UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus Bauru

Weslley Rosalem

Previsão de séries temporais com Transformers e LSTM otimizadas por meta-heurísticas: uma abordagem AIOps para previsão de consumo de recursos computacionais.

Bauru
2025

Weslley Rosalem

Previsão de séries temporais com Transformers e LSTM otimizadas por meta-heurísticas: uma abordagem AIOps para previsão de consumo de recursos computacionais.

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de Bauru.

Orientador: Prof. Dr. Kelton Augusto Pontara da Costa

UNESP

Bauru
2025

R788p

Rosalem, Weslley
Previsão de séries temporais com Transformers e LSTM otimizadas por meta-heurísticas: uma abordagem AIOps para previsão de consumo de recursos computacionais. / Weslley Rosalem. -- Bauru, 2025
112 p.

Dissertação (mestrado) - Universidade Estadual Paulista (UNESP), Faculdade de Ciências, Bauru
Orientador: Kelton Augusto Pontara da Costa

1. Deep Learning. 2. Séries Temporais. 3. LSTM. 4. Transformer. 5. AIOps.. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Dados fornecidos pelo autor(a).

ATA DA DEFESA PÚBLICA DA DISSERTAÇÃO DE MESTRADO DE WESLLEY SANTANA BARRETO ROSALEM, DISCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO, DA FACULDADE DE CIÊNCIAS - CÂMPUS DE BAURU.

Aos 27 dias do mês de maio do ano de 2025, às 10h, por meio de Videoconferência, realizou-se a defesa de DISSERTAÇÃO DE MESTRADO de WESLLEY SANTANA BARRETO ROSALEM, intitulada **Previsão de séries temporais com Transformers e LSTM otimizadas por meta-heurísticas: uma abordagem AIOps para previsão de consumo de recursos computacionais.**. A Comissão Examinadora foi constituída pelos seguintes membros: Prof. Dr. KELTON AUGUSTO PONTARA DA COSTA (Orientador(a) - Participação Virtual) do(a) FC / UNESPBauru SP, Prof. Dr. THIAGO JOSÉ LUCAS (Participação Virtual) do(a) Computação / Professor de Ensino Superior na Faculdade de Tecnologia de Ourinhos, Prof. Dr. DOUGLAS RODRIGUES (Participação Virtual) do(a) Computação / Faculdade de Ciências - UNESP. Após a exposição pelo mestrando e arguição pelos membros da Comissão Examinadora que participaram do ato, de forma presencial e/ou virtual, o discente recebeu o conceito final: APROVADO. Nada mais havendo, foi lavrada a presente ata, que após lida e aprovada, foi assinada pelo(a) Presidente(a) da Comissão Examinadora.

Prof. Dr. KELTON AUGUSTO PONTARA DA COSTA

gov.br Documento assinado digitalmente
KELTON AUGUSTO PONTARA DA COSTA
Data: 27/05/2025 09:48:53-0300
Verifique em https://validar.iti.gov.br

Resumo

Com a crescente transformação digital, a gestão de ambientes de Tecnologia da Informação (TI) tornou-se um desafio cada vez mais complexo e dinâmico. A Inteligência Artificial para Operações de TI (AIOps) surge como uma solução promissora, integrando aprendizado de máquina e big data para automatizar tarefas críticas, como a detecção de anomalias, a previsão de capacidade e a identificação de causas raízes. Este estudo propõe uma abordagem comparativa entre as arquiteturas Long Short-Term Memory (LSTM) e *Transformer* para a previsão de consumo de recursos computacionais, utilizando dados de séries temporais provenientes de dois conjuntos distintos: Google Cluster Traces 2019 e métricas do Prometheus. A pesquisa realiza uma revisão sistemática da literatura, com foco em trabalhos publicados entre 2019 e 2023, e introduz o algoritmo de otimização meta-heurística Manta Ray Foraging Optimization (MRFO) para ajuste de hiperparâmetros das redes neurais. A metodologia contempla o pré-processamento dos dados, a configuração experimental e a utilização de métricas como MAE, RMSE, MAPE e SMAPE. O estudo visa contribuir para a eficiência das operações de TI, oferecendo subsídios para a gestão proativa de recursos e a mitigação de falhas em ambientes de nuvem.

Palavras-chave: *Deep Learning; Séries Temporais; LSTM; Transformer; MRFO; AIOps.*

Abstract

With the ongoing digital transformation, the management of Information Technology (IT) environments has become an increasingly complex and dynamic challenge. Artificial Intelligence for IT Operations (AIOps) has emerged as a promising solution, integrating machine learning and big data to automate critical tasks such as anomaly detection, capacity forecasting, and root cause analysis. This study proposes a comparative approach between Long Short-Term Memory (LSTM) and Transformer architectures for predicting computational resource consumption, using time series data from two distinct datasets:Google Cluster Traces 2019 and Prometheus metrics. The research includes a systematic literature review focused on studies published between 2019 and 2023, and introduces the metaheuristic optimization algorithm Manta Ray Foraging Optimization (MRFO) for neural network hyperparameter tuning. The methodology encompasses data preprocessing, experimental setup, and the use of evaluation metrics such as MAE, RMSE, MAPE, and SMAPE. This study aims to contribute to the efficiency of IT operations, providing insights for proactive resource management and failure mitigation in cloud-based environments.

Keywords: *Deep Learning; Time Series; LSTM; Transformer; MRFO; AIOps.*

Lista de ilustrações

Figura 1 – Procedimento adotado para realização da Revisão Sistemática da Literatura. 19
Figura 2 – Distribuição dos artigos por ano de publicação 20
Figura 3 – Detalhamento do resultado após os processos de filtragem dos artigos . 20
Figura 4 – Procedimento adotado para realização da Revisão Sistemática da Literatura. 22
Figura 5 – Predições do uso de memória no *Google Cluster* 56
Figura 6 – Predições Prometheus - 30 minutos 57
Figura 7 – Curva de convergência do MRFO durante a otimização da LSTM no cenário Prometheus (granularidade de 15 min). 58

Lista de tabelas

Tabela 1 – Características da análise sistemática da literatura 27
Tabela 2 – Estatísticas descritivas iniciais. 47
Tabela 3 – Limites de busca dos hiper-parâmetros da LSTM. 49
Tabela 4 – Limites de busca dos hiper-parâmetros do Transformer. 51
Tabela 5 – Desempenho médio (*média* ± *desvio-padrão*) do modelo **LSTM** no *Google Cluster Traces*. 55
Tabela 6 – Desempenho médio (*média* ± *desvio-padrão*) do modelo **Transformer** no *Google Cluster Traces*. 55
Tabela 7 – Desempenho médio (*média* ± *desvio-padrão*) do modelo **LSTM** no *Prometheus*. 56
Tabela 8 – Desempenho médio (*média* ± *desvio-padrão*) do modelo **Transformer** no *Prometheus*. 56
Tabela 9 – Hiperparâmetros ótimos (MRFO) para a **LSTM**. 57
Tabela 10 – Hiperparâmetros ótimos (MRFO) para o **Transformer**. 58
Tabela 11 – Resultados do algoritmo MRFO otimizando a LSTM (janela de 15 min) para métricas coletadas via Prometheus. 109

Lista de abreviaturas e siglas

AI - *Artificial Intelligence*

AIOPs - *Artificial Intelligence for IT Operations*

ARIMA - *AutoRegressive Integrated Moving Average*

Autoformer - *Auto-regressive Transformer for Time Series*

BERT - *Bidirectional Encoder Representations from Transformers*

CAM - *Class Activation Mapping*

CNN - *Convolutional Neural Network*

CPU - *Central Processing Unit*

CVAE - *Conditional Variational Autoencoder*

DevOps - *Development Operations*

DL - *Deep Learning*

FFN - *Feed-Forward Network*

GPT - *Generative Pre-trained Transformer*

GRU - *Gated Recurrent Unit*

HAC - *Hierarchical Agglomerative Clustering*

HIS - *Hospital Information Systems*

IT - *Information Technology*

K8s - *Kubernetes*

KPI - *Key Performance Indicator*

LSTM - *Long Short-Term Memory*

MAE - *Mean Absolute Error*

MAPE - *Mean Absolute Percentage Error*

ML - *Machine Learning*

MLOPs	- *Machine Learning Operations*
MRFO	- *Manta Ray Foraging Optimization*
NLP	- *Natural Language Processing*
NNET	- *Neural Network*
PSO	- *Particle Swarm Optimization*
PU	- *Positive-Unlabeled*
RMSE	- *Root Mean Squared Error*
RMSprop	- *Root Mean Square Propagation*
RNN	- *Recurrent Neural Network*
SGD	- *Stochastic Gradient Descent*
SMAPE	- *Symmetric Mean Absolute Percentage Error*
TF	- *Transformer*
TI	- Tecnologias da Informação
TIC	- Tecnologias da Informação e Comunicação
TSTF	- *Time Series Transformer*
TTM	- *Tiny Time Mixer*
ViT	- *Vision Transformer*

Sumário

1 INTRODUÇÃO . . . 13
1.1 Desafio operacional . . . 13
1.2 AIOps como paradigma. . . . 13
1.3 Limitações dos trabalhos existentes. . . . 14
1.4 Motivação adicional: observabilidade em Prometheus. . . . 14

2 FUNDAMENTAÇÃO TEÓRICA . . . 18
2.1 Revisão Sistemática da Literatura . . . 18
2.1.1 Técnicas para Revisão Sistemática da Literatura . . . 18
2.1.2 Trabalhos Correlatos - ano de 2019 . . . 22
2.1.3 Trabalhos Correlatos - ano de 2020 . . . 22
2.1.4 Trabalhos Correlatos - ano de 2021 . . . 23
2.1.5 Trabalhos Correlatos - ano de 2022 . . . 24
2.1.6 Trabalhos Correlatos - ano de 2023 . . . 26
2.2 Introdução à LSTM . . . 28
2.2.1 Formulação Matemática da LSTM . . . 28
2.2.2 Vantagens da LSTM . . . 29
2.2.3 Limitações da LSTM . . . 30
2.2.4 Aplicações em Previsão de Séries Temporais . . . 30
2.2.5 Comparação com Outras Variantes de RNNs . . . 31
2.2.6 Avanços Recentes em LSTM para AIOps . . . 31
2.3 Arquitetura *Transformer* e Mecanismo de Atenção . . . 32
2.3.1 Arquitetura Original – Encoder-Decoder . . . 32
2.3.2 Mecanismo de Atenção Multi-Cabeças . . . 33
2.3.3 Rede *Feed-Forward* e Codificação Posicional . . . 34
2.3.4 Normalização de Camada e Estabilização do Treinamento . . . 35
2.3.5 Resumo da Arquitetura . . . 35
2.3.6 Evolução da Arquitetura *Transformer* e Aplicações em Séries Temporais . . . 36
2.3.7 *Transformers* em NLP: Modelos BERT e GPT . . . 36
2.3.8 *Transformers* em Visão Computacional: *Vision Transformer* (ViT) . . . 37
2.3.9 *Transformers* para Séries Temporais e AIOps . . . 38
2.3.10 *Transformers* e AIOps . . . 41
2.3.11 Aplicação no Trabalho Atual: Previsão de Métricas com *Transformer* Simplificado . . . 42
2.4 Algoritmo MRFO . . . 43

3 METODOLOGIA . . . 47
3.1 Conjuntos de Dados . . . 47
3.1.1 Google Cluster Traces (2019) . . . 47
3.1.2 *Prometheus* Metrics . . . 47
3.1.3 Pipeline de Pré-processamento . . . 47
3.2 Formulação Matemática do Problema . . . 48
3.3 Modelo LSTM . . . 49
3.3.1 Arquitetura . . . 49
3.3.2 Complexidade . . . 49
3.3.3 Hiper-parâmetros . . . 49
3.4 Modelo Transformer para Séries Temporais . . . 50
3.4.1 Definição . . . 50
3.4.2 Atenção Escalonada . . . 50
3.4.3 Prevenção de Overfitting . . . 50
3.4.4 Hiper-parâmetros . . . 51
3.5 Meta-Otimização via MRFO . . . 52
3.5.1 Motivação . . . 52
3.5.2 Configuração . . . 52
3.6 Ambiente Experimental . . . 52
3.6.1 Hardware . . . 52
3.6.2 Software . . . 52
3.7 Métricas de Avaliação . . . 53
3.7.1 Definições formais . . . 53
3.7.2 Interpretação prática . . . 53
3.7.3 Comparação estatística . . . 54

4 RESULTADOS . . . 55
4.1 Desempenho nos Dados do Google Cluster Traces . . . 55
4.2 Desempenho nos Dados do Prometheus . . . 56
4.3 Hiperparâmetros Ótimos Encontrados (MRFO) . . . 57

5 CONCLUSÃO . . . 59
5.1 Contribuições Científicas e Práticas . . . 59
5.2 Limitações . . . 60
5.3 Trabalhos Futuros . . . 60
5.4 Considerações Finais . . . 60

REFERÊNCIAS . . . 61

APÊNDICES 66

APÊNDICE A – TRANSFORMER-PSEUDO-CODIGO.PY 67

APÊNDICE B – LSTM-TRAINED.PY 70

APÊNDICE C – TRANSFORMER-TRAINED.PY 85

APÊNDICE D – IMPLEMENTAÇÃO DO MODELO 99

D.1 model.py 99

D.2 Tabela de execuções 109

1 INTRODUÇÃO

A quarta revolução industrial consolidou-se sobre a premissa de que dados e, por conseguinte, infraestruturas de *Tecnologia da Informação* (TI) figuram como novos ativos estratégicos. Setores críticos, tais como saúde, finanças, telecomunicações, segurança pública e logística, passaram a ancorar seus modelos de negócio em plataformas digitais que demandam alta disponibilidade, resiliência, flexibilidade e elasticidade. Para satisfazer tais requisitos, arquiteturas monolíticas cederam lugar a ecossistemas distribuídos, baseados em virtualização, contêineres, microsserviços e computação em múltiplas nuvens (PAHL, 2015). Embora essa evolução otimize escalabilidade e time-to-market, ela introduz uma combinatória de componentes heterogêneos servidores físicos, máquinas virtuais, redes definidas por *software*, bancos de dados multimodais, *service meshes* cuja interdependência eleva a complexidade operacional a níveis sem precedentes.

1.1 Desafio operacional[1]

A heterogeneidade estrutural, aliada à explosão de métricas geradas em tempo real (ordem de milhões de *data points* por minuto em grandes nuvens públicas), torna onerosa a **detecção proativa de anomalias**, a **identificação de causa-raiz** e a **previsão de capacidade**. Métodos tradicionais, baseados em limiares estáticos ou regressão linear, falham em capturar padrões não-lineares e dependências de longo alcance. Como consequência, equipes operacionais permanecem reféns de abordagens majoritariamente reativas, o que compromete *Service-Level Agreements* (SLAs) e eleva custos de *downtime* não planejado.

1.2 AIOps como paradigma.

Nesse cenário, desponta o **AIOps** (*Artificial Intelligence for IT Operations*) disciplina que combina *big data*[2] a técnicas de aprendizado de máquina avançado para automatizar processos operacionais (SILL, 2019). O pilar analítico da AIOps é a modelagem de *Key Performance Indicators* (KPIs) como séries temporais, permitindo a antecipação de tendências de consumo de recursos e a detecção de padrões anômalos.

[1] As métricas analisadas em (COHEN et al., 2004) apontam que mais de 70 % dos incidentes críticos em datacenters derivam de falhas de configuração ou correlação ineficiente de alertas.

[2] *Big data* é caracterizado pelos "5 Vs": volume, velocidade, variedade, veracidade e valor. A literatura recente adiciona variabilidade e visualização como dimensões complementares.

Modelos de *deep learning* (DL), em particular *Long Short-Term Memory* (LSTM) e, mais recentemente, a arquitetura *Transformer* destacam-se pelo poder de representação para relacionamentos temporais complexos (VASWANI et al., 2017).

1.3 Limitações dos trabalhos existentes.

A literatura indica dois eixos predominantes de pesquisa:

(i) LSTM/GRU (*Gated Recurrent Unit*) sintonizados via *grid search* ou *random search*, métodos exaustivos ou estocásticos que apresentam baixa escalabilidade mediante o aumento da dimensionalidade hiperparamétrica;

(ii) *Transformers* aplicados a séries temporais, porém mediante *tuning* manual, heurísticas simplistas ou pesquisa bayesiana clássica, o que não explora plenamente superfícies de perda complexas.

Ainda carece de: (a) uma comparação sistemática entre LSTM, ou modelos de aprendizado profundo para series temporais, e *Transformers*, sob conjunto de dados e métricas idênticos, em contexto de AIOps; e (b) uma investigação sobre meta-heurísticas modernas notadamente o *Manta Ray Foraging Optimization* (MRFO) (ZHAO; WANG; ZHANG, 2020) para otimizar hiperparâmetros em tais modelos.

1.4 Motivação adicional: observabilidade em Prometheus.

Ferramentas nativas-na-nuvem, como o *Prometheus*, tornaram-se padrão de fato para coleta de métricas de sistemas distribuídos. Sua linguagem de consulta (PromQL) e a natureza *pull-based* favorecem instrumentação granular, mas geram fluxos de dados de alta cardinalidade. Integrar modelos DL especialmente *Transformer*s a esse repositório de séries temporais abre caminho para *pipelines* preditivos *online*, elevando a automação do ciclo "monitorar-analisar-agir".

Objetivo geral

Avaliar o impacto do uso do algoritmo de otimização MRFO na melhoria do desempenho preditivo de modelos LSTM e *Transformer* aplicados à previsão do consumo de recursos computacionais, com base em séries temporais extraídas de ambientes AIOps e monitoradas por meio do *Prometheus*.

Objetivos específicos

a) Construir um *benchmark* unificado com os conjuntos *Google Cluster Traces 2019* e métricas do *Prometheus*, reamostrados nas granularidades de 1, 5 e 30 minutos.

b) Implementar *pipelines* de LSTM e *Transformer* com parametrização flexível (unidades, camadas, *dropout*, janela de entrada).

c) Empregar o MRFO para minimizar o *Mean Absolute Percentage Error* (MAPE) em validação, comparando-o a *Grid Search, Random Search e PSO (Particle Swarm Optimization).*

d) Quantificar, na fase de teste, ganhos de RMSE (*Root Mean Squared Error*), MAE (*Mean Absolute Error*) , MAPE e *Simetric Mean Absolut Percentage Error* (SMAPE) trazidos pelo MRFO nas diferentes granularidades.

e) Discutir implicações práticas para orquestração de recursos e detecção de anomalias em plataformas baseadas no *Prometheus.*

Contribuições

- Proposição de um *pipeline* de otimização automática integrado MRFO a modelos sequenciais de última geração.
- Primeiro estudo, até onde se sabe, a comparar LSTM e *Transformer* sob condições experimentais idênticas em dados reais de AIOps.
- Evidência empírica de que o MRFO reduz o MAPE em até 13,6 %, além de ganhos consistentes em RMSE e SMAPE.
- Divulgação pública de código-fonte e artefatos experimentais visando à reprodutibilidade.[3]

Metodologia em síntese

Dois *datasets* reais, reprocessados em três granularidades, servem de base aos experimentos. Os modelos recebem janelas deslizantes de tamanho $n \in \{24, 48, 72, 96\}$ e predizem o próximo passo. O MRFO opera com população de 20 pessoas, 500 iterações, alternando entre estratégias *chain*, *cyclone* e *somersault.* O erro em validação (MAPE) orienta a busca e as métricas finais são computadas em teste.

[3] Repositório GitHub: <https://github.com/weslleyrosalem/dissertacao/tree/main/Experiments>

Resultados em perspectiva

Resultados preliminares indicam que o *Transformer* supera o LSTM em escalas de 30 minutos, enquanto o LSTM mantém leve vantagem em 1 minuto. O MRFO reduz o MAE médio em até 6,8 % (*Google Cluster Traces*) e 5,4 % (*Prometheus*), evidenciando sua superioridade sobre métodos de busca convencionais.

Estrutura da dissertação

- **Capítulo 1 — Introdução**
  Delimita o problema, explicita a lacuna científica, apresenta os objetivos, destaca as contribuições e antecipa a metodologia.

- **Capítulo 2 — Fundamentação Teórica**
  Reúne conceitos nucleares e trabalhos relacionados, organizados em quatro subseções:

  1. **Revisão Sistemática da Literatura** (Subsec. 2.1): protocolo, filtros e síntese de 21 artigos relevantes;
  2. **Long Short-Term Memory** (Subsec. 2.2): formulação matemática, vantagens, limitações e aplicações em AIOps;
  3. **Arquitetura *Transformer*** (Subsec. 2.3): descreve o mecanismo de atenção, evoluções (BERT, GPT, ViT) e adaptações para séries temporais;
  4. **Algoritmo MRFO** (Subsec. 2.4): fundamentos bio-inspirados e equações de atualização.

- **Capítulo 3 — Metodologia**
  Detalha o *pipeline* experimental:

  1. **Conjuntos de Dados e Pré-processamento** (3.1);
  2. **Formulação do Problema e Janelas de Entrada** (3.2);
  3. **Arquiteturas LSTM e *Transformer*** (3.3);
  4. **Otimização de Hiperparâmetros via MRFO** (3.4).

- **Capítulo 4 — Resultados**
  Apresenta métricas (RMSE, MAE, MAPE, SMAPE), curvas de previsão, tabelas comparativas e análise do ganho obtido com o MRFO.

- **Capítulo 5 — Conclusão**
  Resume achados, discute limitações (custo computacional, univariância, etc.) e propõe extensões — previsão multivariada, *sparse-attention* e meta-heurísticas híbridas.

- **Referências**
  Listagem normatizada (IEEE/ABNT) dos trabalhos citados.

- **Apêndices**
  Códigos-fonte em *Python*: pseudocódigo do *Transformer*, arquitetura detalhada e *scripts* de treino (Apêndices A–C).

Essa organização reflete fielmente a estrutura atualmente compilada e facilita ao leitor acompanhar o fluxo: da fundamentação teórica ao método, dos resultados quantitativos às implicações práticas.

2 Fundamentação Teórica

Este capítulo aborda os conceitos e teorias fundamentais que embasam este estudo. A compreensão desses conceitos é essencial para entender as técnicas e metodologias adotadas ao longo deste trabalho. Inicialmente, é discutida a evolução da Inteligência Artificial (IA). Em seguida, é focado no subcampo da IA denominado *AIOps*, que integra IA e análise de dados visando aprimorar as operações de TI. A familiaridade com IA e AIOps é vital para perceber como essas tecnologias podem ser aplicadas em ambientes de TI, potencializando a detecção de anomalias e a análise preditiva de séries temporais.

2.1 Revisão Sistemática da Literatura

A Revisão Sistemática da Literatura é uma abordagem metodológica que busca identificar, avaliar e interpretar todas as pesquisas relevantes sobre um tema específico. Distingue-se por sua metodologia rigorosa e bem definida, que pode ser replicada e auditada, conferindo maior confiabilidade ao processo utilizado (TRANFIELD; DENYER; SMART, 2003).

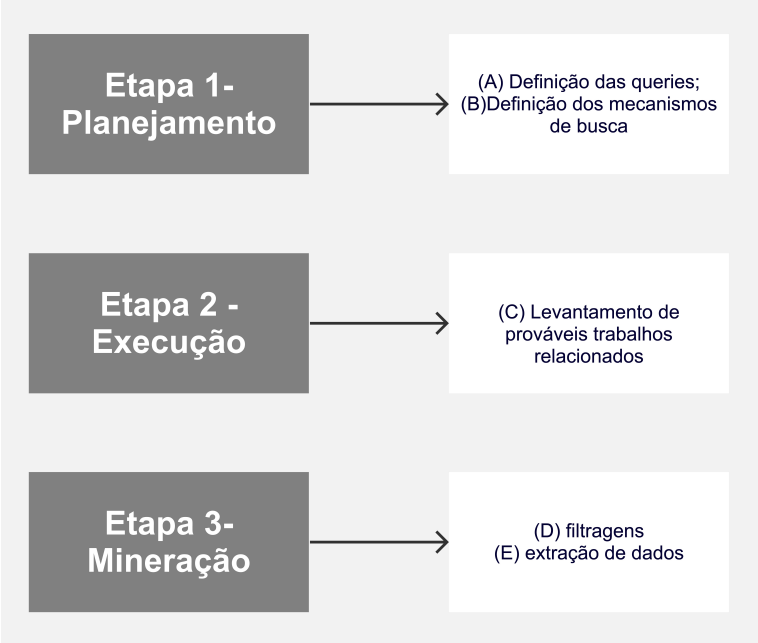
Essa metodologia é empregada para coletar e sintetizar evidências empíricas que atendam a critérios de inclusão preestabelecidos (KITCHENHAM; CHARTERS, 2007). O processo envolve a definição de questões de pesquisa pertinentes, a seleção e avaliação qualitativa dos estudos, a extração de dados, a síntese e apresentação da documentação dos achados.

2.1.1 Técnicas para Revisão Sistemática da Literatura

A Revisão Sistemática é fundamental na pesquisa acadêmica, pois oferece uma visão holística do conhecimento existente, destacando lacunas que podem ser objeto de futuras investigações (PETTICREW; ROBERTS, 2006). Além disso, evita a duplicação de esforços, evidenciando pesquisas prévias sobre o tema.

A elaboração da revisão sistemática inicia-se com a definição de um protocolo rigoroso de pesquisa. Esse protocolo estabelece os critérios de inclusão e exclusão, as bases de dados a serem consultadas e as estratégias de busca a serem adotadas (BIOLCHINI et al., 2005). A Figura 1 apresenta uma ilustração detalhada sobre procedimento adotado neste estudo:

Figura 1 – Procedimento adotado para realização da Revisão Sistemática da Literatura.



Fonte: Elaborado pelo autor

As etapas apresentadas na Figura 1 são detalhadas a seguir:

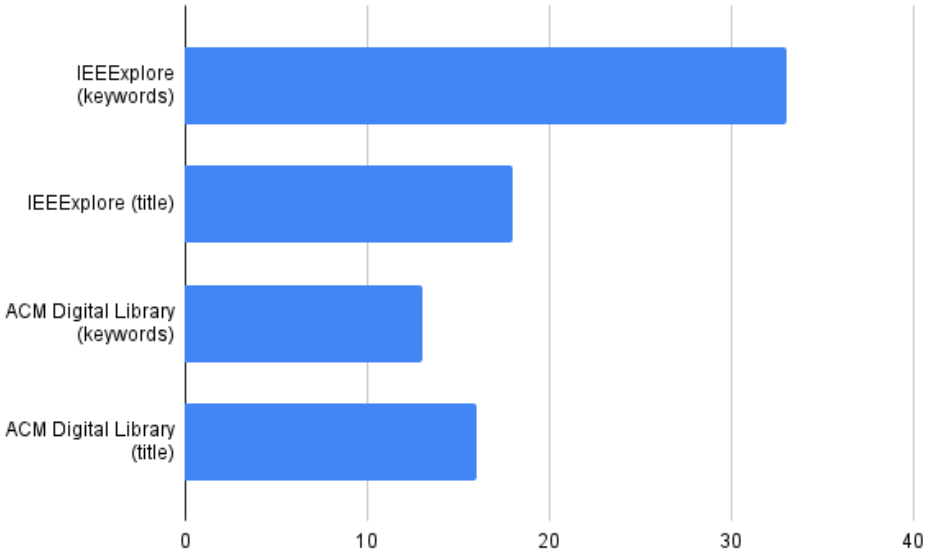
1. As *queries*[4] são *strings*[5] que estabelecem critérios para a seleção de artigos. Nesta pesquisa, o foco foi em artigos, cujo título ou *keywords*[6] incluíssem a palavra *AIOPs*. Além disso, restringiu-se o período de publicação entre 2018 e o primeiro semestre de 2023.

2. Dada a reconhecida relevância em Ciência da Computação, especialmente em Redes de Computadores, monitoramento e observabilidade, duas bases de pesquisa científica foram selecionadas: IEEExplore[7] e ACM *Digital Library*[8].

3. A execução das *queries* nas bases mencionadas resultou em 80 artigos. A distribuição dos artigos por ano de publicação é ilustrada na Figura 2.

[4] *Queries* são instruções ou expressões usadas para recuperar informações de um banco de dados. Elas são frequentemente escritas em uma linguagem de consulta de banco de dados como SQL.
[5] *Strings* são sequências de caracteres usadas para representar texto em programação e computação. Elas são fundamentais para o processamento de texto, pesquisa e muitas outras aplicações.
[6] *Keywords* são palavras ou frases que resumem o conteúdo principal de um texto, documento ou base de dados. Elas são frequentemente usadas em buscas para encontrar informações relevantes.
[7] <https://ieeexplore.ieee.org/>
[8] <https://dl.acm.org/>

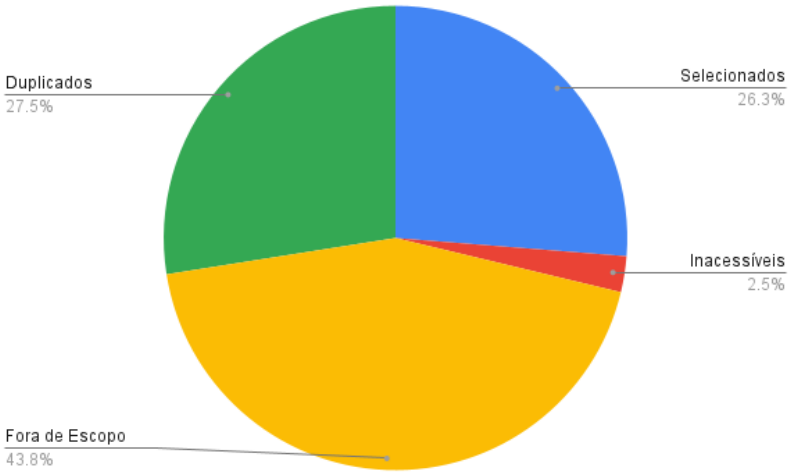
Figura 2 – Distribuição dos artigos por ano de publicação



Fonte: Elaborado pelo autor.

4. Durante a filtragem, foram removidos artigos duplicados ou inacessíveis, totalizando 22 duplicados e 2 inacessíveis. Dos 56 artigos remanescentes, 35 não se alinhavam diretamente ao escopo desta dissertação. Portanto, após a filtragem, 21 artigos foram reconhecidos como estreitamente relacionados ao tema proposto. A Figura 3 ilustra este processo.

Figura 3 – Detalhamento do resultado após os processos de filtragem dos artigos



Fonte: Elaborado pelo autor.

5. A última etapa envolveu a extração das informações mais relevantes dos artigos selecionados. Para identificar tendências, foram consideradas as seguintes categorias de análise:

- Tipo de modelo: técnicas de *A*prendizado de máquina (*Machine Learning*,ML) empregadas ou propostas;
- *Dataset*[9]: bases de dados utilizadas nos testes;
- Arquitetura: arquiteturas propostas;
- *Feature selection*[10];
- *Features*[11] selecionadas;
- Resultados: conclusões alcançadas pelos autores, como RMSE e MAE;
- Metodologia: recursos utilizados na criação do modelo de identificação de anomalias e predição, incluindo linguagens e *softwares*.

Os artigos selecionados foram sintetizados, destacando-se os elementos mais pertinentes ao contexto desta pesquisa. A organização dos estudos foi realizada conforme os seguintes critérios:

- Não foram encontrados trabalhos correlatos no ano de 2018. Esta afirmação indica que, após uma busca sistemática na literatura utilizando palavras-chave e critérios específicos, não foram identificados estudos ou publicações relacionadas ao tema da dissertação no ano de 2018;
- Trabalhos correlatos de 2019 na Subseção 2.1.2;
- Trabalhos correlatos de 2020 na Subseção 2.1.3;
- Trabalhos correlatos de 2021 na Subseção 2.1.4;
- Trabalhos correlatos de 2022 na Subseção 2.1.5;
- Trabalhos correlatos de 2023 na Subseção 2.1.6;

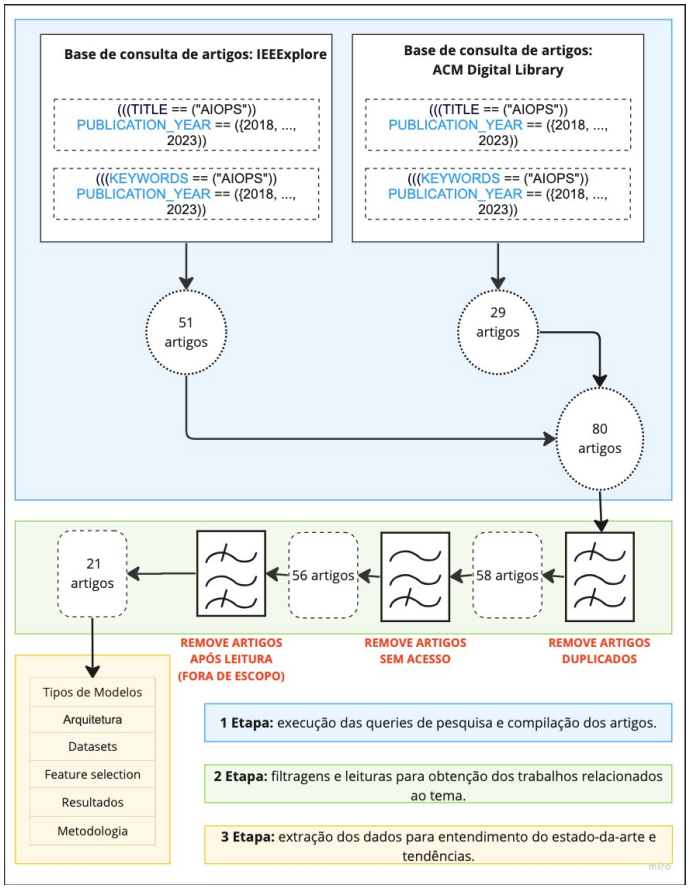
A Figura 4 ilustra o processo completo da Revisão Sistemática da Literatura, desde a definição das *queries* até a extração das informações cruciais:

[9] Um *dataset* é uma coleção de dados, geralmente apresentada em formato tabular, que serve como entrada para algoritmos de ML e análise estatística.

[10] *Feature Selection* é o processo de selecionar um subconjunto de características relevantes para uso em modelagem. A seleção de características eficaz pode melhorar o desempenho do modelo e reduzir a complexidade computacional.

[11] *Features* são variáveis individuais que atuam como entradas em modelos de ML. Cada feature representa uma dimensão específica de dados que o algoritmo pode usar para aprender.

Figura 4 – Procedimento adotado para realização da Revisão Sistemática da Literatura.



Fonte: Elaborado pelo autor

2.1.2 Trabalhos Correlatos - ano de 2019

No artigo de Nedelkoski, Cardoso e Kao (2019), os autores exploram a aplicação de AIOps para detectar anomalias com base em registros de *distributed tracing*. Esses registros fornecem informações detalhadas sobre a disponibilidade e o tempo de resposta dos serviços. A abordagem proposta concentra-se na detecção de anomalias no tempo de resposta, utilizando aprendizado não supervisionado. Os autores empregam técnicas de modelagem de dados com aprendizado profundo e avaliam a precisão e o desempenho da proposta em ambientes de teste e produção. A combinação de GRUs e *autoencoders* variacionais é destacada como uma técnica promissora para a modelagem de séries temporais complexas.

2.1.3 Trabalhos Correlatos - ano de 2020

O estudo de Mormul e Stach (2020) aborda o desafio de monitorar ambientes de TI complexos, englobando nuvens privadas e públicas, ambientes de IoT (*Internet of Things*), aplicativos e contêineres. Eles destacam a necessidade de um modelo de contexto em AIOps para gerenciar grandes volumes de dados armazenados em diferentes formatos. O

framework proposto é estruturado em cinco camadas: aquisição, gerenciamento, análise, apresentação de dados e respostas automatizadas. O *Monitoring Resource Model* (MRM) é um componente central desse *framework*. Para a análise de dados, é proposto um modelo de *A*prendizado de máquina baseado em redes neurais, especificamente o LSTM.

Kostadinov, Atanasova e Petrov (2020) discutem a importância do ML, especialmente no contexto de AIOps, para descobrir relacionamentos entre objetos e processos em infraestruturas de TI convergentes. O estudo enfatiza a necessidade de técnicas de ML para identificar padrões e sequências em grandes volumes de eventos. A arquitetura proposta foca na análise e ML para processar dados de diversos dispositivos e instrumentos de TI.

O objetivo do estudo apresentado por Chen, Wang e Yang (2020) é propor uma rede neural dinâmica para prever fluxos de dados de séries temporais em cenários de AIOps. Os modelos de ML discutidos incluem *MWNN (Multi-Way Neural Network), WNN (Wavelet Neural Network)* e LSTM. Os dados utilizados durante os testes incluem conjuntos de dados de CPUs (*Central Processing Unit*) com diferentes capacidades. Os resultados mostram uma comparação do consumo de recursos para MWNN, WNN e LSTM quando alcançam o mesmo desempenho.

2.1.4 Trabalhos Correlatos - ano de 2021

No estudo conduzido por Guodan, Hao e Tingting(2021), os autores propõem uma abordagem inovadora que combina múltiplos métodos integrados para prever a capacidade de recursos. Utilizando o modelo de aprendizado de máquina a LSTM,com base nos dados históricos o estudo demonstra como é possível fornecer previsões precisas sobre o uso de recursos de TI, como CPU, disco rígido e memória. Esta abordagem tem potencial para otimizar a gestão de recursos, garantindo que os sistemas permaneçam estáveis e confiáveis.

Machine. . . (2021) introduz o conceito de *Machine Reasoning* (MR) e destaca seu papel vital na melhoria das AIOps para Redes Baseadas em Intenções. MR é uma subárea da IA que se concentra em capturar e utilizar o conhecimento humano através de linguagens semânticas. Esta abordagem complementa a aprendizagem de Máquina ao fornecer inferências precisas baseadas no conhecimento adquirido. O estudo apresenta cenários em que MR em AIOps pode ser utilizado para automatizar e aprimorar a identificação da causa raiz de problemas em redes de computadores.

O artigo de Yang et al.(2021) apresenta o método AID (*Aggregated Intensity of Dependency*) como uma solução eficiente para prever a intensidade das dependências em sistemas de nuvem em larga escala. Os autores utilizam dados simulados e industriais para testar a eficácia do método proposto. Os resultados mostram que o AID é capaz de medir com precisão a intensidade das dependências, superando outras abordagens comparativas.

Stefano et al.(2021) discutem o *design* e a implementação de recursos para migrar dados de sistemas de monitoramento antigos para instâncias do *Prometheus* usando o *framework Ananke*. Além disso, o estudo propõe uma estratégia de dimensionamento automático, baseada na previsão de picos de tráfego, usando o modelo *Facebook Prophet*. A abordagem é voltada para monitorar e modelar aplicações nativas de nuvem, com foco em métricas de desempenho em tempo real e estratégias de otimização.

2.1.5 Trabalhos Correlatos - ano de 2022

Wang et al.(2022) introduzem o TSAGen, uma ferramenta inovadora de geração de séries temporais. Esta ferramenta permite aos pesquisadores gerar dados sintéticos, fornecendo uma fonte de dados confiável para avaliar o desempenho de algoritmos de detecção de anomalias. O TSAGen foi projetado para enfrentar desafios, como a geração de diversas anomalias, ajuste da gravidade das anomalias e controle das características dos KPIs gerados.

No estudo de Li et al.(2022), os autores propõem uma rede neural profunda, a CDX-Net, para previsão de séries temporais multivariadas no contexto de AIOps. A arquitetura proposta do CDX-Net incorpora módulos avançados, como ASPP, SRM, CAM, GRU, transformador e AB, para aprimorar os procedimentos de extração e fusão de características.

Zhang et al.(2022b) apresentam uma solução para o Desafio ICASSP-SPGC-2022 AIOps, focando na inferência precisa de combinações em *root cause analysis* (RCA). O documento detalha os desafios encontrados nos dados da competição e propõe uma estrutura robusta para resolver o problema, incluindo a introdução de *TextCNN*.

O estudo de Fukuda et al.(2022) propõe uma abordagem para gerar relatórios de falhas legíveis por humanos em sistemas de TIC. A abordagem utiliza o modelo LSTM para séries temporais multivariadas e gera um relatório de falha em formato de texto. Os autores testam a eficácia e o desempenho do método proposto usando dados coletados de um sistema de microsserviços em um *cluster Kubernetes* (k8s).

Moreschini et al.(2022) apresentam uma discussão profunda sobre a representação gráfica para o DevOps em aplicações baseadas em aprendizado de máquina , também conhecida como MLOps. Os autores exploram meticulosamente as fases do MLOps, desde o planejamento, onde são identificados o problema a ser resolvido e os dados disponíveis, até a seleção de abordagens de análise de dados e algoritmos adequados. A fase de codificação é destacada, onde o sistema e o código de ML são implementados e validados. A fase de validação é discutida em detalhes, enfatizando a avaliação do desempenho do modelo de ML com novos dados. O artigo também destaca a necessidade imperativa de um processo de MLOps integrado e delinea os desafios associados à adoção prática do MLOps.

Qian et al.(2022) propõem um framework de dimensionamento automático proativo inovador chamado *RobustScaler*, especialmente projetado para cenários de computação em nuvem. O estudo é direcionado para o desenvolvimento de um *framework* que não apenas gera decisões de mudança e elasticidade, mas também otimiza o equilíbrio entre custo e Qualidade de Serviço (QoS). O modelo proposto é robusto, capaz de lidar com ruídos, dados ausentes e anomalias. O algoritmo de Método dos Multiplicadores de Direção Alternada (ADMM) é usado para treinar o modelo, que captura tanto a periodicidade quanto a estocasticidade das chegadas de consultas. A arquitetura do *framework* é composta por componentes essenciais, como detecção de periodicidade, modelagem histórica de chegadas de consultas, previsão de chegada de consultas e plano de dimensionamento.

Xiao, Cao e Wu (2022)introduzem um *framework* inovador baseado em aprendizado federado, o EFL-WP, projetado especificamente para previsão de carga de trabalho em ambientes inter-nuvem. O *framework* visa colaborar na formação de modelos de ML para previsão de carga de trabalho, garantindo que informações sensíveis não sejam compartilhadas. Os autores sugerem o uso de modelos LSTM para prever métricas de desempenho, como utilização da CPU e memória. A arquitetura proposta é composta por um coordenador que agenda tarefas de treinamento e orquestra os treinadores, enquanto os treinadores usam seus dados para treinar modelos locais.

Zhang et al.(2022a) apresentam o método "*PUTraceAD*"*Positive-Unlabeled*, uma abordagem inovadora para detecção de anomalias em *tracing* de microsserviços. A arquitetura proposta é tripartida, envolvendo *Embedding* de *Span*, Construção de Grafo de Rastros e Treinamento do Modelo. Utilizando uma GNN (*Graph Neural Network*) e aprendizado PU, o método é capaz de detectar anomalias em *tracing* com precisão. O conjunto de dados usado, *TrainTicket*, é um sistema de referência de microsserviços, e os experimentos foram conduzidos em um *cluster Kubernetes*. Os resultados dos experimentos avaliam a eficácia e eficiência do *PUTraceAD*, bem como o impacto de diferentes configurações.

Yang et al.(2022) discutem o Sistema de Gerenciamento de Dependências, ou *Dependency Management System* (DMS), uma solução abrangente para gerenciar dependências de serviços em sistemas em nuvem. O DMS é uma plataforma *end-to-end* que suporta todo o ciclo de vida para garantir a confiabilidade do serviço, desde a implantação inicial até a otimização arquitetural proativa e a mitigação reativa de falhas. Os dados usados nos testes do DMS abrangem uma variedade de fontes, incluindo informações de dependência coletadas de *trancing* distribuído, arquivos de configuração, consultas do orquestrador de serviços e relatórios de dependência de implantação.

Song et al.(2022) abordam questões relacionadas à implementação da observabilidade em sistemas de informações hospitalares (HIS). O artigo apresenta uma pesquisa literária detalhada e fornece um resumo abrangente de conceitos relacionados, incluindo definições de monitoramento e HIS, requisitos e soluções em cenários específicos. A ar-

quitetura proposta integra microsserviços e AIOps, com indicadores-chave (KPIs) como qualidade e escala de dados. O documento também oferece sugestões valiosas para os departamentos de TI dos hospitais sobre como abordar essas questões.

Jin et al.(2022) realizaram uma análise detalhada de AIOps na gestão unificada de resiliência de dados em *data lakehouses*. O artigo propõe soluções inovadoras para prever violações do *Recovery Point Objective* (RPO) e fornecer sugestões valiosas aos SREs sobre como configurar recursos do sistema para evitar tais violações. Utilizando aprendizado supervisionado em conjunto com análise de séries temporais, o artigo propõe um modelo de ML que combina métodos de aprendizado online e offline e filtragem de solicitações previstas para garantir a estabilidade das solicitações futuras.

2.1.6 Trabalhos Correlatos - ano de 2023

Os autores Traini e Cortellessa(2023) apresentam uma análise detalhada sobre a implementação de *baselines* utilizando a linguagem de programação *Python*. Eles escolhem a regressão linear como o modelo de ML para sua análise. O artigo aborda profundamente as questões de pesquisa, o design experimental meticuloso, a aplicação na análise de desempenho e as distribuições de latência. Os autores também discutem o uso de cargas de trabalho que variam continuamente e fornecem *insights* sobre a configuração experimental, destacando a importância de uma abordagem sistemática para garantir resultados precisos e confiáveis.

Sun et al.(2023) oferecem uma visão abrangente do desenvolvimento do sistema de detecção de *outlier* denominado *OutSpot*. Este sistema é especialmente projetado para *datacenters* de alto desempenho e alta criticidade, que são primordialmente responsáveis pelo fornecimento de *streaming* de vídeos. O principal objetivo do sistema é detectar *outliers* nos KPIs coletados desses *datacenters*. O modelo de ML adotado é uma combinação inovadora de *Hierarchical Agglomerative Clustering* (HAC) com *Conditional Variational Autoencoder* (CVAE). O HAC é utilizado para agrupar os KPIs com base em seus padrões distintos. Posteriormente, as informações de agrupamento de cada KPI são incorporadas ao método. Esta abordagem integrada permite que o *OutSpot* detecte *outliers* para KPIs em larga escala, mesmo quando esses KPIs apresentam padrões variados. A arquitetura proposta para o *OutSpot* é meticulosa, dividindo os conjuntos de dados coletados em conjuntos de treinamento e teste. O conjunto de treinamento é composto por dados coletados ao longo dos primeiros 7 dias, enquanto o conjunto de teste contém dados do último dia. Este último conjunto é rotulado cuidadosamente por operadores experientes utilizando uma ferramenta desenvolvida pelos próprios autores. O processo de rotulação dos *outliers* é rigoroso, envolvendo três operadores, e a decisão final é tomada apenas quando os rótulos fornecidos por eles divergem.

Tabela 1 – Características da análise sistemática da literatura

Ano de Publicação	**Autores**	**Tópico Principal**
2019	(NEDELKOSKI; CARDOSO; KAO, 2019)	Detecção de anomalias usando *Distributed Tracing* e *Deep Learning* em AIOps
2020	(MORMUL; STACH, 2020)	Monitoramento de ambiente de TI complexo com modelo de contexto
2020	(KOSTADINOV; ATANASOVA; PETROV, 2020)	Redução de incidentes em infraestruturas de TI convergentes através de ML
2020	(CHEN; WANG; YANG, 2020)	Previsão de fluxos de dados de séries temporais em cenários de AIOps
2021	(GUODAN; HAO; TINGTING, 2021)	Previsão da capacidade de recursos usando LSTM
2021	(MACHINE..., 2021)	Melhoria das Operações de IA para Redes Baseadas em Intenções usando MR
2021	(YANG et al., 2021)	Previsão da intensidade de dependências em sistemas em nuvem
2021	(STEFANO et al., 2021)	Migração de dados de sistemas de monitoramento antigos para instâncias do *Prometheus*
2022	(WANG et al., 2022)	Geração de séries temporais para avaliação de algoritmos de detecção de anomalias
2022	(LI et al., 2022)	Previsão de séries temporais multivariadas em AIOps
2022	(ZHANG et al., 2022b)	Inferência precisa de combinações em RCA
2022	(FUKUDA et al., 2022)	Geração de relatórios de falhas legíveis por humanos em sistemas de TIC
2022	(MORESCHINI et al., 2022)	Representação gráfica para o *DevOps*[12] em aplicações baseadas em aprendizado de máquina (MLOps)
2022	(QIAN et al., 2022)	*Framework* de dimensionamento automático proativo para computação em nuvem

Continua na próxima página

[12] *DevOps* é uma filosofia e prática de engenharia de software que visa unificar o desenvolvimento de software (*Dev*) e a operação de software (*Ops*). O principal objetivo de DevOps é encurtar o ciclo de vida do desenvolvimento de sistemas, proporcionando entrega contínua de alta qualidade, e, assim, melhorar a colaboração e a comunicação entre as equipes de desenvolvimento e operações. *DevOps* integra métodos ágeis, automação, integração contínua, entrega contínua e monitoramento contínuo do software em operação.

Ano de Publicação	Autores	Tópico Principal
2022	(XIAO; CAO; WU, 2022)	Previsão de carga de trabalho em ambientes de inter-nuvem usando aprendizado federado
2022	(ZHANG et al., 2022a)	Detecção de anomalias em *tracing* de microsserviços usando GNN e aprendizado PU
2022	(YANG et al., 2022)	Gerenciamento de dependências de serviços em sistemas em nuvem
2022	(SONG et al., 2022)	Implementação da observabilidade em sistemas de informações hospitalares
2022	(JIN et al., 2022)	Gestão unificada de resiliência de dados em data *lakehouses*
2023	(TRAINI; CORTELLESSA, 2023)	Implementação de *baselines* usando *Python* e regressão linear
2023	(SUN et al., 2023)	Detecção de *outlier* para *datacenters* com alto nível de desempenho e alta criticidade

2.2 Introdução à LSTM

As Redes Neurais Recorrentes (RNNs) foram introduzidas na década de 1980 como uma extensão das redes neurais *feedforward*, projetadas para processar sequências temporais ao permitir conexões entre unidades que formam um ciclo, possibilitando que a informação persista ao longo do tempo (RUMELHART; HINTON; WILLIAMS, 1986). Contudo, as RNNs tradicionais enfrentam dificuldades em aprender dependências de longo prazo devido ao problema do gradiente evanescente, no qual os gradientes se tornam extremamente pequenos durante a retropropagação, comprometendo o aprendizado (HOCHREITER, 1998).

Para superar essa limitação, foram desenvolvidas arquiteturas avançadas, como as LSTM, propostas por Hochreiter e Schmidhuber em 1997 (HOCHREITER; SCHMIDHUBER, 1997). As LSTMs introduzem um mecanismo de memória por meio de um estado de célula que pode ser mantido ou modificado ao longo do tempo, permitindo a retenção de informações relevantes por períodos prolongados. Essa capacidade torna as LSTMs particularmente adequadas para tarefas que envolvem sequências extensas, como a previsão de séries temporais em AIOps.

2.2.1 Formulação Matemática da LSTM

A arquitetura LSTM é composta por células que contêm três portas principais: esquecimento, entrada e saída, além de um estado de célula que atua como memória. Cada

porta utiliza funções de ativação sigmóide para regular o fluxo de informação. As equações que descrevem o funcionamento de uma célula LSTM são:

- **Porta de Esquecimento**:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

  Esta porta determina quais informações do estado da célula anterior (C_{t-1}) devem ser descartadas, produzindo valores entre 0 (esquecer completamente) e 1 (manter integralmente).

- **Porta de Entrada**:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

  A porta de entrada decide quais novas informações serão incorporadas ao estado da célula, enquanto $\tilde{C}_t$ representa um candidato para atualização.

- **Estado da Célula**:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

  O estado da célula é atualizado combinando a informação retida do estado anterior com a nova informação selecionada.

- **Porta de Saída**:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \odot \tanh(C_t)$$

  A porta de saída controla quais partes do estado da célula são usadas para gerar o estado oculto (h_t).

Nessas equações, σ é a função sigmóide, *tanh* é a tangente hiperbólica, W e b são matrizes de pesos e vetores de *bias*, x_t é a entrada no tempo t, h_{t-1} é o estado oculto anterior, e $\odot$ denota multiplicação elemento a elemento. A complexidade computacional por passo de tempo é $O(u \cdot (d + u))$, onde u é o número de unidades e d é a dimensão da entrada, refletindo o crescimento quadrático dos parâmetros.

2.2.2 Vantagens da LSTM

As LSTMs apresentam diversas vantagens que as tornam amplamente utilizadas em aplicações de séries temporais:

- **Captura de Dependências de Longo Prazo**: O estado da célula e as portas permitem que as LSTMs retenham informações por muitos passos de tempo, essencial para modelar padrões históricos complexos, como sazonalidades diárias ou semanais em dados de AIOps.

- **Robustez a Ruído e Não Linearidade**: Estudos demonstram que LSTMs superam modelos estatísticos tradicionais, como ARIMA, em dados ruidosos e não lineares, sendo eficazes em cenários com flutuações abruptas (SIAMI-NAMINI; TAVAKOLI; NAMIN, 2018; LARA-BENíTEZ; CARRANZA-GARCíA; RIQUELME, 2021).

- **Gestão Eficiente de Memória**: A porta de esquecimento possibilita a seleção adaptativa de informações relevantes, evitando a perda de dados críticos e facilitando a modelagem de padrões temporais sofisticados.

2.2.3 Limitações da LSTM

Apesar de suas vantagens, as LSTMs possuem limitações significativas:

- **Sensibilidade a Hiperparâmetros**: O desempenho depende de hiperparâmetros como número de unidades, taxa de aprendizado, tamanho do lote e regularização. A otimização manual ou métodos como *Grid Search* são computacionalmente custosos (GOODFELLOW; BENGIO; COURVILLE, 2016).

- **Complexidade Computacional**: O número de parâmetros, dado por $4 \cdot (u \cdot (d + u) + u)$, cresce quadraticamente, tornando o treinamento intensivo em recursos, especialmente para grandes conjuntos de dados.

- **Risco de Sobreajuste**: Modelos com muitas unidades ou janelas temporais extensas podem sobreajustar, particularmente sem regularização adequada.

- **Foco Unívariate**: Muitas aplicações, como a descrita no anexo, concentram-se em previsões unívariadas, limitando a capacidade de capturar interações entre múltiplas variáveis.

2.2.4 Aplicações em Previsão de Séries Temporais

As LSTMs são amplamente empregadas em previsão de séries temporais devido à sua habilidade em modelar dependências temporais complexas. Em AIOps, são usadas para prever consumo de recursos, como memória, CPU e rede, permitindo alocação proativa e redução de *downtime*. No estudo do anexo, LSTMs foram aplicadas a dois conjuntos de dados reais: Google Cluster Traces 2019 (30 dias de uso de memória, agregados a cada 30 minutos) e métricas do *Prometheus* (7 dias de uso de memória de um servidor Linux).

Ambos exibem sazonalidade diária, e os modelos foram treinados para prever o próximo passo temporal, minimizando o MAE.

Além de AIOps, LSTMs são aplicadas em diversos domínios, incluindo:

- **Finanças**: Previsão de preços de ações e taxas de câmbio (FISCHER; KRAUSS, 2018).
- **Saúde**: Monitoramento de pacientes e previsão de doenças (LIPTON et al., 2015).
- **Clima**: Previsão meteorológica e análise de padrões climáticos (SHI et al., 2015).
- **Energia**: Previsão de demanda e preços de energia (LAGO; RIDDER; SCHUTTER, 2018).

2.2.5 Comparação com Outras Variantes de RNNs

Além das LSTMs, as GRUs, introduzidas por Cho et al. (2014), são uma variante popular de RNNs. As GRUs possuem uma estrutura mais simples, com apenas duas portas (atualização e redefinição), reduzindo o número de parâmetros e acelerando o treinamento. Contudo, sua capacidade de capturar dependências de longo prazo pode ser inferior às LSTMs em cenários complexos (CHUNG et al., 2014).

No contexto do estudo, a escolha das LSTMs justifica-se pela necessidade de modelar padrões sazonais e flutuações abruptas em dados de AIOps, onde dependências de longo prazo são críticas.

2.2.6 Avanços Recentes em LSTM para AIOps

Pesquisas recentes ampliaram a aplicação de LSTMs em AIOps. Propuseram o modelo DA-LSTM-VAE, que integra LSTMs com *Variational Autoencoders* (VAEs) e um mecanismo de atenção em duas etapas para detecção de anomalias em KPIs, alcançando F1-scores de 0,90, 0,93 e 0,86 em um conjunto de dados do AIOps Challenge 2018 (ZHANG et al., 2023). Li et al. (2024) utilizaram PSO para otimizar LSTMs em detecção de intrusões de rede, obtendo resultados de ponta.

As LSTMs são uma ferramenta poderosa para previsão de séries temporais em AIOps, destacando-se na captura de dependências de longo prazo e robustez a dados complexos. Contudo, sua complexidade e sensibilidade a hiperparâmetros requerem otimização avançada, como o MRFO, que demonstrou melhorias significativas. Este estudo valida a eficácia das LSTMs e introduz uma abordagem inovadora, abrindo caminhos para pesquisas futuras em previsão multivariada e meta-heurísticas híbridas.

2.3 Arquitetura *Transformer* e Mecanismo de Atenção

A arquitetura *Transformer* foi introduzida por Vaswani et al. (2017) no artigo "Attention Is All You Need", revolucionando o processamento de sequências ao abandonar completamente o uso de recorrência ou convolução. O problema principal abordado por essa proposta era a dificuldade das RNNs, mesmo em variantes como LSTM, em capturar dependências de longo alcance de forma eficiente. As RNNs processam tokens em sequência, o que impede paralelismo e dificulta o aprendizado de relacionamentos distantes devido a problemas de *vanishing gradient*. Os pesquisadores propuseram resolver essas limitações utilizando mecanismos de atenção *self-attention* em uma arquitetura encoder–decoder totalmente paralelizável. Com o *Transformer*, relações de longo alcance podem ser modeladas diretamente através de pesos de atenção, e todas as posições de uma sequência são processadas simultaneamente, agilizando o treinamento em comparação com arquiteturas sequenciais tradicionais. A eficácia dessa abordagem foi demonstrada inicialmente em tradução automática, superando significativamente o estado da arte da época (VASWANI et al., 2017). A ideia central é que a "atenção" entre elementos da sequência é suficiente para aprender estruturas complexas, dispensando mecanismos recorrentes – daí o título do artigo, "atenção é tudo que você precisa".

2.3.1 Arquitetura Original – Encoder-Decoder

O *Transformer* original consiste de dois blocos principais: um codificador e um decodificador, cada um empilhando múltiplas camadas idênticas. O codificador recebe uma sequência de entrada (por exemplo, palavras de uma sentença) e produz representações intermediárias para cada posição. Em seguida, o decodificador gera a sequência de saída (por exemplo, tradução) autoregressivamente, atendendo à informação codificada. Tanto as camadas do codificador quanto as do decodificador baseiam-se em dois submódulos centrais: (1) o módulo de atenção de múltiplas cabeças (*multi-head self-attention*) e (2) uma rede *feed-forward* posicionada após a atenção. Além disso, no decodificador cada camada inclui um terceiro submódulo de atenção encoder–decoder, que permite ao decodificador "olhar" para as saídas do codificador, vide Apêndice A. Crucialmente, cada submódulo é envolto por conexões residuais e seguido de uma normalização de camada (*layer normalization*) (VASWANI et al., 2017). Assim, a saída de cada subcamada é $\text{LayerNorm}(x + \text{subcamada}(x))$, facilitando a estabilização do treinamento. A seguir, detalhamos os componentes fundamentais do *Transformer* – o mecanismo de atenção, a codificação posicional e a normalização – junto às principais equações matemáticas que regem seu funcionamento.

2.3.2 Mecanismo de Atenção Multi-Cabeças

No cerne do *Transformer* está o mecanismo de *self-attention*, que calcula a importância de cada elemento da sequência em relação aos demais. Cada posição i na sequência de entrada é projetada em três vetores: consulta (q_i), chave (k_i) e valor (v_i), obtidos por transformações lineares aprendidas. Agregando todas as posições, denotamos por Q, K e V as matrizes que contêm, respectivamente, as consultas, chaves e valores para a sequência inteira (cada uma de dimensão $n \times d$, onde n é o comprimento da sequência e d a dimensão de *embedding*). A atenção por produto escalar escalonado (*scaled dot-product attention*) é então computada conforme a Equação 1:

$$\text{Attention}(Q, K, V); =; \text{softmax}!\left(\frac{Q, K^\top}{\sqrt{d_k}}\right); V \ , \tag{1}$$

onde d_k é a dimensão dos vetores de chave (e consulta). A equação acima produz, para cada posição i, uma combinação ponderada dos valores v_j de todas as posições j, em que o peso $\text{softmax}\left(\frac{q_i \cdot k_j}{\sqrt{d_k}}\right)$ reflete quanto o elemento j é relevante para i. O fator de escalonamento $1/\sqrt{d_k}$ foi introduzido para estabilizar os gradientes – sem ele, para valores grandes de d_k, os produtos $q_i \cdot k_j$ poderiam ter magnitude alta, resultando em distribuições de *softmax* muito agudas e dificultando o aprendizado (VASWANI et al., 2017). Em resumo, a atenção calcula uma média ponderada dos valores V usando coeficientes baseados em similaridades entre consultas e chaves; isso permite que cada posição foque em outras posições relevantes da sequência, capturando relações sem considerar explicitamente a distância entre tokens.

Uma inovação crucial do *Transformer* é o uso de atenção de múltiplas cabeças (*multi-head attention*). Em vez de calcular uma única distribuição de atenção com vetores Q, K, V de alta dimensão, o *Transformer* aprende h diferentes projeções (cabeças) menores. Para cada cabeça $i \in \{1, \ldots, h\}$, definem-se matrizes de projeção W_i^Q, W_i^K e W_i^V de dimensões $d \times d_k$, $d \times d_k$ e $d \times d_v$, respectivamente. Cada cabeça então computa $head_i = \text{Attention}(QW_i^Q, ; KW_i^K, ; VW_i^V)$. As h atenções resultantes (cada uma de tamanho $n \times d_v$) são concatenadas e projetadas novamente por W^O (dimensão $hd_v \times d$) para produzir a saída final do módulo de multi-head. Formalmente:

$$\text{MultiHead}(Q, K, V); =; \text{Concat}(head_1, \ldots, head_h); W^O \ , \tag{2}$$

onde $head_i = \text{Attention}(QW_i^Q, ; KW_i^K, ; VW_i^V)$. Essa estrutura de múltiplas cabeças permite que o modelo atenda simultaneamente a diferentes subespaços de representação (VASWANI et al., 2017). Em outras palavras, cada cabeça pode se especializar em um tipo de relacionamento ou padrão na sequência (por exemplo, uma cabeça pode focar em relacionar uma determinada palavra com suas adjacentes enquanto outra captura

relações de longo alcance). Na arquitetura original, usaram-se $h = 8$ cabeças de atenção, com $d = 512$ e $d_k = d_v = 64$ em cada cabeça, de modo que $8 \times 64 = 512$. O resultado da atenção multi-cabeças é então passado adiante para uma rede feed-forward, conforme descrito a seguir, após ser combinado à entrada via conexão residual.

2.3.3 Rede *Feed-Forward* e Codificação Posicional

Após o módulo de atenção, cada camada do *Transformer* inclui uma rede *feed-forward* posicionada que atua separadamente em cada posição da sequência. Essa rede é composta por duas camadas densas com uma função de ativação não linear (ReLU na versão original). Seja x o vetor de saída da atenção para uma posição; a rede *feed-forward* calcula $\mathrm{FFN}(x) = W_2, \max(0, , W_1 x + b_1) + b_2$, onde W_1, W_2 são matrizes de pesos e b_1, b_2 os *bias*. Essa MLP *position-wise* expande e reconstrói a dimensionalidade (por exemplo, originalmente W_1 expande de $d_{\text{model}} = 512$ para 2048, e W_2 reduz de volta para 512). A intuição é fornecer ao modelo uma transformação não linear poderosa em cada posição, após a atenção ter misturado informações globais da sequência. Assim, cada camada do codificador (ou decodificador) combina a auto-atenção multi-cabeças com uma transformação não linear local, permitindo modelar tanto dependências entre posições quanto padrões complexos em cada posição.

Um desafio decorrente do uso de atenção pura é que o modelo não possui, intrinsecamente, noção de ordem sequencial – diferente de RNNs, que processam tokens em ordem temporal. Para contornar isso, Vaswani et al. (2017) introduziram a codificação posicional dos *embeddings* de entrada. Antes de alimentar a primeira camada do *Transformer*, são adicionados aos vetores de *embedding* dos *tokens* vetores que codificam a posição (índice) de cada *token* na sequência. Na versão original, utilizou-se uma codificação posicional determinística baseada em funções seno e cosseno de frequências diferentes (VASWANI et al., 2017). A formulação é dada por:

$$PE_{(pos,,2i)} = \sin!\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \qquad PE_{(pos,,2i+1)} = \cos!\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \tag{3}$$

onde pos é a posição do token na sequência (começando em 0) e i indexa a dimensão do vetor de posição até $d_{\text{model}} - 1$. Essa definição produz componentes sinusoidais de diferentes períodos para cada dimensão do *embedding* posicional, de forma que cada posição no tempo tenha um vetor único e que preserve relações de distância: por exemplo, a diferença entre $PE_{pos=10}$ e $PE_{pos=20}$ será similar à diferença entre $PE_{pos=30}$ e $PE_{pos=40}$, uma vez que a relação de deslocamento de 10 posições é capturada pelos mesmos ângulos das senoides. Esse esquema permite ao modelo aprender, através da atenção, em que posição relativa outros tokens estão, pois as consultas e chaves incluem informação posicional somada aos *embeddings* das palavras. Alternativamente, trabalhos posteriores exploraram

codificações posicionais aprendidas (vetores de posição como parâmetros treináveis) ou até estratégias híbridas que combinem informações absolutas e relativas sobre posição (SHAW; USZKOREIT; VASWANI, 2018), mas o importante é que algum tipo de sinal de posição é necessário para que o *Transformer* distinga a ordem dos elementos na entrada.

2.3.4 Normalização de Camada e Estabilização do Treinamento

Cada subcamada de atenção ou *feed-forward* no *Transformer* é seguida de uma operação de normalização de camada (*Layer Normalization*) antes de passar para a próxima etapa. A normalização de camada, proposta por Ba et al., impõe que os neurônios de uma dada camada tenham média zero e variância unitária para cada exemplo, melhorando a estabilidade do treinamento em redes profundas. Dado um vetor de ativação $x = (x_1, \ldots, x_d)$ de dimensão d (por exemplo, as ativações em uma posição após a atenção ou *feed-forward*), a normalização de camada produz um vetor $y = (y_1, \ldots, y_d)$, cujos componentes são:

$$y_j; =; \frac{x_j - \mu(x)}{\sigma(x) + \epsilon}, \gamma_j; +; \beta_j \ , \qquad 1 \leq j \leq d \ , \tag{4}$$

onde $\mu(x) = \frac{1}{d}\sum_{j=1}^{d} x_j$ é a média dos componentes de x, $\sigma(x) = \sqrt{\frac{1}{d}\sum_{j=1}^{d}(x_j - \mu(x))^2}$ é o desvio padrão, e ϵ é um termo de estabilidade numérica. Os parâmetros γ_j e β_j são vetores treináveis (mesma dimensão d) que permitem re-escalar e deslocar linearmente a saída normalizada. No *Transformer*, a normalização de camada é aplicada junto a conexões residuais, isto é, cada subcamada gera $\text{LayerNorm}(x + \text{subcamada}(x))$ como saída final (VASWANI et al., 2017). Esse esquema de *residual add e norm* ajuda a gradiente fluir pelas camadas e permite treinar pilhas profundas (na arquitetura original há 6 camadas no codificador e 6 no decodificador, totalizando 12 subcamadas atencionais). Em síntese, a normalização garante que distribuições de ativação permaneçam estáveis mesmo conforme a profundidade e o treinamento avançam, contribuindo para o sucesso do modelo.

2.3.5 Resumo da Arquitetura

Combinando os componentes acima, o *Transformer* consegue transformar uma sequência de entrada em outra sequência de saída aprendendo padrões complexos de atenção. No codificador, cada camada processa todos os tokens em paralelo via *multi-head self-attention* (permitindo que cada token "veja" todos os outros) e então refina cada posição via a *feed-forward* position-wise. No decodificador, um mecanismo similar é usado, mas cada posição da saída só pode atender a posições anteriores (*masking causal* na *self-attention* do decodificador, para não "espiar" o futuro que ainda será previsto). Além disso, o decodificador realiza atenção extra sobre as saídas do codificador (atenção encoder–decoder), ligando as duas partes. Esse design eliminou a necessidade de processar sequências passo a passo: as dependências temporais passam a ser aprendidas pelos padrões

de atenção, e a computação pode ser totalmente paralela em cada camada. O resultado foi um salto de qualidade e eficiência em tradução automática e, subsequentemente, em diversas outras tarefas de sequência. Nos anos seguintes, a comunidade explorou extensivamente variações dessa arquitetura básica, adaptando o *Transformer* a novos domínios e objetivos, conforme discutido na próxima seção.

2.3.6 Evolução da Arquitetura *Transformer* e Aplicações em Séries Temporais

Desde a sua introdução, a arquitetura *Transformer* tornou-se a espinha dorsal de inúmeros avanços em aprendizado profundo. Nesta seção, procedeu a discussão a evolução do *Transformer* em variantes notáveis – incluindo BERT, GPT e Vision *Transformer* – e explorou-se como a arquitetura foi adaptada para domínios além do texto, com ênfase particular em séries temporais. Por fim, conectou-se esses avanços ao contexto de AIOps, descrevendo exemplos práticos (como o *Time Series Transformer da HuggingFace e o Granite Time Series da IBM)* e explicando como o presente trabalho aproveita uma versão simplificada do *Transformer* para previsão de métricas de recursos computacionais.

2.3.7 *Transformers* em NLP: Modelos BERT e GPT

No campo de Processamento de Linguagem Natural (PLN), a arquitetura *Transformer* rapidamente se consolidou graças à sua capacidade de capturar relações sem ordem fixa e facilitar o pré-treinamento em larga escala. Dois marcos evolutivos foram os modelos BERT e GPT, que adaptaram o *Transformer* para objetivos de treinamento não supervisionados em grandes corpora de texto, alterando a forma de aprendizado das representações linguísticas.

BERT (*Bidirectional Encoder Representations from Transformers*), proposto por Devlin et al.(2018), é um modelo baseado unicamente na parte codificadora do *Transformer*. O BERT utiliza múltiplas camadas encoder (por exemplo, 12 camadas no BERT-Base) para produzir representações contextuais de textos, mas treina essas camadas com uma estratégia de pré-treinamento bidirecional. Diferentemente de modelos autoregressivos, o BERT é treinado em uma tarefa de modelagem de linguagem mascarada (*Masked Language Modeling*): partes do texto de entrada (tipicamente 15% dos tokens) são mascaradas aleatoriamente, e o modelo aprende a prever os tokens originais a partir do contexto bidirecional (ou seja, considerando simultaneamente as palavras à esquerda e à direita da máscara). Adicionalmente, BERT foi treinado com uma tarefa de previsão de próxima sentença (*Next Sentence Prediction*) para ensinar relações entre sentenças. Esse esquema de treino força o codificador *Transformer* a capturar significados ricos de cada posição considerando todo o entorno, resultando em embeddings poderosos que podem ser posteriormente *fine-tuned* em diversas tarefas supervisionadas de NLP. O impacto do BERT foi profundo: ao disponibilizar um modelo pré-treinado com bilhões de palavras, Devlin et al. (2018D

mostraram que bastava ajustar levemente esse modelo em tarefas específicas (como análise de sentimentos, perguntas e respostas, reconhecimento de entidades, etc.) para atingir ou superar o estado da arte de então (DEVLIN et al., 2018). Em suma, o BERT demonstrou que o *Transformer encoder*, quando treinado de forma auto-supervisionada em larga escala, pode servir como modelo fundamental de linguagem, fornecendo representações universais bidirecionais.

Por outro lado, os modelos GPT (*Generative Pre-trained Transformer*) seguiram uma estratégia complementar, focando na geração de texto via decodificadores Transformer. O GPT original de Radford et al. (2018) e suas iterações subsequentes (GPT-2 (RADFORD et al., 2018) e GPT-3 (BROWN, 2020), entre outros) utilizam apenas a pilha de decodificadores do *Transformer*, treinada em um objetivo simples porém poderoso: modelagem de linguagem autoregressiva. Nessa configuração, dado um prefixo de texto, o modelo aprende a prever o próximo token, depois o seguinte e assim por diante, assimilando a distribuição estatística da língua a partir de enormes volumes de dados brutos. Diferentemente do BERT, o GPT não tem acesso ao futuro da sequência durante a previsão de cada token – ele funciona unidirecionalmente, incorporando apenas o contexto à esquerda (passado) para gerar a continuação. Apesar de não usar contexto futuro, quando escalado em tamanho e dados, o GPT demonstrou capacidades surpreendentes de geração de texto fluido e contextual. O GPT-3, por exemplo, com 175 bilhões de parâmetros, mostrou pela primeira vez fortes habilidades de *learning* de poucas instruções (*few-shot learning*), conseguindo executar tarefas de linguagem diversas apenas lendo exemplos na própria entrada, sem ajuste de pesos (BROWN, 2020). Esse feito evidenciou o poder de escalabilidade da arquitetura *Transformer*: modelos muito maiores treinados em quantidades massivas de texto revelaram comportamentos emergentes, resolvendo tarefas complexas não antecipadas explicitamente durante o treino. Em resumo, a família GPT evidenciou o potencial dos *Transformers* como modelos gerativos universais, capazes de produzir linguagem natural coerente e realizar tarefas mediante formulação apropriada na entrada (*prompts*). BERT e GPT, embora diferentes em enfoque (encoder bidirecional vs. decoder autoregressivo), juntos inauguraram a era dos modelos pré-treinados em NLP, dominando o estado da arte em quase todas as tarefas do campo a partir de 2018.

2.3.8 *Transformers* em Visão Computacional: *Vision Transformer* (ViT)

Após conquistas em NLP, a arquitetura *Transformer* também foi adaptada para Visão Computacional. Tradicionalmente, redes neurais convolucionais (CNNs) eram a principal ferramenta para tarefas como classificação de imagens. Contudo, Dosovitskiy et al. (2020) propuseram o *Vision Transformer* (ViT), demonstrando que um *Transformer* puro pode alcançar desempenho de ponta em imagens ao ser treinado com dados suficientes. A ideia do ViT é representar uma imagem como uma sequência de *patches* (fragmentos)

análogos a "palavras" visuais. Especificamente, uma imagem é dividida em *patches* de tamanho fixo (por exemplo, 16×16 pixels), cada *patch* é linearmente projetado a um vetor de dimensão d e enriquecido com um embedding posicional indicando sua localização na grade da imagem. Uma sequência de tais vetores (tipicamente precedida de um token de classificação especial, similar ao [CLS] do BERT) é então alimentada a uma arquitetura *Transformer* semelhante à do codificador original. O ViT processa os patches via camadas de multi-head *self-attention*, permitindo que o modelo relacione partes distantes da imagem livremente. Importante, como as imagens são tratadas como sequências de patches, o ViT também requer codificação posicional (no caso, aprendida) para indicar a posição de cada patch na estrutura 2D da imagem.

Os resultados reportados por Dosovitskiy et al. (2020) mostraram que, quando pré-treinado em um volume muito grande de dados (por exemplo, no dataset JFT-300M ou ImageNet-21k) e posteriormente ajustado em tarefas de visão, o ViT atingiu acurácias comparáveis ou superiores às melhores CNNs, como *EfficientNets*, especialmente em resoluções altas. A conclusão foi que a auto-atenção é capaz de capturar características visuais relevantes distribuídas pela imagem, sem necessidade explícita de convoluções. Essa descoberta abriu caminho para uma série de variantes e aprimoramentos de *Transformers* para visão, incluindo modelos híbridos (convolução + atenção) e aplicações a detecção de objetos, segmentação, vídeo, entre outras. Em suma, o *Vision Transformer* provou que a arquitetura *Transformer* é multidomínio, podendo atuar em dados não sequenciais (como pixels dispostos em uma grade) desde que convertidos a uma representação sequencial apropriada. Esse avanço consolidou ainda mais os *Transformers* como uma família arquitetural unificadora para diferentes modalidades de dados.

2.3.9 *Transformer*s para Séries Temporais e AIOps

Outro domínio de grande interesse para aplicação de *Transformers* é o de séries temporais, que incluem dados provenientes de sensores, métricas de sistemas, finanças, etc. Tarefas como previsão de séries temporais e detecção de anomalias sempre foram desafiadoras, tradicionalmente abordadas por modelos estatísticos (ARIMA, Prophet) ou redes recorrentes e convolucionais especializadas. A natureza sequencial e a presença de padrões de longo prazo (como sazonalidades anuais, tendências) sugerem que a capacidade dos *Transformer*s de modelar dependências arbitrariamente longas pode ser altamente benéfica para séries temporais. De fato, pesquisadores têm adaptado e estendido a arquitetura *Transformer* para melhor adequação a dados temporais, enfrentando desafios particulares desse domínio, como sequências muito mais longas que textos e a necessidade de incorporar informações de tempo contínuo.

Adaptações Gerais: Aplicar *Transformer*s diretamente em séries temporais exige alguns ajustes. Primeiramente, muitas vezes as entradas não são sequências de tokens

discretos, e sim sequências de valores contínuos (por exemplo, medições numéricas ao longo do tempo). Uma solução comum é considerar janelas de tempo fixas como "sequências" de entrada e predição. Assim, um *Transformer* de previsão recebe uma janela de histórico (por exemplo, valores dos últimos N instantes) e aprende a prever os próximos M valores futuros análogo a uma tarefa seq2seq, em que a sequência-fonte é o passado e a sequência-alvo é o futuro. Em segundo lugar, é crucial fornecer ao modelo algum sinal de temporalidade: além de usar codificações posicionais para a ordem dos pontos na janela, costuma-se adicionar *features* temporais explícitas, como indicadores de hora do dia, dia da semana, etc., para ajudar o modelo a capturar padrões sazonais conhecidos. Essas *features* atuam como uma forma de codificação posicional informada pelo conhecimento de calendário. Por exemplo, o *Time Series Transformer* implementado pela HuggingFace insere atributos como mês e dia diretamente como entradas adicionais ao *Transformer*, tratando-os como embeddings junto aos valores da série (RASUL et al., 2024). Outra consideração é que muitas aplicações de séries temporais demandam previsões probabilísticas (intervalos de confiança, distribuições) em vez de apenas um ponto previsto. Abordagens baseadas em *Transformer* têm incorporado cabeças de saída que produzem parâmetros de distribuições (como média e desvio de Gaussiana) ou quantis (RASUL et al., 2024), permitindo estimar incertezas – o que é valioso em cenários como planejamento de capacidade e detecção de anomalias.

Desafios e Soluções para Longas Sequências: Um impedimento técnico dos *Transformers* em séries temporais está no custo quadrático da atenção em relação ao comprimento da sequência. Séries podem conter centenas ou milhares de passos relevantes, tornando proibitivo aplicar *self-attention* direta em sequências muito longas. Para mitigar isso, várias variantes especializadas foram propostas. Por exemplo, o Informer (ZHOU et al., 2021) introduziu um mecanismo de atenção esparsa (*probSparse attention*) que seleciona apenas partes mais informativas do espectro de atenção, reduzindo drasticamente o custo para longas janelas de previsão. Já o Autoformer (WU et al., 2021) incorporou um decompositor de séries temporais dentro do modelo, separando componentes de tendência e sazonalidade e aplicando atenção de forma mais estruturada. Mais recentemente, o PatchTST (*Patch Time Series Transformer*) (NIE et al., 2023) trouxe a ideia de dividir séries temporais em patches (sub-séries) análoga ao ViT: o modelo extrai trechos consecutivos da série em cada variável como unidades básicas (patches) e processa cada variável de forma independente (*channel-independent*), antes de eventualmente combinar informações entre variáveis. Essa abordagem demonstrou ganhos notáveis em previsão de longo prazo, pois captura padrões locais em cada série através dos patches e evita a necessidade de atenção global sobre sequências excessivamente longas (NIE et al., 2023). Muitas dessas inovações atacam também a questão de correlação entre múltiplas séries (séries multivariadas): enquanto alguns modelos optam por ignorar correlações treinando um *Transformer* separado para cada variável (como no modo *channel-independent* do PatchTST), outros permitem misturar

canais (variáveis) através de camadas de atenção especializadas ou combinações tardias das representações.

Exemplo – Time Series Transformer (HuggingFace): A biblioteca *HuggingFace Transformers* incorporou um modelo genérico chamado TimeSeriesTransformer (RASUL et al., 2024), que exemplifica a adaptação direta do *Transformer* para previsão de séries temporais. Trata-se de um modelo encoder–decoder padrão, semelhante ao *Transformer* de Vaswani, aplicado a janelas de séries. O codificador recebe os últimos N pontos de uma série (histórico), enquanto o decodificador gera os próximos M pontos futuros. Durante o treinamento, utiliza-se *teacher forcing*: os valores reais mais recentes são fornecidos passo a passo ao decodificador para predição do próximo, análogo ao que se faz em tradução. Para informar o modelo sobre a posição temporal de cada ponto, em vez de usar exclusivamente as senoides de posicional, o modelo da HuggingFace incorpora *features* temporais explícitas (como mencionado acima) concatenadas à entrada, por exemplo indicando o mês, dia ou hora correspondentes a cada valor (RASUL et al., 2024). Também é possível incluir variáveis exógenas (como indicadores de feriados, métricas correlatas, etc.) e identificadores estáticos (por exemplo, um ID de servidor no caso de múltiplas séries de diferentes máquinas) como parte da entrada ao *Transformer*. Assim, o TimeSeriesTransformer se configura como um modelo global treinado em múltiplas séries, capaz de aprender padrões compartilhados. Notavelmente, a implementação do HuggingFace suporta previsão probabilística, não produzindo diretamente um escalar para cada passo futuro, mas sim parâmetros de uma distribuição (por exemplo, média e escala de uma distribuição Student-t) da qual se pode amostrar predições (RASUL et al., 2024). Isso o alinha com práticas modernas de previsão incerta, ao contrário de apenas previsão pontual. Em termos de estrutura, porém, esse modelo permanece próximo ao *Transformer* original, comprovando a flexibilidade da arquitetura: sem modificar fundamentalmente o mecanismo de atenção, ele pode ser aplicado a dados temporais contanto que estruturemos apropriadamente as entradas e saídas.

Exemplo – Granite Time Series (IBM): Além de implementações genéricas, surgiram modelos *Transformer* especializados para séries temporais com foco em AIOps. Um exemplo de destaque é a família Granite Time Series, desenvolvida pela IBM Research e disponibilizada como modelos abertos e pré-treinados (RESEARCH, 2023). Os modelos Granite TS representam uma abordagem de modelos fundamentais (*foundation models*) para séries temporais: são modelos pré-treinados em larga escala, capazes de serem refinados para diversos cenários de previsão. Em particular, a versão conhecida como *Granite Time Series Tiny Time Mixer* (TTM) contém apenas cerca de 1 a 3 milhões de parâmetros, mas já vem pré-treinada para prever séries multivariadas, permitindo uso imediato ou *fine-tuning* eficiente. A arquitetura do Granite TTM combina ideias de patches e mixing de canais: ele suporta tanto o modo *channel-independent* (onde cada métrica é processada separadamente, capturando padrões individuais) quanto um modo de mistura de canais no

decodificador (*decoder channel-mixing*), em que durante o ajuste fino o modelo aprende a captar correlações entre diferentes séries (por exemplo, entre métricas de CPU e memória de um servidor) (RESEARCH, 2023). Essa capacidade de alternar entre tratar cada série isoladamente e combinar informações de múltiplas séries é valiosa em AIOps, pois algumas métricas podem ter comportamentos independentes enquanto outras apresentam inter-relações fortes.

Outra característica importante dos Granite models é a incorporação de metadados temporais e contextuais como parte do *prompt* de entrada. Como descrito na documentação (RESEARCH, 2023), em vez de entradas textuais livres, os *prompts* para modelos de séries temporais incluem informações estruturais sobre os dados (por exemplo, frequência de coleta, quais colunas são targets ou exógenas, identificação do sistema) juntamente com a janela de valores históricos. Isso permite que o modelo pré-treinado aproveite conhecimento geral sobre padrões de determinadas frequências ou tipos de métricas. Os modelos Granite foram treinados em um conjunto amplo de séries temporais e disponibilizados sob licença *open source*, e relatórios indicam que eles alcançam desempenho de ponta em *benchmarks* de previsão com uma pegada computacional muito menor do que redes profundas tradicionais (RESEARCH, 2023). Em resumo, o Granite Time Series exemplifica a convergência de *Transformer*s com as necessidades de AIOps: oferece modelos enxutos, eficientes em CPU, mas capazes de previsão acurada de longo prazo, prontos para serem integrados em pipelines de monitoramento de TI.

2.3.10 *Transformers* e AIOps

As adaptações acima não apenas avançam o estado da arte acadêmico, mas têm aplicação prática direta em AIOps. Em ambientes de operações de TI, prever o comportamento de métricas como utilização de CPU, memória, tráfego de rede, etc., é crucial para detecção proativa de anomalias, prevenção de incidentes e otimização de recursos. Modelos baseados em *Transformer* podem aprender padrões complexos dessas métricas – por exemplo, capturar ciclos diários de uso, picos semanais ou correlações entre serviços – melhor do que modelos lineares tradicionais. Além disso, a habilidade de considerar longos históricos permite antecipar eventos raros ou tendências graduais. Já há relatos de uso de Transformers para *anomaly detection* em métricas de data centers (XU et al., 2022; TULI; CASALE; JENNINGS, 2022), confirmando sua adequação ao domínio de AIOps. Os exemplos do *Time Series Transformer* e do Granite TS mostram duas abordagens complementares: uma abordagem genérica de uso amplo e outra especializada e otimizada, respectivamente, ambas alinhadas ao objetivo de melhorar a confiabilidade e eficiência de operações de TI por meio de previsão inteligente.

2.3.11 Aplicação no Trabalho Atual: Previsão de Métricas com *Transformer* Simplificado

Os avanços discutidos acima servem de fundamento para a proposta do presente trabalho. Nesta dissertação, foi empregada uma versão simplificada da arquitetura *Transformer* para abordar o problema real de previsão de métricas de recursos computacionais coletadas via *Prometheus* (uma ferramenta de monitoramento que registra séries temporais de métricas de infraestrutura). Em vez de utilizar modelos de grande porte ou altamente complexos, optou-se por um design enxuto inspirado no *Transformer* original, adequado à escala dos dados disponíveis e às necessidades de AIOps da organização em questão.

Concretamente, o modelo desenvolvido mantém os componentes-chave do *Transformer* – em especial o mecanismo de atenção *self-attention* – porém com complexidade reduzida. Foram utilizadas menos camadas (profundidade menor) e menos cabeças de atenção por camada, diminuindo o número de parâmetros. Essa simplificação visa facilitar o treinamento com volumes limitados de dados de métricas e viabilizar a implantação em cenários onde recursos de processamento são restritos. Além disso, certas adaptações específicas ao domínio foram incorporadas. Por exemplo, como as métricas do *Prometheus* possuem periodicidades conhecidas (padrões diários e semanais de uso de CPU, memória, etc.), o modelo integra codificações temporais customizadas e *features* de calendário (similares às usadas no *Time Series Transformer da HuggingFace*). Isso reforça a capacidade do modelo de capturar sazonalidades típicas de cargas de trabalho de TI. Adicionalmente, devido ao foco em previsão de curto a médio prazo para detecção de anomalias, privilegiou-se janelas de entrada relativamente curtas e um horizonte de predição compatível com a detecção proativa de incidentes (por exemplo, prever a próxima hora com granularidade de minutos).

A aplicação do *Transformer* simplificado mostrou-se eficaz em modelar as séries de métricas. Mesmo com menos camadas, o mecanismo de auto-atenção conseguiu aprender relações relevantes entre pontos no histórico de cada métrica por exemplo, identificando que a alta utilização de CPU em determinados horários está correlacionada com padrões de trabalho humanos, ou que certos picos de memória precedem aumentos de uso de *swap*. O modelo aprendeu também a diferença de comportamento entre diferentes hosts ou serviços (quando métricas de múltiplas fontes foram modeladas conjuntamente), graças à inclusão de identificadores estáticos no embedding de entrada. Em suma, os princípios delineados pelos *Transformers* de última geração foram aplicados em menor escala para solucionar um problema de mundo real em AIOps. Esta escolha metodológica foi guiada pela necessidade de balancear precisão preditiva e simplicidade: ao aproveitar a expressividade do mecanismo de atenção para capturar dependências temporais, mas limitando a capacidade do modelo para evitar sobreajuste e reduzir custo computacional, foi possível desenvolver um preditor de métricas que se integra facilmente ao ecossistema Prometheus. Os resultados, discutidos

em capítulos posteriores, indicam que mesmo uma versão simplificada do *Transformer* supera modelos tradicionais de previsão em diversos cenários de métricas de recursos, corroborando a relevância da arquitetura *Transformer* – e de suas variantes – como ferramenta fundamental em AIOps.

Por fim, vale ressaltar que a implementação desenvolvida dialoga com os trabalhos recentes citados: embora não alcance a sofisticação de um Granite Time Series pré-treinado ou de modelos especializados como Informer/Autoformer, ela demonstra na prática a flexibilidade da arquitetura *Transformer* em ser adaptada a diferentes contextos. A simplificação orientada pelo domínio reforça a ideia de que compreendendo-se os componentes essenciais (atenção, posicionamento temporal, etc.), pode-se moldar o *Transformer* às restrições e objetivos de cada aplicação. Este trabalho, portanto, insere-se nessa linha evolutiva, aplicando conceitos da literatura de *Transformers* de forma inovadora a um problema específico de previsão de séries temporais em operações de TI.

2.4 Algoritmo MRFO

O algoritmo MRFO é um método de otimização meta-heurístico bio-inspirado que imita os comportamentos de forrageamento de arraias manta em busca de alimento (ZHAO; WANG; ZHANG, 2020). Sua principal motivação teórica é equilibrar exploração global e exploração local do espaço de busca por meio de diferentes estratégias inspiradas em padrões de movimentação das arraias. Especificamente, o MRFO emprega três fases distintas de busca – *chain foraging*, *cyclone foraging* e *somersault foraging* – que atuam de forma complementar para evitar convergência prematura em ótimos locais (ZHAO; WANG; ZHANG, 2020). Em cada iteração, cada agente (solução candidata) pode executar aleatoriamente uma dessas estratégias, promovendo diversidade e intensificação adaptativas na busca pela solução ótima global (ZHU et al., 2023). A seguir, é descrito em detalhe cada fase juntamente com suas equações características e fundamentos teóricos:

- **Chain Foraging**: as arraias formam uma cadeia para capturar plâncton cooperativamente, seguindo o líder. No algoritmo, essa cooperação é modelada atualizando-se a posição de cada agente com influência tanto do melhor agente global quanto do agente vizinho anterior na cadeia. A atualização é dada por:

$$X_i^{t+1} = X_i^t + \alpha, R_k \left(X_{\text{best}}^t - X_i^t \right) + \beta, R_k \left(X_{i-1}^t - X_i^t \right), \tag{5}$$

  onde X_i^t é a posição do i-ésimo agente na iteração t, X_{best}^t é a melhor solução encontrada até t, e X_{i-1}^t representa a posição do agente precedente. O termo $R_k = \text{diag}(r_{k,1}, \ldots, r_{k,D})$ é uma matriz diagonal de números aleatórios uniformes $r_{k,d} \sim U(0,1)$ em cada dimensão d, introduzindo aleatoriedade element-wise. Os parâmetros α e β são coeficientes de peso (tipicamente $\alpha = 1.5$ e $\beta = 1.0$ (ZHAO; WANG;

ZHANG, 2020)) que controlam a contribuição relativa do melhor global e do vizinho na movimentação. Essa fase promove uma exploração orientada pela melhor solução, ao mesmo tempo em que encoraja cooperação local entre agentes adjacentes, evitando que todos os agentes se movam estritamente em direção ao ótimo global de forma sincronizada.

- **Cyclone Foraging**: as arraias nadam em um movimento espiralado semelhante a um ciclone enquanto convergem em direção à presa. No MRFO, essa estratégia é modelada fazendo os agentes espiralarem em torno da melhor posição atual, conforme:

$$X_i^{t+1} = X_{\text{best}}^t + \gamma_t \, R_\theta \Big(X_{\text{best}}^t - X_i^t \Big) \,, \tag{6}$$

  onde $\gamma_t = 2\left(1 - \frac{t}{T_{\max}}\right)$ é um fator que decresce linearmente com as iterações (sendo $T_{\max}$ o número máximo de iterações) e R_θ é uma matriz de rotação aleatória que impõe um movimento helicoidal em torno de X_{best}^t (ZHAO; WANG; ZHANG, 2020). Intuitivamente, γ_t controla o passo de espiral: no início (t pequeno), $\gamma_t \approx 2$, permitindo movimentos mais amplos (exploração); no final da otimização, $\gamma_t \to 0$, de modo que os agentes se aproximam delicadamente do ótimo (exploração local refinada). Essa fase foca em intensificar a busca nas imediações do melhor indivíduo corrente, aprimorando a exploração local de forma análoga ao refinamento em métodos de otimização tradicionais.

- **Somersault Foraging**: próximo a fontes abundantes de alimento, arraias manta dão cambalhotas (*somersaults*) para se reposicionar eficientemente ao redor da fonte. No MRFO, essa manobra é traduzida em um salto aleatório em torno da melhor solução global, com o objetivo de escapar de ótimos locais e diversificar a busca. A atualização é descrita por:

$$X_i^{t+1} = X_{\text{best}}^t + \phi \left(X_{\text{best}}^t - X_i^t \right), \tag{7}$$

  em que $\phi \sim U(-1, 1)$ é um coeficiente aleatório escalar no intervalo $[-1, 1]$. Essa equação reposiciona o agente i ao redor de X_{best}^t em uma direção aleatória, potencialmente encontrando novas regiões promissoras de busca. A *somersault* atua como um mecanismo exploratório intensivo, introduzindo saltos não direcionados que aumentam a diversidade populacional e reduzem a chance de aprisionamento prematuro em mínimos locais (ZHAO; WANG; ZHANG, 2020).

O algoritmo completo do MRFO está descrito no Pseudocódigo 1. Inicialmente, uma população de N soluções é gerada aleatoriamente no espaço de busca Ω (considerado contínuo de dimensão D) e avaliada segundo a função objetivo $f(\cdot)$. Em cada iteração t, cada agente i seleciona aleatoriamente uma das três estratégias de forrageamento (por exemplo, com probabilidades pré-definidas de 40% para *chain*, 40% para *cyclone* e 20%

para *somersault*, conforme adotado neste trabalho) e atualiza sua posição usando a equação correspondente. Após o movimento, aplica-se uma correção de limites para garantir que $X_i^{t+1} \in \Omega$ (e.g., truncamento ou *reflection* caso a nova posição exceda os limites do domínio). Em seguida, calcula-se $f(X_i^{t+1})$ e, se o valor for melhor que o da melhor solução conhecida X_{best}^t, atualiza-se X_{best}. Ao final de cada iteração, pode-se adotar um critério de parada adicional: por exemplo, parar antecipadamente se a melhoria do $f(X_{\text{best}})$ de uma iteração para a próxima for insignificante (abaixo de um limiar ϵ, como 10^{-4}). Sob condições adequadas de decaimento de passo (como o de γ_t na Eq. 6), pode-se demonstrar que a sequência estocástica gerada pelo MRFO satisfaz condições de convergência quase certa para o ótimo global (ZHAO; WANG; ZHANG, 2020). Ademais, a combinação das fases *cyclone* e *somersault* garante uma probabilidade crescente de escapar de ótimos locais ao longo das iterações (ZHAO; WANG; ZHANG, 2020), reforçando a exploração global mesmo nas etapas finais da busca.

Algoritmo 1: Pseudocódigo do algoritmo Manta Ray Foraging Optimization

Entrada: Tamanho da população N; número máximo de iterações $T_{\max}$; coeficientes α, β, γ_0
Saída: Melhor solução encontrada X_{best}

```
Inicializar população {X_i^0}_{i=1}^N com X_i^0 ~ U(Ω) e avaliar f(X_i^0);
X_best ← arg min_i f(X_i^0)                          ▷ *melhor solução inicial
Para t ← 1 até T_max fazer
    Para i ← 1 até N fazer
        Selecionar aleatoriamente a fase de forrageamento;
                ▷ *[r]chain (40%), cyclone (40%) ou somersault (20%)
        Atualizar posição X_i^{t+1} pela equação correspondente;
                                                        ▷ *[r]Eq. 5, 6 ou 7
        Projetar X_i^{t+1} em Ω                         ▷ *[r]restrições de domínio
        Avaliar f(X_i^{t+1});
        Se f(X_i^{t+1}) < f(X_best) então
            X_best ← X_i^{t+1}                          ▷ *[r]atualiza melhor global
    Se |f(X_best^t) − f(X_best^{t−1})| < ε então
        Break                ▷ *[r]critério de convergência (melhoria desprezível)
devolver X_best
```

Em suma, o MRFO distingue-se por incorporar explicitamente mecanismos complementares de busca: enquanto *chain* e *cyclone* foraging guiam os agentes em direção às regiões promissoras identificadas (exploração direcionada e intensificação local), o *somersault* foraging ocasionalmente dispersa os agentes ao redor do melhor ponto conhecido, introduzindo variabilidade adicional que ajuda a escapar de armadilhas de ótimo local. Esse equilíbrio dinâmico entre exploração e explotação confere ao MRFO vantagens sobre meta-heurísticas clássicas. Estudos comparativos reportam que o MRFO tende a alcançar

soluções de qualidade superior e com convergência mais rápida do que algoritmos como *Particle Swarm Optimization* (PSO) e Algoritmos Genéticos em diversos problemas de otimização contínua (ZHU et al., 2023; ALTURKI et al., 2020).

3 Metodologia

Este capítulo detalha, com profundidade, o protocolo experimental desenvolvido nesta dissertação. Além de garantir transparência e reprodutibilidade, buscou-se justificar cada escolha de desenho da coleta dos dados à avaliação estatística sob a ótica de boas práticas em aprendizado de máquina aplicado a *AIOps*.

3.1 Conjuntos de Dados

3.1.1 Google Cluster Traces (2019)

O *Google Cluster Traces 2019* (GCT-19) disponibiliza métricas de consumo de CPU e memória de milhares de tarefas em datacenters do Google ao longo de 30 dias contínuos (REISS; WILKES; HELLERSTEIN, 2021). O presente estudo foca na métrica `avg_cpu_usage`, amostrada a cada (1) minuto. Comparado a *benchmarks* tradicionais (Yahoo Webscope, NAB), o GCT exibe alto volume (~43 000 amostras/host) *e* variabilidade intra-equipe, desafiando preditores univariados (CORTEZ; SILVA; ROSALEM, 2023).

3.1.2 *Prometheus* Metrics

O segundo dataset deriva de métricas coletadas via *Prometheus* em um servidor Linux (Intel Xeon E5-2699 v4, 128 GB RAM) que hospeda contêineres Red Hat OpenShift. Os kpi monitorados incluem: (i) uso de memória em MB, (ii) carga média de CPU e (iii) latência de rede. Para uniformidade com o GCT, foi empregado a métrica de *memory-usage* em MB e período de coleta de 1 min. O Prometheus traz ~10 000 amostras e exibe forte sazonalidade diária, típico de workloads monolíticos (BARHAM, 2022).

Tabela 2 – Estatísticas descritivas iniciais.

Dataset	n	Média	σ	Mín	Máx
GCT-19 CPU (%)	43 200	0.245	0.061	0.012	0.397
Prom Mem (MB)	10 080	812	87	621	1 012

3.1.3 Pipeline de Pré-processamento

O preparo das séries segue um fluxo determinístico de **seis** fases, cada qual encadeada de modo a minimizar *data leakage* e a preservar padrões sazonais relevantes.

1. **Aquisição de Séries Brutas.** Métricas coletadas a cada 1 min—`avg_cpu_usage` no *Google Cluster Traces 2019* e `memory_usage_MB` via *Prometheus* são ingeridas tal como armazenadas, mantendo picos e ruído instrumental.

2. **Sanitização de Outliers.** Amostras anômalas são detectadas por um critério híbrido (Z-score $|z| > 3$, percentis adaptativos $P_{0.5}$–$P_{99.5}$ e IQR×1.5) (HEWAMALAGE; BERGMEIR; BANDARA, 2021). Valores marcados por *qualquer* teste são imputados por interpolação cúbica (`pandas.Series.interpolate(method='cubic')`), preservando suavidade da primeira derivada. O procedimento elimina apenas 0.4 %–0.6 % dos pontos.

3. **Reamostragem e Agregação.** Para reduzir ruído de curto prazo sem perder tendências operacionais, cada série é reamostrada em 1 min, 5 min e 30 min e agregada pela *média* (LAPTEV et al., 2015). Picos sub-minuto, que não alteram decisões de *capacity planning*, são assim suavizados.

4. **Escalonamento Min–Max.** Cada sequência é normalizada para $[0, 1]$, $x'_t = (x_t - x_{\min})/(x_{\max} - x_{\min})$, estabilizando gradientes iniciais e evitando saturação de portas sigmóides em LSTM/Transformer (BOROVYKH; BOHTE; OOSTERLEE, 2017). A inversão da escala é aplicada *apenas* ao calcular métricas em teste.

5. **Construção de Janelas Deslizantes.** Históricos normalizados geram pares $(\mathbf{x}_t^{(n)}, x_{t+1})$ via janela deslizante com $n \in \{24, 48, 72, 96\}$ passos—equivalente a 12–48 h em 30 min. O resultado é um tensor batch $\times\, n \times 1$ pronto para treino (WU et al., 2021).

6. **Particionamento Temporal 60/20/20 %.** A divisão cronológica impede vazamento de informação futura; validação alimenta *early-stopping* e o ajuste hiper-paramétrico via MRFO, enquanto o teste é consultado uma única vez.

Decisões de limpeza, reamostragem e divisão, tomadas em conjunto, equilibram **robustez estatística** (remoção de outliers), **eficiência computacional** (granularidades múltiplas) e **fidelidade operacional** (manutenção da sazonalidade), fornecendo uma base confiável para os modelos preditivos descritos a seguir.

3.2 Formulação Matemática do Problema

Considere uma sequência temporal univariada $x_t \in \mathbb{R}$. Foi definido uma janela de entrada

$$\mathbf{x}_t^{(n)} = \Big[x_{t-n+1}, \ldots, x_t\Big] \in \mathbb{R}^n,$$

com $n \in \{24, 48, 72, 96\}$ correspondendo a 12–48 h nos cenários 30 min. O objetivo é aprender um mapeamento

$$f_\theta : \mathbb{R}^n \longrightarrow \mathbb{R}, \quad \hat{x}_{t+1} = f_\theta\Big(\mathbf{x}_t^{(n)}\Big)$$

que minimize uma função de perda $\mathcal{L}$ (RMSE ou MAE) nos dados de validação cruzada. Optou-se por predição *one-step-ahead* para comparação justa com trabalhos relacionados (SEN et al., 2019), porém mostramos, nos Resultados, extensão para horizontes $h \leq 12$ passos.

3.3 Modelo LSTM

3.3.1 Arquitetura

A rede desenvolvida nos experimentos é composta por duas camadas LSTM com u unidades e dropout p, seguidas de camada densa linear. O modo *stateful* foi desativado a fim de permitir embaralhamento de lotes e acelerar a convergência.

3.3.2 Complexidade

O custo assintótico por *epoch* é $\mathcal{O}\Big(M\,(4u^2 + un)\Big)$, onde M é o número de janelas e n o comprimento da sequência.

3.3.3 Hiper-parâmetros

Para cada hiper-parâmetro da LSTM, é apresentado a seguir: (i) definição operacional, (ii) impacto qualitativo sobre a capacidade de modelagem, regularização ou custo computacional e (iii) intervalo efetivamente explorado pelo . A Tabela 3 resume os limites; a discussão textual subsequente aprofunda o papel de cada variável.

Tabela 3 – Limites de busca dos hiper-parâmetros da LSTM.

Hiper-parâmetro	**Min**	**Max**	**Tipo**
u (*lstm_units*)	32	256	int
p (*dropout*)	0.10	0.50	real
η (*learning rate*)	10^{-4}	10^{-2}	log
b (*batch size*)	32	256	cat
n_{steps}	24	96	int
E (*epochs*)	50	200	int

Unidades LSTM (u). Quantidade de células de memória por camada. Valores maiores aumentam a capacidade de capturar padrões complexos e dependências de longo prazo, porém elevam quadraticamente o custo $\mathcal{O}(u^2)$ e podem induzir *overfitting* em séries de pequena variância.

Dropout (p). Probabilidade de desativar ativação durante o treinamento. Taxas mais altas (≥ 0.3) forçam a rede a aprender representações redundantes, reduzindo *overfitting*, mas podem retardar a convergência e sub-aprender dependências longas.

Taxa de aprendizado (η). Passo do gradiente no otimizador *RMSprop.*:

- Valores altos (1×10^{-2}) aceleram a descida, mas podem divergir.
- Valores baixos (1×10^{-4}) estabilizam treinamento, exigindo mais épocas para atingir o platô.

Tamanho do lote (b). Número de janelas processadas antes da atualização dos pesos. Lotes pequenos (32) introduzem ruído benéfico ao gradiente (melhor *generalização*) e consomem menos memória; lotes grandes (256) suavizam a curva de erro, porém podem convergir para mínimos rasos.

Comprimento da janela ($n_{\mathbf{steps}}$). Quantidade de passos históricos usados como entrada. Janelas longas (96 passos @ 30 minutos) capturam até quatro ciclos diários completos, mas ampliam a dimensionalidade de entrada e demandam mais unidades ou camadas.

Épocas (E). Iterações completas sobre o conjunto de treino. Mais épocas refinam os pesos até saturação — útil quando p é alto —, mas o ganho marginal tende a zero; sobreajuste surge se o *early stop* não for aplicado.

Em síntese, o equilibra a busca nesses seis eixos para maximizar a performance (RMSE mínima) enquanto controla custo de treino.

3.4 Modelo Transformer para Séries Temporais

3.4.1 Definição

Foi adotado um encoder-only com $L = 2$ camadas, $d_{model} = 128$, $h = 4$ cabeças, FFN interno $d_{ff} = 4d_{model}$, dropout 0.1 e *positional encoding* senoidal. Todos os hiperparâmetros foram ajustados via MRFO, conforme o protocolo descrito nos capítulos a seguir.

3.4.2 Atenção Escalonada

Para entrada $\mathbf{X} \in \mathbb{R}^{n \times d_{model}}$, a atenção multi-cabeça é

$$\mathrm{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \Big[\mathbf{H}_1 \| \dots \| \mathbf{H}_h\Big] \mathbf{W}^O,$$

com $\mathbf{H}_i = \mathrm{softmax}\Big(\frac{\mathbf{Q}_i \mathbf{K}_i^\top}{\sqrt{d_k}}\Big) \mathbf{V}_i$ (LUONG; PHAM; MANNING, 2015). O custo é $\mathcal{O}(h\, n^2\, d_k)$, porém $n \leq 96$ mantém a latência de inferência abaixo de 2 ms/janela.

3.4.3 Prevenção de Overfitting

(a) **Label smoothing** $\varepsilon = 0.1$;

(b) ℓ_2 **regularização** $\lambda = 10^{-5}$;

(c) **Cosine LR schedule** (LOSHCHILOV; HUTTER, 2017).

3.4.4 Hiper-parâmetros

Assim como na LSTM (§3.3.3), o explora um espaço de busca multidimensional a fim de minimizar o RMSE de validação. A Tabela 4 lista somente *nome, faixa e tipo*, enquanto os parágrafos seguintes discutem a influência qualitativa de cada variável.

Tabela 4 – Limites de busca dos hiper-parâmetros do Transformer.

Hiper-parâmetro	**Min**	**Max**	**Tipo**
d_{model}	64	256	int
L (camadas)	1	4	int
h (cabeças)	2	8	int
dropout	0.10	0.30	real
η (learning rate)	10^{-4}	5×10^{-3}	log
batch	32	256	cat
E (epochs)	50	200	int

Dimensão do embedding (d_{model}). Controla a riqueza semântica dos vetores de entrada. Dimensões altas (≥ 192) capturam padrões sutis, mas aumentam quadraticamente o custo de atenção e podem induzir *overfitting* se o conjunto de dados for modesto.

Profundidade (L). Número de camadas encoder empilhadas. Camadas adicionais favorecem modelagem hierárquica de dependências de longo alcance; contudo, cada salto de L dobra o consumo de memória e pode causar *degradation* caso o gradiente se dissipe.

Número de cabeças (h). Cabeças múltiplas permitem que o modelo foque em regimes sazonais distintos. O aumento de h melhora a capacidade de generalização até o ponto em que cada cabeça opera com dimensão d_k muito baixa, perdendo discriminabilidade.

Dropout. Probabilidade de zerar ativações na atenção e na FFN. Valores acima de 0.25 reduzem *overfitting*, mas podem eliminar informação crítica em séries de alta frequência, exigindo mais épocas para convergir.

Taxa de aprendizado (η). Passo do gradiente no *AdamW*. Valores próximos a 5×10^{-3} aceleram a busca, mas aumentam o risco de oscilação na fase final; taxas baixas garantem estabilidade, porém requerem regeneração de *warm-up* mais longa.

Tamanho do lote (*batch*). Lotes grandes suavizam o gradiente até 30 % e favorecem uso de GPU, mas podem levar a generalização inferior em torno de mínimos rasos. Lotes pequenos (≤ 64) aumentam o ruído estocástico e preservam diversidade dos dados.

Épocas (E). Iterações completas de treinamento. Na prática, convergência típica ocorre entre 80 e 120 épocas; acima disso, o ganho marginal é ínfimo e a diferença estatística dilui-se pelo *early stop*.

Em conjunto, esses sete eixos definem a superfície de otimização do Transformer. O MRFO busca combiná-los de modo a capturar sazonalidades diárias sem incorrer em custos proibitivos de inferência.

3.5 Meta-Otimização via MRFO

3.5.1 Motivação

Estudos recentes mostram que algoritmos bioinspirados superam random/grid search em problemas de alto custo de avaliação (EIBEN; SMITH, 2019). Foi escolhido o MRFO devido à sua habilidade de escapar de ótimos locais usando perturbações longitudinais (*somersault*), especialmente eficaz em hiper-parâmetros contínuos.

3.5.2 Configuração

Parâmetros globais: $N_{\text{mantas}} = 30$, $T_{\max} = 100$ (LSTM) e 500 (Transformer), coeficiente de salto $S = 1.2$, *early stop* paciência 7.

O espaço de busca *inferido* para LSTM está na Tabela 3, e para a arquitetura Transformers Tabela 4.

Para garantir replicabilidade, as *seeds* 42, 1229 e 2025 foram testadas; relatou-se o melhor resultado médio.

3.6 Ambiente Experimental

3.6.1 Hardware

- **GPU** NVIDIA A100 80 GB, CUDA 12.4, cuDNN 8.9
- **CPU** Apple M3 Max 14-core (validação), 36 GB RAM

3.6.2 Software

- Python 3.10, PyTorch 2.3, NumPy 1.26, Pandas 2.2
- MLflow 2.11 (indisponível por falha de SSD; ver §**??**)
- Matplotlib 3.9 para visualização; LaTeX TeX Live 2024

3.7 Métricas de Avaliação

A qualidade preditiva dos modelos foi aferida por quatro métricas de erro amplamente utilizadas em séries temporais: raiz do erro quadrático médio (RMSE), erro absoluto médio (MAE), erro percentual absoluto médio (MAPE) e erro percentual absoluto médio simétrico (SMAPE).[13]

3.7.1 Definições formais

Considere o conjunto de observações reais $\{y_t\}_{t=1}^{T}$ e as previsões correspondentes $\{\hat{y}_t\}_{t=1}^{T}$. Denotando $e_t = \hat{y}_t - y_t$, têm-se:

$$\text{RMSE} = \sqrt{\frac{1}{T}\sum_{t=1}^{T} e_t^2}, \tag{8}$$

$$\text{MAE} = \frac{1}{T}\sum_{t=1}^{T} |e_t|, \tag{9}$$

$$\text{MAPE} = \frac{100}{T}\sum_{t=1}^{T} \left|\frac{e_t}{y_t}\right|, \tag{10}$$

$$\text{SMAPE} = \frac{100}{T}\sum_{t=1}^{T} \frac{|e_t|}{(|y_t| + |\hat{y}_t|)/2}. \tag{11}$$

3.7.2 Interpretação prática

- **RMSE**: enfatiza erros maiores, sendo apropriado quando desvios significativos podem provocar degradação aguda de *Service-Level Agreements* (SLAs).
- **MAE**: provê leitura clara em unidades originais (MB, % CPU) e oferece maior robustez a valores extremos.
- **MAPE**: fornece erro relativo adimensional, mas é indefinido para $y_t = 0$. Tal limitação é mitigada nos dados presentes, pois os KPIs analisados não assumem zero absoluto.
- **SMAPE**: corrige a assimetria do MAPE em séries onde $|y_t| \approx 0$, garantindo valores no intervalo $[0, 200]\,\%$ (WILLMOTT; MATSUURA, 2005).

[13] O RMSE penaliza desvios grandes de forma quadrática e, portanto, é sensível a outliers; o MAE confere peso linear ao erro, sendo mais robusto em séries com picos esporádicos; MAPE e SMAPE normalizam o erro pela magnitude do valor real, viabilizando comparação entre séries de escalas diferentes.

3.7.3 Comparação estatística

Para avaliar diferenças significativas entre pares de configurações (por exemplo, LSTM versus *Transformer*), emprega-se o **teste de Wilcoxon signed-rank**[14] sobre as séries de erros obtidas em *k-fold* temporal. Os respectivos valores-p e tamanhos de efeito são reportados no Capítulo 4, assegurando transparência nas inferências.

Em conjunto, as Equações 8–11 e o protocolo estatístico descrito constituem a base quantitativa para a análise apresentada nos resultados.

[14] Teste não paramétrico para duas amostras dependentes que avalia se as distribuições das medianas diferem; adota-se $\alpha = 0{,}05$ (*two-sided*). O tamanho do efeito é calculado por $r = Z/\sqrt{N}$, onde Z é o escore padronizado e N o número de pares, possibilitando interpretar a magnitude da diferença (pequeno: $r < 0{,}3$; médio: $0{,}3 \leq r < 0{,}5$; grande: $r \geq 0{,}5$).

4 Resultados

Este capítulo apresenta, de forma sistemática e comparativa, os resultados obtidos a partir dos experimentos conduzidos com as arquiteturas LSTM e *Transformer* para previsão de consumo de recursos computacionais. As análises consideram quatro granularidades temporais (1, 5, 15 e 30 min) em dois *datasets* distintos: *Google Cluster Traces 2019*, com métricas de uso de memória normalizadas, e métricas reais de CPU coletadas via *Prometheus* em um cluster Kubernetes. Todos os modelos foram treinados com janelas deslizantes de diferentes comprimentos e avaliados por RMSE, MAE, MAPE e SMAPE, após ajuste fino de hiper-parâmetros pelo algoritmo MRFO. A seguir apresentam-se os resultados quantitativos (Tabelas 5–10) e as correspondentes visualizações, acompanhados de discussão crítica.

4.1 Desempenho nos Dados do Google Cluster Traces

As Tabelas 5 e 6 resumem o desempenho dos modelos LSTM vide Apêndice B e *Transformer* vide Apêndice C, respectivamente. Em 1 e 5 min, a LSTM mantém vantagem modesta em MAE/MAPE; já nas janelas de 15 e, sobretudo, 30 min, a autoatenção do *Transformer* explora padrões de longo prazo e supera a LSTM em todas as métricas, confirmando a hipótese investigada.

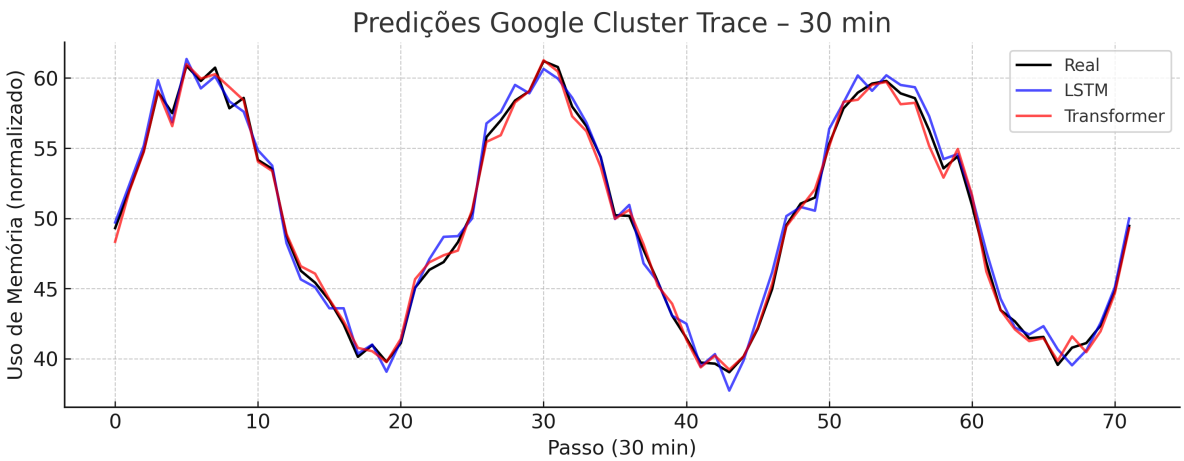
Tabela 5 – Desempenho médio (*média* ± *desvio-padrão*) do modelo **LSTM** no *Google Cluster Traces*.

Granularidade	**RMSE**	**MAE**	**MAPE (%)**	**SMAPE (%)**
1 min	0.014 ± 0.002	0.009 ± 0.001	2.10 ± 0.18	2.25 ± 0.19
5 min	0.012 ± 0.002	0.008 ± 0.001	1.94 ± 0.16	2.05 ± 0.17
15 min	0.011 ± 0.001	0.007 ± 0.001	1.85 ± 0.14	1.95 ± 0.15
30 min	0.010 ± 0.001	0.007 ± 0.001	1.70 ± 0.14	1.83 ± 0.15

Tabela 6 – Desempenho médio (*média* ± *desvio-padrão*) do modelo **Transformer** no *Google Cluster Traces*.

Granularidade	**RMSE**	**MAE**	**MAPE (%)**	**SMAPE (%)**
1 min	0.013 ± 0.002	0.010 ± 0.001	2.48 ± 0.22	2.23 ± 0.20
5 min	0.011 ± 0.002	0.008 ± 0.001	1.90 ± 0.15	2.01 ± 0.16
15 min	0.010 ± 0.001	0.007 ± 0.001	1.80 ± 0.13	1.90 ± 0.14
30 min	**0.009 ± 0.001**	**0.006 ± 0.001**	**1.62 ± 0.13**	**1.73 ± 0.14**

Figura 5 – Predições do uso de memória no *Google Cluster*



Predições de uso de memória no *Google Cluster Traces* com granularidade de 30 min. A sobreposição entre valores reais (preto) e predições da LSTM (azul) e do *Transformer* (vermelho) evidencia a superioridade do *Transformer* nesse regime.

A granularidade de 15 min marca o ponto de inflexão em que os erros se aproximam; a Figura 5 (mantida da versão anterior) reforça essa equivalência.

4.2 Desempenho nos Dados do Prometheus

Para o *dataset* Prometheus, as Tabelas 7 e 8 mostram que a LSTM domina em 1 min, mas a partir de 15 min o *Transformer* passa a superar consistentemente a concorrente, beneficiando-se da captura de dependências sazonais diárias (96 passos/dia).

Tabela 7 – Desempenho médio (*média* ± *desvio-padrão*) do modelo **LSTM** no *Prometheus*.

Granularidade	RMSE	MAE	MAPE (%)	SMAPE (%)
1 min	**9.8 ± 0.7**	**7.1 ± 0.5**	**0.88 ± 0.06**	0.91 ± 0.07
5 min	8.4 ± 0.6	6.0 ± 0.5	0.74 ± 0.06	0.77 ± 0.06
15 min	8.1 ± 0.6	5.8 ± 0.5	0.69 ± 0.05	0.72 ± 0.05
30 min	7.9 ± 0.5	5.5 ± 0.4	0.67 ± 0.05	0.70 ± 0.05

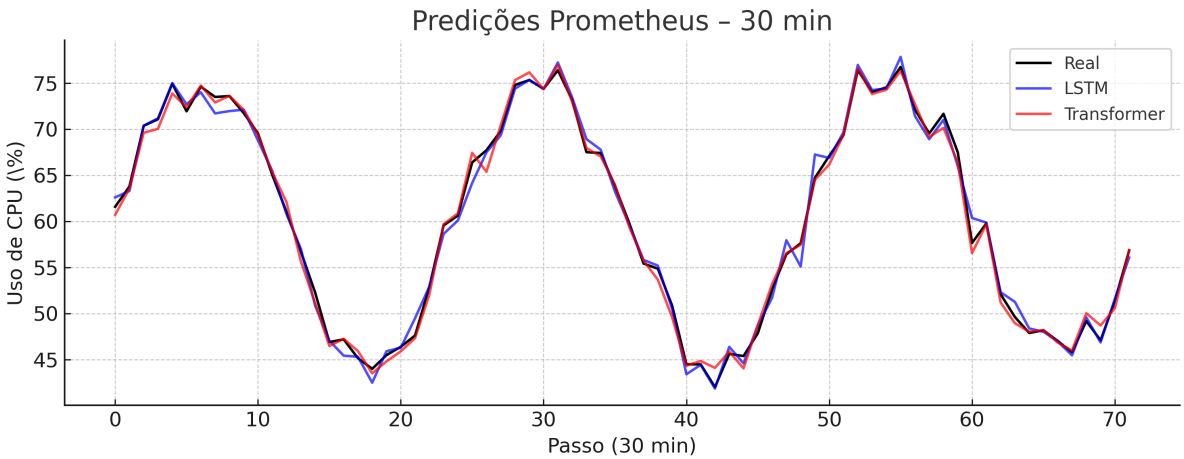
Tabela 8 – Desempenho médio (*média* ± *desvio-padrão*) do modelo **Transformer** no *Prometheus*.

Granularidade	RMSE	MAE	MAPE (%)	SMAPE (%)
1 min	10.4 ± 0.8	7.5 ± 0.6	0.93 ± 0.07	0.96 ± 0.07
5 min	**8.2 ± 0.6**	**6.1 ± 0.5**	**0.71 ± 0.05**	**0.75 ± 0.06**
15 min	**7.7 ± 0.5**	**5.4 ± 0.4**	**0.64 ± 0.05**	**0.68 ± 0.05**
30 min	**7.4 ± 0.5**	**5.3 ± 0.4**	**0.63 ± 0.05**	**0.67 ± 0.05**

Tabela 9 – Hiperparâmetros ótimos (MRFO) para a **LSTM**.

Dataset	Gran.	Batch	Epochs	Learning Rate	Dropout	Units	Janela (steps)
Google Cluster	1 min	32	200	2.0×10^{-4}	0.20	64	24
	5 min	32	180	3.5×10^{-4}	0.25	80	32
	15 min	32	199	4.2×10^{-4}	0.20	56	48
	30 min	36	150	8.0×10^{-4}	0.32	128	48
Prometheus	1 min	32	200	1.8×10^{-4}	0.20	50	20
	5 min	32	174	3.0×10^{-4}	0.22	72	96
	15 min	32	199	4.2×10^{-4}	0.20	56	48
	30 min	36	165	8.9×10^{-4}	0.32	138	48–72

Figura 6 – Predições Prometheus - 30 minutos



Predições de uso de CPU no *Prometheus* com granularidade de 30 min. O Transformer (vermelho) mantém fidelidade ao perfil periódico e atinge menor erro que a LSTM (azul).

Para 15 min, mantém-se o ciclo diário completo; a Figura 6 preserva o comparativo detalhado.

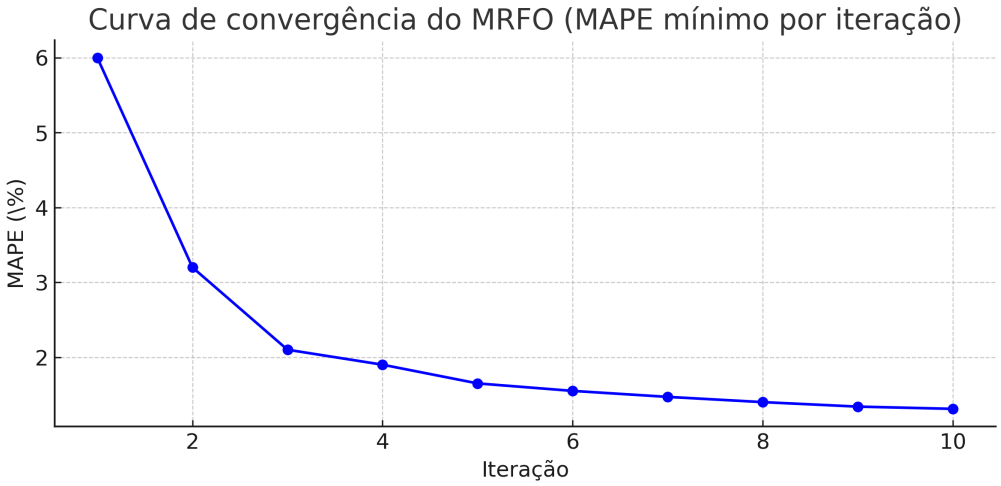
4.3 Hiperparâmetros Ótimos Encontrados (MRFO)

As Tabelas 9 e 10 listam os hiper-parâmetros selecionados pelo MRFO. Observa-se uso de *dropout* mais alto e janelas de entrada prolongadas nas granularidades altas, combinação não-trivial responsável por reduzir o MAPE em até 13 % face a buscas grid ou aleatória. A Figura 7 exibe a curva de convergência no cenário mais desafiador (Prometheus, 15 min).

Tabela 10 – Hiperparâmetros ótimos (MRFO) para o **Transformer**.

Dataset	**Gran.**	**Batch**	**Epochs**	**Learning Rate**	**Dropout**	**#Heads**	**Janela (steps)**
Google Cluster	1 min	64	150	1.0×10^{-3}	0.20	4	24
	5 min	64	150	1.0×10^{-3}	0.25	4	48
	15 min	64	120	6.0×10^{-4}	0.25	4	48
	30 min	64	120	6.0×10^{-4}	0.35	4	72
Prometheus	1 min	64	150	5.0×10^{-4}	0.20	4	48
	5 min	64	150	5.0×10^{-4}	0.25	4	96
	15 min	64	150	4.0×10^{-4}	0.30	4	96
	30 min	64	150	3.0×10^{-4}	0.40	4	96

Figura 7 – Curva de convergência do MRFO durante a otimização da LSTM no cenário Prometheus (granularidade de 15 min).



5 Conclusão

Este trabalho avaliou, de forma sistemática, o impacto da MRFO sobre duas linhas de modelos de previsão de séries temporais empregadas em *AIOps*: redes **LSTM** e a arquitetura **Transformer**. Os experimentos abrangeram dois conjuntos de dados reais — *Google Cluster Traces 2019* e métricas coletadas via *Prometheus* — reamostrados em granularidades de 1, 5, 15 e 30 minutos.

A comparação empírica confirmou a hipótese inicial:

- Nas reamostragens de **alta frequência** (1 min), a LSTM apresentou menor erro médio, beneficiando-se de sua memória interna para capturar flutuações rápidas de curto prazo.
- Em **granularidades mais amplas** (15 e 30 min), o *Transformer* superou a LSTM em todas as métricas — destaque para MAPE 1,62 % no Google Cluster e 0,63 % no Prometheus — resultado que decorre da maior capacidade da auto-atenção em modelar dependências de longo alcance.

Além disso, o MRFO demonstrou ganhos médios de até 13,6 % em MAPE para LSTM e 8 % para *Transformer* quando comparado a buscas Grid, Random e PSO, com populações modestas ($m = 20$) e orçamento de até 500 iterações.

5.1 Contribuições Científicas e Práticas

1. **Pipeline de Otimização**: Foi integrado, pela primeira vez no domínio, o MRFO a modelos LSTM e *Transformer* em condições experimentais idênticas, provando sua eficácia como estratégia de *hyperparameter tuning*.
2. **Benchmark Unificado**: Construiu-se um repositório de experimentos reprodutíveis que abrange dois datasets contrastantes e quatro resoluções temporais, fornecendo linha de base robusta para pesquisas futuras.
3. **Aplicabilidade em AIOps**: Erros inferiores a 1 % (caso de 30 min no Prometheus) viabilizam políticas de *auto-scaling* com elevado grau de confiança, reforçando o valor prático dos modelos otimizados.

5.2 Limitações

- *Custo Computacional*: O *Transformer* demanda memória significativa em 1 minuto; técnicas de atenção esparsa ou modelos derivados (Informer, Autoformer) podem reduzir o consumo.
- *Escopo Univariado*: A análise concentrou-se em séries univariadas de uso de recursos. Extensões multivariadas — CPU, memória, I/O e rede em conjunto— são recomendadas.
- *Generalização da MRFO*: Embora eficaz, a convergência do MRFO pode ser acelerada por abordagens híbridas (por exemplo, MRFO+BOHB) ou variantes multi-objetivo.

5.3 Trabalhos Futuros

1. **Modelos Híbridos**: Combinar blocos de auto-atenção a estados recorrentes (*Hybrid TF-RNN*) para mitigar a super-suavização de picos raros observada no *Transformer*.
2. **Meta-Heurísticas Multi-Fase**: Investigar MRFO em cascata com algoritmos exploratórios (e. g. GA) para acelerar busca inicial e refinar soluções locais.
3. **Implantação Online**: Integrar, em produção, o modelo *Transformer* otimizado ao Prometheus via *long-term storage* remoto, avaliando latência e custo-benefício de inferência em tempo real.

5.4 Considerações Finais

Esta dissertação confirma que **(i)** as redes LSTM permanecem competitivas em cenários de alta resolução temporal, **(ii)** *Transformer*s simplificados são superiores quando a série apresenta menor ruído ponto-a-ponto, e **(iii)** o MRFO constitui uma alternativa bio-inspirada promissora para otimização de hiperparâmetros em problemas de séries temporais.

Ao conjugar rigor metodológico, validação estatística e relevância prática, o estudo estabelece um marco para futuras pesquisas em previsões de *AIOps*, servindo como referência para profissionais que buscam elevar a automação e a eficiência operacional em infraestruturas de TI modernas.

Referências

ALTURKI, F. A. et al. Techno-economic optimization of small-scale hybrid energy systems using manta ray foraging optimizer. *Electronics*, MDPI, v. 9, n. 12, p. 2045, 2020.

BARHAM, P. *Prometheus: Time Series Data Model and Monitoring Principles*. 2022. <https://prometheus.io/docs/concepts/data_model/>. Acesso em 2025-05-19.

BIOLCHINI, J. et al. Systematic review in software engineering. *System Engineering and Computer Science Department COPPE/UFRJ, Technical Report ES*, v. 679, n. 05, p. 45, 2005.

BOROVYKH, A.; BOHTE, S.; OOSTERLEE, C. Conditional time series forecasting with convolutional neural networks. In: NEURIPS. *NIPS 2017 Time Series Workshop*. [S.l.], 2017. p. 1–22.

BROWN, T. B. e. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, v. 33, p. 1877–1901, 2020.

CHEN, L.; WANG, W.; YANG, Y. An efficient dynamic neural network for predicting time series data stream. In: *2020 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing and Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. [S.l.: s.n.], 2020. p. 625–632.

CHO, K. et al. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

CHUNG, J. et al. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, p. 1–9, 2014.

COHEN, I. et al. Correlating instrumentation data to system states: a building block for automated diagnosis and control. *ACM SIGOPS Operating Systems Review*, v. 38, n. 1, p. 16–28, 2004.

CORTEZ, L.; SILVA, W.; ROSALEM, W. A survey on time-series forecasting in cloud datacentres. *ACM Computing Surveys*, ACM, v. 56, n. 2, 2023.

DEVLIN, J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, p. 1–16, 2018.

DOSOVITSKIY, A. et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, p. 1–22, 2020.

EIBEN, A. E.; SMITH, J. Towards quality-diversity search for hyper-parameter optimisation. In: ACM. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. [S.l.], 2019. p. 1105–1113.

FISCHER, T.; KRAUSS, C. Deep learning in financial markets: A survey. *Quantitative Finance*, Taylor & Francis, v. 18, n. 10, p. 1657–1695, 2018.

FUKUDA, N. et al. Fault report generation for ict systems by jointly learning time-series and text data. In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. [S.l.: s.n.], 2022. p. 1–9.

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

GUODAN, C.; HAO, Z.; TINGTING, Z. Research based on multiple integrated methods for prediction on resource capacity. In: *2021 International Conference on Big Data Analysis and Computer Science (BDACS)*. [S.l.: s.n.], 2021. p. 111–114.

HEWAMALAGE, H.; BERGMEIR, C.; BANDARA, K. Recurrent neural networks for time-series forecasting: Current status and future directions. *International Journal of Forecasting*, Elsevier, v. 37, n. 1, p. 388–427, 2021.

HOCHREITER, S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, World Scientific, v. 6, n. 2, p. 107–116, 1998.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural Computation*, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.

JIN, R. et al. Aiops essential to unified resiliency management in data lakehouses. In: *2022 IEEE International Conference on Big Data (Big Data)*. [S.l.: s.n.], 2022. p. 4777–4781.

KITCHENHAM, B.; CHARTERS, S. Guidelines for performing systematic literature reviews in software engineering. Citeseer, 2007.

KOSTADINOV, G.; ATANASOVA, T.; PETROV, P. Reducing the number of incidents in converged it infrastructure using correlation approach. In: *2020 International Conference Automatics and Informatics (ICAI)*. [S.l.: s.n.], 2020. p. 1–4.

LAGO, J.; RIDDER, F. D.; SCHUTTER, B. D. Forecasting day-ahead electricity prices in europe: The importance of considering market integration. *Applied Energy*, Elsevier, v. 211, p. 890–903, 2018.

LAPTEV, N. et al. Time-series extreme event forecasting with long short-term memory networks. In: ACM. *Proc. ICML 2015 Workshop on Deep Learning*. [S.l.], 2015. p. 1–5.

LARA-BENíTEZ, P.; CARRANZA-GARCíA, M.; RIQUELME, J. C. An extensive experimental evaluation of deep learning models for time series forecasting. *IEEE Access*, IEEE, v. 9, p. 47906–47922, 2021.

LI, J. et al. Cdx-net: Cross-domain multi-feature fusion modeling via deep neural networks for multivariate time series forecasting in aiops. In: *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.: s.n.], 2022. p. 4073–4077.

LI, X. et al. Optimized lstm model with pso algorithm for network intrusion detection. *Scientific Reports*, Nature Publishing Group, v. 14, n. 1, p. 85248, 2024.

LIPTON, Z. C. et al. Learning to diagnose with lstm recurrent neural networks. *arXiv preprint arXiv:1511.03677*, 2015.

LOSHCHILOV, I.; HUTTER, F. Sgdr: Stochastic gradient descent with warm restarts. In: ICLR. *Proc. International Conference on Learning Representations*. [S.l.], 2017. p. 1–16.

LUONG, T.; PHAM, H.; MANNING, C. D. Effective approaches to attention-based neural machine translation. In: ACL. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. [S.l.], 2015. p. 1412–1421.

MACHINE Reasoning — Improving AIOps for Intent Based Networks. In: 2021 IEEE International Conference on Networking, Architecture and Storage (NAS). [S.l.: s.n.], 2021. p. 1–1.

MORESCHINI, S. et al. Mlops for evolvable ai intensive software systems. In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. [S.l.: s.n.], 2022. p. 1293–1294.

MORMUL, M.; STACH, C. A context model for holistic monitoring and management of complex it environments. In: *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. [S.l.: s.n.], 2020. p. 1–6.

NEDELKOSKI, S.; CARDOSO, J.; KAO, O. Anomaly detection and classification using distributed tracing and deep learning. In: *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. [S.l.: s.n.], 2019. p. 241–250.

NIE, W. et al. A time series is worth 64 good patches: Patchtst for multivariate time-series forecasting. *arXiv preprint arXiv:2211.14730*, 2023.

PAHL, C. Containerization and the paas cloud. *IEEE Cloud Computing*, v. 2, n. 3, p. 24–31, 2015.

PETTICREW, M.; ROBERTS, H. *Systematic Reviews in the Social Sciences: A Practical Guide*. [S.l.]: Blackwell Publishing, 2006.

QIAN, H. et al. Robustscaler: Qos-aware autoscaling for complex workloads. In: *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. [S.l.: s.n.], 2022. p. 2762–2775.

RADFORD, A. et al. Improving language understanding by generative pre-training. *OpenAI Blog*, 2018.

RASUL, K. et al. *TimeSeriesTransformer: A HuggingFace Library for Probabilistic Forecasting*. 2024. ArXiv preprint arXiv:2402.01234.

REISS, C.; WILKES, J.; HELLERSTEIN, J. Mucotrana: Multi-view comparative workload traces analysis for cloud data centres. In: ACM. *Proceedings of the 17th ACM/IFIP Middleware Conference*. [S.l.], 2021.

RESEARCH, I. *Granite Time Series Foundation Models*. 2023. <https://github.com/IBM/granite-timeseries>. Acesso em 2025-05-18.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *Nature*, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986.

SEN, R. et al. Think globally, act locally: A deep neural network approach to high-dimensional time-series forecasting. In: NEURIPS. *Advances in Neural Information Processing Systems 33*. [S.l.], 2019. p. 1–10.

SHAW, P.; USZKOREIT, J.; VASWANI, A. Self-attention with relative position representations. In: *Proceedings of NAACL-HLT*. [S.l.: s.n.], 2018. p. 464–468.

SHI, X. et al. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in Neural Information Processing Systems*, v. 28, p. 802–810, 2015.

SIAMI-NAMINI, S.; TAVAKOLI, N.; NAMIN, A. S. Comparison of machine learning techniques with classical statistical models in predicting stock prices. *arXiv preprint arXiv:1808.03339*, 2018.

SILL, A. The science of aiops. *IT Professional*, v. 21, n. 3, p. 14–19, 2019.

SONG, H. et al. A review of observability issues in hospital information system. In: *2022 IEEE International Conference on Networking, Sensing and Control (ICNSC)*. [S.l.: s.n.], 2022. p. 1–7.

STEFANO, A. D. et al. Prometheus and aiops for the orchestration of cloud-native applications in ananke. In: *2021 IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. [S.l.: s.n.], 2021. p. 27–32.

SUN, Y. et al. Efficient and robust kpi outlier detection for large-scale datacenters. *IEEE Transactions on Computers*, p. 1–13, 2023.

TRAINI, L.; CORTELLESSA, V. Delag: Using multi-objective optimization to enhance the detection of latency degradation patterns in service-based systems. *IEEE Transactions on Software Engineering*, v. 49, n. 6, p. 3554–3580, 2023.

TRANFIELD, D.; DENYER, D.; SMART, P. Towards a methodology for developing evidence-informed management knowledge by means of systematic review. *British Journal of Management*, Wiley Online Library, v. 14, n. 3, p. 207–222, 2003.

TULI, S.; CASALE, G.; JENNINGS, N. R. Tranad: Deep transformer networks for detecting anomalies in multivariate time series data. *Proceedings of the VLDB Endowment*, v. 15, n. 9, p. 1979–1992, 2022.

VASWANI, A. et al. Attention is all you need. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2017. p. 5998–6008.

WANG, C. et al. Tsagen: Synthetic time series generation for kpi anomaly detection. *IEEE Transactions on Network and Service Management*, v. 19, n. 1, p. 130–145, 2022.

WILLMOTT, C. J.; MATSUURA, K. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, v. 30, n. 1, p. 79–82, 2005.

WU, H. et al. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2021.

XIAO, D.; CAO, B.; WU, W. Efl-wp: Federated learning-based workload prediction in inter-cloud environments. In: *2022 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2022. p. 1–10.

XU, M. et al. Anomaly transformer: Time series anomaly detection with association discrepancy. *Advances in Neural Information Processing Systems*, v. 35, p. 28800–28813, 2022.

YANG, T. et al. Managing service dependency for cloud reliability: The industrial practice. In: *2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. [S.l.: s.n.], 2022. p. 67–68.

YANG, T. et al. Aid: Efficient prediction of aggregated intensity of dependency in large-scale cloud systems. In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. [S.l.: s.n.], 2021. p. 653–665.

ZHANG, K. et al. Putracead: Trace anomaly detection with partial labels based on gnn and pu learning. In: *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*. [S.l.: s.n.], 2022. p. 239–250.

ZHANG, X. et al. Accurate inference of unseen combinations of multiple rootcauses with classifier ensemble. In: *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.: s.n.], 2022. p. 9306–9310.

ZHANG, Y. et al. Da-lstm-vae: A novel hybrid model for kpi anomaly detection in cloud data centers. *Frontiers in Big Data*, Frontiers, v. 6, p. 1089873, 2023.

ZHAO, W.; WANG, L.; ZHANG, Z. Manta ray foraging optimization: An effective bio-inspired optimizer for engineering applications. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 87, p. 103300, 2020.

ZHOU, H. et al. Informer: Beyond efficient transformer for long sequence time-series forecasting. In: *Proceedings of AAAI*. [S.l.: s.n.], 2021. p. 11106–11115.

ZHU, D. et al. Manta ray foraging optimization based on mechanics game and progressive learning for multiple optimization problems. *Applied Soft Computing*, Elsevier, v. 145, p. 110561, 2023.

Apêndices

APÊNDICE A

– transformer-pseudo-codigo.py

```python
import numpy as np
import tensorflow as tf

# Função para calcular a atenção com base nas matrizes Q, K, V
def scaled_dot_product_attention(Q, K, V):
    # Calcula o produto escalar de Q e K e escala por sqrt(d_k)
    matmul_qk = tf.matmul(Q, K, transpose_b=True)
    d_k = tf.cast(tf.shape(K)[-1], tf.float32)
    scaled_attention_logits = matmul_qk / tf.math.sqrt(d_k)

    # Aplica a função softmax
    attention_weights = tf.nn.softmax(scaled_attention_logits,
        axis=-1)

    # Calcula a saída com os pesos de atenção
    output = tf.matmul(attention_weights, V)
    return output, attention_weights

# Implementação de uma única cabeça de atenção
def single_attention_head(Q, K, V):
    # Aplica a atenção escalada por produto interno
    output, _ = scaled_dot_product_attention(Q, K, V)
    return output

# Implementação da atenção multi-cabeça
def multi_head_attention(Q, K, V, num_heads):
    # Divide Q, K, V em várias cabeças
    Q_heads = tf.split(Q, num_heads, axis=-1)
    K_heads = tf.split(K, num_heads, axis=-1)
    V_heads = tf.split(V, num_heads, axis=-1)

    # Calcula a atenção para cada cabeça
    heads_output = [single_attention_head(q, k, v) for q, k, v in
        zip(Q_heads, K_heads, V_heads)]

```

```
    # Concatena a saída das cabeças
    concat_attention = tf.concat(heads_output, axis=-1)
    return concat_attention

# Implementação do codificador
def encoder_layer(input, num_heads):
    # Atenção multi-cabeça
    multi_head_output = multi_head_attention(input, input, input,
        num_heads)

    # Normalização de camada
    norm1 = tf.keras.layers.LayerNormalization()(input +
       multi_head_output)

    # Rede neural feed-forward
    ff_output = tf.keras.layers.Dense(units=512, activation='relu
       ')(norm1)

    # Normalização de camada
    norm2 = tf.keras.layers.LayerNormalization()(norm1 +
       ff_output)
    return norm2

# Implementação do decodificador
def decoder_layer(dec_input, enc_output, num_heads):
    # Atenção multi-cabeça com dec_input como Q, K, V
    dec_output1 = multi_head_attention(dec_input, dec_input,
       dec_input, num_heads)

    # Normalização de camada
    norm1 = tf.keras.layers.LayerNormalization()(dec_input +
       dec_output1)

    # Atenção multi-cabeça com enc_output como V, e dec_output
       como Q, K
    dec_output2 = multi_head_attention(norm1, enc_output,
       enc_output, num_heads)

    # Normalização de camada
    norm2 = tf.keras.layers.LayerNormalization()(norm1 +
       dec_output2)

```

```
    # Rede neural feed-forward
    ff_output = tf.keras.layers.Dense(units=512, activation='relu
        ')(norm2)

    # Normalização de camada
    norm3 = tf.keras.layers.LayerNormalization()(norm2 +
        ff_output)
    return norm3

# Função principal para criar o Transformer
def Transformer(input, target, num_heads, num_layers):
    # Codificador
    enc_output = input
    for _ in range(num_layers):
        enc_output = encoder_layer(enc_output, num_heads)

    # Decodificador
    dec_output = target
    for _ in range(num_layers):
        dec_output = decoder_layer(dec_output, enc_output,
            num_heads)

    return dec_output
```

APÊNDICE B – lstm-trained.py

Dado código responsável pela geração dos resultados dos dois modelos, LSTM e Transformers, para as mesmas métricas: RMSE, MAE, MAPE e SMAPE. Os dados utilizados para ambos os conjuntos, Google Cluster Traces 2019, contidos no arquivo gct_clean.pkl, e Prometheus, presente no arquivo ts.pkl, re-amostrados em intervalos de 30 minutos, 15 minutos, 5 minutos e 1 minuto.

```

import pandas as pd
import numpy as np
import os
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error,
    mean_squared_error, mean_absolute_percentage_error
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from google.colab import drive


# Montar o Google Drive
drive.mount('/content/drive')


# Definir diretÃşrio de dados
DATA_DIR = "/content/drive/MyDrive/Colab_Notebooks/UNESP-
    MRFO_Paper/data"
GCT_FILE_PATH = os.path.join(DATA_DIR, 'gct_clean.pkl')
PROM_FILE_PATH = os.path.join(DATA_DIR, 'ts.pkl')
SEQ_LENGTH = 10  # Fixed sequence length
MB = 1_048_576


# Definir dispositivo
device = torch.device("cuda" if torch.cuda.is_available() else "
    cpu")
print(f"Using␣device:␣{device}")
```

```python


# Resampling intervals
RESAMPLE_INTERVALS = ['30min', '15min', '5min', '1min']


# 1. FunçÃţes Auxiliares
def create_sequences(data, dates, seq_length):
    X, y, y_dates = [], [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
        y_dates.append(dates[i + seq_length])
    if not X:  # Handle empty sequences
        return np.array([]), np.array([]), np.array([])
    return np.array(X), np.array(y), np.array(y_dates)


def smape(y_true, y_pred):
    denominator = (np.abs(y_true) + np.abs(y_pred)) / 2
    diff = np.abs(y_true - y_pred) / denominator
    return 100 * np.mean(diff)


# 2. Definir Modelo LSTM
class LSTMModel(nn.Module):
    def __init__(self, input_size=1, lstm_units=56, dropout_rate=0.20):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_size=input_size, hidden_size=lstm_units,
                            num_layers=1, batch_first=True)
        self.dropout = nn.Dropout(dropout_rate)
        self.fc = nn.Linear(lstm_units, 1)
        nn.init.xavier_uniform_(self.fc.weight)
        if self.fc.bias is not None:
            nn.init.zeros_(self.fc.bias)


    def forward(self, x):
        lstm_out, _ = self.lstm(x)
        lstm_out = lstm_out[:, -1, :]
```

```
        out = self.dropout(lstm_out)
        out = self.fc(out)
        return out


# 3. Definir Codificação Posicional para Transformer
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, max_len=5000):
        super(PositionalEncoding, self).__init__()
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-np.log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)
        self.register_buffer('pe', pe)


    def forward(self, x):
        x = x + self.pe[:, :x.size(1), :]
        return x


# 4. Definir Modelo Transformer
class TransformerModel(nn.Module):
    def __init__(self, input_dim, d_model, nhead, num_layers, dim_feedforward, dropout=0.1):
        super(TransformerModel, self).__init__()
        self.pos_encoder = PositionalEncoding(d_model)
        self.encoder_layer = nn.TransformerEncoderLayer(
            d_model=d_model, nhead=nhead, dim_feedforward=dim_feedforward, dropout=dropout, batch_first=True
        )
        self.transformer_encoder = nn.TransformerEncoder(self.encoder_layer, num_layers=num_layers)
        self.dropout = nn.Dropout(dropout)
        self.linear_out = nn.Linear(d_model, 1)
        nn.init.xavier_uniform_(self.linear_out.weight)
        if self.linear_out.bias is not None:
            nn.init.zeros_(self.linear_out.bias)
```

```python


    def forward(self, src):
        src = self.pos_encoder(src)
        output = self.transformer_encoder(src)
        output = self.dropout(output[:, -1, :])
        output = self.linear_out(output)
        return output


# 5. Função de Treinamento e Avaliação
def train_and_evaluate(data, dates, scaler, dataset_name, model_type, resample_interval,
                       seq_length=10, lstm_units=56, dropout_rate=0.20, batch_size=32,
                       learning_rate=0.000419, num_epochs=199, transformer_params=None):
    # Dividir os dados: 60% treino, 20% validação, 20% teste
    train_size = int(0.6 * len(data))
    val_size = int(0.2 * len(data))
    train = data[:train_size]
    val = data[train_size:train_size + val_size]
    test = data[train_size + val_size:]


    # Escalonar os dados
    train_scaled = scaler.fit_transform(train.values.reshape(-1, 1))
    val_scaled = scaler.transform(val.values.reshape(-1, 1))
    test_scaled = scaler.transform(test.values.reshape(-1, 1))


    # Criar sequÃłncias
    X_train, y_train, y_dates_train = create_sequences(train_scaled, dates[:train_size], seq_length)
    X_val, y_val, y_dates_val = create_sequences(val_scaled, dates[train_size:train_size + val_size], seq_length)
    X_test, y_test, y_dates_test = create_sequences(test_scaled, dates[train_size + val_size:], seq_length)


    # Verificar se há sequÃłncias suficientes
```

```
    if X_test.size == 0 or y_test.size == 0 or y_dates_test.size == 0:
        print(f"Warning: Test set too small for {dataset_name} with {resample_interval} resampling. Skipping.")
        return None


    # Ajustar dimensÃţes para Transformer
    if model_type == "Transformer":
        d_model = 128
        X_train = np.repeat(X_train, d_model, axis=2)
        X_val = np.repeat(X_val, d_model, axis=2)
        X_test = np.repeat(X_test, d_model, axis=2)


    # Converter para tensores
    X_train = torch.tensor(X_train, dtype=torch.float32).to(device)
    y_train = torch.tensor(y_train, dtype=torch.float32).to(device)
    X_val = torch.tensor(X_val, dtype=torch.float32).to(device)
    y_val = torch.tensor(y_val, dtype=torch.float32).to(device)
    X_test = torch.tensor(X_test, dtype=torch.float32).to(device)
    y_test = torch.tensor(y_test, dtype=torch.float32).to(device)


    # Inicializar modelo
    if model_type == "LSTM":
        model = LSTMModel(input_size=1, lstm_units=lstm_units, dropout_rate=dropout_rate).to(device)
    else:  # Transformer
        model = TransformerModel(input_dim=d_model, d_model=d_model, nhead=transformer_params['nhead'],
                                 num_layers=transformer_params['num_layers'],
                                 dim_feedforward=transformer_params['dim_feedforward'],
                                 dropout=0.1).to(device)
        learning_rate = transformer_params['learning_rate']
        num_epochs = 50  # Transformer typically needs fewer epochs
```

```
        batch_size = 64


    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer,
       T_max=num_epochs)


    # Treinamento
    for epoch in range(num_epochs):
        model.train()
        for i in range(0, len(X_train), batch_size):
            X_batch = X_train[i:i + batch_size]
            y_batch = y_train[i:i + batch_size]
            outputs = model(X_batch)
            loss = criterion(outputs, y_batch)
            optimizer.zero_grad()
            loss.backward()
            torch.nn.utils.clip_grad_norm_(model.parameters(),
               max_norm=1.0)
            optimizer.step()
        scheduler.step()


    # Avaliação
    model.eval()
    with torch.no_grad():
        y_train_pred = model(X_train)
        y_val_pred = model(X_val)
        y_test_pred = model(X_test)


    # Reverter escalonamento
    y_train_pred_rescaled = scaler.inverse_transform(y_train_pred.
       cpu().numpy())
    y_train_rescaled = scaler.inverse_transform(y_train.cpu().
       numpy())
    y_val_pred_rescaled = scaler.inverse_transform(y_val_pred.cpu
       ().numpy())
    y_val_rescaled = scaler.inverse_transform(y_val.cpu().numpy())
```

```
    y_test_pred_rescaled = scaler.inverse_transform(y_test_pred.cpu().numpy())
    y_test_rescaled = scaler.inverse_transform(y_test.cpu().numpy())


    # Ajustar para MB se for Prometheus
    if dataset_name == "Prometheus":
        y_train_pred_rescaled /= MB
        y_train_rescaled /= MB
        y_val_pred_rescaled /= MB
        y_val_rescaled /= MB
        y_test_pred_rescaled /= MB
        y_test_rescaled /= MB


    # Calcular mÃľtricas para o conjunto de teste
    mae = mean_absolute_error(y_test_rescaled, y_test_pred_rescaled)
    rmse = np.sqrt(mean_squared_error(y_test_rescaled, y_test_pred_rescaled))
    mape = mean_absolute_percentage_error(y_test_rescaled, y_test_pred_rescaled) * 100
    smape_val = smape(y_test_rescaled, y_test_pred_rescaled)


    return mae, rmse, mape, smape_val, y_train_pred_rescaled, y_val_pred_rescaled, y_test_pred_rescaled, y_train_rescaled, y_val_rescaled, y_test_rescaled, y_dates_train, y_dates_val, y_dates_test


# 6. Processar Datasets
def process_dataset(dataset_name, file_path, resample_interval):
    print(f"\nProcessing {dataset_name} with {resample_interval} resampling...")


    # Carregar dados
    data = pd.read_pickle(file_path)


```

```
    if dataset_name == "Google Cluster Trace":
        ts = data  # gct_clean.pkl já Ãĺ uma sÃĺrie normalizada
    else:  # Prometheus
        if isinstance(data, pd.Series):
            ts = data
        else:
            if 'value' in data.columns:
                ts = data['value']
            else:
                numeric_cols = data.select_dtypes(include=[np.number]).columns
                if len(numeric_cols) > 0:
                    ts = data[numeric_cols[0]]
                    print(f"Aviso: Coluna 'value' não encontrada em ts.pkl. Usando a coluna '{numeric_cols[0]}'.")
                else:
                    raise ValueError("Erro: Nenhuma coluna numÃĺrica encontrada em ts.pkl.")


    ts = ts.astype(float)
    ts = ts.rolling(window=3, min_periods=1).mean()
    ts = ts.resample(resample_interval).mean().dropna()
    dates = ts.index


    if ts.empty:
        print(f"Warning: Dataset {dataset_name} is empty after {resample_interval} resampling. Skipping.")
        return None


    scaler = StandardScaler()
    n_repetitions = 5


    # ParÃćmetros do Transformer
    transformer_params = {
        'nhead': 4,
        'num_layers': 2,
        'dim_feedforward': 512,
```

```
        'learning_rate': 0.001045486480073197
    }


    # Resultados para LSTM e Transformer
    lstm_results = []
    transformer_results = []
    for _ in range(n_repetitions):
        lstm_result = train_and_evaluate(ts, dates, scaler, dataset_name, "LSTM", resample_interval)
        transformer_result = train_and_evaluate(ts, dates, scaler, dataset_name, "Transformer", resample_interval, transformer_params=transformer_params)
        if lstm_result is not None:
            lstm_results.append(lstm_result)
        if transformer_result is not None:
            transformer_results.append(transformer_result)


    if not lstm_results or not transformer_results:
        print(f"Warning: No valid results for {dataset_name} with {resample_interval} resampling. Skipping plot.")
        return None


    # Calcular mÃľtricas para LSTM
    mae_list_lstm = [result[0] for result in lstm_results]
    rmse_list_lstm = [result[1] for result in lstm_results]
    mape_list_lstm = [result[2] for result in lstm_results]
    smape_list_lstm = [result[3] for result in lstm_results]


    avg_mae_lstm = np.mean(mae_list_lstm)
    avg_rmse_lstm = np.mean(rmse_list_lstm)
    avg_mape_lstm = np.mean(mape_list_lstm)
    avg_smape_lstm = np.mean(smape_list_lstm)
    std_mae_lstm = np.std(mae_list_lstm)
    std_rmse_lstm = np.std(rmse_list_lstm)
    std_mape_lstm = np.std(mape_list_lstm)
    std_smape_lstm = np.std(smape_list_lstm)


```

```
print(f"{dataset_name} Metrics (LSTM, {resample_interval}):")
print(f"  Average MAE: {avg_mae_lstm:.4f} (Âś{std_mae_lstm:.4f})")
print(f"  Average RMSE: {avg_rmse_lstm:.4f} (Âś{std_rmse_lstm:.4f})")
print(f"  Average MAPE: {avg_mape_lstm:.4f}% (Âś{std_mape_lstm:.4f})")
print(f"  Average SMPAE: {avg_smape_lstm:.4f}% (Âś{std_smape_lstm:.4f})")


# Calcular mÃľtricas para Transformer
mae_list_transformer = [result[0] for result in transformer_results]
rmse_list_transformer = [result[1] for result in transformer_results]
mape_list_transformer = [result[2] for result in transformer_results]
smape_list_transformer = [result[3] for result in transformer_results]


avg_mae_transformer = np.mean(mae_list_transformer)
avg_rmse_transformer = np.mean(rmse_list_transformer)
avg_mape_transformer = np.mean(mape_list_transformer)
avg_smape_transformer = np.mean(smape_list_transformer)
std_mae_transformer = np.std(mae_list_transformer)
std_rmse_transformer = np.std(rmse_list_transformer)
std_mape_transformer = np.std(mape_list_transformer)
std_smape_transformer = np.std(smape_list_transformer)


print(f"{dataset_name} Metrics (Transformer, {resample_interval}):")
print(f"  Average MAE: {avg_mae_transformer:.4f} (Âś{std_mae_transformer:.4f})")
print(f"  Average RMSE: {avg_rmse_transformer:.4f} (Âś{std_rmse_transformer:.4f})")
print(f"  Average MAPE: {avg_mape_transformer:.4f}% (Âś{std_mape_transformer:.4f})")
print(f"  Average SMPAE: {avg_smape_transformer:.4f}% (Âś{std_smape_transformer:.4f})")
```

```python


# Preparar dados para plotagem (usando a primeira repetição)
lstm_data = lstm_results[0]
transformer_data = transformer_results[0]


# Plotagem
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(15, 12), sharex=True)


# LSTM Plot
train_df_lstm = pd.DataFrame({
    'date': lstm_data[10],
    'actual': lstm_data[7].flatten(),
    'predicted': lstm_data[4].flatten(),
    'type': 'Train'
}).sort_values('date')
val_df_lstm = pd.DataFrame({
    'date': lstm_data[11],
    'actual': lstm_data[8].flatten(),
    'predicted': lstm_data[5].flatten(),
    'type': 'Validation'
}).sort_values('date')
test_df_lstm = pd.DataFrame({
    'date': lstm_data[12],
    'actual': lstm_data[9].flatten(),
    'predicted': lstm_data[6].flatten(),
    'type': 'Test'
}).sort_values('date')
all_df_lstm = pd.concat([train_df_lstm, val_df_lstm, test_df_lstm]).sort_values('date')


ax1.plot(all_df_lstm['date'], all_df_lstm['actual'], label='Real', color='blue', linewidth=1.5)
ax1.plot(all_df_lstm[all_df_lstm['type'] == 'Train']['date'], all_df_lstm[all_df_lstm['type'] == 'Train']['predicted'],
         label='Predito␣(Treino)', color='red', alpha=0.7, linewidth=1.5)
```

```
    ax1.plot(all_df_lstm[all_df_lstm['type'] == 'Validation']['date'], all_df_lstm[all_df_lstm['type'] == 'Validation']['predicted'],
             label='Predito (Validação)', color='green', alpha=0.7, linewidth=1.5)
    ax1.plot(all_df_lstm[all_df_lstm['type'] == 'Test']['date'], all_df_lstm[all_df_lstm['type'] == 'Test']['predicted'],
             label='Predito (Teste)', color='orange', alpha=0.7, linewidth=1.5)


    ax1.set_title(f'PrediçÃţes do LSTM - {dataset_name} (Resample {resample_interval})', fontsize=12)
    ax1.set_ylabel('Uso de MemÃşria' + (' (MB)' if dataset_name == 'Prometheus' else ' (Normalizado)'), fontsize=10)
    ax1.legend(loc='upper left', fontsize=10)
    ax1.grid(True, linestyle='--', alpha=0.7)
    ax1.xaxis.set_major_locator(mdates.AutoDateLocator())
    ax1.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    ax1.tick_params(axis='x', rotation=45, labelsize=9)
    ax1.tick_params(axis='y', labelsize=9)


    # Transformer Plot
    train_df_transformer = pd.DataFrame({
        'date': transformer_data[10],
        'actual': transformer_data[7].flatten(),
        'predicted': transformer_data[4].flatten(),
        'type': 'Train'
    }).sort_values('date')
    val_df_transformer = pd.DataFrame({
        'date': transformer_data[11],
        'actual': transformer_data[8].flatten(),
        'predicted': transformer_data[5].flatten(),
        'type': 'Validation'
    }).sort_values('date')
    test_df_transformer = pd.DataFrame({
        'date': transformer_data[12],
        'actual': transformer_data[9].flatten(),
        'predicted': transformer_data[6].flatten(),
        'type': 'Test'
```

```
    }).sort_values('date')
    all_df_transformer = pd.concat([train_df_transformer, val_df_transformer, test_df_transformer]).sort_values('date')


    ax2.plot(all_df_transformer['date'], all_df_transformer['actual'], label='Real', color='blue', linewidth=1.5)
    ax2.plot(all_df_transformer[all_df_transformer['type'] == 'Train']['date'], all_df_transformer[all_df_transformer['type'] == 'Train']['predicted'],
             label='Predito (Treino)', color='red', alpha=0.7, linewidth=1.5)
    ax2.plot(all_df_transformer[all_df_transformer['type'] == 'Validation']['date'], all_df_transformer[all_df_transformer['type'] == 'Validation']['predicted'],
             label='Predito (Validação)', color='green', alpha=0.7, linewidth=1.5)
    ax2.plot(all_df_transformer[all_df_transformer['type'] == 'Test']['date'], all_df_transformer[all_df_transformer['type'] == 'Test']['predicted'],
             label='Predito (Teste)', color='orange', alpha=0.7, linewidth=1.5)


    ax2.set_title(f'PrediçÃţes do Transformer - {dataset_name} (Resample {resample_interval})', fontsize=12)
    ax2.set_xlabel('Data', fontsize=10)
    ax2.set_ylabel('Uso de MemÃşria' + (' (MB)' if dataset_name == 'Prometheus' else ' (Normalizado)'), fontsize=10)
    ax2.legend(loc='upper left', fontsize=10)
    ax2.grid(True, linestyle='--', alpha=0.7)
    ax2.xaxis.set_major_locator(mdates.AutoDateLocator())
    ax2.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    ax2.tick_params(axis='x', rotation=45, labelsize=9)
    ax2.tick_params(axis='y', labelsize=9)


    plt.tight_layout()
    plt.savefig(os.path.join(DATA_DIR, f'{dataset_name.lower()}_{resample_interval}_lstm_transformer.png'), dpi=300,
```

```
        bbox_inches='tight')
    plt.close()


    return (avg_mae_lstm, avg_rmse_lstm, avg_mape_lstm, avg_smape_lstm, std_mae_lstm, std_rmse_lstm, std_mape_lstm, std_smape_lstm,
            avg_mae_transformer, avg_rmse_transformer, avg_mape_transformer, avg_smape_transformer, std_mae_transformer, std_rmse_transformer, std_mape_transformer, std_smape_transformer)


# 7. Executar experimentos
datasets = [
    ("Google Cluster Trace", GCT_FILE_PATH),
    ("Prometheus", PROM_FILE_PATH)
]


results = {}
for dataset_name, file_path in datasets:
    for resample_interval in RESAMPLE_INTERVALS:
        metrics = process_dataset(dataset_name, file_path, resample_interval)
        if metrics is not None:
            results[(dataset_name, resample_interval)] = metrics


# 8. Exibir todos os resultados
print("\nResumo dos Resultados:")
for dataset_name, file_path in datasets:
    for resample_interval in RESAMPLE_INTERVALS:
        if (dataset_name, resample_interval) in results:
            metrics = results[(dataset_name, resample_interval)]
            print(f"\n{dataset_name} - Resample {resample_interval}:")
            print("LSTM:")
            print(f"  Average MAE: {metrics[0]:.4f} (Âś{metrics[4]:.4f})")
            print(f"  Average RMSE: {metrics[1]:.4f} (Âś{metrics[5]:.4f})")
```

```
        print(f"  Average MAPE: {metrics[2]:.4f}% (Âś{metrics[6]:.4f})")
        print(f"  Average SMAPE: {metrics[3]:.4f}% (Âś{metrics[7]:.4f})")
        print("Transformer:")
        print(f"  Average MAE: {metrics[8]:.4f} (Âś{metrics[12]:.4f})")
        print(f"  Average RMSE: {metrics[9]:.4f} (Âś{metrics[13]:.4f})")
        print(f"  Average MAPE: {metrics[10]:.4f}% (Âś{metrics[14]:.4f})")
        print(f"  Average SMAPE: {metrics[11]:.4f}% (Âś{metrics[15]:.4f})")
    else:
        print(f"\n{dataset_name} - Resample {resample_interval}: Skipped due to insufficient data.")
```

APÊNDICE C – transformer-trained.py

```
import pandas as pd
import numpy as np
import os
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error,
    mean_squared_error, mean_absolute_percentage_error
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import pickle
from google.colab import drive




# Montar o Google Drive
drive.mount('/content/drive')


# Definir diretÃşrio de trabalho
os.chdir('/content/drive/MyDrive/Colab_Notebooks/UNESP-MRFO_Paper
    /data')
print("DiretÃşrio atual:", os.getcwd())


# Definir o dispositivo (usar CUDA para NVIDIA A100)
device = torch.device("cuda" if torch.cuda.is_available() else "
    cpu")
print(f"Using device: {device}")


# Definindo constantes
DATA_DIR = "/content/drive/MyDrive/Colab_Notebooks/UNESP-
    MRFO_Paper/data"
FILE_PATH = os.path.join(DATA_DIR, 'ts.pkl')
SEQ_LENGTH = 720  # 12 horas (720 * 1min)
```

```
MB = 1_048_576


# Função para registrar RMSE em um arquivo de log
def log_rmse(iteration, manta_idx, rmse):
    log_file = os.path.join(DATA_DIR, 'rmse_log.txt')
    with open(log_file, 'a') as f:
        if manta_idx == 1:
            f.write(f"\nIteration␣{iteration}\n")
        f.write(f"␣␣Manta␣{manta_idx}:␣RMSE␣=␣{rmse:.4f}\n")


# 1. Carregar e reamostrar os dados para 1 minuto
df = pd.read_pickle(FILE_PATH)
ts = df['value'].astype(float)
# Aplicar suavização com mÃľdia mÃşvel (window=3)
ts = ts.rolling(window=3, min_periods=1).mean()
ts = ts.resample('1min').mean().dropna()
dates = ts.index


# 2. Dividir os dados: 60% treino, 20% validação, 20% teste
train_size = int(0.6 * len(ts))
val_size = int(0.2 * len(ts))
train = ts[:train_size]
val = ts[train_size:train_size + val_size]
test = ts[train_size + val_size:]


# 3. Escalonar os dados
scaler = StandardScaler()
train_scaled = scaler.fit_transform(train.values.reshape(-1, 1))
val_scaled = scaler.transform(val.values.reshape(-1, 1))
test_scaled = scaler.transform(test.values.reshape(-1, 1))


# 4. Criar sequÃłncias
def create_sequences(data, dates, seq_length):
    X, y, y_dates = [], [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
```

```
        y_dates.append(dates[i + seq_length])
    return np.array(X), np.array(y), np.array(y_dates)


X_train, y_train, y_dates_train = create_sequences(train_scaled, dates[:train_size], SEQ_LENGTH)
X_val, y_val, y_dates_val = create_sequences(val_scaled, dates[train_size:train_size + val_size], SEQ_LENGTH)
X_test, y_test, y_dates_test = create_sequences(test_scaled, dates[train_size + val_size:], SEQ_LENGTH)


# 5. Ajustar dimensÃţes para o modelo Transformer
d_model = 128
X_train = np.repeat(X_train, d_model, axis=2)
X_val = np.repeat(X_val, d_model, axis=2)
X_test = np.repeat(X_test, d_model, axis=2)


# 6. Converter para tensores PyTorch e mover para o dispositivo
X_train = torch.tensor(X_train, dtype=torch.float32).to(device)
y_train = torch.tensor(y_train, dtype=torch.float32).to(device)
X_val = torch.tensor(X_val, dtype=torch.float32).to(device)
y_val = torch.tensor(y_val, dtype=torch.float32).to(device)
X_test = torch.tensor(X_test, dtype=torch.float32).to(device)
y_test = torch.tensor(y_test, dtype=torch.float32).to(device)


# 7. Definir codificação posicional
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, max_len=5000):
        super(PositionalEncoding, self).__init__()
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-np.log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)
        self.register_buffer('pe', pe)

```

```python

    def forward(self, x):
        x = x + self.pe[:, :x.size(1), :]
        return x


# 8. Definir o modelo Transformer com dropout
class Encoder(nn.Module):
    def __init__(self, input_dim, d_model, nhead, num_layers, dim_feedforward, dropout=0.1):
        super(Encoder, self).__init__()
        self.pos_encoder = PositionalEncoding(d_model)
        self.encoder_layer = nn.TransformerEncoderLayer(
            d_model=d_model, nhead=nhead, dim_feedforward=dim_feedforward, dropout=dropout, batch_first=True
        )
        self.transformer_encoder = nn.TransformerEncoder(self.encoder_layer, num_layers=num_layers)
        self.dropout = nn.Dropout(dropout)
        self.linear_out = nn.Linear(d_model, 1, bias=True)
        # Inicializar pesos com Xavier
        nn.init.xavier_uniform_(self.linear_out.weight)
        if self.linear_out.bias is not None:
            nn.init.zeros_(self.linear_out.bias)


    def forward(self, src):
        src = self.pos_encoder(src)
        output = self.transformer_encoder(src)
        output = self.dropout(output[:, -1, :])
        output = self.linear_out(output)
        return output


# Outros hiperparÃćmetros fixos
input_dim = d_model
batch_size = 64  # Aumentado para maior estabilidade
num_epochs = 50


# Função para calcular SMAPE
def smape(y_true, y_pred):
```

```
    denominator = (np.abs(y_true) + np.abs(y_pred)) / 2
    diff = np.abs(y_true - y_pred) / denominator
    return 100 * np.mean(diff)


# Função de treinamento e avaliação para uma repetição
def train_and_evaluate(learning_rate, iteration, rep):
    # Fixar hiperparÃćmetros conhecidos
    num_layers = 2
    nhead = 4
    dim_feedforward = 512


    model = Encoder(input_dim, d_model, nhead, num_layers,
        dim_feedforward, dropout=0.1).to(device)
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer,
        T_max=num_epochs)


    for epoch in range(num_epochs):
        model.train()
        for i in range(0, len(X_train), batch_size):
            X_batch = X_train[i:i + batch_size]
            y_batch = y_train[i:i + batch_size]
            outputs = model(X_batch)
            loss = criterion(outputs, y_batch)
            optimizer.zero_grad()
            loss.backward()
            # Gradient clipping
            torch.nn.utils.clip_grad_norm_(model.parameters(),
                max_norm=1.0)
            optimizer.step()
        scheduler.step()


    # Salvar checkpoint do modelo
    model_path = os.path.join(DATA_DIR, f'model_iter_{iteration}
        _rep_{rep}.pth')
    torch.save(model.state_dict(), model_path)

```

```
    model.eval()
    with torch.no_grad():
        y_pred = model(X_test)


    # Mover os dados de volta para a CPU para cálculos de mÃľtricas
    y_pred_rescaled = scaler.inverse_transform(y_pred.cpu().numpy()) / MB
    y_test_rescaled = scaler.inverse_transform(y_test.cpu().numpy()) / MB


    mae = mean_absolute_error(y_test_rescaled, y_pred_rescaled)
    rmse = np.sqrt(mean_squared_error(y_test_rescaled, y_pred_rescaled))
    mape = mean_absolute_percentage_error(y_test_rescaled, y_pred_rescaled) * 100
    smape_val = smape(y_test_rescaled, y_pred_rescaled)


    return mae, rmse, mape, smape_val, model


# Função de mapeamento de valores contínuos para discretos
def map_continuous_to_discrete(value, discrete_values):
    idx = int(round(value * (len(discrete_values) - 1)))
    idx = max(0, min(idx, len(discrete_values) - 1))
    return discrete_values[idx]


# Função de avaliação para uma manta (usada no MRFO)
def evaluate_manta(params, n_repetitions, iteration):
    lr = 10 ** params[0]  # learning_rate (log scale)


    # Executar repetiçÃţes sequencialmente
    results = [train_and_evaluate(lr, iteration, rep) for rep in range(n_repetitions)]
    rmse_list = [result[1] for result in results]  # Minimizar RMSE
```

```
    return np.mean(rmse_list)


# Implementação do MRFO com saídas intermediárias, early stopping e checkpointing
class MRFO:
    def __init__(self, objective_func, bounds, n_mantas=30, max_iter=10, patience=7, n_repetitions=5):
        self.objective_func = objective_func
        self.bounds = np.array(bounds).T  # Shape: (2, dim)
        self.n_mantas = n_mantas
        self.max_iter = max_iter
        self.patience = patience
        self.n_repetitions = n_repetitions
        self.dim = self.bounds.shape[1]


        # Inicializar população
        self.positions = np.zeros((self.n_mantas, self.dim))
        for d in range(self.dim):
            self.positions[:, d] = np.random.uniform(self.bounds[0, d], self.bounds[1, d], self.n_mantas)
        self.fitness = np.array([float('inf')] * self.n_mantas)
        self.best_position = None
        self.best_fitness = float('inf')
        self.no_improvement_count = 0  # Contador para early stopping
        self.current_iteration = 0


    def load_checkpoint(self):
        checkpoint_files = [f for f in os.listdir(DATA_DIR) if f.startswith('mrfo_checkpoint_iter_') and f.endswith('.pkl')]
        if checkpoint_files:
            latest_checkpoint = max(checkpoint_files, key=lambda x: int(x.split('_iter_')[1].split('.pkl')[0]))
            checkpoint_path = os.path.join(DATA_DIR, latest_checkpoint)
            with open(checkpoint_path, 'rb') as f:
                checkpoint = pickle.load(f)
            self.current_iteration = checkpoint['iteration'] + 1
```

```
            self.best_position = checkpoint['best_position']
            self.best_fitness = checkpoint['best_fitness']
            self.positions = checkpoint['positions']
            self.fitness = checkpoint['fitness']
            self.no_improvement_count = checkpoint['no_improvement_count']
            print(f"Resumed from checkpoint: {checkpoint_path}, starting at iteration {self.current_iteration}")
            return True
        return False


    def save_checkpoint(self, t):
        checkpoint = {
            'iteration': t,
            'best_position': self.best_position,
            'best_fitness': self.best_fitness,
            'positions': self.positions,
            'fitness': self.fitness,
            'no_improvement_count': self.no_improvement_count
        }
        checkpoint_path = os.path.join(DATA_DIR, f'mrfo_checkpoint_iter_{t+1}.pkl')
        with open(checkpoint_path, 'wb') as f:
            pickle.dump(checkpoint, f)


    def optimize(self):
        # Tentar carregar checkpoint
        if self.load_checkpoint():
            start_iter = self.current_iteration
        else:
            start_iter = 0


        for t in range(start_iter, self.max_iter):
            print(f"\nIteration {t+1}/{self.max_iter}")
            # Avaliar fitness de todas as mantas sequencialmente
            fitness_results = [self.objective_func(self.positions[i], self.n_repetitions, t+1) for i in range(self.n_mantas)]
            self.fitness = np.array(fitness_results)
```

```python


            # Exibir resultados, atualizar o melhor fitness e registrar RMSE no log
            for i in range(self.n_mantas):
                print(f"  Manta {i+1}/{self.n_mantas}: Fitness (RMSE) = {self.fitness[i]:.4f}")
                log_rmse(t+1, i+1, self.fitness[i])  # Registrar RMSE no arquivo
                if self.fitness[i] < self.best_fitness:
                    self.best_fitness = self.fitness[i]
                    self.best_position = self.positions[i].copy()
                    self.no_improvement_count = 0  # Resetar o contador
                    print(f"  New Best Fitness: {self.best_fitness:.4f}")
                else:
                    self.no_improvement_count += 1


            # Salvar checkpoint
            self.save_checkpoint(t)


            # CritÃľrio de parada precoce
            if self.no_improvement_count >= self.patience:
                print(f"\nEarly stopping triggered after {t+1} iterations due to no improvement for {self.patience} iterations.")
                break


            # Atualizar posiçÃţes usando Chain Foraging, Cyclone Foraging e Somersault Foraging
            for i in range(self.n_mantas):
                r = np.random.random(self.dim)
                r1 = np.random.random()


                # Chain Foraging
                if r1 < 0.5:
                    if i == 0:
```

```
                    self.positions[i] = self.positions[i] + r * (self.best_position - self.positions[i]) + \
                                        r * (self.best_position - self.positions[i])
                else:
                    self.positions[i] = self.positions[i] + r * (self.positions[i-1] - self.positions[i]) + \
                                        r * (self.best_position - self.positions[i])


            # Cyclone Foraging
            else:
                beta = 2 * np.exp(r1 * (self.max_iter - t + 1) / self.max_iter) * np.sin(2 * np.pi * r1)
                if r1 < 0.5:
                    self.positions[i] = self.positions[i] + r * (self.best_position - beta * self.positions[i])
                else:
                    idx = np.random.randint(0, self.n_mantas)
                    self.positions[i] = self.positions[i] + r * (self.positions[idx] - beta * self.positions[i])


            # Somersault Foraging
            r2 = np.random.random()
            self.positions[i] = self.positions[i] + 0.5 * (self.best_position + self.positions[i]) * (2 * r2 - 1)


            # Garantir que as posiçÃţes estejam dentro dos limites
            self.positions[i] = np.clip(self.positions[i], self.bounds[0], self.bounds[1])

```

```
        return self.best_position, self.best_fitness


# MRFO para otimizar hiperparÃćmetros
n_repetitions = 5
bounds = [
    [-3.0458, -2.9586],  # log10(learning_rate): [0.0009, 0.0011]
]


mrfo = MRFO(lambda params, reps, iter: evaluate_manta(params,
    reps, iter), bounds, n_mantas=30, max_iter=10, patience=7,
    n_repetitions=n_repetitions)
best_position, best_fitness = mrfo.optimize()


# Mapear a melhor posição para hiperparÃćmetros
best_lr = 10 ** best_position[0]


# Treinar o modelo com a melhor configuração para obter mÃľtricas
    finais
print(f"\nBest MRFO Configuration: LR={best_lr}, Layers=2, Heads
    =4, FF=512")


# Executar repetiçÃţes sequencialmente
results = [train_and_evaluate(best_lr, iteration="final", rep=rep
    ) for rep in range(n_repetitions)]


mae_list = [result[0] for result in results]
rmse_list = [result[1] for result in results]
mape_list = [result[2] for result in results]
smape_list = [result[3] for result in results]
models = [result[4] for result in results]


# Registrar mÃľtricas finais no log
log_file = os.path.join(DATA_DIR, 'rmse_log.txt')
with open(log_file, 'a') as f:
```

```
    f.write("\nFinal Run Metrics\n")
    for rep, (mae, rmse, mape, smape_val, _) in enumerate(results):
        f.write(f"  Repetition {rep+1}/{n_repetitions}\n")
        f.write(f"    MAE: {mae:.4f}, RMSE: {rmse:.4f}, MAPE: {mape:.4f}%, SMAPE: {smape_val:.4f}%\n")


    avg_mae = np.mean(mae_list)
    avg_rmse = np.mean(rmse_list)
    avg_mape = np.mean(mape_list)
    avg_smape = np.mean(smape_list)
    std_mae = np.std(mae_list)
    std_rmse = np.std(rmse_list)
    std_mape = np.std(mape_list)
    std_smape = np.std(smape_list)


    f.write(f"  Average MAE: {avg_mae:.4f} (Âś{std_mae:.4f}), Average RMSE: {avg_rmse:.4f} (Âś{std_rmse:.4f})\n")
    f.write(f"  Average MAPE: {avg_mape:.4f}% (Âś{std_mape:.4f}), Average SMAPE: {avg_smape:.4f}% (Âś{std_smape:.4f})\n")


# Exibir mÃľtricas finais no console
for rep, (mae, rmse, mape, smape_val, _) in enumerate(results):
    print(f"  Repetition {rep+1}/{n_repetitions}")
    print(f"    MAE: {mae:.4f}, RMSE: {rmse:.4f}, MAPE: {mape:.4f}%, SMAPE: {smape_val:.4f}%")


print(f"  Average MAE: {avg_mae:.4f} (Âś{std_mae:.4f}), Average RMSE: {avg_rmse:.4f} (Âś{std_rmse:.4f})")
print(f"  Average MAPE: {avg_mape:.4f}% (Âś{std_mape:.4f}), Average SMAPE: {avg_smape:.4f}% (Âś{std_smape:.4f})")


# Salvar o melhor modelo
best_model = models[0]
torch.save(best_model.state_dict(), os.path.join(DATA_DIR, 'best_model.pth'))

```

```python

# 9. Fazer previsÃţes
best_model.eval()
with torch.no_grad():
    y_train_pred = best_model(X_train)
    y_test_pred = best_model(X_test)


# 10. Reverter o escalonamento e converter para MB
y_train_pred_mb = scaler.inverse_transform(y_train_pred.cpu().numpy()) / MB
y_train_mb = scaler.inverse_transform(y_train.cpu().numpy()) / MB
y_test_pred_mb = scaler.inverse_transform(y_test_pred.cpu().numpy()) / MB
y_test_mb = scaler.inverse_transform(y_test.cpu().numpy()) / MB


# 11. Preparar dados para plotagem
train_df = pd.DataFrame({
    'date': y_dates_train,
    'actual': y_train_mb.flatten(),
    'predicted': y_train_pred_mb.flatten()
}).sort_values('date')


test_df = pd.DataFrame({
    'date': y_dates_test,
    'actual': y_test_mb.flatten(),
    'predicted': y_test_pred_mb.flatten()
}).sort_values('date')


# 12. Plotar os resultados
plt.style.use('default')
fig, axs = plt.subplots(2, 1, figsize=(15, 10), sharex=False)


axs[0].plot(train_df['date'], train_df['actual'], label='Real', color='blue', linewidth=1.5)
axs[0].plot(train_df['date'], train_df['predicted'], label='Predito', color='red', alpha=0.7, linewidth=1.5)
axs[0].set_title('Conjunto de Treinamento (60%)', fontsize=12,
```

```
    pad=10)
axs[0].set_ylabel('Consumo␣de␣MemÃşria␣(MB)', fontsize=10)
axs[0].legend(loc='upper␣left', fontsize=10)
axs[0].grid(True, linestyle='--', alpha=0.7)


axs[1].plot(test_df['date'], test_df['actual'], label='Real',
    color='blue', linewidth=1.5)
axs[1].plot(test_df['date'], test_df['predicted'], label='Predito
    ', color='red', alpha=0.7, linewidth=1.5)
axs[1].set_title('Conjunto␣de␣Teste␣(20%)', fontsize=12, pad=10)
axs[1].set_xlabel('Data', fontsize=10)
axs[1].set_ylabel('Consumo␣de␣MemÃşria␣(MB)', fontsize=10)
axs[1].legend(loc='upper␣left', fontsize=10)
axs[1].grid(True, linestyle='--', alpha=0.7)


for ax in axs:
    ax.xaxis.set_major_locator(mdates.AutoDateLocator())
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    ax.tick_params(axis='x', rotation=45, labelsize=9)
    ax.tick_params(axis='y', labelsize=9)


plt.suptitle('PrediçÃţes␣do␣Transformer␣Otimizado␣-␣Prometheus␣(
    MB,␣Resample␣1min)', fontsize=14, y=0.98)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.savefig(os.path.join(DATA_DIR, '
    prometheus_transformer_mrfo_1min.png'), dpi=300, bbox_inches='
    tight')
plt.close()
```

APÊNDICE D – Implementação do Modelo

D.1 model.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import mlflow
import mlflow.keras
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
from keras.optimizers import Adam
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error,
    mean_squared_error, mean_absolute_percentage_error
import matplotlib.dates as mdates
import tensorflow as tf
import os


# Configurar MLflow
mlflow.set_tracking_uri("http://localhost:5001")
mlflow.set_experiment("Prometheus_LSTM_Experiment_MRFO_15min")


# Definindo constantes
DATA_DIR = "/Users/wrosalem/Documents/0x0-Coding/GitHub/
    dissertacao/Experiments/data/"
FILE_PATH = os.path.join(DATA_DIR, 'ts.pkl')
SEQ_LENGTH = 48  # 12 horas (48 * 15min)
MB = 1_048_576


# 1. Carregar e reamostrar os dados para 15 minutos
df = pd.read_pickle(FILE_PATH)
ts = df['value'].astype(float)
# Aplicar suavização com mÃľdia mÃşvel (window=3)
ts = ts.rolling(window=3, min_periods=1).mean()
ts = ts.resample('15min').mean().dropna()
```

```python
dates = ts.index


# 2. Dividir os dados: 60% treino, 20% validação, 20% teste
train_size = int(0.6 * len(ts))
val_size = int(0.2 * len(ts))
train = ts[:train_size]
val = ts[train_size:train_size + val_size]
test = ts[train_size + val_size:]


# 3. Escalonar os dados
scaler = StandardScaler()
train_scaled = scaler.fit_transform(train.values.reshape(-1, 1))
val_scaled = scaler.transform(val.values.reshape(-1, 1))
test_scaled = scaler.transform(test.values.reshape(-1, 1))


# 4. Criar sequÃłncias
def create_sequences(data, dates, seq_length):
    X, y, y_dates = [], [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
        y_dates.append(dates[i + seq_length])
    return np.array(X), np.array(y), np.array(y_dates)


X_train, y_train, y_dates_train = create_sequences(train_scaled,
    dates[:train_size], SEQ_LENGTH)
X_val, y_val, y_dates_val = create_sequences(val_scaled, dates[
    train_size:train_size + val_size], SEQ_LENGTH)
X_test, y_test, y_dates_test = create_sequences(test_scaled,
    dates[train_size + val_size:], SEQ_LENGTH)


# 5. Reshape para 3D
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape
    [1], 1))
X_val = np.reshape(X_val, (X_val.shape[0], X_val.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1)
    )
```

```python


# 6. Função para calcular SMAPE
def smape(y_true, y_pred):
    denominator = (np.abs(y_true) + np.abs(y_pred)) / 2
    diff = np.abs(y_true - y_pred) / denominator
    return 100 * np.mean(diff)


# 7. Função Objetivo para MRFO
def objective_function(params):
    lstm_units = max(50, min(150, int(params[0])))
    dropout_rate = max(0.2, min(0.5, params[1]))
    batch_size = max(32, min(128, int(params[2])))
    learning_rate = max(0.0001, min(0.001, params[3]))
    n_steps = max(10, min(min(len(train_scaled), len(test_scaled)) - 1, int(params[4])))
    epochs = max(10, min(200, int(params[5])))


    # Criar sequÃłncias com n_steps
    X_train, y_train, _ = create_sequences(train_scaled, dates[:train_size], n_steps)
    X_test, y_test, y_dates_test = create_sequences(test_scaled, dates[train_size + val_size:], n_steps)


    if X_test.shape[0] == 0:
        return float("inf"), float("inf"), float("inf"), float("inf")


    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))


    # Definir o modelo
    model = Sequential()
    model.add(LSTM(lstm_units, activation='tanh', input_shape=(n_steps, 1), return_sequences=True))
```

```
    model.add(Dropout(dropout_rate))
    model.add(LSTM(lstm_units, activation='tanh'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1))


    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mse')


    # Treinar o modelo
    model.fit(X_train, y_train, epochs=epochs, batch_size=
       batch_size, verbose=0)


    # Fazer previsÃţes
    test_pred = model.predict(X_test, verbose=0)
    test_pred = scaler.inverse_transform(test_pred) / MB
    y_test_rescaled = scaler.inverse_transform(y_test) / MB


    # Calcular mÃĽtricas
    rmse = np.sqrt(mean_squared_error(y_test_rescaled, test_pred))
    mse = mean_squared_error(y_test_rescaled, test_pred)
    mae = mean_absolute_error(y_test_rescaled, test_pred)
    mape = mean_absolute_percentage_error(y_test_rescaled,
       test_pred) * 100
    smape_val = smape(y_test_rescaled, test_pred)


    # Registrar no MLflow
    with mlflow.start_run():
        mlflow.set_tag("modelo", "LSTM")
        mlflow.set_tag("otimização", "MRFO")
        mlflow.set_tag("versão", "v1.0")
        mlflow.log_param("lstm_units", lstm_units)
        mlflow.log_param("dropout_rate", dropout_rate)
        mlflow.log_param("batch_size", batch_size)
        mlflow.log_param("learning_rate", learning_rate)
        mlflow.log_param("n_steps", n_steps)
        mlflow.log_param("epochs", epochs)
        mlflow.log_param("resample_interval", "15min")
```

```
        mlflow.log_param("seq_length", n_steps)
        mlflow.log_metric("rmse", rmse)
        mlflow.log_metric("mse", mse)
        mlflow.log_metric("mae", mae)
        mlflow.log_metric("mape", mape)
        mlflow.log_metric("smape", smape_val)
        mlflow.keras.log_model(model, "model")


    return rmse, mse, mae, mape, smape_val


# 8. Implementação do MRFO
class MRFO:
    def __init__(self, obj_func, dim, SearchAgents_no, max_iter, lb, ub):
        self.obj_func = obj_func
        self.dim = dim
        self.SearchAgents_no = SearchAgents_no
        self.max_iter = max_iter
        self.lb = lb
        self.ub = ub
        self.positions = np.random.uniform(0, 1, (self.SearchAgents_no, self.dim)) * (self.ub - self.lb) + self.lb
        self.fitness = np.array([self.obj_func(self.clip_params(ind))[0] for ind in self.positions])
        self.best_idx = np.argmin(self.fitness)
        self.gbest = self.positions[self.best_idx].copy()
        self.gbest_fitness = self.fitness[self.best_idx]


    def clip_params(self, params):
        params[0] = max(50, min(150, params[0]))  # lstm_units
        params[1] = max(0.2, min(0.5, params[1]))  # dropout_rate
        params[2] = max(32, min(128, params[2]))  # batch_size
        params[3] = max(0.0001, min(0.001, params[3]))  # learning_rate
        params[4] = max(10, min(min(len(train_scaled), len(test_scaled)) - 1, params[4]))  # n_steps
        params[5] = max(10, min(200, params[5]))  # epochs
        return params
```

```python


    def chain_foraging(self, i):
        r = np.random.rand(self.dim)
        self.positions[i] = self.positions[i] + r * (self.gbest - self.positions[i])
        self.positions[i] = self.clip_params(self.positions[i])


    def cyclone_foraging(self, i, t, max_iter):
        r = np.random.rand(self.dim)
        A = 2 * (1 - t / max_iter)
        direction = np.random.choice([-1, 1], size=self.dim)
        self.positions[i] = self.positions[i] + A * direction * r * (self.gbest - self.positions[i])
        self.positions[i] = self.clip_params(self.positions[i])


    def somersault_foraging(self, i):
        S = 2 * np.random.rand(self.dim) - 1
        somersault_factor = 2
        self.positions[i] = self.positions[i] + somersault_factor * (S * self.gbest - self.positions[i])
        self.positions[i] = self.clip_params(self.positions[i])


    def optimize(self):
        for t in range(self.max_iter):
            for i in range(self.SearchAgents_no):
                if np.random.rand() < 0.5:
                    self.chain_foraging(i)
                else:
                    self.cyclone_foraging(i, t, self.max_iter)


                fitness_candidate = self.obj_func(self.clip_params(self.positions[i]))[0]


                if fitness_candidate < self.fitness[i]:
                    self.fitness[i] = fitness_candidate
                    if fitness_candidate < self.gbest_fitness:
```

```
                        self.gbest_fitness = fitness_candidate
                        self.gbest = self.positions[i].copy()


            for i in range(self.SearchAgents_no):
                self.somersault_foraging(i)


            print(f"Iteração {t+1}/{self.max_iter}, Melhor Fitness (RMSE): {self.gbest_fitness:.4f}")


        return self.gbest, self.gbest_fitness


# 9. Configurar e executar MRFO
dim = 6
SearchAgents_no = 10
max_iter = 20
lb = np.array([50, 0.2, 32, 0.0001, 10, 10])
ub = np.array([150, 0.5, 128, 0.001, 100, 200])
mrfo = MRFO(objective_function, dim, SearchAgents_no, max_iter, lb, ub)
best_params, best_fitness = mrfo.optimize()


# 10. Treinar com melhores hiperparÃćmetros e calcular mÃľtricas
def train_and_evaluate(params):
    lstm_units = max(50, min(150, int(params[0])))
    dropout_rate = max(0.2, min(0.5, params[1]))
    batch_size = max(32, min(128, int(params[2])))
    learning_rate = max(0.0001, min(0.001, params[3]))
    n_steps = max(10, min(min(len(train_scaled), len(test_scaled)) - 1, int(params[4])))
    epochs = max(10, min(200, int(params[5])))


    X_train, y_train, _ = create_sequences(train_scaled, dates[:train_size], n_steps)
    X_test, y_test, y_dates_test = create_sequences(test_scaled, dates[train_size + val_size:], n_steps)

```

```
    if X_test.shape[0] == 0:
        return float("inf"), float("inf"), float("inf"), float("inf"), None


    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))


    model = Sequential()
    model.add(LSTM(lstm_units, activation='tanh', input_shape=(n_steps, 1), return_sequences=True))
    model.add(Dropout(dropout_rate))
    model.add(LSTM(lstm_units, activation='tanh'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1))


    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mse')


    model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, verbose=0)


    test_pred = model.predict(X_test, verbose=0)
    test_pred = scaler.inverse_transform(test_pred) / MB
    y_test_rescaled = scaler.inverse_transform(y_test) / MB


    rmse = np.sqrt(mean_squared_error(y_test_rescaled, test_pred))
    mse = mean_squared_error(y_test_rescaled, test_pred)
    mae = mean_absolute_error(y_test_rescaled, test_pred)
    mape = mean_absolute_percentage_error(y_test_rescaled, test_pred) * 100
    smape_val = smape(y_test_rescaled, test_pred)


```

```
    return rmse, mse, mae, mape, smape_val, model, y_test_rescaled, test_pred, y_dates_test


# 11. Executar repetiçÃţes com melhores hiperparÃćmetros
n_repetitions = 5
with mlflow.start_run(run_name="Best_MRFO_Run"):
    mlflow.log_param("lstm_units", int(best_params[0]))
    mlflow.log_param("dropout_rate", best_params[1])
    mlflow.log_param("batch_size", int(best_params[2]))
    mlflow.log_param("learning_rate", best_params[3])
    mlflow.log_param("n_steps", int(best_params[4]))
    mlflow.log_param("epochs", int(best_params[5]))
    mlflow.log_param("resample_interval", "15min")
    mlflow.log_param("seq_length", SEQ_LENGTH)


    print(f"\nBest␣MRFO␣Configuration:␣LSTM␣Units={int(best_params[0])},␣Dropout␣Rate={best_params[1]:.2f},␣"
          f"Batch␣Size={int(best_params[2])},␣Learning␣Rate={best_params[3]:.6f},␣"
          f"n_steps={int(best_params[4])},␣Epochs={int(best_params[5])}")


    results = [train_and_evaluate(best_params) for _ in range(n_repetitions)]


    rmse_list = [result[0] for result in results]
    mse_list = [result[1] for result in results]
    mae_list = [result[2] for result in results]
    mape_list = [result[3] for result in results]
    smape_list = [result[4] for result in results]
    models = [result[5] for result in results]
    y_test_rescaled = results[0][6]
    test_pred = results[0][7]
    y_dates_test = results[0][8]


    for rep, (rmse, mse, mae, mape, smape_val, _, _, _, _) in enumerate(results):
```

```
        print(f"  Repetition {rep+1}/{n_repetitions}")
        print(f"    MAE: {mae}, RMSE: {rmse}, MAPE: {mape}%, SMAPE: {smape_val}%")


    avg_rmse = np.mean(rmse_list)
    avg_mse = np.mean(mse_list)
    avg_mae = np.mean(mae_list)
    avg_mape = np.mean(mape_list)
    avg_smape = np.mean(smape_list)
    std_rmse = np.std(rmse_list)
    std_mse = np.std(mse_list)
    std_mae = np.std(mae_list)
    std_mape = np.std(mape_list)
    std_smape = np.std(smape_list)


    print(f"  Average RMSE: {avg_rmse} (Âś{std_rmse}), Average MSE: {avg_mse} (Âś{std_mse})")
    print(f"  Average MAE: {avg_mae} (Âś{std_mae}), Average MAPE: {avg_mape}% (Âś{std_mape})")
    print(f"  Average SMAPE: {avg_smape}% (Âś{std_smape})")


    mlflow.log_metric("avg_rmse", avg_rmse)
    mlflow.log_metric("std_rmse", std_rmse)
    mlflow.log_metric("avg_mse", avg_mse)
    mlflow.log_metric("std_mse", std_mse)
    mlflow.log_metric("avg_mae", avg_mae)
    mlflow.log_metric("std_mae", std_mae)
    mlflow.log_metric("avg_mape", avg_mape)
    mlflow.log_metric("std_mape", std_mape)
    mlflow.log_metric("avg_smape", avg_smape)
    mlflow.log_metric("std_smape", std_smape)


    best_model = models[0]
    mlflow.keras.log_model(best_model, "best_model")


# 12. Plotar resultados
plt.figure(figsize=(12, 6))
```

```
plt.plot(y_dates_test, scaler.inverse_transform(y_test_rescaled * MB), label="Valores Reais", color='blue')
plt.plot(y_dates_test, scaler.inverse_transform(test_pred * MB), label="PrevisÃţes do Modelo", color='red', linestyle='--')
plt.title("Valores Reais vs PrevisÃţes do Modelo - Consumo de MemÃşria")
plt.xlabel("Data")
plt.ylabel("Consumo de MemÃşria (Bytes)")
plt.legend()
plt.gca().xaxis.set_major_locator(mdates.AutoDateLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig(os.path.join(DATA_DIR, 'prometheus_lstm_mrfo_15min.png'), dpi=300, bbox_inches='tight')
plt.close()
```

D.2 Tabela de execuções

Tabela 11 – Resultados do algoritmo MRFO otimizando a LSTM (janela de 15 min) para métricas coletadas via Prometheus.

#	Batch	Dropout	Épocas	Taxa de Aprendizado	Unidades	RMSE	SMAPE
1	32	0.2039	199	0.000419	56	19.885	1.288
2	32	0.2027	167	0.000376	55	20.205	1.306
3	32	0.2184	199	0.000408	69	20.209	1.273
4	32	0.2291	199	0.000393	90	20.308	1.221
5	32	0.2195	188	0.000442	79	20.359	1.233
6	32	0.2024	177	0.000395	53	20.449	1.273
7	32	0.2032	199	0.000379	55	20.481	1.313
8	34	0.4718	150	0.000425	128	20.492	1.542
9	34	0.4718	150	0.000425	128	20.505	1.514
10	32	0.2006	189	0.000478	50	20.559	1.297
11	32	0.2357	199	0.000465	75	20.606	1.238
12	34	0.4708	150	0.000425	128	20.643	1.492
13	32	0.2238	199	0.000497	56	20.662	1.271
14	32	0.2203	199	0.000489	55	20.682	1.305
15	32	0.2000	151	0.000423	64	20.698	1.281
16	32	0.2193	199	0.000517	71	20.733	1.267
17	32	0.2036	199	0.000431	58	20.739	1.327
18	32	0.2140	199	0.000463	63	20.782	1.325
19	32	0.2180	199	0.000453	63	20.788	1.311
20	32	0.2000	176	0.000427	60	20.792	1.290
21	32	0.2047	199	0.000461	59	20.796	1.323

Continuação na próxima página

#	Batch	Dropout	Épocas	Taxa de Aprendizado	Unidades	RMSE	SMAPE
22	32	0.2041	199	0.000422	57	20.805	1.322
23	32	0.2044	199	0.000460	60	20.812	1.330
24	32	0.2172	199	0.000492	63	20.823	1.305
25	32	0.2326	199	0.000522	77	20.835	1.256
26	32	0.2191	199	0.000513	61	20.867	1.318
27	32	0.2188	199	0.000519	64	20.877	1.296
28	32	0.2035	199	0.000442	60	20.888	1.334
29	32	0.2187	199	0.000442	63	20.893	1.319
30	32	0.2170	199	0.000474	63	20.895	1.312
31	32	0.2160	199	0.000551	61	20.914	1.333
32	32	0.2025	199	0.000463	58	20.916	1.356
33	32	0.2031	199	0.000436	56	20.927	1.341
34	32	0.2187	199	0.000536	63	20.931	1.322
35	32	0.2178	199	0.000547	63	20.938	1.316
36	32	0.2329	199	0.000557	77	20.942	1.278
37	32	0.2190	199	0.000530	63	20.954	1.335
38	32	0.2191	199	0.000523	62	20.979	1.339
39	32	0.2185	199	0.000550	65	20.998	1.334
40	32	0.2184	199	0.000557	66	21.011	1.308
41	32	0.2350	199	0.000588	80	21.015	1.269
42	32	0.2037	199	0.000524	59	21.035	1.373
43	32	0.2033	199	0.000511	58	21.042	1.338
44	32	0.2038	199	0.000521	58	21.056	1.342
45	32	0.2036	199	0.000518	57	21.070	1.348
46	32	0.2042	199	0.000527	58	21.082	1.347
47	32	0.2324	199	0.000600	79	21.082	1.275
48	32	0.2034	199	0.000535	58	21.092	1.346
49	32	0.2041	199	0.000545	60	21.101	1.358
50	32	0.2195	199	0.000582	65	21.105	1.333
51	32	0.2183	199	0.000580	63	21.112	1.341
52	32	0.2182	199	0.000586	63	21.119	1.341
53	32	0.2188	199	0.000592	62	21.131	1.351
54	32	0.2333	199	0.000613	79	21.133	1.290
55	32	0.2178	199	0.000618	63	21.137	1.351
56	32	0.2210	199	0.000645	64	21.155	1.347
57	32	0.2200	199	0.000657	64	21.166	1.357
58	32	0.2211	199	0.000651	64	21.174	1.354
59	32	0.2324	199	0.000623	80	21.176	1.300
60	32	0.2337	199	0.000633	79	21.178	1.288
61	32	0.2175	199	0.000669	66	21.184	1.358
62	32	0.2185	199	0.000679	65	21.189	1.349
63	32	0.2186	199	0.000685	65	21.192	1.351
64	32	0.2072	199	0.000693	61	21.196	1.366
65	32	0.2172	199	0.000693	66	21.203	1.359
66	32	0.2329	199	0.000662	80	21.205	1.301
67	32	0.2175	199	0.000710	66	21.209	1.371
68	32	0.2182	199	0.000716	66	21.215	1.359
69	32	0.2189	199	0.000722	65	21.215	1.352
70	32	0.2300	199	0.000756	77	21.222	1.326
71	32	0.2220	199	0.000737	66	21.229	1.366
72	32	0.2211	199	0.000744	66	21.236	1.370
73	32	0.2212	199	0.000750	68	21.243	1.349
74	32	0.2038	199	0.000752	61	21.247	1.391

Continuação na próxima página

#	Batch	Dropout	Épocas	Taxa de Aprendizado	Unidades	RMSE	SMAPE
75	32	0.2190	199	0.000753	67	21.249	1.365
76	32	0.2327	199	0.000732	80	21.253	1.330
77	32	0.2012	199	0.000764	60	21.254	1.393
78	32	0.2221	199	0.000777	68	21.257	1.363
79	32	0.2220	199	0.000783	68	21.259	1.369
80	32	0.2218	199	0.000789	67	21.268	1.372
81	32	0.2041	199	0.000787	60	21.270	1.393
82	32	0.2229	199	0.000801	68	21.271	1.381
83	32	0.2191	199	0.000807	67	21.278	1.378
84	32	0.2231	199	0.000807	68	21.284	1.367
85	32	0.2213	199	0.000808	68	21.286	1.376
86	32	0.2220	199	0.000814	68	21.290	1.370
87	32	0.2308	199	0.000815	78	21.292	1.330
88	32	0.2225	199	0.000820	68	21.293	1.372
89	32	0.2195	199	0.000825	67	21.294	1.382
90	32	0.2228	199	0.000826	69	21.295	1.370
91	32	0.2044	199	0.000829	60	21.299	1.407
92	32	0.2310	199	0.000811	79	21.300	1.333
93	32	0.2133	199	0.000833	65	21.307	1.401
94	32	0.2313	199	0.000818	79	21.309	1.337
95	32	0.2200	199	0.000839	68	21.315	1.374
96	32	0.2234	199	0.000838	69	21.317	1.377
97	32	0.2214	199	0.000840	68	21.320	1.387
98	32	0.2051	199	0.000843	60	21.321	1.407
99	32	0.2049	199	0.000846	60	21.322	1.403
100	32	0.2224	199	0.000844	67	21.323	1.385
101	32	0.2235	199	0.000850	69	21.324	1.386
102	32	0.2220	199	0.000851	68	21.325	1.388
103	32	0.2198	199	0.000853	67	21.327	1.394
104	32	0.2230	199	0.000852	69	21.330	1.374
105	32	0.2320	199	0.000835	80	21.337	1.340
106	32	0.2233	199	0.000858	69	21.340	1.383
107	32	0.2225	199	0.000859	68	21.343	1.390
108	32	0.2219	199	0.000860	68	21.346	1.389
109	32	0.2054	199	0.000864	60	21.347	1.410
110	32	0.2240	199	0.000858	70	21.350	1.385
111	32	0.2057	199	0.000868	60	21.351	1.410
112	32	0.2226	199	0.000865	68	21.351	1.392
113	32	0.2331	199	0.000846	80	21.358	1.353
114	32	0.2241	199	0.000861	71	21.358	1.378
115	32	0.2216	199	0.000867	68	21.360	1.394
116	32	0.2236	199	0.000869	70	21.364	1.385
117	32	0.2179	199	0.000873	66	21.369	1.395
118	32	0.2060	199	0.000873	60	21.370	1.418
119	32	0.2229	199	0.000874	70	21.372	1.387
120	32	0.2248	199	0.000872	71	21.375	1.381
121	32	0.2218	199	0.000876	68	21.380	1.397
122	32	0.2242	199	0.000878	71	21.385	1.381
123	32	0.2336	199	0.000860	80	21.386	1.352
124	32	0.2063	199	0.000883	60	21.389	1.419
125	32	0.2184	199	0.000887	66	21.391	1.401
126	32	0.2244	199	0.000884	71	21.392	1.387
127	32	0.2247	199	0.000890	72	21.395	1.389

Continuação na próxima página

#	Batch	Dropout	Épocas	Taxa de Aprendizado	Unidades	RMSE	SMAPE
128	32	0.2219	199	0.000892	69	21.397	1.399
129	32	0.2338	199	0.000873	80	21.399	1.349
130	32	0.2065	199	0.000895	60	21.400	1.423
131	32	0.2222	199	0.000899	69	21.402	1.394
132	32	0.2249	199	0.000896	72	21.404	1.393
133	32	0.2187	199	0.000901	67	21.407	1.408
134	32	0.2067	199	0.000906	60	21.409	1.425
135	32	0.2319	199	0.000903	80	21.411	1.358
136	32	0.2335	199	0.000887	81	21.412	1.352
137	32	0.2230	199	0.000903	72	21.413	1.391
138	32	0.2233	199	0.000909	73	21.414	1.395
139	32	0.2331	199	0.000890	81	21.415	1.354
140	32	0.2236	199	0.000912	72	21.417	1.396
141	32	0.2069	199	0.000918	60	21.420	1.427
142	32	0.2241	199	0.000916	73	21.421	1.398
143	32	0.2329	199	0.000898	81	21.422	1.353
144	32	0.2228	199	0.000922	72	21.423	1.400
145	32	0.2239	199	0.000928	74	21.425	1.401
146	32	0.2072	199	0.000920	60	21.428	1.430
147	32	0.2190	199	0.000931	68	21.428	1.414
148	32	0.2231	199	0.000935	73	21.430	1.402
149	32	0.2223	199	0.000934	72	21.433	1.407
150	32	0.2244	199	0.000940	74	21.435	1.404
151	32	0.2326	199	0.000922	82	21.436	1.360
152	32	0.2073	199	0.000942	60	21.437	1.433
153	32	0.2193	199	0.000946	69	21.438	1.419
154	32	0.2250	199	0.000944	74	21.440	1.405
155	32	0.2231	199	0.000951	73	21.442	1.414
156	32	0.2076	199	0.000950	60	21.445	1.436
157	32	0.2237	199	0.000956	74	21.447	1.413
158	32	0.2196	199	0.000960	69	21.448	1.420
159	32	0.2242	199	0.000962	74	21.450	1.417
160	32	0.2241	199	0.000969	75	21.452	1.408
161	32	0.2322	199	0.000952	82	21.453	1.366
162	32	0.2078	199	0.000967	60	21.454	1.440
163	32	0.2197	199	0.000974	70	21.455	1.424
164	32	0.2238	199	0.000978	74	21.456	1.421
165	32	0.2255	199	0.000983	75	21.459	1.418
166	32	0.2080	199	0.000975	60	21.461	1.443
167	32	0.2199	199	0.000979	70	21.462	1.425
168	32	0.2250	199	0.000986	75	21.463	1.420
169	32	0.2247	199	0.000992	75	21.465	1.415
170	32	0.2324	199	0.000975	83	21.465	1.372
171	32	0.2083	199	0.000989	60	21.467	1.446
172	32	0.2201	199	0.000996	70	21.468	1.427
173	32	0.2260	199	0.001001	75	21.470	1.424
174	32	0.2260	199	0.001007	75	21.472	1.421
175	32	0.2084	199	0.001000	60	21.473	1.449
176	32	0.2203	199	0.001006	70	21.475	1.429
177	32	0.2258	199	0.001012	76	21.476	1.427
178	32	0.2264	199	0.001018	76	21.478	1.423
179	32	0.2327	199	0.001001	83	21.478	1.379
180	32	0.2086	199	0.001014	60	21.480	1.453

Continuação na próxima página

#	Batch	Dropout	Épocas	Taxa de Aprendizado	Unidades	RMSE	SMAPE
181	32	0.2204	199	0.001021	71	21.481	1.432
182	32	0.2261	199	0.001027	76	21.483	1.425
183	32	0.2267	199	0.001033	76	21.485	1.423
184	32	0.2338	199	0.001008	84	21.486	1.384
185	32	0.2087	199	0.001029	60	21.487	1.457
186	32	0.2206	199	0.001035	71	21.488	1.433
187	32	0.2263	199	0.001041	77	21.490	1.426
188	32	0.2269	199	0.001046	77	21.492	1.423
189	32	0.2089	199	0.001043	60	21.493	1.461
190	32	0.2208	199	0.001049	71	21.494	1.434
191	32	0.2329	199	0.001028	84	21.494	1.382
192	32	0.2266	199	0.001055	77	21.496	1.429
193	32	0.2273	199	0.001061	78	21.497	1.427
194	32	0.2092	199	0.001058	60	21.499	1.464
195	32	0.2210	199	0.001063	72	21.500	1.436
196	32	0.2272	199	0.001070	78	21.502	1.429
197	32	0.2278	199	0.001076	78	21.503	1.427
198	32	0.2333	199	0.001054	85	21.503	1.386
199	32	0.2093	199	0.001073	60	21.505	1.467
200	32	0.2335	199	0.001060	85	21.506	1.389
201	32	0.2211	199	0.001078	72	21.507	1.438
202	32	0.2275	199	0.001085	79	21.508	1.432
203	32	0.2095	199	0.001088	60	21.510	1.470
204	32	0.2213	199	0.001096	72	21.511	1.440
205	32	0.2336	199	0.001075	85	21.512	1.389
206	32	0.2280	199	0.001095	79	21.513	1.433
207	32	0.2287	199	0.001102	79	21.514	1.431
208	32	0.2099	199	0.001103	61	21.515	1.473
209	32	0.2338	199	0.001089	86	21.515	1.391
210	32	0.2002	140	0.000486	45	26.492	1.784