder et al. **Exploratory data analysis**. Reading, MA: Addison-wesley, 1977.

7. Apêndice A1 (Código)

```
import numpy as np
import pandas as pd
import pywt
# Carrega os dados
caminho_arquivo = r"C:\Users\-\dados_vibracao.csv"
dados = pd.read_csv(caminho_arquivo)
# Frequência de amostragem
frequencia_amostragem = 25600
# Função para calcular a MODWPT e retornar as frequências e
energias relativas
def calcular_modwpt(sinal, fs, wavelet='sym4'):
    pacote = pywt.WaveletPacket(data=sinal, wavelet=wavelet,
mode='symmetric', maxlevel=None)
    nos = pacote.get_level(pacote.maxlevel, order='freq')

    energias = np.array([np.sum(np.abs(no.data) ** 2) for no
in nos])
    energia_total = np.sum(energias)
    energias_relativas = energias / energia_total

    # Calcula a frequência central de cada nó com base no
caminho binário
    frequencias = np.array([
        (fs / 2) * (int(''.join('0' if letra == 'a' else '1'
for letra in no.path), 2) / 2**pacote.maxlevel)
        for no in nos
    ])
    return frequencias, energias_relativas
# Lista para armazenar a energia abaixo de Fs/4 para cada
amostra
lista_energia_abaixo_quarto = []
# Analisa todas as colunas (amostras)
for i, nome_coluna in enumerate(dados.columns):
    sinal = dados[nome_coluna].values
    frequencias, energias_relativas = calcular_modwpt(sinal,
frequencia_amostragem)
    # Calcula a porcentagem de energia abaixo de 1/4 da
frequência de amostragem
    energia_abaixo_quarto =
np.sum(energias_relativas[frequencias <=
(frequencia_amostragem / 4)]) * 100
    lista_energia_abaixo_quarto.append(energia_abaixo_quarto)
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Carrega os dados reamostrados
caminho_arquivo = r"C:\-\dados_vibracao.csv"
#caminho_arquivo =
"C:\Users\geoma\Desktop\TCC\Gearbox_vibration_data\XAll_output
.csv"
dados = pd.read_csv(caminho_arquivo)

# Frequência de amostragem após reamostragem
frequencia_amostragem_nova = 25600

# Inicializa o tempo inicial
tempo_inicial = 0

# Cria a figura
plt.figure(figsize=(12, 6))

# Plota cada sinal de forma cumulativa
for i in range(dados.shape[1]):
    sinal = dados.iloc[:, i].values
    tempo = (np.arange(len(sinal)) /
frequencia_amostragem_nova) + tempo_inicial
    plt.plot(tempo, sinal, label=f'Dia {i+1}')
    tempo_inicial = tempo[-1]

# Adiciona título e rótulos
plt.title("Monitoramento Cumulativo de Vibração em rolamento")
plt.xlabel("Tempo (segundos) – 2,08 segundos por dia durante
100 dias")
plt.ylabel("Aceleração")
plt.tight_layout()

# Exibe o gráfico
plt.show()
```

```
import numpy as np
import pandas as pd
from kymatio.torch import Scattering1D
import torch

# Carrega os dados de vibração reamostrados (assumindo que o
CSV está corretamente formatado)
caminho_arquivo =
r"C:\Users\geoma\Desktop\TCC\Gearbox_vibration_data\dados_vibra
cao.csv"
dados = pd.read_csv(caminho_arquivo)

# Taxa de amostragem
frequencia_amostragem = 25600  # Reamostrado para metade da
taxa original

# Tamanho do quadro (frame) e taxa de sobreposição
tamanho_quadro = int(1.04 * frequencia_amostragem)       # 1
segundo por quadro
taxa_quadro = int(0.52 * frequencia_amostragem)        # 0.5
segundo de sobreposição

# Número de quadros e dias
num_quadros = int((dados.shape[0] - tamanho_quadro) /
taxa_quadro + 1)
num_dias = dados.shape[1]

# Prepara a matriz para armazenar os dados segmentados
matriz_dados = np.zeros((tamanho_quadro, num_quadros *
num_dias), dtype=np.float32)

indice_coluna = 0

# Processa o sinal de cada dia
for dia in range(num_dias):
    sinal = dados.iloc[:, dia].values
    for quadro in range(num_quadros):
        indice_inicio = quadro * taxa_quadro
        indice_fim = indice_inicio + tamanho_quadro
        matriz_dados[:, indice_coluna] =
sinal[indice_inicio:indice_fim]
        indice_coluna += 1
```

```
# Parâmetros de qualidade (número de filtros por oitava)
qualidade_por_camada = (4, 1)
# Cálculo do número de escalas (J) para alcançar a escala
de invariância desejada
# invariance_scale ≈ 2^J / fs  =>  J ≈ log2(fs *
escala_invariancia)
import math
J = math.ceil(math.log2(frequencia_amostragem *
escala_invariancia))
# Cria o objeto de scattering
rede_scat = Scattering1D(
    J=J,
    shape=tamanho_sinal,
    Q=qualidade_por_camada,
    oversampling=1,
    max_order=2)
print(f"Rede de scattering criada com: J={J},
Q={qualidade_por_camada}, tamanho_sinal={tamanho_sinal}")
# Carregar os dados segmentados
caminho_arquivo =
r"C:\Users\geoma\Desktop\TCC\Gearbox_vibration_data\XAll_o
utput.csv"
dados_segmentados =
pd.read_csv(caminho_arquivo).values.astype(np.float32)  #
# Transpor para shape: (200, 26624) → 200 janelas de 1
segundo
dados_janelas = torch.from_numpy(dados_segmentados.T)
# Certifique-sede que a rede está em modo correto
rede_scat.eval()
# Aplicar scattering a cada janela
scat_features = []

with torch.no_grad():
    for janela in dados_janelas:
        coef_scat = rede_scat(janela.unsqueeze(0))  #
shape: (1, canais, tempo)
        coef_scat = coef_scat[:, 1:, :]  # Remove ordem
zero (canal 0)
        scat_features.append(coef_scat.squeeze(0))  #
shape: (canais-1, tempo)
# Empilha os coeficientes
```