Previsão de Preços de Fechamento do Café Arábica no Mercado Financeiro Utilizando Redes Neurais LSTM Otimizadas por Algoritmos Genéticos Transgênicos

Lynyker Carvalho de Oliveira Magalhães

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2026

Lynyker Carvalho de Oliveira Magalhães

Previsão de Preços de Fechamento do Café Arábica no Mercado Financeiro Utilizando Redes Neurais LSTM Otimizadas por Algoritmos Genéticos Transgênicos

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Prof. Dr. Laurence Rodrigues do Amaral
Coorientador: Prof. Dr. Murillo Guimarães Carneiro

Uberlândia
2026

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

M188 Magalhães, Lynyker Carvalho de Oliveira, 1994-
2026 Previsão de Preços de Fechamento do Café Arábica no Mercado Financeiro Utilizando Redes Neurais LSTM Otimizadas por Algoritmos Genéticos Transgênicos [recurso eletrônico] / Lynyker Carvalho de Oliveira Magalhães. - 2026.

Orientador: Laurence Rodrigues do Amaral.
Coorientador: Murillo Guimarães Carneiro.
Dissertação (Mestrado) - Universidade Federal de Uberlândia, Pós-graduação em Ciência da Computação.
Modo de acesso: Internet.
DOI http://doi.org/10.14393/ufu.di.2026.620
Inclui bibliografia.
Inclui ilustrações.

1. Computação. I. Amaral, Laurence Rodrigues do ,1978-, (Orient.). II. Carneiro, Murillo Guimarães ,1988-, (Coorient.). III. Universidade Federal de Uberlândia. Pós-graduação em Ciência da Computação. IV. Título.

CDU: 681.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:
Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Computação
Av. João Naves de Ávila, 2121, Bloco 1A, Sala 243 - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4470 - www.ppgco.facom.ufu.br - cpgfacom@ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Dissertação, 20/2026, PPGCO				
Data:	30 de Julho de 2026	Hora de início:	09:00	Hora de encerramento:	11:27
Matrícula do Discente:	12322CCP017				
Nome do Discente:	Lynyker Carvalho de Oliveira Magalhães				
Título do Trabalho:	Um Algoritmo Genético Transgênico para Otimização de Hiperparâmetros de Redes Long Short-Term Memory Aplicado à Previsão de Preços de Café Arábica				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Inteligência Artificial				
Projeto de Pesquisa de vinculação:	----------				

Reuniu-se por videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Murillo Guimarães Carneiro - FACOM/UFU (Coorientador), Claudiney Ramos Tinoco - FACOM/UFU, Joslaine Cristina Jeske de Freitas - UFJ e Laurence Rodrigues do Amaral - FACOM/UFU, orientador(a) do(a) candidato(a).

Os examinadores participaram desde as seguintes localidades: Laurence Rodrigues do Amaral - Patos de Minas/MG, Joslaine Cristina Jeske de Freitas - Jataí/GO, Murillo Guimarães Carneiro e Claudiney Ramos Tinoco em Uberlândia/MG. O aluno(a) Lynyker Carvalho de Oliveira Magalhães participou de Ribeirão Preto/SP.

Iniciando os trabalhos o(a) presidente da mesa, Prof. Dr. Laurence Rodrigues do Amaral, apresentou a Comissão Examinadora e o(a) candidato(a), agradeceu a presença do público, e concedeu ao(à) Discente a palavra para a exposição do seu trabalho.

A seguir o senhor(a) presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o(a) candidato(a). Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato(a):

Aprovado

Esta defesa faz parte dos requisitos necessários à obtenção do título de Mestre.

Dedico este trabalho a Deus, que tornou possível a realização deste sonho; ao meu pai, in memoriam, cuja presença permanece viva em meus ensinamentos e lembranças; à minha mãe, pelo incentivo incansável à educação; à minha esposa, pelo amor, compreensão e companheirismo; e às minhas irmãs, pelo apoio constante e por compartilharem comigo a certeza de que o conhecimento transforma vidas.

Agradecimentos

A Deus, por me conceder a oportunidade, a força e a perseverança necessárias para realizar este sonho e estudar uma área pela qual sempre tive grande interesse.

À minha esposa Cleidiane, por todo amor, companheirismo, compreensão e paciência ao longo desta trajetória. Sua presença foi fundamental, especialmente nos momentos em que precisei dedicar muitas horas aos estudos e ao desenvolvimento desta pesquisa. Agradeço por compreender minhas ausências, por me apoiar nas dificuldades e por permanecer ao meu lado durante todo esse processo.

À minha mãe, Maria Valtervam, por sempre incentivar seus filhos a estudarem e por nos ensinar a importância da educação, da dedicação e da busca pelo conhecimento. Seu apoio e seus ensinamentos foram essenciais para minha formação pessoal e acadêmica.

Às minhas irmãs, Kaliny Alice e Kethlyn, pelo incentivo, pelo apoio e pela motivação durante esta caminhada. A trajetória de nossa família, com os três irmãos dedicando-se aos estudos na área das Engenharias, representa a força dos ensinamentos recebidos e da valorização da educação em nossas vidas.

Ao meu pai, Vanderval, que, embora já não esteja fisicamente presente, permanece vivo em minha memória e em meu coração. Agradeço por todo apoio, incentivo e confiança que sempre depositou em mim. Seus ensinamentos e seu exemplo contribuíram profundamente para que eu chegasse até aqui.

Ao meu orientador Dr. Laurence, pela paciência, pela disponibilidade e por todos os ensinamentos compartilhados durante o desenvolvimento deste trabalho. Agradeço pelas orientações, pelas contribuições e pela confiança, fundamentais para meu crescimento acadêmico e para a realização desta pesquisa.

Por fim, agradeço a todas as pessoas que, direta ou indiretamente, contribuíram para esta conquista e estiveram ao meu lado ao longo desta jornada.

"O fracasso é apenas a oportunidade de começar de novo, desta vez de forma mais inteligente."

(Henry Ford)

Resumo

A previsão de séries temporais financeiras constitui um desafio devido à elevada volatilidade, à não linearidade e à complexidade inerentes aos mercados. Nesse contexto, esta dissertação investiga a utilização de redes neurais *Long Short-Term Memory* (LSTM) para a previsão dos preços de fechamento dos contratos futuros de café arábica, tendo como principal objetivo analisar a contribuição da otimização automática de hiperparâmetros por algoritmos evolutivos para a construção de modelos capazes de conciliar qualidade preditiva, desempenho financeiro e custo computacional. Para isso, foram comparadas três abordagens: uma LSTM com hiperparâmetros fixos, uma LSTM otimizada por Algoritmo Genético (LSTM-AG) e uma LSTM otimizada por uma adaptação do Algoritmo Genético Transgênico (LSTM-AGT). A metodologia proposta incorpora um histórico genético probabilístico e quatro módulos transgênicos ao processo evolutivo, visando modificar a dinâmica da busca por configurações promissoras. Os experimentos foram realizados utilizando preços históricos dos contratos futuros de café arábica (KC=F), considerando as granularidades temporais de quatro horas, diária e semanal. A avaliação contemplou métricas estatísticas de previsão, análise da convergência e da diversidade populacional, complexidade arquitetural, custo computacional, eficiência arquitetural e validação financeira por meio de *backtesting*. Os resultados demonstraram que a otimização evolutiva automatizou de forma eficaz a seleção de hiperparâmetros, produzindo arquiteturas competitivas e evidenciando que diferentes frequências temporais demandam configurações distintas. Observou-se ainda que melhorias nas métricas preditivas não implicam necessariamente melhor desempenho financeiro, reforçando a importância de uma avaliação integrada entre precisão estatística, desempenho financeiro, complexidade arquitetural e custo computacional. A adaptação do AGT mostrou-se competitiva na geração de arquiteturas estruturalmente mais compactas, preservando a diversidade populacional e produzindo modelos com elevada eficiência arquitetural. Conclui-se que a metodologia proposta representa uma alternativa promissora para a otimização automática de hiperparâmetros de redes LSTM, contribuindo para o avanço da otimização evolutiva aplicada à Inteligência Artificial e à previsão de séries temporais financeiras.

Palavras-chave: Previsão de Séries Temporais; Redes Neurais LSTM; Algoritmos Evolutivos; Algoritmo Genético Transgênico; Mercado Financeiro..

Abstract

Financial time series forecasting remains a challenging problem due to the high volatility, nonlinearity, and complexity of financial markets. This dissertation investigates the use of *Long Short-Term Memory* (LSTM) neural networks to forecast Arabica coffee futures closing prices, aiming to analyze the contribution of evolutionary hyperparameter optimization to the development of models capable of balancing predictive performance, financial performance, and computational cost. Three approaches were compared: a baseline LSTM with fixed hyperparameters, an LSTM optimized by a Genetic Algorithm (LSTM-GA), and an LSTM optimized by an adapted Transgenetic Genetic Algorithm (LSTM-TGA). The proposed methodology incorporates a probabilistic genetic history and four transgenetic modules into the evolutionary process in order to modify the search dynamics for promising hyperparameter configurations. Experiments were conducted using historical closing prices of Arabica coffee futures contracts (KC=F) considering three temporal resolutions: 4-hour, daily, and weekly. Model evaluation included predictive accuracy metrics, convergence and population diversity analyses, architectural complexity, computational cost, architectural efficiency, and financial validation through backtesting. The results demonstrate that evolutionary optimization effectively automated hyperparameter selection, producing competitive neural architectures and showing that different temporal resolutions require distinct configurations. Furthermore, improvements in predictive performance were not necessarily associated with superior financial performance, highlighting the importance of an integrated evaluation involving predictive accuracy, financial performance, architectural complexity, and computational cost. The adapted TGA proved to be a competitive optimization strategy by generating structurally simpler architectures, preserving population diversity, and producing models with high architectural efficiency. It is concluded that the proposed methodology represents a promising approach for automatic LSTM hyperparameter optimization, contributing to the advancement of evolutionary optimization applied to Artificial Intelligence and financial time series forecasting.

Keywords: Time Series Forecasting; Long Short-Term Memory (LSTM); Evolutionary Algorithms; Transgenetic Genetic Algorithm; Financial Market..

Lista de ilustrações

Figura 1 – O Módulo repetitivo de uma célula *Long Short-Term Memory* (LSTM), composto pelas portas de esquecimento (σ), entrada (σ), modulação de estado (tanh) e saída (σ), responsáveis pelo controle do fluxo de informações, atualização do estado da célula e geração do estado oculto ao longo da sequência temporal, conforme apresentado em (OLAH, 2015). 38

Figura 2 – Exploração conceitual do espaço de busca por diferentes estratégias de otimização de hiperparâmetros. A figura ilustra, de forma esquemática, como métodos tradicionais e evolutivos percorrem o espaço de busca. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 42

Figura 3 – Representação conceitual do processo evolutivo empregado na otimização dos hiperparâmetros das redes neurais LSTM utilizando AGs. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 43

Figura 4 – Representação conceitual da estrutura histórica utilizada pelo operador Transgênico proposto por Amaral e Jr (2014). Para cada gene do cromossomo são armazenados o valor considerado promissor e sua frequência de ocorrência entre os indivíduos associados à melhor aptidão observada. A estrutura também registra o melhor valor da função de aptidão obtido até a geração corrente. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)), com base em Amaral e Jr (2014). 47

Figura 5 – Representação conceitual da operação transgênica proposta por Amaral e Jr (2014). O histórico evolutivo fornece os genes e suas frequências de ocorrência, que são utilizados para selecionar probabilisticamente os valores genéticos a serem transferidos aos indivíduos da população. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)), com base em Amaral e Jr (2014). 49
Figura 6 – Representação conceitual dos módulos do operador Transgênico proposto por Amaral e Jr (2014). Cada módulo atua sobre uma cópia independente da população corrente, diferenciando-se pela quantidade de genes modificados durante a operação transgênica. Após a avaliação das quatro populações resultantes, os melhores indivíduos são selecionados para compor a nova população. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)), com base em Amaral e Jr (2014). 50
Figura 7 – Construção do histórico transgênico a partir dos indivíduos preservados pelo elitismo. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 85
Figura 8 – Fluxo geral do Algoritmo Genético Transgênico implementado nesta dissertação. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 89
Figura 9 – Evolução da função perda da LSTM Clássica durante o treinamento nas granularidades de 4 horas, diária e semanal. 98
Figura 10 – Espaço de busca definido para a otimização dos hiperparâmetros da rede LSTM utilizando Algoritmo Genético. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 102
Figura 11 – Evolução do melhor valor de *fitness* do Algoritmo Genético nas granularidades de 4 horas, diária e semanal. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)), ... 103
Figura 12 – Evolução da diversidade cromossômica e da entropia média dos genes durante a otimização pelo Algoritmo Genético.Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 105
Figura 13 – Evolução do melhor cromossomo identificado em cada geração do Algoritmo Genético para as granularidades de 4 horas, diária e semanal. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 107

Figura 14 – Evolução do melhor valor de *fitness* do Algoritmo Genético Transgênico nas granularidades de 4 horas, diária e semanal.Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 112
Figura 15 – Evolução da diversidade cromossômica e da entropia média dos genes durante a otimização pelo Algoritmo Genético Transgênico. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 113
Figura 16 – Evolução do melhor cromossomo identificado em cada geração do Algoritmo Genético Transgênico para as granularidades de 4 horas, diária e semanal.Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 115
Figura 17 – Ativação dos módulos do operador transgênico ao longo das gerações do processo evolutivo.Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). . 117
Figura 18 – Proporção de genes efetivamente alterados em relação aos genes selecionados pelo operador transgênico ao longo das gerações. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 118
Figura 19 – Evolução da diversidade cromossômica e da entropia média dos genes durante a otimização pelo AGT. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 126
Figura 20 – Evolução do melhor cromossomo identificado em cada geração do AGT para as granularidades de 4 horas, diária e semanal. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 127
Figura 21 – Ativação dos módulos do operador transgênico ao longo das gerações do processo evolutivo. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 128
Figura 22 – Proporção de genes efetivamente alterados em relação aos genes selecionados pelo operador transgênico ao longo das gerações. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 129
Figura 23 – Quantidade de parâmetros treináveis das arquiteturas finais por frequência temporal e modelo. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 143

Figura 24 – Relação tridimensional entre simplicidade arquitetural, desempenho preditivo e desempenho financeiro. O posicionamento dos modelos representa qualitativamente o comportamento predominante observado nos experimentos, enquanto a região central indica o conceito de Equilíbrio Arquitetural. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). . . . 144
Figura 25 – Eficiência arquitetural das arquiteturas finais, expressa pelo capital final obtido por mil parâmetros treináveis. Valores mais elevados indicam maior retorno financeiro relativo à complexidade estrutural da rede neural. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 147
Figura 26 – Comparação do lucro médio por operação obtido pelas arquiteturas LSTM Clássica, LSTM+AG e LSTM+AGT nas frequências de 4 horas, diária e semanal. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)). 149

Lista de tabelas

Tabela 1 – Comparação entre diferentes estratégias de otimização de hiperparâmetros encontradas na literatura e a proposta desta dissertação. 41
Tabela 2 – Configuração dos conjuntos de dados utilizados na etapa de busca de hiperparâmetros. 73
Tabela 3 – Configuração da LSTM clássica utilizada como modelo de referência. . 77
Tabela 4 – Genes utilizados na representação cromossômica do Algoritmo Genético. 79
Tabela 5 – Comparação entre o operador transgênico original e a implementação proposta nesta dissertação. 90
Tabela 6 – Configurações experimentais utilizadas no *backtesting*. 93
Tabela 7 – Hiperparâmetros utilizados na LSTM Clássica. 96
Tabela 8 – Resultados obtidos pela LSTM Clássica nas diferentes granularidades temporais. 97
Tabela 9 – Síntese dos principais resultados obtidos pela LSTM Clássica. 100
Tabela 10 – Configuração utilizada no processo de otimização com o Algoritmo Genético. 101
Tabela 11 – Hiperparâmetros utilizados no treinamento final da LSTM otimizada pelo Algoritmo Genético. 106
Tabela 12 – Resultados obtidos pela LSTM otimizada pelo Algoritmo Genético. . . 108
Tabela 13 – Configuração utilizada no processo de otimização com o Algoritmo Genético Transgênico. 111
Tabela 14 – Hiperparâmetros selecionados pelo AGT para o treinamento final. . . . 114
Tabela 15 – Resultados obtidos pela LSTM otimizada pelo Algoritmo Genético Transgênico. 116
Tabela 16 – Comparação entre LSTM Clássica, AG e AGT. 120
Tabela 17 – Resultados financeiros obtidos no *backtesting*. 122
Tabela 18 – Síntese integrada do desempenho preditivo e financeiro dos modelos. . 133
Tabela 19 – Arquiteturas finais selecionadas pelos algoritmos evolutivos. 134

Tabela 20 – Complexidade das arquiteturas finais em quantidade de parâmetros treináveis. 141
Tabela 21 – Eficiência arquitetural das arquiteturas finais em relação ao desempenho financeiro. 146
Tabela 22 – Indicadores operacionais das estratégias de negociação. 148
Tabela 23 – Síntese dos principais resultados experimentais obtidos pelos modelos avaliados. 151
Tabela 24 – Síntese da verificação das hipóteses da pesquisa. 154
Tabela 25 – Principais contribuições da dissertação. 158

Lista de abreviaturas e siglas

AG	Algoritmo Genético
AGT	Algoritmo Genético Transgênico
CNN	Convolutional Neural Network
KC=F	Ticker do contrato futuro de café arábica no Yahoo Finance
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MLP	Multilayer Perceptron
MSE	Mean Squared Error
PSO	Particle Swarm Optimization
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network

Sumário

1 INTRODUÇÃO 27
1.1 Motivação 28
1.2 Objetivos e Desafios da Pesquisa 29
1.3 Hipótese 31
1.4 Contribuições 32
1.5 Organização da Dissertação 33

2 FUNDAMENTAÇÃO TEÓRICA 35
2.1 Mercado Financeiro 36
2.2 *Long Short-Term Memory* (LSTM) 37
2.3 Otimização de Hiperparâmetros 39
2.4 Algoritmos Genéticos (AGs) 42
2.5 Algoritmo Genético Transgênico 46
2.5.1 Operação Transgênica 48
2.5.2 Módulos M1, M2, M3 e M4 49
2.5.3 Resultados e Limitações do Operador Transgênico 51
2.6 Métricas de Avaliação 52
2.6.1 Mean Absolute Percentage Error (MAPE) 53
2.6.2 Mean Squared Error (MSE) 53
2.6.3 Root Mean Squared Error (RMSE) 54
2.6.4 R-squared (Coeficiente de Determinação) 54
2.7 Dinâmica dos mercados de *commodities* e fundamentos da previsão 55
2.8 Modelos tradicionais e evolução das abordagens preditivas . . . 56
2.9 Redes neurais aplicadas à previsão de séries temporais financeiras 58
2.10 Modelos baseados em LSTM 60
2.11 Previsão de preços de *commodities* agrícolas e café arábica . . 62

2.12 Otimização de hiperparâmetros . . . 65
2.13 Lacunas identificadas na literatura e proposta da dissertação . 68

3 MÉTODO . . . 71
3.1 Delineamento da Pesquisa . . . 72
3.2 Base de Dados e Aquisição . . . 72
3.3 Pré-processamento e Normalização . . . 73
3.4 Construção das Janelas Temporais . . . 74
3.5 Divisão Temporal dos Dados . . . 75
3.6 Arquitetura Geral dos Modelos LSTM . . . 76
3.6.1 LSTM Clássica . . . 77
3.6.2 LSTM Otimizada por AG (LSTM-GA) . . . 78
3.6.3 Representação Cromossômica . . . 78
3.6.4 Inicialização da População . . . 79
3.6.5 Avaliação dos Indivíduos e Função de Aptidão . . . 80
3.6.6 Seleção por Torneio . . . 80
3.6.7 *Crossover* de Um Ponto . . . 81
3.6.8 Mutação . . . 81
3.6.9 Elitismo . . . 81
3.6.10 Formação da Nova População e Critério Evolutivo . . . 82
3.6.11 Construção Dinâmica da Rede LSTM . . . 82
3.6.12 LSTM Otimizada por Algoritmo Genético Transgênico (LSTM-AGT) . 83
3.6.13 Construção do Histórico Transgênico . . . 84
3.6.14 Operador Transgênico Modular . . . 86
3.6.15 Taxas Transgênicas . . . 88
3.6.16 Inserção dos Indivíduos Transgênicos na População . . . 88
3.6.17 Fluxo Geral do Algoritmo Genético Transgênico . . . 89
3.6.18 Adaptações em Relação ao Operador Transgênico Original . . . 90
3.6.19 Treinamento Final da Rede . . . 91
3.6.20 Avaliação de Desempenho . . . 91
3.6.21 Metodologia de *Backtesting* . . . 92

4 EXPERIMENTOS E ANÁLISE DOS RESULTADOS . . . 95
4.1 Resultados da LSTM Clássica . . . 95
4.1.1 Desempenho preditivo da LSTM Clássica . . . 97
4.1.2 Análise da convergência do treinamento . . . 98
4.1.3 Análise do custo computacional . . . 100
4.2 Otimização dos hiperparâmetros utilizando Algoritmo Genético 100
4.2.1 Configuração do processo de otimização . . . 101
4.2.2 Espaço de busca dos hiperparâmetros . . . 101

4.2.3 Evolução do *fitness* ao longo das gerações . . . 103
4.2.4 Hiperparâmetros selecionados pelo Algoritmo Genético . . . 106
4.2.5 Desempenho preditivo da LSTM otimizada pelo Algoritmo Genético . . 108
4.2.6 Análise do custo computacional . . . 108
4.2.7 Síntese dos resultados obtidos pelo Algoritmo Genético . . . 109
4.3 Otimização dos hiperparâmetros utilizando Algoritmo Genético Transgênico . . . 110
4.3.1 Configuração do processo de otimização . . . 110
4.3.2 Evolução do *fitness* ao longo das gerações . . . 111
4.3.3 Hiperparâmetros selecionados pelo Algoritmo Genético Transgênico . . . 114
4.3.4 Desempenho preditivo da LSTM otimizada pelo Algoritmo Genético Transgênico . . . 115
4.3.5 Análise da dinâmica do operador transgênico . . . 117
4.3.6 Análise do custo computacional . . . 118
4.3.7 Síntese dos resultados obtidos pelo Algoritmo Genético Transgênico . . 119
4.4 Análise comparativa entre as abordagens . . . 120
4.4.1 Comparação do custo computacional . . . 121
4.4.2 Síntese comparativa das abordagens . . . 121
4.5 Avaliação financeira por meio de *backtesting* . . . 122
4.5.1 Resultados do *backtesting* na granularidade de 4 horas . . . 122
4.5.2 Resultados do *backtesting* na granularidade diária . . . 123
4.5.3 Resultados do *backtesting* na granularidade semanal . . . 124
4.5.4 Relação inicial entre desempenho preditivo e desempenho financeiro . . 124
4.6 Análise da dinâmica do operador transgênico . . . 125
4.6.1 Diversidade populacional e evolução dos cromossomos . . . 125
4.6.2 Convergência da população durante a otimização . . . 130
4.6.3 Impacto do operador transgênico sobre a convergência da busca . . . 131
4.7 Integração entre desempenho preditivo, desempenho financeiro e complexidade das arquiteturas . . . 132
4.7.1 Panorama geral dos resultados experimentais . . . 133
4.7.2 Arquiteturas finais selecionadas pelos algoritmos evolutivos . . . 134
4.7.3 Relação entre desempenho preditivo e desempenho financeiro . . . 135
4.7.4 Influência da otimização evolutiva na complexidade das arquiteturas . . 137
4.7.5 Análise quantitativa da complexidade das arquiteturas . . . 139
4.7.6 Relação entre complexidade arquitetural, desempenho preditivo e desempenho financeiro . . . 143
4.7.7 Eficiência arquitetural . . . 145
4.7.8 Análise das operações de negociação . . . 148
4.7.9 Síntese dos resultados experimentais . . . 150

4.7.10 Verificação das hipóteses de pesquisa 151

5 CONCLUSÃO 155
5.1 Considerações finais 155
5.2 Principais contribuições da pesquisa 157
5.3 Limitações da Pesquisa 157
5.4 Trabalhos Futuros 159

REFERÊNCIAS 161

APÊNDICES 167

APÊNDICE A – LSTM CLÁSSICA 169

APÊNDICE B – LSTM AG 177

APÊNDICE C – LSTM AGT 191

APÊNDICE D – TREINAMENTO FINAL 209

APÊNDICE E – BACKTESTING 217

Capítulo 1

Introdução

O mercado financeiro constitui um ambiente dinâmico e complexo, no qual a previsão de preços de ativos pode contribuir para processos de tomada de decisão, gestão de riscos e planejamento de estratégias de negociação. Com o crescimento da disponibilidade de dados históricos e o avanço das técnicas de aprendizado de máquina, modelos computacionais têm sido amplamente investigados para identificar padrões não lineares e dependências temporais presentes em séries financeiras (PAIVA, 2014).

Entre as técnicas de aprendizado profundo aplicadas a esse contexto, destacam-se as redes neurais recorrentes (*Recurrent Neural Networks* – RNNs), especialmente as redes *Long Short-Term Memory* (LSTM). Essas redes foram desenvolvidas para lidar com sequências de dados e dependências temporais de longo prazo, apresentando características adequadas à modelagem de séries financeiras (NELSON, 2017; MORAIS, 2023).

Apesar de sua capacidade preditiva, o desempenho das redes LSTM depende diretamente da definição de sua arquitetura e de seus hiperparâmetros de treinamento. O número de camadas, a quantidade de neurônios, a taxa de aprendizado, o tamanho do lote e o número de épocas influenciam o processo de convergência, a capacidade de generalização e o custo computacional do modelo. A seleção manual dessas configurações ou a utilização de procedimentos exaustivos torna-se dispendiosa quando o espaço de busca apresenta grande quantidade de combinações possíveis.

Nesse contexto, os Algoritmos Genéticos (AGs) constituem uma alternativa para automatizar a busca por configurações adequadas. Esses algoritmos exploram o espaço de soluções por meio de mecanismos inspirados na evolução natural, incluindo seleção, cruzamento, mutação e elitismo. Por não dependerem de informações de gradiente e permitirem a manipulação simultânea de variáveis discretas e contínuas, os AGs são adequados à otimização de hiperparâmetros de redes neurais (KIM; AHN, 2012).

Entretanto, AGs convencionais podem apresentar perda progressiva de diversidade populacional e convergência prematura. Como alternativa, este trabalho investiga a adaptação do AGT proposto por Amaral e Jr (2014) ao problema de otimização de hiperparâmetros de redes LSTM. A abordagem desenvolvida utiliza um histórico genético proba-

bilístico e quatro módulos transgênicos, denominados M1, M2, M3 e M4, que modificam diferentes quantidades de genes dos cromossomos candidatos.

No método proposto, cada cromossomo representa uma configuração completa da rede LSTM, formada por cinco genes: número de camadas LSTM, número de neurônios, taxa de aprendizado, tamanho do lote e número de épocas. O operador transgênico utiliza informações acumuladas durante a evolução para introduzir valores genéticos na população, com o objetivo de modificar a dinâmica da busca e ampliar a exploração de configurações alternativas.

A avaliação experimental compara três abordagens: uma LSTM clássica com hiperparâmetros fixos, uma LSTM otimizada por Algoritmo Genético convencional (LSTM-AG) e uma LSTM otimizada pelo Algoritmo Genético Transgênico proposto (LSTM-AGT). Os experimentos utilizam preços históricos de fechamento dos contratos futuros de café arábica, identificados pelo código KC=F, em três resoluções temporais: quatro horas, diária e semanal.

As abordagens são analisadas sob três perspectivas complementares. A primeira considera a qualidade preditiva, por meio das métricas MSE, RMSE, MAPE e coeficiente de determinação (R^2). A segunda examina a dinâmica da busca evolutiva, incluindo convergência, diversidade populacional, evolução dos genes e atuação dos módulos transgênicos. A terceira avalia a aplicabilidade financeira dos modelos por meio de *backtesting* realizado com dados não utilizados durante a busca de hiperparâmetros e o treinamento final.

O café arábica foi escolhido como estudo de caso devido à sua relevância econômica e à elevada variabilidade de seus preços, que são influenciados por fatores climáticos, produtivos, cambiais e comerciais (RODRIGUES; REIS; TAVARES, 2014). Essas características tornam a série adequada para investigar métodos computacionais capazes de modelar relações temporais complexas e apoiar a análise de estratégias financeiras.

Assim, a principal contribuição deste trabalho não se restringe à previsão dos preços do café arábica. A pesquisa também investiga como um mecanismo evolutivo baseado em histórico probabilístico e operadores transgênicos modifica a busca por hiperparâmetros de redes LSTM, relacionando a qualidade das soluções encontradas à precisão estatística, à complexidade arquitetural, ao custo computacional e ao desempenho financeiro.

1.1 Motivação

A previsão de preços no mercado financeiro é um desafio contínuo devido à sua natureza volátil e influenciada por múltiplos fatores. No caso específico do café arábica, um dos produtos agrícolas mais comercializados globalmente, a precisão na previsão de preços de fechamento é crucial para produtores, investidores e *traders* (CARRASCO-GUTIERREZ; ALMEIDA, 2013). O café arábica, cultivado em diversas regiões do Brasil, é altamente sensível a variáveis climáticas, políticas comerciais e flutuações na oferta e demanda.

Dessa forma, previsões mais precisas podem auxiliar na tomada de decisões estratégicas, contribuindo para a gestão de riscos e para a definição de estratégias de negociação.

Nesse contexto, as Redes Neurais do tipo *Long Short-Term Memory* (LSTM) destacam-se por sua capacidade de modelar dependências temporais de longo prazo, sendo amplamente empregadas em problemas de previsão de séries temporais financeiras (LEANDRO, 2021; MORAIS, 2023). Entretanto, o desempenho dessas redes depende diretamente da definição de seus hiperparâmetros, como número de camadas, quantidade de neurônios, taxa de aprendizado, tamanho do lote e número de épocas de treinamento.

A otimização desses hiperparâmetros representa um problema complexo, pois diferentes combinações podem produzir modelos com capacidades preditivas e custos computacionais distintos. Nesse cenário, Algoritmos Genéticos têm sido utilizados como mecanismos de busca automática de configurações mais adequadas, reduzindo a dependência de procedimentos empíricos de ajuste manual (KIM; AHN, 2012; MENDES, 2021). Entretanto, limitações como convergência prematura e perda de diversidade populacional motivam a investigação de estratégias evolutivas alternativas, como o operador transgênico proposto por (AMARAL; JR, 2014).

Esta dissertação é motivada pela necessidade de investigar estratégias de otimização automática de hiperparâmetros para redes LSTM aplicadas à previsão de séries temporais financeiras. Para isso, são comparadas três abordagens: uma LSTM com hiperparâmetros fixos, uma LSTM otimizada por Algoritmo Genético convencional e uma LSTM otimizada por uma adaptação do Algoritmo Genético Transgênico proposta neste trabalho.

Além da avaliação da precisão preditiva, a pesquisa busca analisar como diferentes estratégias evolutivas influenciam o processo de busca por hiperparâmetros, considerando aspectos relacionados ao desempenho estatístico, à complexidade das arquiteturas encontradas, ao custo computacional e aos resultados obtidos em simulações de *backtesting*. O mercado futuro de café arábica é utilizado como estudo de caso para validar experimentalmente o método proposto.

1.2 Objetivos e Desafios da Pesquisa

Esta dissertação investiga a utilização de redes neurais do tipo *Long Short-Term Memory* (LSTM) para a previsão dos preços de fechamento do café arábica no mercado financeiro, comparando três estratégias distintas: uma LSTM com hiperparâmetros fixos, uma LSTM otimizada por Algoritmo Genético convencional (LSTM-AG) e uma LSTM otimizada por uma adaptação do Algoritmo Genético Transgênico (LSTM-AGT).

A principal contribuição desta pesquisa consiste na adaptação do operador transgênico proposto por Amaral e Jr (2014) ao problema de otimização de hiperparâmetros de redes LSTM. Para isso, foi desenvolvido um mecanismo baseado em histórico genético probabilístico e em quatro módulos transgênicos (M1–M4), responsáveis por introduzir

diferentes quantidades de genes em indivíduos selecionados durante o processo evolutivo.

Além da avaliação da qualidade preditiva dos modelos, o método proposto investiga o comportamento da estratégia evolutiva por meio da análise da convergência da busca, das arquiteturas encontradas, do custo computacional e do desempenho financeiro obtido por procedimentos sistemáticos de *backtesting*. Dessa forma, o mercado futuro de café arábica é utilizado como estudo de caso para validar experimentalmente o método desenvolvido, buscando compreender o comportamento da estratégia evolutiva proposta e sua contribuição para a otimização de hiperparâmetros de redes neurais LSTM.

Ao investigar a adaptação do operador transgênico ao problema de otimização de hiperparâmetros de redes LSTM, esta pesquisa busca contribuir para a área de Inteligência Artificial ao analisar não apenas a qualidade preditiva dos modelos gerados, mas também o comportamento da estratégia evolutiva empregada durante a busca por soluções. Assim, o mercado futuro de café arábica constitui o domínio de aplicação utilizado para validar experimentalmente o método proposto.

Objetivos específicos:

- Implementar e comparar três estratégias de modelagem para previsão de séries temporais financeiras: uma LSTM com hiperparâmetros fixos, uma LSTM otimizada por Algoritmo Genético convencional (LSTM-AG) e uma LSTM otimizada por Algoritmo Genético Transgênico adaptado (LSTM-AGT);
- Investigar a adaptação do operador transgênico proposto por Amaral e Jr (2014) ao processo de otimização de hiperparâmetros de redes LSTM, analisando sua influência sobre a dinâmica da busca evolutiva, as arquiteturas encontradas e a qualidade das soluções obtidas;
- Realizar *backtesting* dos modelos propostos em diferentes períodos históricos, simulando estratégias baseadas nas previsões geradas, a fim de avaliar o desempenho, a robustez e a estabilidade de cada abordagem em condições reais de mercado, considerando indicadores de rentabilidade, risco e robustez;
- Testar e validar os modelos em resoluções temporais de quatro horas, diária e semanal, analisando sua capacidade de generalização, consistência preditiva e aplicabilidade prática no contexto do mercado financeiro.
- Analisar o comportamento da busca evolutiva durante o processo de otimização, considerando a evolução das arquiteturas encontradas, a dinâmica dos operadores genéticos, a convergência da população e o custo computacional das estratégias avaliadas.

Desafios da pesquisa:

- Manuseio de dados voláteis e não lineares: Prever preços em mercados financeiros para *commodities* como o café arábica, devido à alta volatilidade e à natureza não linear dos dados;
- Ajuste fino dos hiperparâmetros das redes neurais LSTM: Identificar e ajustar os parâmetros ideais para a rede artificial LSTM, como número de camadas, quantidade de neurônios, taxa de aprendizado, tamanho do lote e número de épocas, ajustes esses que demandam tempo e são tarefas complexas;
- Explorar eficientemente um espaço de busca composto por diferentes combinações de hiperparâmetros, equilibrando qualidade das soluções encontradas e custo computacional da otimização.
- Avaliar como a adaptação do operador transgênico influencia a dinâmica da busca evolutiva durante a otimização de hiperparâmetros, analisando seu comportamento em conjunto com os operadores tradicionais do Algoritmo Genético.
- Relacionar os ganhos obtidos na otimização dos hiperparâmetros com a qualidade preditiva, o desempenho financeiro e o custo computacional, investigando os compromissos existentes entre essas diferentes métricas de avaliação.

1.3 Hipótese

Diante da complexidade inerente às séries temporais financeiras, caracterizadas por não estacionariedade, elevada volatilidade e relações não lineares, este trabalho parte do pressuposto de que técnicas de aprendizado profundo aliadas a métodos de otimização evolutiva podem proporcionar avanços significativos na qualidade das previsões. Em particular, investiga-se a adaptação do operador transgênico proposto por Amaral e Jr (2014) ao processo de otimização de hiperparâmetros de redes LSTM aplicadas à previsão dos preços de fechamento do café arábica no mercado financeiro.

Nesse contexto, formula-se a hipótese central e as hipóteses específicas a seguir, as quais orientam o desenvolvimento metodológico, a análise dos resultados e a validação empírica do estudo.

A hipótese central desta pesquisa é que a adaptação do operador transgênico ao processo de otimização de hiperparâmetros de redes LSTM constitui uma estratégia competitiva para a construção de modelos de previsão de séries temporais financeiras, produzindo arquiteturas capazes de conciliar qualidade preditiva, desempenho financeiro e custo computacional. Essa hipótese geral é desdobrada nas seguintes hipóteses específicas:

1- A adaptação do operador transgênico ao processo de otimização de hiperparâmetros pode produzir configurações de redes LSTM competitivas para a previsão dos preços

de fechamento do café arábica quando comparadas às obtidas por um Algoritmo Genético convencional.

2- A utilização de Algoritmos Genéticos para a otimização dos hiperparâmetros de redes LSTM pode proporcionar desempenho preditivo superior ao da LSTM clássica, quando avaliado por métricas estatísticas como MSE, RMSE, MAPE e coeficiente de determinação (R^2).

3- O impacto da otimização evolutiva pode ser mais significativo em séries temporais de maior frequência, como a de quatro horas, devido à maior complexidade temporal e variabilidade dos dados.

4- Modelos LSTM otimizados por Algoritmos Genéticos poderão alcançar desempenho preditivo comparável ou superior ao da LSTM clássica utilizando arquiteturas de menor complexidade, caracterizadas por um número reduzido de camadas e neurônios.

5- Melhorias no desempenho estatístico das previsões geradas por modelos LSTM otimizados podem refletir em melhor desempenho financeiro quando essas previsões são utilizadas em estratégias de negociação avaliadas por meio de *backtesting*.

1.4 Contribuições

Esta dissertação apresenta as seguintes contribuições científicas:

- Adaptação do operador transgênico proposto por Amaral e Jr (2014) ao problema de otimização de hiperparâmetros de redes neurais LSTM aplicadas à previsão de séries temporais financeiras.
- Desenvolvimento de uma estratégia de otimização baseada em histórico genético probabilístico e quatro módulos transgênicos (M1–M4), incorporados ao processo evolutivo para exploração do espaço de hiperparâmetros das redes LSTM.
- Proposição de uma metodologia experimental que compara três estratégias distintas (LSTM, LSTM-AG e LSTM-AGT), considerando simultaneamente qualidade preditiva, arquiteturas encontradas, custo computacional e desempenho financeiro por meio de *backtesting*.
- Analisar aspectos como convergência, diversidade populacional, evolução das arquiteturas e custo computacional da busca evolutiva durante a otimização dos hiperparâmetros.
- Permitir analisar a robustez da estratégia proposta em diferentes escalas temporais utilizando contratos futuros de café arábica em três resoluções temporais (4 horas, diária e semanal).

– Disponibilização de uma estrutura experimental capaz de ser adaptada para outros ativos financeiros e outros problemas de otimização de hiperparâmetros baseados em redes neurais recorrentes.

1.5 Organização da Dissertação

Esta dissertação está estruturada em cinco capítulos, além dos elementos pré-textuais e pós-textuais.

O Capítulo 2 apresenta a fundamentação teórica, abordando os principais conceitos relacionados à previsão de séries temporais financeiras, às redes neurais recorrentes, com ênfase nas arquiteturas *Long Short-Term Memory* (LSTM), aos algoritmos evolutivos para otimização de hiperparâmetros, ao Algoritmo Genético Transgênico e aos trabalhos correlatos encontrados na literatura.

O Capítulo 3 descreve detalhadamente o método empregado na pesquisa, incluindo a caracterização da base de dados, o processo de preparação dos dados, a arquitetura dos modelos desenvolvidos, a adaptação proposta para o Algoritmo Genético Transgênico, a configuração dos experimentos, os parâmetros evolutivos, as métricas de avaliação preditiva e financeira, bem como os procedimentos utilizados para validação dos resultados.

O Capítulo 4 apresenta e discute os resultados obtidos. São analisados o comportamento evolutivo dos algoritmos, o desempenho preditivo dos modelos, a validação financeira por meio de *backtesting*, a complexidade arquitetural, o custo computacional, a eficiência arquitetural e a relação entre essas diferentes dimensões de avaliação, permitindo uma análise integrada das arquiteturas desenvolvidas.

Por fim, o Capítulo 5 apresenta as conclusões da pesquisa, destacando as principais contribuições alcançadas, as limitações identificadas durante o desenvolvimento do trabalho e as perspectivas para pesquisas futuras relacionadas à otimização evolutiva de redes neurais aplicadas à previsão de séries temporais financeiras.

Capítulo 2

Fundamentação Teórica

Neste capítulo são apresentados os fundamentos teóricos e os principais trabalhos relacionados que sustentam a proposta desta dissertação. Inicialmente, são discutidos os conceitos associados ao mercado financeiro e à previsão de séries temporais de commodities, contextualizando o problema investigado. Em seguida, são apresentados os fundamentos das redes neurais LSTM, da otimização de hiperparâmetros e dos algoritmos evolutivos, com ênfase nos Algoritmos Genéticos e no operador transgênico proposto por Amaral e Jr (2014). Por fim, são discutidos os principais trabalhos relacionados e as lacunas identificadas na literatura que motivam a proposta metodológica desta pesquisa. Inicialmente, a Seção 2.1 apresenta uma contextualização do mercado financeiro, destacando a dinâmica de negociação de commodities, os fatores que influenciam a formação de preços e os desafios associados à previsão de preços do café arábica no mercado financeiro. Em seguida, a Seção 2.2 aborda os conceitos fundamentais das redes neurais do tipo *Long Short-Term Memory* (LSTM), descrevendo sua arquitetura, mecanismos internos de memória e aplicações em problemas de previsão de séries temporais. Na sequência, é discutido o problema da otimização de hiperparâmetros em modelos de aprendizado profundo, destacando sua importância para o desempenho de redes neurais. Posteriormente, a Seção 2.4 apresenta os fundamentos dos Algoritmos Genéticos como estratégia de otimização.

Na sequência, a Seção 2.6 apresenta as métricas de avaliação utilizadas nesta pesquisa, contemplando medidas estatísticas amplamente empregadas na literatura de previsão de séries temporais financeiras, como MAPE, MSE, RMSE e o coeficiente de determinação (R^2). Essas métricas são fundamentais para quantificar o desempenho preditivo dos modelos desenvolvidos e permitir comparações consistentes entre as diferentes abordagens analisadas. Além de serem utilizadas na avaliação estatística dos modelos, parte dessas métricas também desempenha papel fundamental no processo de otimização, sendo o erro quadrático médio (MSE) empregado como função de aptidão durante a busca evolutiva.

A Seção 2.7 discute a dinâmica dos mercados de commodities e os fundamentos relacionados à previsão de preços, enfatizando fatores econômicos, financeiros e estruturais que

influenciam o comportamento dessas séries temporais. Em seguida, a Seção 2.8 apresenta a evolução das abordagens preditivas, contemplando desde modelos estatísticos tradicionais até métodos híbridos e arquiteturas modernas baseadas em aprendizado profundo. Já a Seção 2.9 reúne estudos relacionados à aplicação de redes neurais artificiais no mercado financeiro sem o uso de algoritmos genéticos, destacando diferentes técnicas de Inteligência Artificial utilizadas na modelagem de séries temporais financeiras.

Posteriormente, a Seção 2.10 aprofunda a discussão sobre modelos baseados em LSTM aplicados à previsão de séries temporais, abordando diferentes arquiteturas, estratégias de decomposição de sinais, utilização de variáveis exógenas e modelos híbridos. Na sequência, a Seção 2.11 apresenta trabalhos específicos voltados à previsão de *commodities* agrícolas e preços do café arábica, discutindo abordagens univariadas, multivariadas, híbridas e estratégias de previsão *one-step ahead* e *multi-step ahead*. Já a Seção 2.12 reúne estudos relacionados à otimização de hiperparâmetros utilizando algoritmos evolutivos, incluindo Algoritmos Genéticos, *Particle Swarm Optimization* (PSO), *Variational Mode Decomposition* (VMD) e diferentes técnicas híbridas aplicadas à otimização de redes neurais recorrentes.

Por fim, a Seção 2.13 apresenta as principais lacunas identificadas na literatura, destacando limitações metodológicas observadas nos trabalhos correlatos e contextualizando a proposta desta dissertação, baseada na utilização de redes neurais LSTM otimizadas por algoritmos genéticos transgênicos aplicados à previsão de preços de fechamento do café arábica no mercado financeiro, bem como à avaliação de desempenho preditivo e financeiro por meio de estratégias de *backtesting*.

2.1 Mercado Financeiro

O mercado financeiro corresponde ao ambiente em que ocorrem negociações de ativos financeiros, valores mobiliários, câmbio e *commodities*, desempenhando papel fundamental na alocação de recursos e no desenvolvimento econômico. Além de possibilitar investimentos e captação de recursos, esse mercado reflete continuamente as expectativas dos agentes econômicos em relação às condições macroeconômicas, sendo frequentemente considerado um indicador da atividade econômica de um país (NELSON, 2017).

Entre os ativos negociados nesse mercado destacam-se as *commodities*, produtos padronizados cujos preços são influenciados por fatores econômicos, climáticos e geopolíticos. O café arábica constitui uma das principais *commodities* agrícolas comercializadas internacionalmente, apresentando elevada variabilidade de preços decorrente de fatores como condições climáticas, oferta e demanda global e políticas econômicas. Essa combinação de fatores torna a previsão de seus preços um problema complexo, motivando o desenvolvimento de modelos computacionais capazes de capturar padrões presentes nas séries

temporais financeiras (MATSUMOTO, 2019).

Diante desse contexto, modelos capazes de aprender padrões temporais complexos tornam-se alternativas promissoras para a previsão de preços em mercados financeiros. Entre essas abordagens destacam-se as redes neurais recorrentes do tipo *Long Short-Term Memory* (LSTM), discutidas na próxima seção.

2.2 *Long Short-Term Memory* (LSTM)

As redes neurais LSTM são um tipo avançado de redes RNNs (*Recurrent Neural Networks*) projetadas para modelar sequências de dados temporais ou de séries temporais, superando limitações das RNNs tradicionais, como o problema de curto alcance na memória. Introduzidas por Hochreiter e Schmidhuber (1997), as LSTM se destacam por sua habilidade de lembrar informações por longos períodos, tornando-se ferramentas eficazes para uma ampla gama de aplicações, incluindo previsão de preços no mercado financeiro, processamento de linguagem natural, e reconhecimento de fala (KARPATHY, 2015). Sua arquitetura inclui células de memória que permitem armazenar, esquecer ou atualizar informações de maneira controlada, o que as torna altamente eficazes na captura de padrões complexos em dados sequenciais. A Figura 1 apresenta a estrutura do módulo repetitivo de uma rede *Long Short-Term Memory* (LSTM) ao longo do tempo, conforme proposto por Olah (2015). Diferentemente das redes neurais recorrentes tradicionais, nas quais o módulo recorrente é composto por uma única camada, a LSTM utiliza um conjunto de quatro camadas interativas que regulam de forma explícita o fluxo de informação temporal. Essa arquitetura permite que a rede preserve dependências de longo prazo, ao mesmo tempo em que descarta informações irrelevantes, tornando-a particularmente adequada para a modelagem de séries temporais complexas.

Na Figura 1, cada bloco identificado pela letra A representa uma célula LSTM em um determinado instante de tempo. Esses blocos são repetidos ao longo da sequência temporal, compartilhando os mesmos pesos. As entradas x_{t-1}, x_t e x_{t+1} representam os vetores de entrada da série temporal, enquanto h_{t-1}, h_t e h_{t+1} correspondem aos estados ocultos propagados entre as células, responsáveis por transportar informação ao longo do tempo.

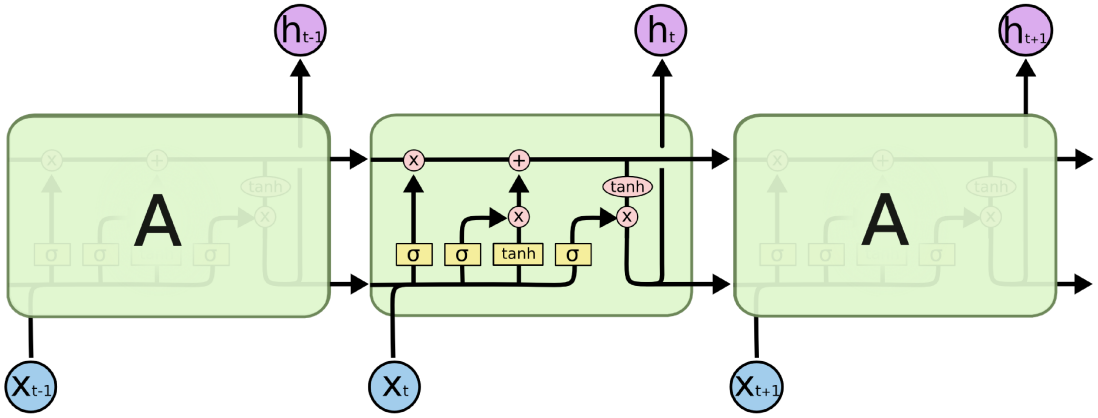

Figura 1 – O Módulo repetitivo de uma célula *Long Short-Term Memory* (LSTM), composto pelas portas de esquecimento (σ), entrada (σ), modulação de estado (tanh) e saída (σ), responsáveis pelo controle do fluxo de informações, atualização do estado da célula e geração do estado oculto ao longo da sequência temporal, conforme apresentado em (OLAH, 2015).

Os principais componentes da arquitetura LSTM apresentados na Figura 1 são descritos a seguir.

- **Os blocos verdes** identificados pela letra A representam as células LSTM individuais. Cada célula contém internamente os mecanismos responsáveis pela atualização da memória e pela geração da saída no instante de tempo correspondente. A repetição desses blocos ao longo da sequência temporal caracteriza a natureza recorrente da arquitetura(OLAH, 2015).

- **Os vetores** x_{t-1}, x_t e x_{t+1}, representados na parte inferior da figura 1, correspondem às observações da série temporal em cada instante de tempo. Esses vetores alimentam diretamente a célula LSTM, juntamente com o estado oculto anterior, influenciando o processo de atualização da memória interna(OLAH, 2015).

- **Os vetores** h_{t-1}, h_t e h_{t+1}, indicados na parte superior da figura 1, representam o estado oculto da LSTM. Esse estado corresponde à saída da célula em cada instante de tempo e é propagado tanto para a próxima célula da sequência quanto para camadas subsequentes da rede, como camadas densas de previsão(OLAH, 2015).

- **A linha horizontal** que atravessa toda a célula representa o estado da célula (C_t), frequentemente interpretado como a memória de longo prazo da LSTM. Essa estrutura permite que informações relevantes sejam transportadas ao longo de muitos passos temporais com mínimas modificações, facilitando a propagação de gradientes durante o treinamento.(OLAH, 2015).

- **Os principais operadores internos da célula** são responsáveis pelo processamento das informações ao longo da sequência temporal. As operações de multipli-

cação elemento a elemento ("×") atuam como mecanismos de filtragem controlados pelas portas da LSTM, enquanto a operação de soma ("+") combina o conteúdo preservado da memória anterior com as novas informações geradas no instante atual. As funções de ativação sigmoide (σ) implementam as portas de esquecimento, entrada e saída, produzindo coeficientes no intervalo $[0, 1]$ que regulam o fluxo de informações. Já a função tanh gera o vetor candidato de memória, limitando seus valores ao intervalo $[-1, 1]$ e contribuindo para a estabilidade do treinamento (OLAH, 2015).

O funcionamento de uma célula LSTM pode ser compreendido em quatro etapas principais. Inicialmente, a porta de esquecimento (*forget gate*) determina quais informações do estado da célula anterior (C_{t-1}) serão preservadas ou descartadas. Em seguida, a porta de entrada (*input gate*) define quais novas informações, produzidas pela camada de modulação (tanh), serão incorporadas ao estado da célula.

Na terceira etapa, o estado da célula é atualizado por meio da combinação entre as informações preservadas e o novo conteúdo gerado. Por fim, a porta de saída (*output gate*) determina quais informações do estado atualizado serão disponibilizadas como estado oculto (h_t), que será propagado para o próximo instante da sequência temporal e utilizado pela rede para produzir suas previsões.

Essas características tornam as redes LSTM particularmente adequadas para problemas de previsão de séries temporais financeiras, nos quais relações temporais complexas, padrões não lineares e dependências de longo prazo são frequentemente observados. Entretanto, o desempenho dessas redes está diretamente relacionado à escolha adequada de seus hiperparâmetros, como número de camadas, quantidade de neurônios, taxa de aprendizado, tamanho do lote e número de épocas de treinamento. Dessa forma, diferentes estratégias de otimização têm sido propostas na literatura com o objetivo de identificar configurações capazes de maximizar o desempenho preditivo das redes LSTM. Esses métodos são apresentados nas próximas seções.

2.3 Otimização de Hiperparâmetros

O desempenho de modelos de aprendizado profundo depende não apenas da arquitetura da rede neural, mas também da adequada configuração de seus hiperparâmetros. Diferentemente dos parâmetros do modelo, que são ajustados automaticamente durante o treinamento por algoritmos de otimização, os hiperparâmetros são definidos previamente e exercem influência direta sobre a capacidade de aprendizado, a generalização do modelo e o custo computacional envolvido no treinamento (KIM; AHN, 2012).

Nas redes neurais LSTM, hiperparâmetros como o número de camadas recorrentes, a quantidade de neurônios por camada, a taxa de aprendizado (*learning rate*), o tamanho

do lote (*batch*) e o número de épocas de treinamento influenciam significativamente o desempenho preditivo da rede. Trabalhos como os de Dantas (2017), Chung e Shin (2018) e Huang et al. (2021) demonstram que diferentes configurações desses hiperparâmetros podem produzir resultados bastante distintos para um mesmo problema de previsão.

Tradicionalmente, a escolha desses hiperparâmetros é realizada por tentativa e erro ou baseada na experiência do pesquisador. Entretanto, à medida que aumenta o número de hiperparâmetros e suas possíveis combinações, o espaço de busca cresce rapidamente, tornando inviável uma exploração manual ou exaustiva das configurações possíveis (DANTAS, 2017).

Com o avanço do aprendizado profundo, diferentes estratégias automáticas passaram a ser utilizadas para apoiar esse processo de otimização. Entre elas destacam-se métodos baseados em otimização Bayesiana, *Particle Swarm Optimization* (PSO), *Whale Optimization Algorithm* (WOA), *Variational Mode Decomposition* (VMD) e Algoritmos Genéticos, os quais procuram explorar o espaço de busca de forma mais eficiente do que abordagens puramente empíricas (CHENG et al., 2019; SHAO et al., 2019; LAHMIRI, 2026).

A literatura apresenta diferentes estratégias para a otimização automática de hiperparâmetros, cada uma com características próprias quanto à exploração do espaço de busca, custo computacional e capacidade de adaptação a diferentes problemas. A Tabela 1 apresenta uma comparação sintética entre os principais métodos encontrados na literatura revisada (CHENG et al., 2019; SHAO et al., 2019; LAHMIRI, 2026; CHUNG; SHIN, 2018; HUANG et al., 2021).

Tabela 1 – Comparação entre diferentes estratégias de otimização de hiperparâmetros encontradas na literatura e a proposta desta dissertação.

Método	**Principais características**	**Limitações**
Grid Search	Busca exaustiva das combinações previamente definidas.	Elevado custo computacional quando o espaço de busca é grande.
Random Search	Seleciona configurações de forma aleatória, reduzindo o custo em relação ao Grid Search.	Pode não explorar regiões promissoras do espaço de busca.
Otimização Bayesiana	Modela probabilisticamente a função objetivo para direcionar a busca.	Maior complexidade de implementação e custo de atualização do modelo probabilístico.
PSO / WOA	Métodos bio-inspirados baseados em inteligência coletiva para exploração contínua do espaço de busca.	Podem apresentar convergência prematura dependendo da configuração adotada.
Algoritmos Genéticos	Exploram simultaneamente diferentes regiões do espaço de busca por meio de operadores evolutivos como seleção, cruzamento e mutação.	Demandam maior custo computacional devido às sucessivas avaliações da função de aptidão.
Algoritmo Genético Transgênico (AGT)	Incorpora operadores transgênicos, histórico evolutivo e módulos especializados para intensificar a exploração de regiões promissoras do espaço de busca, buscando maior eficiência na otimização dos hiperparâmetros da LSTM.	Maior complexidade de implementação e necessidade de definição dos mecanismos de ativação dos operadores transgênicos.

Observa-se que não existe um método universalmente superior para todos os problemas de otimização de hiperparâmetros. A escolha da estratégia depende das características do espaço de busca, da função objetivo e do custo computacional disponível. Entre essas abordagens, os AGs destacam-se pela capacidade de explorar espaços de busca extensos e altamente não lineares, sem exigir hipóteses sobre a forma da função objetivo, característica particularmente interessante na otimização de arquiteturas de redes neurais profundas (GOLDBERG, 1989; EIBEN; SMITH, 2003).

Dentre essas estratégias, os AGs têm recebido especial atenção devido à sua capacidade de explorar espaços de busca extensos, descontínuos e altamente não lineares, características frequentemente presentes na otimização de redes neurais profundas. Diversos estudos recentes empregaram AGs para otimizar hiperparâmetros de redes LSTM aplicadas à previsão de séries temporais financeiras, obtendo melhorias significativas na qualidade das previsões (CHUNG; SHIN, 2018; HUANG et al., 2021; SHA, 2024; WU et al., 2025; CHOUDHARY et al., 2025).

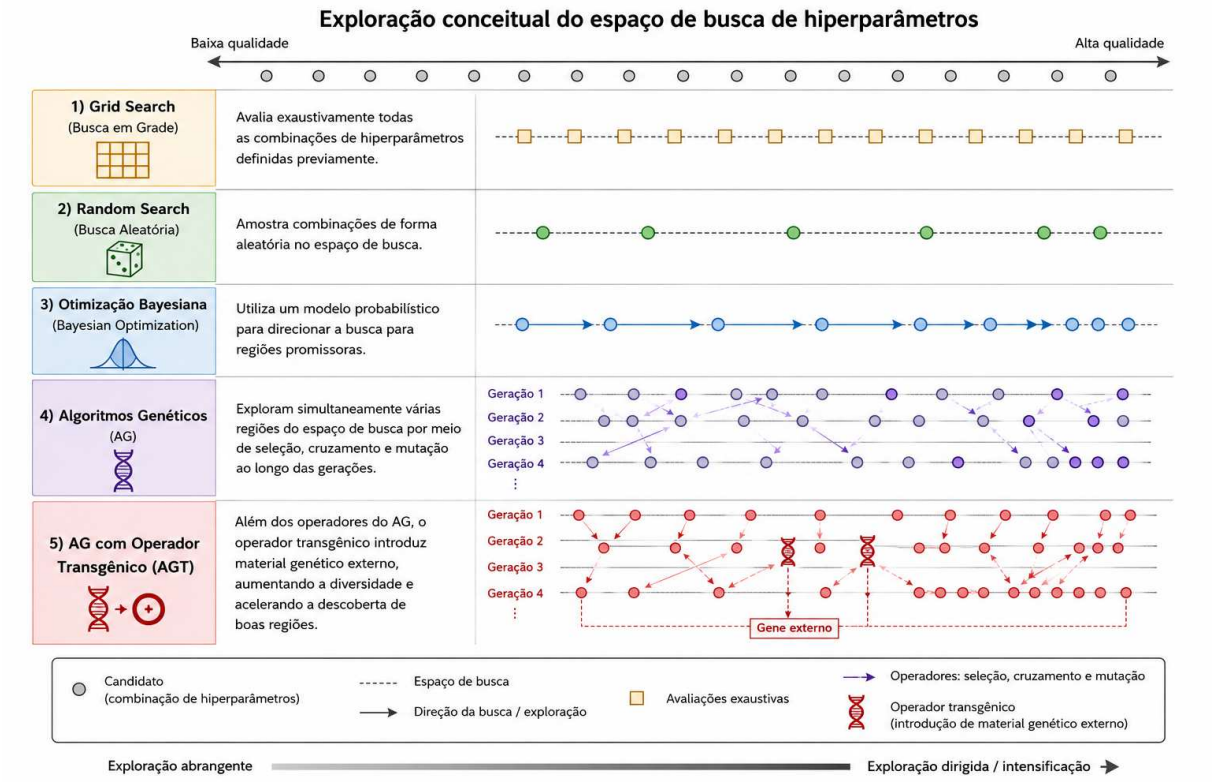

Figura 2 – Exploração conceitual do espaço de busca por diferentes estratégias de otimização de hiperparâmetros. A figura ilustra, de forma esquemática, como métodos tradicionais e evolutivos percorrem o espaço de busca. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

Nesse contexto, os AGs constituem uma estratégia promissora para automatizar a seleção de hiperparâmetros, reduzindo a dependência da configuração manual e aumentando a probabilidade de obtenção de arquiteturas com elevado desempenho preditivo. Seus fundamentos são apresentados na seção seguinte.

2.4 Algoritmos Genéticos (AGs)

Conforme discutido na seção anterior, diferentes estratégias podem ser empregadas para a otimização automática de hiperparâmetros. Entre elas, os AGs destacam-se pela capacidade de explorar espaços de busca complexos utilizando mecanismos inspirados na evolução natural. Esta seção apresenta os principais fundamentos dessa técnica de otimização.

Os AGs constituem uma classe de algoritmos de otimização inspirados nos princípios da seleção natural e da evolução biológica, proposta inicialmente por Holland (1975). Inseridos no contexto da Computação Evolutiva, esses algoritmos são amplamente em-

pregados na resolução de problemas de otimização caracterizados por espaços de busca extensos, funções objetivo não lineares e múltiplos ótimos locais, situações nas quais métodos determinísticos frequentemente apresentam limitações.

O AG simula o processo de evolução biológica, onde uma população de possíveis soluções é refinada ao longo de várias gerações até atingir um critério de parada, que pode ser a convergência para uma solução ótima ou o término de um número pré-definido de iterações. O processo geral de um AG segue os seguintes passos básicos:

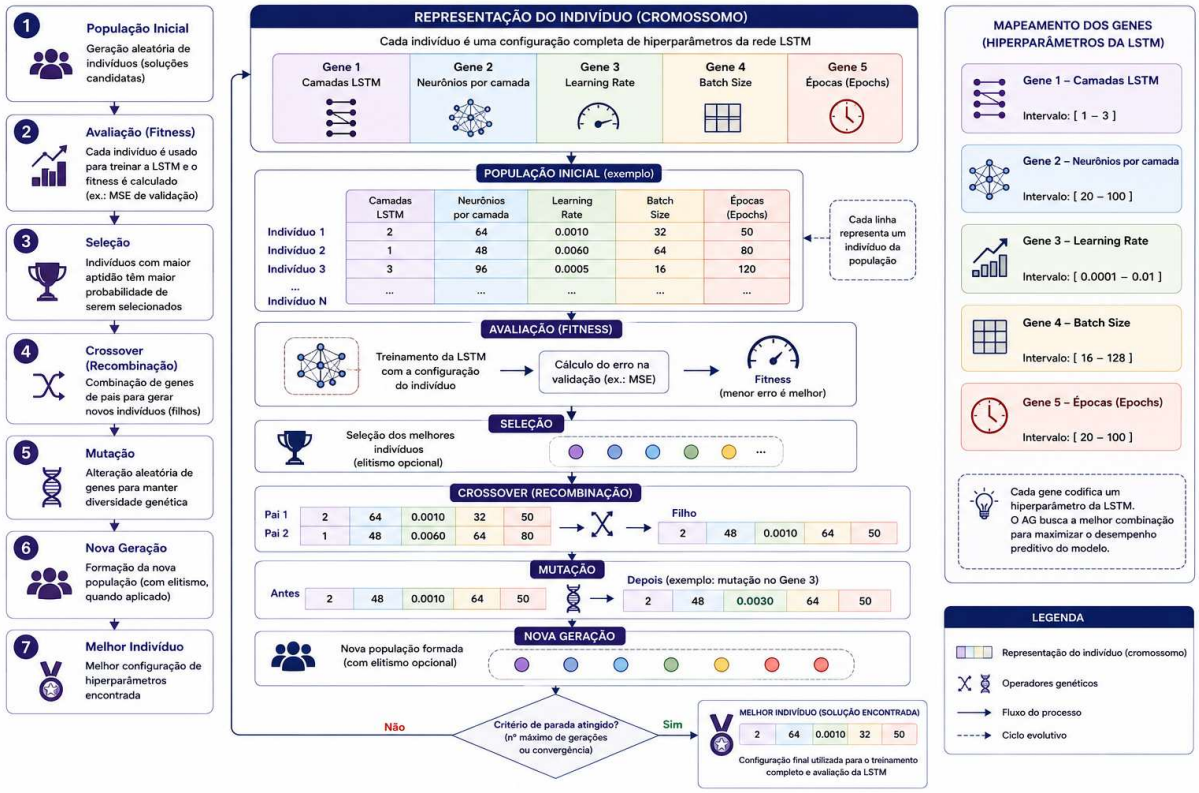

Figura 3 – Representação conceitual do processo evolutivo empregado na otimização dos hiperparâmetros das redes neurais LSTM utilizando AGs. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

A Figura 3 apresenta uma representação conceitual do processo evolutivo empregado nesta pesquisa para otimizar automaticamente os hiperparâmetros das redes neurais LSTM. Inicialmente, é gerada uma população composta por indivíduos, sendo que cada indivíduo representa uma possível configuração da arquitetura da rede. Cada cromossomo é formado por cinco genes, correspondentes aos hiperparâmetros considerados na otimização: número de camadas LSTM, número de neurônios por camada, taxa de aprendizado (*learning rate*), tamanho do lote (*batch size*) e número de épocas de treinamento.

Cada indivíduo é então utilizado para instanciar e treinar uma rede LSTM, cuja qua-

lidade é avaliada por meio da função de aptidão (*fitness*), baseada no erro obtido durante a etapa de validação. Em seguida, os indivíduos mais aptos são selecionados para reprodução, dando origem a uma nova população por meio dos operadores de cruzamento (*crossover*) e mutação.

Esse processo evolutivo é repetido sucessivamente até que seja atingido o número máximo de gerações estabelecido ou outro critério de parada previamente definido. Ao final da busca, o indivíduo com melhor valor de aptidão representa a configuração de hiperparâmetros selecionada para o treinamento final da rede LSTM e posterior avaliação no conjunto de teste e no processo de *backtesting*. Embora a figura represente o fluxo geral do Algoritmo Genético, a adaptação proposta nesta dissertação, baseada no operador transgênico, será apresentada na Seção 2.5, onde são descritas as modificações introduzidas no processo evolutivo tradicional.

1. **Inicialização da População:** O processo começa com a geração de uma população inicial de indivíduos, onde cada indivíduo representa uma solução possível para o problema em questão. Esses indivíduos são representados por cromossomos, que podem ser codificados de diversas maneiras (binária, real, entre outras) (MITCHELL, 1998);

2. **Avaliação (Função *Fitness*):** Cada indivíduo da população é avaliado por uma função de aptidão (*fitness*), que mede a qualidade de cada solução. A função *fitness* é essencial para determinar quais indivíduos possuem as melhores características para a solução do problema (GOLDBERG, 1989);

3. **Seleção:** Após a avaliação, os indivíduos são selecionados para reprodução com base em sua aptidão. Técnicas comuns de seleção incluem a roleta, classificação, elitismo e o torneio, onde indivíduos com maior aptidão têm uma chance maior de serem selecionados (EIBEN; SMITH, 2003);

4. ***Crossover* (Recombinação):** Os indivíduos selecionados passam por um processo de recombinação, no qual partes de seus cromossomos são trocadas para gerar novos indivíduos. O *crossover* simula o processo de reprodução sexual e visa combinar as boas características de duas soluções diferentes (BÄCK, 1996). Técnicas de *crossover* comuns incluem o *crossover* de um ponto, dois pontos e o *crossover Partially Mapped Crossover* (PMX);

5. **Mutação:** Para garantir a diversidade genética na população e evitar a convergência prematura para uma solução subótima, ocorre o processo de mutação, onde alguns genes dos indivíduos podem ser alterados aleatoriamente (HOLLAND, 1975).

Após a aplicação dos operadores de cruzamento e mutação, os novos indivíduos são inseridos na população, processo conhecido como reinserção. Em muitas implementa-

ções utiliza-se o elitismo, estratégia que preserva os melhores indivíduos da geração anterior, reduzindo a probabilidade de perda de soluções promissoras durante a evolução (MITCHELL, 1998). O ciclo evolutivo é repetido sucessivamente até que um critério de parada seja satisfeito, como o número máximo de gerações ou a convergência da população para soluções estáveis (GOLDBERG, 1989).

Além desses processos e operadores, pode-se salientar outros elementos importantes dentre os algoritmos Bio-inspirados no AG (MENDES, 2021):

1. **Gene:** unidade básica que compõe um cromossomo;
2. **Cromossomo:** conjunto organizado de genes;
3. **Indivíduo:** estrutura composta por um ou mais cromossomos, representando uma possível solução para o problema;
4. **População:** conjunto de indivíduos manipulados durante a execução do AG;
5. **Geração:** é uma etapa onde a população atual de indivíduos passa por processos de seleção, *crossover* e mutação, gerando uma nova população.

Diversos trabalhos recentes têm empregado AGs na otimização de redes neurais artificiais, especialmente em arquiteturas LSTM aplicadas à previsão de séries temporais. Em Huang et al. (2021), um AG foi utilizado para otimizar parâmetros da *Variational Mode Decomposition* (VMD). Já Chung e Shin (2018) empregaram AGs para otimizar hiperparâmetros da LSTM, incluindo a janela temporal e o número de camadas ocultas. Em Lin e Chen (2018), a otimização concentrou-se nos pesos internos da rede LSTM, enquanto Dantas (2017) propôs uma estratégia para determinar automaticamente a quantidade de neurônios em cada camada oculta.

Esses estudos evidenciam a versatilidade dos AGs na otimização de diferentes componentes de redes neurais profundas, demonstrando sua capacidade de explorar espaços de busca complexos e identificar configurações competitivas em problemas de previsão de séries temporais.

Os AGs são amplamente utilizados em diversas áreas, como otimização de funções complexas, aprendizado de máquina e sistemas de controle (BÄCK, 1996). Sua capacidade de explorar grandes espaços de busca e encontrar soluções próximas do ótimo em problemas não lineares e com múltiplos picos globais e locais torna os AGs uma ferramenta eficaz e robusta.

Dessa forma, os AGs consolidaram-se como uma das principais técnicas de otimização empregadas em problemas de aprendizado profundo, especialmente quando a definição

manual dos hiperparâmetros se torna inviável devido à elevada dimensionalidade do espaço de busca. Entretanto, apesar de sua reconhecida capacidade exploratória, diferentes estratégias têm sido propostas para aumentar sua eficiência durante o processo evolutivo. Entre elas destaca-se o operador transgênico, tema da próxima seção e principal elemento de inovação investigado nesta dissertação.

2.5 Algoritmo Genético Transgênico

Os AGs convencionais utilizam operadores de seleção, cruzamento e mutação para promover a evolução da população ao longo das gerações. Embora esses operadores apresentem bom desempenho em diversos problemas de otimização, eles podem sofrer com convergência prematura, perda de diversidade genética e dificuldade para escapar de ótimos locais, especialmente em espaços de busca de alta dimensionalidade (GOLDBERG, 1989; EIBEN; SMITH, 2003).

Com o objetivo de reduzir essas limitações, Amaral e Jr (2014) propuseram um novo operador evolutivo denominado *Transgenic*, inspirado nos princípios da engenharia genética e dos organismos geneticamente modificados (OGMs). Diferentemente dos operadores clássicos, o operador transgênico permite a inserção artificial de características consideradas promissoras em indivíduos da população, aumentando a pressão seletiva sobre regiões potencialmente relevantes do espaço de busca sem substituir os mecanismos convencionais do AG (AMARAL; JR, 2014).

A motivação do operador está baseada no fato de que, em diversos problemas de otimização, determinados genes exercem influência significativamente maior sobre a qualidade das soluções encontradas. Dessa forma, ao invés de depender exclusivamente da recombinação aleatória promovida pelo cruzamento e pela mutação, o operador transgênico utiliza informações provenientes das melhores soluções encontradas durante o processo evolutivo para reforçar artificialmente a presença desses genes nas gerações seguintes (AMARAL; JR, 2014).

No trabalho original, quando existe conhecimento prévio sobre quais genes são relevantes para o problema, esses genes podem ser diretamente inseridos nos indivíduos selecionados. Entretanto, quando esse conhecimento não está disponível, o operador constrói automaticamente uma estrutura histórica capaz de identificar quais genes e respectivos valores aparecem com maior frequência entre os indivíduos de melhor desempenho, utilizando essas informações para direcionar a evolução da população (AMARAL; JR, 2014).

O funcionamento do operador transgênico baseia-se na construção de uma estrutura histórica responsável por armazenar informações provenientes dos indivíduos de melhor desempenho encontrados durante o processo evolutivo. Essa estrutura constitui o principal diferencial em relação ao Algoritmo Genético tradicional, pois permite utilizar o

conhecimento acumulado ao longo das gerações para orientar a transferência de genes considerados promissores (AMARAL; JR, 2014).

No trabalho original, a estrutura histórica é composta por um conjunto de registros associados a cada gene do cromossomo. Para cada gene são armazenadas duas informações principais: o valor considerado mais relevante para aquele gene e a quantidade de vezes que esse valor apareceu entre os melhores indivíduos encontrados até o momento. Além disso, a estrutura mantém registrada a melhor avaliação (*fitness*) observada durante a evolução, permitindo que o histórico seja atualizado apenas quando novas soluções superiores são encontradas (AMARAL; JR, 2014).

Para representar essa estrutura histórica de forma mais intuitiva, a Figura 4 apresenta uma adaptação conceitual baseada na proposta original de Amaral e Jr (2014).

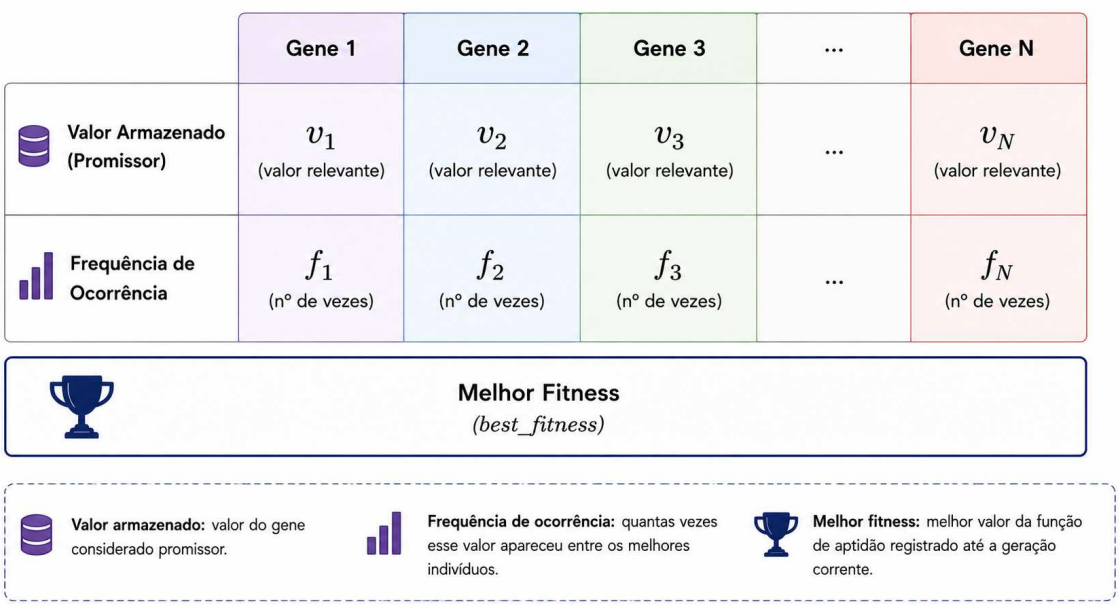

Figura 4 – Representação conceitual da estrutura histórica utilizada pelo operador Transgênico proposto por Amaral e Jr (2014). Para cada gene do cromossomo são armazenados o valor considerado promissor e sua frequência de ocorrência entre os indivíduos associados à melhor aptidão observada. A estrutura também registra o melhor valor da função de aptidão obtido até a geração corrente. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)), com base em Amaral e Jr (2014).

A Figura 4 apresenta a organização conceitual da estrutura histórica utilizada pelo operador transgênico. Cada posição do histórico corresponde a um gene do cromossomo e armazena duas informações: o valor genético armazenado para aquele gene e sua frequência de ocorrência entre os indivíduos associados à melhor aptidão. Além dessas informações, a estrutura mantém registrado o melhor valor da função de aptidão encontrado até a geração corrente, servindo como referência para a atualização do histórico ao

longo do processo evolutivo.

A atualização dessa estrutura ocorre sempre que indivíduos com desempenho superior ao melhor valor anteriormente registrado são encontrados. Nessa situação, o histórico é reinicializado e passa a armazenar exclusivamente as informações associadas às novas melhores soluções. Caso nenhuma solução apresente valor de aptidão superior ao melhor já registrado, o histórico não é reinicializado. Nesse caso, apenas as frequências de ocorrência dos valores genéticos associados aos indivíduos que apresentam a melhor aptidão continuam sendo atualizadas, permitindo identificar quais características permanecem recorrentes entre as melhores soluções encontradas durante a evolução (AMARAL; JR, 2014).

2.5.1 Operação Transgênica

Após a construção da estrutura histórica, o operador transgênico passa a utilizar as informações nela armazenadas para realizar a transferência dirigida de material genético aos indivíduos da população. Diferentemente da mutação tradicional, na qual as alterações ocorrem de forma essencialmente aleatória, a operação transgênica utiliza o conhecimento adquirido durante a evolução da população para direcionar a modificação dos cromossomos, privilegiando genes associados às melhores soluções encontradas (AMARAL; JR, 2014).

O processo inicia-se pela seleção probabilística dos genes que serão modificados. Essa seleção é realizada com base nas frequências armazenadas na estrutura histórica, de modo que genes com maior frequência de ocorrência entre os indivíduos associados à melhor aptidão apresentam maior probabilidade de serem escolhidos durante a operação transgênica. Dessa forma, o operador utiliza informações acumuladas ao longo das gerações para orientar a exploração do espaço de busca, reduzindo a dependência de alterações puramente aleatórias (AMARAL; JR, 2014).

Após a seleção do gene, o operador substitui o valor originalmente presente no cromossomo pelo valor armazenado na estrutura histórica. Esse processo caracteriza a transferência artificial de material genético, preservando características consideradas promissoras e aumentando sua probabilidade de propagação nas gerações subsequentes. Diferentemente do cruzamento, que depende da combinação entre dois indivíduos, a operação transgênica atua diretamente sobre um único cromossomo utilizando informações provenientes do histórico evolutivo (AMARAL; JR, 2014).

A Figura 5 apresenta uma representação conceitual desse processo. Inicialmente, o histórico evolutivo fornece as frequências de ocorrência dos genes associados às melhores soluções. Em seguida, essas frequências são utilizadas para selecionar probabilisticamente quais genes serão transferidos. Após a substituição dos respectivos valores no cromossomo,

obtém-se um novo indivíduo transgênico, que posteriormente será avaliado juntamente com os demais indivíduos da população.

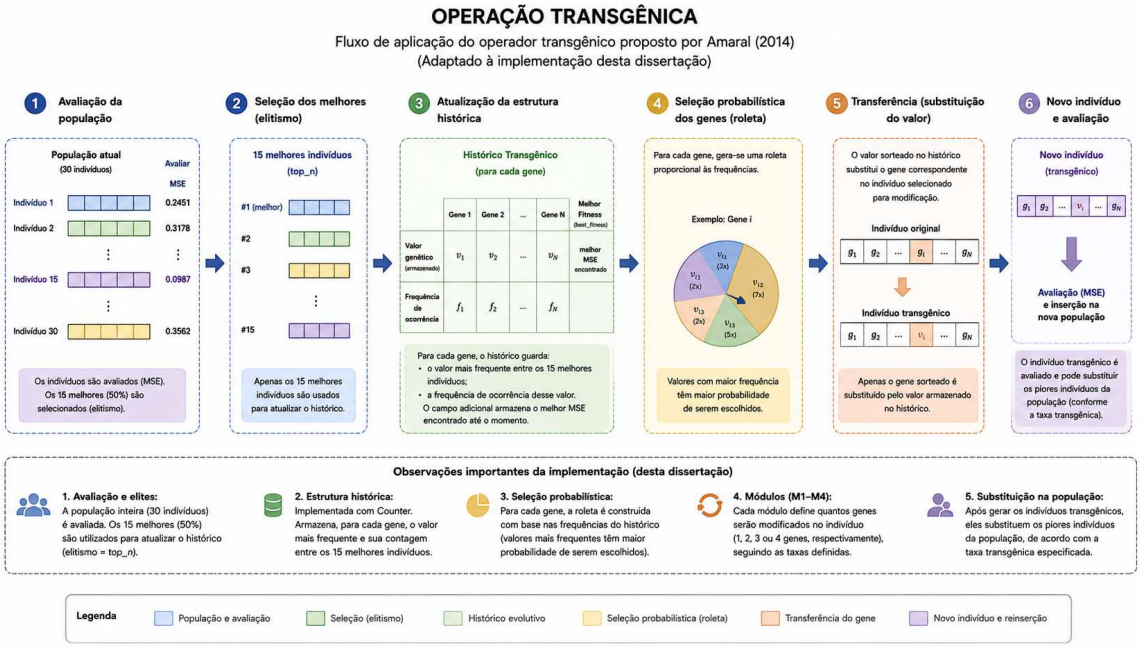

Figura 5 – Representação conceitual da operação transgênica proposta por Amaral e Jr (2014). O histórico evolutivo fornece os genes e suas frequências de ocorrência, que são utilizados para selecionar probabilisticamente os valores genéticos a serem transferidos aos indivíduos da população. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)), com base em Amaral e Jr (2014).

2.5.2 Módulos M1, M2, M3 e M4

Na proposta original de Amaral e Jr (2014), o operador Transgênico é organizado em quatro módulos independentes, denominados M_1, M_2, M_3 e M_4. Cada módulo recebe uma cópia independente da população corrente e utiliza as informações armazenadas na estrutura histórica para selecionar probabilisticamente os genes que serão transferidos aos indivíduos submetidos à operação transgênica.

A diferença entre os módulos está na quantidade de genes selecionados e modificados em cada indivíduo. No módulo M_1, a roleta probabilística é acionada uma única vez, resultando na seleção de um gene. O valor associado a esse gene na estrutura histórica é então inserido em N_1 indivíduos da cópia da população correspondente ao módulo.

No módulo M_2, a roleta é acionada duas vezes, permitindo a seleção de dois genes. Os valores armazenados no histórico para esses genes são utilizados para atualizar N_2 indivíduos. De modo análogo, o módulo M_3 seleciona três genes e transfere seus respectivos valores para N_3 indivíduos, enquanto o módulo M_4 seleciona quatro genes e modifica N_4 indivíduos da população correspondente (AMARAL; JR, 2014).

A atuação dos módulos pode ser sintetizada da seguinte forma: M_1 seleciona e transfere um gene; M_2 seleciona e transfere dois genes; M_3 seleciona e transfere três genes; e M_4 seleciona e transfere quatro genes.

Como a seleção é probabilística, os genes mais frequentes no histórico apresentam maior probabilidade de serem escolhidos, mas sua transferência não é determinística. Assim, o operador privilegia características recorrentes entre as melhores soluções sem eliminar completamente a natureza estocástica do processo evolutivo (AMARAL; JR, 2014).

Cada módulo produz uma nova população modificada de maneira independente. Após a aplicação da operação transgênica, os indivíduos das quatro populações resultantes são avaliados por meio da função de aptidão. Em seguida, os melhores indivíduos são selecionados para compor uma nova população transgênica com o mesmo tamanho da população utilizada pelo Algoritmo Genético (AMARAL; JR, 2014).

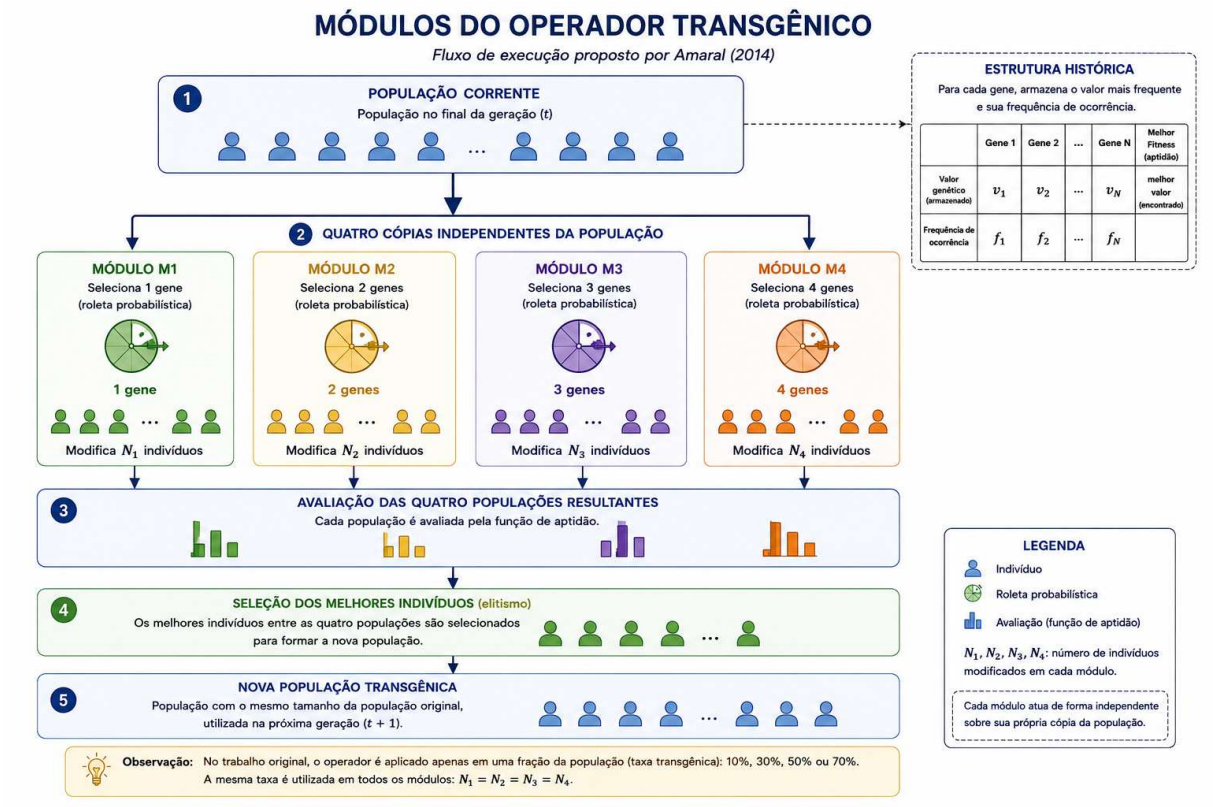

Figura 6 – Representação conceitual dos módulos do operador Transgênico proposto por Amaral e Jr (2014). Cada módulo atua sobre uma cópia independente da população corrente, diferenciando-se pela quantidade de genes modificados durante a operação transgênica. Após a avaliação das quatro populações resultantes, os melhores indivíduos são selecionados para compor a nova população. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)), com base em Amaral e Jr (2014).

No estudo original, o operador não foi aplicado a todos os indivíduos. Os autores analisaram diferentes taxas transgênicas, correspondentes a 10%, 30%, 50% e 70% da população. Em cada experimento, a mesma taxa foi aplicada aos quatro módulos, isto é, $N_1 = N_2 = N_3 = N_4$, possibilitando avaliar o efeito da intensidade da intervenção transgênica sobre o comportamento evolutivo (AMARAL; JR, 2014).

Para facilitar a compreensão da atuação dos quatro módulos do operador transgênico, a Figura 6 apresenta uma representação conceitual do fluxo de execução proposto por Amaral e Jr (2014). Observa-se que cada módulo atua de forma independente sobre uma cópia da população corrente, diferenciando-se apenas pela quantidade de genes modificados em cada indivíduo submetido à operação transgênica.

A Figura 6 sintetiza a organização dos quatro módulos que compõem o operador transgênico. Inicialmente, a população corrente é duplicada em quatro cópias independentes, permitindo que cada módulo realize modificações distintas sobre os cromossomos. Enquanto o módulo M_1 modifica apenas um gene por indivíduo, os módulos M_2, M_3 e M_4 modificam, respectivamente, dois, três e quatro genes selecionados probabilisticamente a partir da estrutura histórica. Após a aplicação da operação transgênica, cada população é avaliada por meio da função de aptidão e, posteriormente, os melhores indivíduos são selecionados para formar a população utilizada na próxima geração do Algoritmo Genético (AMARAL; JR, 2014).

2.5.3 Resultados e Limitações do Operador Transgênico

O operador Transgênico foi avaliado por Amaral e Jr (2014) em problemas de indução de regras para classificação, utilizando bases sintéticas e uma base de dados relacionada à *Gene Ontology*. Os experimentos compararam o desempenho do operador com o Algoritmo Genético tradicional, considerando diferentes taxas de aplicação da operação transgênica ao longo do processo evolutivo.

De maneira geral, os resultados obtidos indicaram que o operador Transgênico foi capaz de produzir indivíduos com melhor qualidade quando comparado ao Algoritmo Genético convencional. Em diversos experimentos, observou-se aumento da função de aptidão (*fitness*), maior capacidade de seleção de atributos relevantes e geração de regras de classificação mais compactas, evidenciando que a utilização das informações armazenadas na estrutura histórica contribuiu para direcionar a evolução da população para regiões mais promissoras do espaço de busca (AMARAL; JR, 2014).

Entretanto, os autores também destacam algumas limitações da abordagem proposta. A principal delas está relacionada ao aumento do custo computacional, decorrente da construção da estrutura histórica, da execução independente dos quatro módulos transgênicos e da avaliação das populações resultantes. Além disso, o operador introduz uma

pressão seletiva adicional sobre a população, o que pode favorecer a convergência prematura caso a diversidade genética não seja preservada adequadamente ao longo da evolução (AMARAL; JR, 2014).

Os experimentos também mostraram que o desempenho do operador depende da taxa transgênica adotada. Taxas muito reduzidas limitam o impacto da transferência dirigida de genes, enquanto taxas excessivamente elevadas podem reduzir a diversidade populacional e aumentar o risco de convergência para ótimos locais. Dessa forma, a definição da intensidade da operação transgênica constitui um fator importante para o equilíbrio entre exploração e intensificação do processo evolutivo (AMARAL; JR, 2014).

Embora o operador Transgênico tenha sido originalmente desenvolvido para problemas de indução de regras de classificação, seus princípios são suficientemente gerais para serem empregados em outros problemas de otimização. Em particular, a utilização de uma estrutura histórica para direcionar probabilisticamente a transferência de genes pode ser adaptada para diferentes representações cromossômicas, desde que seja possível identificar genes associados às melhores soluções encontradas durante a evolução.

Nesta dissertação, esses princípios são empregados na otimização de hiperparâmetros de redes neurais LSTM aplicadas à previsão dos preços de fechamento do café arábica no mercado financeiro. As adaptações realizadas em relação ao operador originalmente proposto por Amaral e Jr (2014), bem como sua implementação computacional e integração ao processo de otimização das redes neurais, são apresentadas detalhadamente no Capítulo 3.

2.6 Métricas de Avaliação

Esta seção apresenta as métricas estatísticas empregadas para avaliar o desempenho preditivo dos modelos desenvolvidos nesta dissertação. Como o objetivo do estudo consiste em comparar diferentes estratégias de otimização de hiperparâmetros para redes neurais LSTM, torna-se necessário utilizar medidas capazes de quantificar tanto a magnitude dos erros de previsão quanto a capacidade de generalização dos modelos.

As métricas selecionadas contemplam diferentes perspectivas de avaliação. O MAPE permite analisar o erro percentual médio das previsões, o MSE e o RMSE quantificam a magnitude dos erros absolutos, atribuindo maior penalização a erros de maior intensidade, enquanto o coeficiente de determinação (R^2) mede a capacidade do modelo em explicar a variabilidade dos dados observados. Em conjunto, essas métricas fornecem uma avaliação abrangente do desempenho dos modelos e permitem comparar, de forma consistente, a LSTM clássica, a LSTM otimizada por Algoritmo Genético (LSTM-AG) e a LSTM otimizada por Algoritmo Genético Transgênico (LSTM-AGT).

2.6.1 Mean Absolute Percentage Error (MAPE)

O MAPE (Erro Percentual Médio Absoluto) mede a precisão de uma previsão calculando a média dos erros percentuais absolutos entre os valores reais e preditos. Esse indicador é útil para avaliar o erro em termos percentuais. O cálculo do MAPE é apresentado na Equação (1).

$$MAPE = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \tag{1}$$

Melhor valor: O valor ideal para o MAPE é 0%, que indica uma previsão perfeita (sem erro percentual).

Interpretação: Valores mais baixos de MAPE indicam que o modelo tem menor erro percentual médio em relação aos valores reais. No entanto, o MAPE pode ser instável quando há valores reais próximos de zero, devido à divisão na fórmula. Por isso, o MAPE é mais confiável quando todos os valores reais são significativamente diferentes de zero.

Nesta dissertação, o MAPE é utilizado como uma métrica complementar para avaliar o erro percentual médio das previsões produzidas pelos diferentes modelos. Como os preços de fechamento do café arábica permanecem sempre positivos, a limitação associada à divisão por valores próximos de zero não compromete sua utilização neste estudo.

2.6.2 Mean Squared Error (MSE)

O MSE (Erro Quadrático Médio) é uma métrica que avalia a média dos quadrados dos erros entre os valores reais e preditos. Por elevar os erros ao quadrado, penaliza mais severamente previsões com maiores desvios, sendo sensível à presença de *outliers*. O cálculo do MSE é apresentado na Equação (2).

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2}$$

Melhor valor: O valor ideal para o MSE é 0, indicando que não há diferença entre os valores reais e preditos.

Interpretação: Quanto mais próximo de zero, menor o erro quadrático médio do modelo. Como o MSE penaliza erros maiores de forma mais severa (devido ao quadrado), valores altos de MSE indicam que há predições com grandes desvios em relação aos valores reais.

O MSE também desempenha um papel importante nesta pesquisa por ser a principal métrica utilizada durante o processo de otimização dos modelos. Ao penalizar mais fortemente erros de maior magnitude, essa métrica favorece a seleção de arquiteturas capazes de produzir previsões mais próximas dos valores reais, tornando-se adequada para

a função de aptidão empregada na comparação entre diferentes configurações das redes neurais.

2.6.3 Root Mean Squared Error (RMSE)

O RMSE (Raiz do Erro Quadrático Médio) corresponde à raiz quadrada do MSE e expressa o erro médio na mesma unidade dos dados originais, facilitando a interpretação dos resultados. Assim como o MSE, essa métrica penaliza erros de maior magnitude e, por isso, também é sensível à presença de *outliers*. O cálculo do RMSE é apresentado na Equação (3).

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3}$$

Melhor valor: O valor ideal para o RMSE é 0, o que indica uma previsão sem erro.

Interpretação: Valores de RMSE mais baixos indicam que o erro médio é menor. Como o RMSE está na mesma unidade dos valores reais, é intuitivo para interpretar o quão distante, em média, as predições estão dos valores reais. Assim como o MSE, o RMSE é sensível a *outliers*.

Como o RMSE possui a mesma unidade da variável prevista, sua interpretação torna-se mais intuitiva quando comparada ao MSE. Nesta dissertação, essa métrica será utilizada para comparar diretamente a qualidade das previsões produzidas pelas diferentes arquiteturas de redes neurais avaliadas.

2.6.4 R-squared (Coeficiente de Determinação)

O coeficiente de determinação (*R-squared* ou R^2) mede a proporção da variabilidade dos valores reais que é explicada pelo modelo preditivo. Em geral, valores mais próximos de 1 indicam maior capacidade do modelo em explicar a variação dos dados observados. Entretanto, essa métrica também pode assumir valores negativos quando o modelo apresenta desempenho inferior ao de um preditor constante baseado na média dos valores observados. O cálculo do coeficiente de determinação é apresentado na Equação (4).

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{4}$$

Melhor valor: O valor ideal para o R^2 é 1, o que indica que o modelo explica perfeitamente a variabilidade dos dados reais.

Interpretação: O R^2 varia de 0 a 1, onde:

- Um valor próximo de 1 significa que o modelo explica bem a variabilidade dos dados.

- Um valor próximo de 0 indica que o modelo tem pouco ou nenhum poder preditivo para os dados.
- Em alguns casos, R^2 pode até ser negativo (caso o modelo preditivo tenha um desempenho pior do que um modelo constante).

Embora as métricas de erro permitam quantificar a proximidade entre valores observados e previstos, elas não indicam quanto da variabilidade da série temporal é efetivamente explicada pelo modelo. Por esse motivo, o coeficiente de determinação (R^2) é utilizado como uma medida complementar para avaliar a capacidade explicativa das diferentes arquiteturas investigadas.

As métricas apresentadas nesta seção serão utilizadas em diferentes etapas da avaliação experimental desta dissertação. Inicialmente, servirão para comparar o desempenho preditivo das arquiteturas obtidas durante a otimização dos hiperparâmetros. Posteriormente, serão empregadas na análise do treinamento final dos modelos e, por fim, seus resultados serão confrontados com os indicadores financeiros obtidos durante o processo de *backtesting*, permitindo investigar a relação entre desempenho estatístico e desempenho operacional das estratégias de negociação.

2.7 Dinâmica dos mercados de *commodities* e fundamentos da previsão

A previsão de preços no mercado de *commodities*, especialmente no caso do café arábica, está inserida em um contexto caracterizado por elevada volatilidade, não linearidade e influência simultânea de fatores econômicos, financeiros e estruturais. A literatura recente evidencia que esses mercados apresentam características que dificultam sua modelagem, destacando-se a interdependência entre diferentes regiões produtoras, a crescente participação de investidores institucionais e a existência de padrões temporais exploráveis (MELO-VELANDIA; OTERO; RAMÍREZ-GONZÁLEZ, 2025; SUÁREZ-RODRÍGUEZ; MANOTAS-DUQUE; LARGO-AVILA, 2026; COULEAU; TRUJILLO-BARRERA; ETIENNE, 2026).

O estudo de Melo-Velandia, Otero e Ramírez-González (2025) analisa a propagação de choques de risco entre mercados globais de café, evidenciando que esses mercados operam de forma interconectada. Utilizando medidas de dependência baseadas em cópulas, os autores demonstram que eventos extremos em uma região podem impactar significativamente outras regiões produtoras. Esse resultado indica que o comportamento dos preços não pode ser analisado de forma isolada, sendo necessário considerar a estrutura global do mercado.

Sob outra perspectiva, Suárez-Rodríguez, Manotas-Duque e Largo-Avila (2026) investigam o processo de financeirização do mercado de café arábica. Os autores utilizam dados de contratos futuros e posições especulativas para demonstrar que a atuação de investidores institucionais influencia diretamente a formação dos preços. Um dos principais resultados do estudo é a identificação de regimes persistentes de contango, nos quais os preços futuros permanecem sistematicamente acima dos preços à vista. Isso evidencia que os preços podem se afastar dos fundamentos econômicos, dificultando a modelagem preditiva.

Já Couleau, Trujillo-Barrera e Etienne (2026) utilizam dados de alta frequência para analisar a existência de padrões de *momentum* intradiário. Por meio de testes de autocorrelação e análise de retornos, os autores demonstram que há previsibilidade no curto prazo, especialmente em intervalos intradiários. Esse resultado é relevante, pois contraria a hipótese de eficiência de mercado em sua forma mais forte, sugerindo que modelos preditivos podem explorar essas ineficiências.

Embora abordem perspectivas distintas, os três estudos convergem ao demonstrar que a dinâmica dos mercados de *commodities* não pode ser explicada por um único fator. Enquanto Melo-Velandia, Otero e Ramírez-González (2025) evidenciam a interdependência entre mercados globais de café, Suárez-Rodríguez, Manotas-Duque e Largo-Avila (2026) mostram que a financeirização modifica o processo de formação dos preços, e Couleau, Trujillo-Barrera e Etienne (2026) identificam padrões temporais exploráveis em séries intradiárias. Em conjunto, esses resultados reforçam que a previsão de preços do café arábica exige modelos capazes de representar relações não lineares e múltiplas dependências temporais.

De forma geral, os trabalhos analisados indicam que, apesar da elevada volatilidade dos mercados de *commodities*, existem padrões temporais e estruturas de dependência que podem ser explorados por modelos de previsão (MELO-VELANDIA; OTERO; RAMÍREZ-GONZÁLEZ, 2025; SUÁREZ-RODRÍGUEZ; MANOTAS-DUQUE; LARGO-AVILA, 2026; COULEAU; TRUJILLO-BARRERA; ETIENNE, 2026). Entretanto, a complexidade dessas séries temporais limita a utilização de modelos puramente lineares, motivando o desenvolvimento de abordagens baseadas em aprendizado de máquina, particularmente modelos baseados em redes neurais profundas, discutidos nas seções seguintes.

2.8 Modelos tradicionais e evolução das abordagens preditivas

Historicamente, a previsão de preços de *commodities* foi baseada em modelos estatísticos lineares, como *Autoregressive Integrated Moving Average* (ARIMA) e *Vector Autoregression* (VAR), amplamente utilizados para representar dependências temporais em

séries econômicas. Entretanto, a elevada volatilidade, a presença de relações não lineares e a influência simultânea de múltiplos fatores reduziram a capacidade desses modelos de representar adequadamente o comportamento dos preços, motivando o desenvolvimento de abordagens baseadas em aprendizado de máquina e aprendizado profundo (DEINA et al., 2022; CELIK; CELIK, 2025; AMALIA et al., 2026).

Nesse contexto, Deina et al. (2022) realizam uma comparação entre modelos lineares e não lineares aplicados à previsão de preços de *commodities*. Utilizando dados históricos e diferentes configurações de treinamento, os autores demonstram que o modelo *Extreme Learning Machine* (ELM), uma rede neural de treinamento rápido, apresenta desempenho superior ao ARIMA em termos de erro médio absoluto (MAE) e erro quadrático médio (RMSE). Esse resultado reforça a necessidade de utilização de modelos não lineares em séries complexas.

De forma semelhante, Kamocsai e Ormos (2026) propõem um modelo baseado em fatores de volatilidade, incorporando efeitos de alavancagem e heterocedasticidade. Os resultados mostram que a inclusão desses fatores melhora significativamente a previsão da volatilidade, especialmente em períodos de alta instabilidade.

Já Celik e Celik (2025) apresentam um modelo híbrido que combina ARIMA, LSTM e componentes estocásticos. A principal contribuição do estudo é demonstrar que nenhum modelo isolado é capaz de capturar todas as características da série, sendo necessário combinar diferentes abordagens. Os resultados indicam que o modelo híbrido apresenta menor erro preditivo em comparação com os modelos individuais.

Os três estudos anteriores evidenciam uma evolução gradual das abordagens preditivas. Enquanto Deina et al. (2022) demonstram as limitações dos modelos lineares frente a métodos baseados em redes neurais, Kamocsai e Ormos (2026) incorporam fatores de volatilidade para melhorar a modelagem estatística, e Celik e Celik (2025) mostram que a combinação de diferentes técnicas pode produzir modelos mais robustos. Em conjunto, esses trabalhos indicam uma tendência de substituição de modelos isolados por abordagens híbridas capazes de representar melhor a dinâmica das séries financeiras.

Além disso, Cortazar et al. (2019) utilizam modelos baseados em fatores econômicos, considerando variáveis como taxa de juros e expectativas de mercado. Os autores demonstram que a inclusão dessas variáveis melhora a capacidade de previsão, especialmente em horizontes mais longos. Já Wang e Li (2018) utilizam a técnica de *Singular Spectrum Analysis* (SSA) para decompor a série temporal em componentes de tendência e ruído, permitindo uma modelagem mais precisa.

Esses trabalhos ampliam a perspectiva dos modelos híbridos ao incorporar informações econômicas e técnicas de decomposição de séries temporais. Enquanto Cortazar et al. (2019) exploram variáveis macroeconômicas para representar fatores fundamentais do

mercado, Wang e Li (2018) utilizam técnicas de decomposição para separar componentes estruturais e reduzir a influência do ruído nos dados.

Mais recentemente, Amalia et al. (2026) utilizam o modelo *Temporal Fusion Transformer* (TFT) para previsão *multi-step* de preços de *commodities* alimentares. O modelo combina mecanismos de atenção com redes neurais profundas, permitindo capturar dependências temporais complexas. De forma semelhante, Sediyono et al. (2025) propõem um modelo baseado em *Transformer* para previsão e detecção de anomalias, demonstrando superioridade em relação a modelos recorrentes tradicionais.

Observa-se, portanto, uma evolução das abordagens preditivas, desde modelos estatísticos lineares até arquiteturas profundas baseadas em mecanismos de atenção. Essa evolução reflete o aumento da capacidade computacional e a necessidade de modelar relações temporais cada vez mais complexas presentes nas séries financeiras (AMALIA et al., 2026; SEDIYONO et al., 2025).

Apesar dos avanços obtidos por arquiteturas mais recentes, como os modelos baseados em *Transformers*, essas abordagens apresentam maior custo computacional e, em muitos casos, menor interpretabilidade quando comparadas a modelos recorrentes tradicionais (AMALIA et al., 2026; SEDIYONO et al., 2025). Nesse contexto, arquiteturas recorrentes, como as redes LSTM, permanecem como alternativas competitivas para problemas de previsão de séries temporais financeiras, motivando a discussão apresentada na seção seguinte.

2.9 Redes neurais aplicadas à previsão de séries temporais financeiras

As redes neurais artificiais passaram a desempenhar papel de destaque na previsão de séries temporais financeiras, principalmente devido à sua capacidade de modelar relações não lineares e capturar padrões complexos presentes nos dados. Entre as primeiras aplicações nessa área destaca-se o trabalho de Paiva (2014), no qual foi utilizada uma rede *Multilayer Perceptron* (MLP) para prever os valores futuros de 28 ativos da bolsa brasileira. O estudo considerou 16 variáveis de entrada, avaliou 50 diferentes configurações da rede e simulou uma estratégia de investimento com capital inicial de R$ 1.000.000,00 e limite diário de operação de R$ 250.000,00. Como resultado, o modelo obteve retorno acumulado de 70,56%, evidenciando o potencial das redes neurais para aplicações financeiras.

Esse trabalho é relevante para a presente pesquisa por demonstrar que modelos de Inteligência Artificial podem ser avaliados não apenas por métricas estatísticas, mas também por indicadores financeiros obtidos em simulações de investimento. Embora utilize uma

arquitetura MLP, que não possui mecanismos internos de memória temporal, o estudo evidencia a importância de aproximar a avaliação preditiva de cenários reais de negociação, perspectiva também adotada nesta dissertação por meio da utilização de procedimentos de *backtesting*.

Outro estudo de destaque é o desenvolvido por Matsumoto (2019), que comparou o desempenho das técnicas (ARIMA) e *Long Short-Term Memory* (LSTM) na previsão de preços de ações. Utilizando como variáveis de entrada os dados de *open*, *high*, *low*, *close* e *volume*, os autores analisaram ações das empresas *Amazon*, *Ebay*, *Facebook*, *HP* e *Google* no período de janeiro de 2010 a dezembro de 2017. Os resultados indicaram que a LSTM apresentou uma média de acurácia de 70,9%, superando de forma significativa o modelo ARIMA, que obteve uma média de 51,5%.

Os resultados apresentados por Matsumoto (2019) representam um avanço em relação às arquiteturas neurais tradicionais, ao evidenciar que redes recorrentes do tipo LSTM conseguem explorar dependências temporais inexistentes em modelos como o ARIMA. Esse resultado reforça a importância da memória temporal na previsão de séries financeiras e justifica o crescente interesse por arquiteturas recorrentes em problemas de previsão de preços.

De forma mais abrangente, o estudo conduzido por Nabipour et al. (2020) realizou uma comparação entre diferentes modelos de aprendizado de máquina, incluindo *Decision Tree*, *Bagging Classifier*, *Random Forest*, *AdaBoost*, *Gradient Boosting*, *XGBoost*, *Support Vector Classification*, *Naive Bayes*, *KNN*, *Logistic Regression* e redes neurais artificiais, além de dois modelos de aprendizado profundo, RNN e LSTM. Os experimentos foram conduzidos com dados do mercado acionário da bolsa de Teerã, no Irã, abrangendo o período de novembro de 2009 a novembro de 2019 e envolvendo ativos dos setores de petróleo, finanças diversificadas, metais básicos e minerais não metálicos. Para a modelagem, foram utilizados dez indicadores técnicos como variáveis de entrada. Os piores desempenhos foram observados nos modelos *Naive Bayes* e *Decision Tree*, com acurácia em torno de 68%, enquanto os melhores resultados foram alcançados pelas redes RNN e LSTM, ambas com acurácia de aproximadamente 86%. Esses resultados reforçam o elevado potencial das arquiteturas recorrentes, em especial da LSTM, na modelagem de séries temporais financeiras complexas.

Em conjunto, os estudos analisados evidenciam uma evolução das arquiteturas neurais aplicadas à previsão financeira. Enquanto Paiva (2014) demonstra a viabilidade da utilização de redes neurais artificiais em estratégias de investimento, Matsumoto (2019) mostram a superioridade das redes recorrentes em relação aos modelos estatísticos tradicionais, e Nabipour et al. (2020) confirmam esse comportamento ao comparar diferentes algoritmos de aprendizado de máquina e aprendizado profundo. Esses resultados reforçam que arquiteturas recorrentes, especialmente as redes LSTM, constituem alternativas

promissoras para problemas de previsão de séries temporais financeiras.

Dentre as diferentes arquiteturas neurais apresentadas na literatura, as redes LSTM destacam-se por sua capacidade de representar dependências temporais de longo prazo, característica particularmente relevante para séries financeiras. Por esse motivo, a seção seguinte concentra-se especificamente nos trabalhos que empregam arquiteturas LSTM para previsão de séries temporais.

2.10 Modelos baseados em LSTM

Com o avanço do aprendizado profundo, as redes neurais recorrentes, em especial a arquitetura *Long Short-Term Memory* (LSTM), passaram a ocupar posição de destaque na previsão de séries temporais financeiras e de *commodities*. A literatura recente evidencia que essas redes vêm sendo combinadas com diferentes estratégias, como utilização de variáveis exógenas, decomposição de sinais, modelos híbridos e técnicas de *ensemble*, buscando ampliar sua capacidade preditiva em problemas caracterizados por elevada complexidade temporal (ESANGBEDO et al., 2024; YANG et al., 2024; LAHMIRI, 2026).

Nesse contexto, Esangbedo et al. (2024) propõem um modelo baseado em LSTM aplicado à previsão de preços de *commodities* metálicas, incorporando variáveis exógenas e técnicas de *ensemble learning*. Os autores utilizam dados multivariados e demonstram que a combinação de múltiplos modelos melhora significativamente a acurácia, reduzindo métricas como RMSE e MAE. Esse resultado reforça que, em séries financeiras e de *commodities*, o uso de informações externas pode auxiliar a rede neural na identificação de padrões que não seriam capturados apenas pelo histórico da própria variável prevista.

De forma complementar, Yang et al. (2024) utilizam uma abordagem híbrida baseada na decomposição multiescala da série temporal por meio do método MEEMD (*Multivariate Ensemble Empirical Mode Decomposition*), seguida da aplicação de modelos LSTM e MLP. Essa estratégia permite separar a série em componentes de diferentes frequências, reduzindo o ruído e facilitando o aprendizado do modelo. A proposta é relevante porque mostra que o desempenho de redes profundas pode ser ampliado quando a série é previamente tratada e decomposta em estruturas mais simples.

Embora utilizem estratégias distintas, os estudos de Esangbedo et al. (2024) e Yang et al. (2024) apresentam um objetivo comum: aumentar a capacidade preditiva das redes LSTM por meio da incorporação de informações adicionais antes da etapa de treinamento. Enquanto o primeiro utiliza variáveis exógenas e técnicas de *ensemble*, o segundo emprega decomposição multiescala da série temporal, evidenciando duas abordagens distintas para reduzir limitações da modelagem baseada apenas na série original.

Já Avinash et al. (2024) propõem a integração entre Modelos Ocultos de Markov

(HMM) e redes neurais profundas, permitindo identificar estados latentes da série temporal. Esse tipo de abordagem é particularmente relevante para mercados financeiros, nos quais mudanças de regime são frequentes. Ao incorporar estados ocultos como variáveis adicionais, o modelo amplia sua capacidade de reconhecer períodos de alta, baixa ou estabilidade, contribuindo para maior robustez na previsão.

No trabalho de Sarangi et al. (2025), diferentes modelos de aprendizado de máquina são comparados, incluindo LSTM, *Random Forest* e LASSO. Embora o LSTM apresente bom desempenho, os autores não detalham aspectos fundamentais como o tamanho da janela temporal ou o número de *lags* utilizados, o que limita a reprodutibilidade do estudo. Essa limitação é recorrente em parte da literatura e reforça a necessidade de explicitar, em pesquisas futuras, a forma como as entradas temporais são construídas.

Os estudos de Avinash et al. (2024) e Sarangi et al. (2025) evidenciam outra preocupação recorrente na literatura: além do desenvolvimento de modelos cada vez mais sofisticados, torna-se necessário descrever detalhadamente aspectos relacionados à construção das entradas temporais e à configuração experimental. A ausência dessas informações compromete a reprodutibilidade dos resultados e dificulta comparações entre diferentes trabalhos.

No setor energético, Lahmiri (2026) utilizam um modelo híbrido CNN-LSTM-GRU para previsão de preços de petróleo, considerando uma janela temporal de 30 períodos e realizando previsão do tipo *one-step ahead*. Os resultados indicam que a combinação de diferentes arquiteturas melhora a capacidade de captura de padrões complexos, uma vez que a CNN extrai padrões locais, a LSTM captura dependências temporais de longo prazo e a GRU atua como uma estrutura recorrente mais compacta.

Já Cheng et al. (2019) utilizam dois conjuntos de dados distintos: dados horários, totalizando 720 observações, e dados de alta frequência com intervalo de 10 minutos. O modelo utiliza a série horária para modelagem geral e os dados de 10 minutos para previsão *multi-step* de até 30 minutos à frente, utilizando três passos de previsão. Essa abordagem evidencia a importância da granularidade dos dados na modelagem preditiva, pois séries em menor intervalo temporal podem capturar variações mais finas do comportamento do mercado.

Os trabalhos de Lahmiri (2026) e Cheng et al. (2019) mostram que o desempenho das arquiteturas LSTM também depende das características dos dados utilizados. Enquanto o primeiro combina diferentes arquiteturas neurais para explorar padrões espaciais e temporais, o segundo demonstra que diferentes resoluções temporais influenciam diretamente a capacidade preditiva do modelo.

De forma semelhante, Sayed et al. (2024) utilizam uma rede LSTM aplicada à previsão de preços de carbono, considerando um conjunto multivariado de dados, como preço

de abertura, máxima, mínima, volume e variação. Embora o estudo apresente elevada acurácia, não há especificação clara do horizonte de previsão, o que limita a interpretação dos resultados e dificulta comparações diretas com estudos que adotam previsão *one-step ahead* ou *multi-step ahead*.

Yang, Liu e Li (2023) propõem um modelo baseado em *ensemble* multiobjetivo para previsão de petróleo, combinando múltiplos algoritmos de aprendizado. A proposta busca equilibrar viés e variância, utilizando diferentes modelos para aumentar a estabilidade das previsões. Já Liang e Jia (2022) incorporam variáveis exógenas baseadas em dados de busca online, demonstrando que informações de comportamento digital podem contribuir para melhorar a previsão de preços futuros.

De forma geral, observa-se que a literatura recente tem direcionado esforços para aumentar o desempenho das redes LSTM por diferentes caminhos, incluindo modelos híbridos, utilização de variáveis exógenas, decomposição de sinais, técnicas de *ensemble* e integração com outras arquiteturas de aprendizado profundo. Entretanto, essas estratégias normalmente são acompanhadas por aumento da complexidade computacional e da quantidade de hiperparâmetros que precisam ser ajustados durante o treinamento (LAHMIRI, 2026; YANG; LIU; LI, 2023; LIANG; JIA, 2022).

Os trabalhos analisados evidenciam que o desempenho de modelos baseados em LSTM depende de diversos fatores, entre eles a qualidade dos dados de entrada, a utilização de variáveis exógenas, a definição da janela temporal, o horizonte de previsão adotado e a estratégia empregada para configuração dos hiperparâmetros (ESANGBEDO et al., 2024; YANG et al., 2024; LAHMIRI, 2026; SAYED et al., 2024). Esses aspectos são particularmente relevantes para esta dissertação, uma vez que a proposta consiste em investigar a otimização automática dos hiperparâmetros de redes LSTM por meio de um Algoritmo Genético Transgênico aplicado à previsão dos preços de fechamento do café arábica no mercado financeiro.

2.11 Previsão de preços de *commodities* agrícolas e café arábica

No contexto específico das *commodities* agrícolas, diferentes abordagens têm sido propostas para lidar com a volatilidade, a sazonalidade e a não linearidade das séries temporais. No caso do café arábica, essa discussão torna-se ainda mais relevante devido à importância econômica dessa *commodity* e à influência simultânea de fatores produtivos, climáticos e financeiros sobre a formação de seus preços. Como consequência, a literatura apresenta uma ampla diversidade de estratégias de modelagem, abrangendo desde modelos estatísticos tradicionais até arquiteturas híbridas baseadas em aprendizado profundo (JIN; XU, 2024; NAVEENA et al., 2017; WANG et al., 2021).

Jin e Xu (2024) propõem um modelo baseado em *Gaussian Process Regression* (GPR) com hiperparâmetros ajustados por otimização bayesiana. O estudo utiliza uma série temporal univariada composta exclusivamente por preços diários do café no período entre 2013 e 2024. Embora o artigo não especifique explicitamente o tamanho da janela temporal utilizada, a estrutura metodológica caracteriza um problema do tipo *one-step ahead*. Os resultados mostram elevada precisão preditiva, com erro relativo quadrático médio inferior a 3%, indicando forte aderência entre valores previstos e observados.

Yashavanth et al. (2017) utilizam modelos VAR para analisar relações entre diferentes mercados de café. Diferentemente de abordagens univariadas, o modelo considera múltiplas variáveis simultaneamente, permitindo capturar interdependências entre mercados regionais. Essa perspectiva contribui para mostrar que os preços do café podem ser influenciados por dinâmicas cruzadas entre mercados distintos.

De forma complementar, Naveena et al. (2017) propõem um modelo híbrido combinando ARIMA e redes neurais artificiais. O ARIMA é utilizado para capturar componentes lineares da série, enquanto a rede neural modela os padrões não lineares restantes. Os autores mostram que a combinação das duas abordagens apresenta desempenho superior aos modelos isolados, reforçando a importância de estratégias híbridas em séries financeiras complexas.

Os estudos de Jin e Xu (2024), Yashavanth et al. (2017) e Naveena et al. (2017) ilustram três perspectivas distintas para a previsão de preços do café. Enquanto o primeiro emprega um modelo não linear com otimização bayesiana em uma abordagem univariada, o segundo considera relações entre diferentes mercados por meio de um modelo VAR, e o terceiro combina modelos estatísticos e redes neurais em uma abordagem híbrida. Esses resultados evidenciam que não existe uma única estratégia dominante para esse problema.

Um estudo particularmente relevante é apresentado por Wang et al. (2021), que utilizam o modelo *Nonlinear Grey Bernoulli Model* (NGBM (1,1)) combinado com séries de Fourier para previsão de preços do café no Vietnã. O aspecto mais interessante desse trabalho está no fato de os autores utilizarem apenas seis observações anuais como conjunto de treinamento. A escolha do modelo Grey é justificada justamente pela reduzida quantidade de dados disponíveis, uma vez que esse tipo de modelo é adequado para pequenas amostras e sistemas com informação incompleta. O modelo realiza previsão *multi-step*, gerando projeções anuais para o período entre 2020 e 2025. Os resultados indicam desempenho superior do modelo híbrido NGBM + Fourier em comparação com versões tradicionais do modelo Grey.

Os trabalhos de Wang et al. (2021) e Xu e Zhang (2022) mostram que diferentes estruturas de modelagem podem ser utilizadas mesmo quando o objetivo permanece o mesmo: prever preços de *commodities* agrícolas. Enquanto o primeiro desenvolve um modelo voltado para pequenas amostras de dados, o segundo utiliza redes neurais autoregressivas

para explorar dependências temporais presentes nas séries históricas.

Já Xu e Zhang (2022) utilizam redes neurais autoregressivas não lineares para previsão de *commodities* agrícolas, explorando relações temporais complexas presentes nas séries históricas. A estrutura autoregressiva permite que valores passados da própria série sejam utilizados como entrada para prever valores futuros, aproximando-se da lógica de construção de janelas temporais em modelos supervisionados.

De forma semelhante, Gu et al. (2022) propõem um modelo LSTM com entrada dual (*dual input*), combinando a série principal de preços com variáveis exógenas adicionais. Essa abordagem permite incorporar múltiplas fontes de informação ao processo de previsão, ampliando a capacidade do modelo de capturar fatores externos que influenciam os preços agrícolas.

Murugesan, Mishra e Krishnan (2022) comparam diferentes arquiteturas baseadas em LSTM aplicadas a séries agrícolas, demonstrando que pequenas alterações estruturais na rede podem impactar significativamente o desempenho preditivo. Esse tipo de estudo reforça a importância de avaliar não apenas o uso da LSTM em si, mas também suas variações arquiteturais, número de camadas, unidades ocultas e forma de organização das entradas.

Os estudos de Gu et al. (2022) e Murugesan, Mishra e Krishnan (2022) direcionam a atenção para aspectos internos da arquitetura das redes neurais. Enquanto o primeiro amplia a quantidade de informações disponíveis por meio da utilização de entradas duplas, o segundo evidencia que pequenas alterações estruturais nas redes LSTM podem produzir diferenças significativas no desempenho preditivo.

Além disso, Taghipour, Rezaee e Hajati (2025) utilizam modelos híbridos para previsão de *commodities* financeiras, reforçando a importância da combinação de técnicas estatísticas, aprendizado profundo e otimização. Embora o estudo não seja especificamente sobre café, sua contribuição está em evidenciar que *commodities* financeiras altamente voláteis podem se beneficiar de modelos híbridos multiescala.

De maneira geral, os trabalhos analisados evidenciam que ainda não existe consenso na literatura sobre a melhor estratégia para previsão dos preços do café arábica. As diferenças observadas envolvem a escolha entre modelos univariados e multivariados, a utilização de variáveis exógenas, o horizonte de previsão adotado, a arquitetura dos modelos e a estratégia de configuração dos hiperparâmetros (JIN; XU, 2024; GU et al., 2022; MURUGESAN; MISHRA; KRISHNAN, 2022; TAGHIPOUR; REZAEE; HAJATI, 2025). Essa diversidade metodológica reforça a importância de investigar novas estratégias de otimização capazes de selecionar automaticamente configurações mais adequadas para redes neurais LSTM, objetivo central desta dissertação.

2.12 Otimização de hiperparâmetros

Embora as redes LSTM apresentem elevado potencial na modelagem de séries temporais, seu desempenho depende fortemente da configuração adequada de seus hiperparâmetros. Entre esses hiperparâmetros destacam-se o número de unidades ocultas, o tamanho da janela temporal, a taxa de aprendizado, o número de épocas, o tamanho do lote e a própria arquitetura da rede. A definição inadequada desses elementos pode comprometer a capacidade de generalização do modelo, motivando o desenvolvimento de diferentes estratégias automáticas de otimização, especialmente aquelas baseadas em algoritmos evolutivos (SARI et al., 2024; MANOGNA; DHARMAJI; SARANG, 2025; PENG et al., 2018).

Nesse contexto, algoritmos evolutivos têm sido amplamente utilizados para automatizar o processo de otimização. Sari et al. (2024) demonstram que AGs podem otimizar automaticamente o número de *lags* e a arquitetura da rede, reduzindo a necessidade de ajuste manual e permitindo que o modelo encontre combinações mais adequadas ao comportamento da série.

Já Manogna, Dharmaji e Sarang (2025) investigam o impacto do tamanho da janela temporal no desempenho preditivo, utilizando janelas com até 1000 *lags*. Os autores mostram que a quantidade de memória histórica influencia significativamente os resultados obtidos. Entretanto, janelas muito longas podem aumentar o custo computacional e introduzir ruído, o que torna necessária uma escolha criteriosa.

Hwase e Fofanah (2021) utilizam dados diários de mercado financeiro e demonstram superioridade das arquiteturas LSTM em relação a modelos tradicionais. Embora o estudo reforce a capacidade da LSTM em capturar dependências temporais, a ausência de detalhamento sobre a janela utilizada limita comparações diretas com outros trabalhos.

Peng et al. (2018) utilizam evolução diferencial para otimização automática da janela temporal. Essa abordagem é relevante porque trata a janela de entrada como hiperparâmetro a ser aprendido, em vez de defini-la manualmente. De forma complementar, (BENIWAL; SINGH; KUMAR, 2023) demonstram que estratégias de otimização melhoram a capacidade de generalização da rede, especialmente em previsões de longo prazo.

Os estudos de Sari et al. (2024), Manogna, Dharmaji e Sarang (2025), Peng et al. (2018) e Beniwal, Singh e Kumar (2023) mostram que diferentes hiperparâmetros exercem influência direta sobre o desempenho das redes LSTM. Enquanto alguns autores concentram a otimização na janela temporal, outros investigam simultaneamente aspectos estruturais da rede, evidenciando que não existe um único hiperparâmetro dominante para todos os problemas de previsão.

No trabalho de Sha (2024), um modelo híbrido GA–LSTM é utilizado para previsão de preços de ações. O algoritmo genético é empregado na otimização de hiperparâmetros da

rede LSTM, buscando melhorar o desempenho preditivo. Os resultados mostram redução progressiva do erro absoluto médio ao longo do treinamento. Entretanto, o estudo não especifica claramente os operadores genéticos utilizados nem o processo detalhado de otimização, o que limita a reprodutibilidade.

Além disso, Sun et al. (2024) apresentam uma abordagem integrada que combina AGs, redes MLP e procedimentos de *backtesting* financeiro. Diferentemente de estudos focados apenas em métricas estatísticas, os autores incorporam indicadores financeiros, como *Sharpe Ratio*, *Sortino Ratio*, *Calmar Ratio* e *Alpha*, diretamente na função *fitness* do AG. Esse aspecto aproxima significativamente o estudo da proposta desta dissertação, uma vez que o presente trabalho também pretende avaliar não apenas métricas estatísticas, mas o desempenho financeiro das previsões realizadas.

Chung e Shin (2018) utilizam AGs para otimizar simultaneamente o tamanho da janela temporal e o número de unidades ocultas da rede LSTM aplicada ao índice KOSPI. Os hiperparâmetros são codificados em vetores binários, permitindo busca automática no espaço de soluções. Essa abordagem é importante porque demonstra que a janela temporal pode ser otimizada em conjunto com a arquitetura da rede, e não definida de forma arbitrária.

Já Chen e Zhou (2021) utilizam AGs para seleção automática de características, identificando subconjuntos ótimos de variáveis técnicas e financeiras. Essa estratégia contribui para reduzir dimensionalidade e eliminar variáveis pouco informativas. De forma semelhante, Pang e Sun (2025) propõem um modelo híbrido GA–BP, no qual o algoritmo genético é utilizado para otimização global dos pesos e termos de *bias* da rede neural.

No contexto brasileiro, Cruvinel (2024) aplicam um modelo GA–LSTM para previsão dos preços das ações da Petrobras. O algoritmo genético é utilizado para otimizar hiperparâmetros como número de unidades ocultas, taxa de aprendizado e número de épocas. Os resultados mostram melhoria significativa em relação à LSTM tradicional, evidenciando o potencial da combinação entre redes recorrentes e algoritmos evolutivos em dados do mercado brasileiro.

Os trabalhos de Sha (2024), Sun et al. (2024), Chung e Shin (2018), Chen e Zhou (2021), Pang e Sun (2025) e Cruvinel (2024) evidenciam a ampla utilização de AGs na otimização de modelos preditivos. Entretanto, observa-se que as estratégias empregadas concentram-se predominantemente em operadores evolutivos tradicionais, como seleção, cruzamento e mutação, sem explorar operadores alternativos capazes de modificar a dinâmica da busca evolutiva.

Além dos AGs, outras estratégias evolutivas também têm sido exploradas. Shao et al. (2019) utilizam *Particle Swarm Optimization* (PSO) para otimizar hiperparâmetros da rede LSTM aplicada à previsão de preços do níquel. O estudo utiliza preços médios

mensais de fechamento negociados na London Metal Exchange entre 2006 e 2018. O PSO é utilizado para otimizar parâmetros como número de neurônios e tamanho da janela temporal, definida em 15 períodos.

Já Choudhary et al. (2025) utilizam decomposição *Variational Mode Decomposition* (VMD) combinada com AGs. Inicialmente, a série temporal é decomposta em diferentes componentes de frequência por meio do VMD. Em seguida, o algoritmo genético otimiza os parâmetros do modelo preditivo aplicado a cada componente. Segundo os autores, essa abordagem reduz ruído e melhora significativamente a qualidade da previsão.

Bu et al. (2025) investigam o impacto do tamanho da janela temporal na previsão de preços da soja, realizando testes com janelas variando entre 5 e 120 períodos. Os resultados mostram que janelas curtas podem apresentar melhor desempenho dependendo das características da série analisada. Esse resultado é importante para esta dissertação porque reforça que a definição da janela temporal não deve ser tratada como escolha fixa e universal.

Mais recentemente, Wu et al. (2025) ampliam essa linha de pesquisa ao integrar redes LSTM otimizadas por AGs com análise textual baseada em modelos BERT. O estudo incorpora notícias financeiras e análise de sentimento ao modelo preditivo, demonstrando que informações textuais podem melhorar significativamente a previsão de preços.

Além dos AGs, a literatura recente também investiga diferentes estratégias de otimização, incluindo métodos baseados em enxames de partículas, decomposição de sinais e integração com modelos de linguagem. Esses trabalhos demonstram que a otimização automática dos hiperparâmetros permanece como um tema ativo de pesquisa, refletindo a dificuldade de definir configurações universais para diferentes séries temporais (SHAO et al., 2019; CHOUDHARY et al., 2025; BU et al., 2025; WU et al., 2025).

De maneira geral, os trabalhos analisados evidenciam que a otimização automática de hiperparâmetros constitui um dos principais fatores para melhoria do desempenho de redes LSTM aplicadas à previsão de séries temporais financeiras. Entretanto, observa-se que a quase totalidade dos estudos utiliza operadores evolutivos clássicos, baseados em mecanismos tradicionais de seleção, cruzamento e mutação (SHA, 2024; CHUNG; SHIN, 2018; CRUVINEL, 2024). Nenhum dos trabalhos analisados investiga a utilização do operador genético transgênico proposto por Amaral e Jr (2014) como mecanismo de otimização de hiperparâmetros para redes LSTM. Essa lacuna constitui uma das principais motivações desta dissertação, que propõe adaptar esse operador ao problema de otimização de arquiteturas neurais aplicadas à previsão dos preços de fechamento do café arábica no mercado financeiro.

2.13 Lacunas identificadas na literatura e proposta da dissertação

A revisão da literatura realizada neste capítulo evidencia avanços significativos na aplicação de técnicas de Inteligência Artificial à previsão de séries temporais financeiras e de *commodities*. Entretanto, também permite identificar limitações recorrentes relacionadas à modelagem dos dados, à configuração dos modelos e à avaliação dos resultados. Essas limitações dificultam tanto a comparação entre diferentes abordagens quanto a reprodução dos experimentos apresentados na literatura.

Observa-se também que grande parte das pesquisas limita-se à avaliação por métricas estatísticas tradicionais, como RMSE, MAE e MAPE, sem considerar o impacto financeiro das previsões em cenários reais de negociação. Embora alguns estudos, como Paiva (2014) e Sun et al. (2024), avancem nesse sentido ao incorporar simulações financeiras ou métricas de desempenho econômico, essa ainda não é uma prática consolidada na literatura sobre previsão de *commodities* agrícolas e café arábica.

Outra lacuna identificada está relacionada aos algoritmos evolutivos utilizados. A maior parte dos estudos emprega operadores genéticos clássicos, havendo pouca exploração de estratégias mais robustas de diversidade genética. Nesse contexto, observa-se que nenhum dos trabalhos analisados emprega o operador genético transgênico proposto por Amaral e Jr (2014) para otimização de hiperparâmetros de redes neurais LSTM aplicadas à previsão de preços de *commodities* financeiras. Dessa forma, permanece aberta a possibilidade de investigar o potencial desse operador como estratégia alternativa aos mecanismos evolutivos tradicionais.

Diante desse panorama, observa-se que a previsão de preços de fechamento do café arábica no mercado financeiro ainda apresenta desafios relevantes, especialmente no que se refere à escolha e à otimização dos hiperparâmetros de modelos baseados em redes neurais do tipo LSTM. Embora a literatura evidencie avanços no uso de AGs para esse fim, persiste uma lacuna no desenvolvimento de estratégias de otimização mais robustas, particularmente no que diz respeito à utilização de operadores genéticos avançados. Ademais, observa-se que grande parte dos estudos limita-se à avaliação do desempenho por meio de métricas estatísticas, como RMSE, MAE e MAPE, sem investigar de forma aprofundada os impactos práticos dessas previsões em contextos de tomada de decisão financeira.

Nesse sentido, esta dissertação propõe um modelo baseado em redes neurais LSTM otimizadas por Algoritmo Genético Transgênico para previsão dos preços de fechamento do café arábica no mercado financeiro. A proposta busca investigar se a utilização do operador transgênico pode produzir configurações de hiperparâmetros capazes de melhorar o desempenho preditivo em relação às abordagens tradicionais baseadas em LSTM e

LSTM otimizada por Algoritmo Genético clássico.

Além da avaliação por métricas estatísticas, o trabalho também incorpora procedimentos de *backtesting*, permitindo analisar o desempenho financeiro das previsões geradas pelos diferentes modelos. Dessa forma, serão comparados os modelos LSTM clássico, LSTM otimizado por Algoritmo Genético (AG) e LSTM otimizado por Algoritmo Genético Transgênico (AGT), buscando avaliar tanto sua capacidade preditiva quanto seu potencial de aplicação em estratégias de negociação no mercado financeiro.

Assim, as lacunas identificadas ao longo desta revisão fundamentam a proposta metodológica desenvolvida nesta dissertação, apresentada no capítulo seguinte, no qual são descritos os procedimentos experimentais, os conjuntos de dados utilizados, a adaptação do operador genético transgênico para otimização de hiperparâmetros e o protocolo de avaliação estatística e financeira empregado.

Capítulo 3

Método

Este capítulo descreve os procedimentos metodológicos e computacionais adotados para comparar três abordagens de previsão do preço de fechamento do café arábica: Foram avaliadas três abordagens: a LSTM clássica, configurada com hiperparâmetros fixos, a LSTM com hiperparâmetros selecionados por Algoritmo Genético (LSTM-AG) e a LSTM com hiperparâmetros selecionados por Algoritmo Genético Transgênico (LSTM-AGT). Para os modelos LSTM-AG e LSTM-AGT, a seleção dos hiperparâmetros foi realizada automaticamente pelos algoritmos evolutivos a partir de um espaço de busca previamente definido. Os intervalos e valores admissíveis para cada hiperparâmetro foram estabelecidos com base nas configurações identificadas nos trabalhos analisados na literatura e nas limitações dos recursos computacionais disponíveis para a execução dos experimentos.

Inicialmente, são apresentados o delineamento da pesquisa, a fonte dos dados e as três granularidades temporais consideradas. Em seguida, são descritos o pré-processamento, a divisão cronológica dos conjuntos de dados, a normalização e a construção das janelas temporais utilizadas como entrada das redes.

Posteriormente, são detalhadas a arquitetura de referência da LSTM clássica, a representação cromossômica adotada nos algoritmos evolutivos, a função de aptidão, os operadores do AG convencional e as adaptações realizadas para incorporar o operador Transgênico à otimização dos hiperparâmetros. Também são apresentados os registros produzidos durante a busca evolutiva, o procedimento de treinamento final e as métricas empregadas na avaliação preditiva.

Por fim, descreve-se o protocolo de *backtesting* utilizado para transformar as previsões dos modelos em sinais de negociação e avaliar o comportamento financeiro das nove configurações resultantes da combinação entre os três modelos e as três granularidades temporais.

3.1 Delineamento da Pesquisa

Esta pesquisa caracteriza-se como aplicada, de abordagem quantitativa e desenvolvida por meio de experimentação computacional. Seu caráter aplicado decorre da utilização de técnicas de aprendizado profundo e computação evolutiva na construção de modelos destinados à previsão do preço de fechamento do contrato futuro do café arábica.

A investigação possui natureza quantitativa porque o desempenho das abordagens é examinado por meio de métricas numéricas de erro, medidas de capacidade explicativa, registros de custo computacional e indicadores financeiros obtidos por meio de *backtesting*. A dimensão experimental está relacionada à implementação e à comparação controlada de três estratégias de modelagem submetidas aos mesmos dados, procedimentos de pré-processamento e protocolo de avaliação.

As três abordagens comparadas são:

1. LSTM clássica, com hiperparâmetros definidos previamente;
2. LSTM com hiperparâmetros selecionados por Algoritmo Genético convencional (LSTM-AG);
3. LSTM com hiperparâmetros selecionados por Algoritmo Genético Transgênico (LSTM-AGT).

Após a definição dos hiperparâmetros de cada abordagem, os modelos são submetidos ao treinamento final, à avaliação preditiva fora da amostra e ao *backtesting* financeiro. O desenho experimental procura manter constantes os demais procedimentos, de modo que as diferenças observadas possam ser analisadas principalmente em função da estratégia adotada para a definição dos hiperparâmetros.

3.2 Base de Dados e Aquisição

Os dados utilizados nesta pesquisa correspondem às cotações históricas do contrato futuro do café arábica, identificado pelo código `KC=F`. A aquisição foi realizada diretamente da plataforma Yahoo Finance por meio da biblioteca *yfinance*, utilizada nos códigos para solicitar e organizar as séries temporais do ativo.

Foram consideradas três granularidades temporais:

- ❏ dados intradiários com intervalo de quatro horas (`4h`);
- ❏ dados diários (`1d`);
- ❏ dados semanais (`1wk`).

Cada granularidade foi tratada como um conjunto de dados independente. Essa decisão permite analisar o comportamento dos modelos em séries com diferentes frequências de observação, sem combinar registros intradiários, diários e semanais em uma única base.

Nos códigos destinados à busca de hiperparâmetros e à avaliação inicial dos modelos, a série de quatro horas foi obtida utilizando uma janela relativa de 500 dias. Para os intervalos diário e semanal, foram solicitados dados entre 1º de janeiro de 2000 e 31 de dezembro de 2020. Essas configurações são definidas diretamente pelas variáveis `PERIOD`, `START_DATE`, `END_DATE` e `INTERVAL` presentes nos códigos experimentais.

Em todas as três granularidades, somente o preço de fechamento (*Close*) foi utilizado como variável de entrada e como base para a construção do valor-alvo. Assim, a modelagem preditiva é univariada: os 30 preços de fechamento anteriores são empregados para estimar o preço de fechamento imediatamente seguinte. Os preços de abertura, máxima e mínima são utilizados posteriormente no procedimento de *backtesting*, e não como entradas das redes durante a busca de hiperparâmetros.

Após a aquisição, os registros com valores ausentes foram removidos por meio da operação `dropna()`, e o índice da série foi convertido para o formato de data e hora. Os dados foram mantidos em ordem cronológica durante todas as etapas do experimento.

A Tabela 2 resume as configurações empregadas na etapa de busca e avaliação inicial.

Tabela 2 – Configuração dos conjuntos de dados utilizados na etapa de busca de hiperparâmetros.

Granularidade	**Parâmetro de coleta**	**Variável de entrada**	**Alvo**
4 horas	Últimos 500 dias	*Close*	Próximo *Close*
Diária	01/01/2000 a 31/12/2020	*Close*	Próximo *Close*
Semanal	01/01/2000 a 31/12/2020	*Close*	Próximo *Close*

3.3 Pré-processamento e Normalização

O pré-processamento foi executado de forma equivalente nas três abordagens avaliadas. Inicialmente, a série de preços de fechamento foi convertida em amostras supervisionadas, formadas por janelas temporais de entrada e pelo respectivo preço-alvo. Em seguida, as amostras foram divididas cronologicamente em conjuntos de treinamento e teste.

A normalização foi realizada por meio do *MinMaxScaler*. Para evitar vazamento de informação, o escalonador foi ajustado exclusivamente com os valores pertencentes ao conjunto de treinamento. No ajuste foram considerados tanto os valores presentes nas janelas de entrada quanto os respectivos valores-alvo de treinamento.

Após esse ajuste, o mesmo escalonador foi utilizado para transformar:

- ❏ as janelas de entrada do conjunto de treinamento;
- ❏ as janelas de entrada do conjunto de teste;
- ❏ os valores-alvo de treinamento;
- ❏ os valores-alvo de teste.

Esse procedimento impede que os valores mínimos e máximos do período de teste participem da definição da escala utilizada durante o treinamento. Os dados normalizados foram mapeados para o intervalo $[0, 1]$, e o escalonador ajustado foi preservado para a posterior desnormalização das previsões e utilização no treinamento final e no *backtesting*.

3.4 Construção das Janelas Temporais

A série histórica foi transformada em um problema de aprendizado supervisionado por meio de janelas temporais deslizantes. Em todos os experimentos foi adotado o parâmetro `TIME_STEP = 30`. Assim, cada amostra de entrada contém 30 preços de fechamento consecutivos, e o valor-alvo corresponde ao fechamento imediatamente posterior.

Considerando uma série temporal de preços de fechamento

$$\{x_1, x_2, \ldots, x_T\}, \tag{5}$$

a amostra de entrada associada ao instante t é definida por:

$$X_t = [x_{t-29}, x_{t-28}, \ldots, x_t], \tag{6}$$

enquanto o valor-alvo é dado por:

$$y_t = x_{t+1}. \tag{7}$$

O valor 30 representa uma quantidade fixa de observações, e não necessariamente 30 dias. Desse modo, as janelas correspondem a:

- ❏ 30 registros de quatro horas no conjunto intradiário;
- ❏ 30 registros diários no conjunto diário;
- ❏ 30 registros semanais no conjunto semanal.

A manutenção da mesma quantidade de observações nas três granularidades permite aplicar um procedimento uniforme de construção das entradas, embora cada janela represente extensões cronológicas distintas.

Após a normalização, as entradas foram reorganizadas no formato tridimensional exigido pelas camadas LSTM:

$$(\text{amostras}, \text{passos temporais}, \text{atributos}). \tag{8}$$

Como a modelagem utiliza apenas o preço de fechamento, a última dimensão possui tamanho igual a 1. Portanto, o formato de entrada de cada conjunto é:

$$(\text{número de amostras}, 30, 1). \tag{9}$$

3.5 Divisão Temporal dos Dados

As amostras foram divididas cronologicamente, sem embaralhamento, preservando a ordem temporal da série. O ponto de corte foi definido pela proporção de 80% para treinamento e 20% para teste. As primeiras 80% das amostras foram destinadas ao desenvolvimento dos modelos, enquanto as 20% finais foram mantidas para avaliação fora da amostra.

A divisão foi aplicada após a construção das janelas temporais. Dessa forma, tanto as entradas quanto os respectivos valores-alvo foram separados pelo mesmo índice cronológico:

$$n_{\text{treino}} = \lfloor 0{,}8n \rfloor , \tag{10}$$

em que n representa a quantidade total de amostras supervisionadas geradas a partir da série.

O conjunto de teste não foi utilizado para selecionar os hiperparâmetros dos modelos. Na LSTM clássica, os hiperparâmetros foram previamente fixados, e o modelo foi treinado com o conjunto destinado ao treinamento. Nos modelos LSTM-AG e LSTM-AGT, a avaliação dos cromossomos foi realizada somente sobre o conjunto de treinamento, utilizando validação temporal interna.

Para essa validação interna foi empregado o método *TimeSeriesSplit*, configurado com duas divisões temporais. Em cada divisão, o modelo é ajustado com observações anteriores e validado em um bloco cronologicamente posterior. O MSE obtido nas duas divisões é posteriormente utilizado para calcular o valor médio de aptidão do indivíduo.

O conjunto correspondente às 20% finais permaneceu separado durante a busca evolutiva. Esse procedimento permite avaliar os modelos em observações posteriores às utilizadas no ajuste dos hiperparâmetros, reduzindo o risco de incorporar informações futuras ao processo de seleção.

3.6 Arquitetura Geral dos Modelos LSTM

As três abordagens avaliadas nesta dissertação utilizam redes neurais recorrentes do tipo *Long Short-Term Memory* (LSTM) como modelo preditivo. A implementação foi realizada por meio da biblioteca TensorFlow/Keras, utilizando o modelo sequencial (*Sequential*) para a organização das camadas da rede.

Em todas as abordagens, a entrada é constituída por janelas temporais contendo 30 preços de fechamento consecutivos. Como a modelagem utiliza apenas a variável *Close*, cada amostra possui um único atributo, sendo organizada no formato tridimensional

$$(\text{número de amostras}, 30, 1). \tag{11}$$

A saída corresponde à previsão do preço de fechamento imediatamente posterior à janela de entrada. Para isso, a última camada da rede é composta por um único neurônio totalmente conectado (*Dense*), responsável por produzir um valor contínuo.

Nos modelos LSTM-AG e LSTM-AGT, a quantidade de camadas LSTM, o número de neurônios por camada e a taxa de aprendizado variam conforme os valores codificados em cada cromossomo. Quando a arquitetura contém mais de uma camada recorrente, as camadas anteriores à última são configuradas para retornar a sequência completa por meio do parâmetro `return_sequences=True`, permitindo que suas saídas sejam utilizadas como entrada pela camada LSTM seguinte. A última camada recorrente retorna apenas o estado oculto final, que é encaminhado para a camada de saída.

A taxa de *dropout* foi mantida fixa em 0,2 nas abordagens avaliadas. Esse hiperparâmetro não integra o cromossomo utilizado pelos algoritmos evolutivos. Nos modelos construídos durante a busca com AG e AGT, camadas de *dropout* são adicionadas após as camadas LSTM intermediárias quando a arquitetura possui mais de duas camadas. No treinamento final, a camada de *dropout* é aplicada após o conjunto de camadas recorrentes, conforme a função utilizada para reconstruir a arquitetura selecionada.

Todos os modelos são compilados utilizando o otimizador Adam. Na LSTM clássica, a taxa de aprendizado é previamente definida, enquanto nas abordagens LSTM-AG e LSTM-AGT esse valor é determinado pelo cromossomo avaliado. O erro quadrático médio (*Mean Squared Error* – MSE) é utilizado como função de perda durante o treinamento das redes.

Apesar de compartilharem a mesma finalidade preditiva e o mesmo formato de entrada e saída, as três abordagens diferem quanto ao procedimento empregado para definir os hiperparâmetros. Na LSTM clássica, esses valores são fixados previamente. Na LSTM-AG, são selecionados por meio de um Algoritmo Genético convencional. Na LSTM-AGT, a busca evolutiva incorpora a adaptação do operador Transgênico fundamentado na proposta de Amaral e Jr (2014).

3.6.1 LSTM Clássica

A LSTM clássica foi utilizada como modelo de referência para a comparação com as abordagens evolutivas. Nessa configuração, os hiperparâmetros foram definidos previamente e permaneceram fixos durante todos os experimentos, sem a utilização de mecanismos automáticos de busca ou otimização.

A arquitetura implementada é composta por duas camadas LSTM empilhadas. A primeira camada possui 50 neurônios e utiliza o parâmetro `return_sequences=True`, pois sua saída completa é encaminhada para a segunda camada recorrente. A segunda camada também possui 50 neurônios, mas utiliza `return_sequences=False`, retornando apenas o estado oculto final da sequência.

Após as camadas recorrentes, foi adicionada uma camada de *dropout* com taxa de 0,2. Em seguida, a rede possui uma camada totalmente conectada com 25 neurônios e uma camada de saída *Dense* composta por um único neurônio, responsável por produzir a previsão do preço de fechamento imediatamente posterior à janela temporal de entrada.

A rede foi compilada utilizando o otimizador Adam, com taxa de aprendizado igual a 0,001, e o erro quadrático médio (*Mean Squared Error* – MSE) como função de perda. O treinamento foi realizado durante 50 épocas, utilizando lotes de 32 amostras.

Tabela 3 – Configuração da LSTM clássica utilizada como modelo de referência.

Componente ou hiperparâmetro	**Configuração**
Número de camadas LSTM	2
Neurônios em cada camada LSTM	50
Retorno da primeira camada LSTM	Sequência completa
Retorno da segunda camada LSTM	Estado oculto final
Taxa de *dropout*	0,2
Camada densa intermediária	25 neurônios
Camada de saída	1 neurônio
Taxa de aprendizado	0,001
Tamanho do lote	32
Número de épocas	50
Otimizador	Adam
Função de perda	MSE

O treinamento foi realizado com o conjunto destinado ao aprendizado, sem utilização de embaralhamento temporal na preparação das amostras. Após o ajuste dos pesos, o modelo produziu previsões para os conjuntos de treinamento e teste. As previsões foram posteriormente desnormalizadas para a escala original dos preços, permitindo o cálculo das métricas preditivas descritas no Capítulo 2.

A adoção dessa configuração fixa permite utilizar a LSTM clássica como referência experimental. Assim, seus resultados podem ser comparados com aqueles obtidos pelas redes cujos hiperparâmetros foram selecionados pelo Algoritmo Genético convencional e pelo Algoritmo Genético Transgênico.

3.6.2 LSTM Otimizada por AG (LSTM-GA)

Nesta seção é apresentado o Algoritmo Genético (AG) desenvolvido para automatizar a seleção dos hiperparâmetros da rede LSTM. Diferentemente da abordagem clássica, na qual os hiperparâmetros permanecem fixos durante todo o treinamento, o AG realiza uma busca evolutiva por diferentes configurações da arquitetura, avaliando sucessivamente soluções candidatas até identificar aquela que apresenta o melhor desempenho preditivo.

Cada indivíduo da população representa uma configuração completa da rede LSTM. Os genes codificam diretamente os hiperparâmetros utilizados na construção da arquitetura, de modo que cada cromossomo corresponde a uma rede neural distinta. Durante a execução do algoritmo, cada indivíduo é convertido em uma arquitetura LSTM, treinado utilizando o conjunto de treinamento e avaliado por validação temporal.

O desempenho obtido por cada arquitetura é utilizado como função de aptidão do indivíduo. Com base nessa avaliação, são aplicados os operadores evolutivos de seleção, cruzamento, mutação e elitismo, produzindo sucessivas gerações de soluções. Ao final do processo evolutivo, é selecionado o indivíduo que apresentou o menor erro médio durante toda a execução do algoritmo. Essa configuração é utilizada posteriormente para reconstruir a rede LSTM e realizar o treinamento final do modelo.

3.6.3 Representação Cromossômica

Cada indivíduo do Algoritmo Genético representa uma configuração completa da rede LSTM por meio de um cromossomo composto por cinco genes. Cada gene codifica diretamente um hiperparâmetro utilizado durante a construção da arquitetura da rede neural, permitindo que diferentes combinações de valores sejam avaliadas ao longo do processo evolutivo.

A Tabela 4 apresenta os genes utilizados, os respectivos hiperparâmetros representados e os intervalos considerados durante a busca evolutiva.

Tabela 4 – Genes utilizados na representação cromossômica do Algoritmo Genético.

Gene	Hiperparâmetro	Intervalo
1	Número de camadas LSTM	1 a 4
2	Número de neurônios	10 a 100
3	Taxa de aprendizado	0,0001 a 0,01
4	*Batch size*	16 a 128
5	Número de épocas	10 a 100

Os valores dos genes são gerados respeitando os limites estabelecidos para cada hiperparâmetro. Durante a execução do algoritmo, cada cromossomo é convertido automaticamente em uma arquitetura LSTM. O número de camadas recorrentes, a quantidade de neurônios em cada camada, a taxa de aprendizado do otimizador Adam, o tamanho do lote utilizado no treinamento e o número de épocas são definidos diretamente pelos valores armazenados no indivíduo.

Essa estratégia permite que cada cromossomo representa uma configuração distinta de hiperparâmetros, a partir da qual é construída automaticamente uma arquitetura LSTM, possibilitando ao AG explorar diferentes combinações de hiperparâmetros de forma automática ao longo do processo evolutivo.

3.6.4 Inicialização da População

Após a definição da representação cromossômica, é gerada a população inicial do Algoritmo Genético. Cada indivíduo é criado de forma aleatória, respeitando os limites estabelecidos para os cinco genes apresentados na Tabela 4. Dessa forma, a população inicial contém arquiteturas LSTM distintas, permitindo que diferentes regiões do espaço de busca sejam exploradas desde o início do processo evolutivo.

Neste trabalho, a população é composta por 30 indivíduos. Cada cromossomo é gerado independentemente dos demais, não sendo utilizada qualquer estratégia de inicialização heurística ou conhecimento prévio sobre arquiteturas consideradas promissoras. Essa abordagem favorece a diversidade genética inicial, característica importante para reduzir o risco de convergência prematura do algoritmo.

O processo evolutivo é executado durante cinco gerações. Em cada geração, todos os indivíduos são convertidos em arquiteturas LSTM, treinados e avaliados utilizando validação temporal. Após a avaliação, são aplicados os operadores evolutivos responsáveis pela construção da nova população.

Durante toda a execução do algoritmo, é mantido o registro do melhor indivíduo global encontrado. Assim, a solução final não corresponde necessariamente ao melhor indivíduo da última geração, mas sim ao cromossomo que apresentou o menor erro médio dentre todos os indivíduos avaliados ao longo do processo evolutivo. Essa estratégia garante que

soluções de melhor desempenho não sejam perdidas devido à natureza estocástica dos operadores genéticos.

3.6.5 Avaliação dos Indivíduos e Função de Aptidão

Após a geração da população inicial, cada indivíduo é avaliado de forma independente. Como cada cromossomo representa uma configuração específica da rede LSTM, seus genes são utilizados para construir dinamicamente a arquitetura correspondente. Em seguida, a rede é compilada, treinada e avaliada utilizando os dados da série temporal.

A avaliação dos indivíduos é realizada por meio de validação temporal utilizando o método *TimeSeriesSplit*, preservando a ordem cronológica das observações durante a divisão dos dados. Neste trabalho foram utilizadas duas divisões temporais (*n_splits*=2). Em cada divisão, a rede é treinada utilizando apenas informações do passado e posteriormente avaliada sobre observações futuras, evitando vazamento de informações entre treinamento e validação.

Para cada divisão é calculado o Erro Quadrático Médio (*Mean Squared Error* – MSE) entre os valores observados e os valores previstos pela rede. Ao final das duas divisões, calcula-se o MSE médio, que representa o desempenho daquele indivíduo durante o processo de validação.

Esse valor médio é utilizado como função de aptidão do Algoritmo Genético. Como o objetivo do problema consiste em minimizar o erro de previsão, indivíduos que apresentam menores valores de MSE são considerados mais aptos e possuem maior probabilidade de serem preservados durante o processo evolutivo.

Durante toda a execução do algoritmo, o menor MSE médio obtido é continuamente monitorado. Sempre que um indivíduo apresenta desempenho superior ao melhor registrado até então, sua configuração é armazenada como o melhor indivíduo global, sendo posteriormente utilizada para reconstrução e treinamento final da arquitetura LSTM selecionada.

3.6.6 Seleção por Torneio

Após a avaliação de todos os indivíduos da população, é aplicado o operador de seleção por torneio para determinar quais cromossomos participarão da reprodução. Neste trabalho foi adotado tamanho de torneio igual a $k = 3$.

Em cada torneio, três indivíduos são selecionados aleatoriamente da população. Entre eles, é escolhido aquele que apresenta o menor valor de MSE médio, sendo esse indivíduo utilizado como progenitor para a etapa de reprodução. Esse procedimento é repetido até que sejam obtidos os indivíduos necessários para a aplicação do operador de *crossover*.

A utilização da seleção por torneio permite favorecer indivíduos de melhor desempenho sem eliminar completamente a diversidade genética da população, uma vez que a escolha dos participantes do torneio permanece aleatória ao longo de toda a execução do algoritmo.

3.6.7 *Crossover* de Um Ponto

Os indivíduos selecionados na etapa anterior são recombinados utilizando o operador de *crossover* de um ponto. Nesse procedimento, um ponto de corte é escolhido aleatoriamente ao longo do cromossomo e os genes localizados após esse ponto são trocados entre dois indivíduos progenitores.

Como cada cromossomo é composto por cinco genes, o ponto de corte pode ocorrer entre quaisquer duas posições consecutivas. Os descendentes resultantes combinam hiperparâmetros provenientes de ambos os progenitores, permitindo que novas arquiteturas LSTM sejam exploradas nas gerações seguintes.

Após o *crossover*, os descendentes produzidos passam para a etapa de mutação, onde pequenas alterações aleatórias podem ser introduzidas em seus genes.

3.6.8 Mutação

Após o *crossover*, cada descendente é submetido ao operador de mutação com probabilidade de 10%. Quando a mutação ocorre, uma das cinco posições do cromossomo é selecionada aleatoriamente.

O valor do gene selecionado é substituído por um novo valor gerado aleatoriamente dentro do intervalo definido para o respectivo hiperparâmetro. Dessa forma, a mutação pode alterar o número de camadas LSTM, o número de neurônios, a taxa de aprendizado, o tamanho do lote ou o número de épocas.

Apenas um gene é alterado em cada ocorrência do operador de mutação. Quando o valor aleatório utilizado no teste de probabilidade não satisfaz a taxa estabelecida, o descendente permanece inalterado. Esse procedimento introduz novas configurações na população e mantém os valores dos genes dentro dos limites estabelecidos para o espaço de busca.

3.6.9 Elitismo

Após a avaliação dos indivíduos, a população é ordenada de forma crescente de acordo com o MSE médio, de modo que as configurações com menor erro ocupem as primeiras posições. Em seguida, são preservados os 50% melhores indivíduos da geração.

Considerando uma população de 30 indivíduos, o procedimento mantém diretamente os 15 indivíduos com menor MSE médio. Esses indivíduos são copiados sem alterações

para a população seguinte, enquanto as 15 posições restantes são preenchidas pelos descendentes produzidos por seleção, *crossover* e mutação.

A utilização do elitismo assegura que as melhores soluções encontradas em cada geração não sejam perdidas durante a aplicação dos operadores evolutivos. Entretanto, o melhor indivíduo global também é armazenado separadamente, permitindo preservar a melhor configuração identificada ao longo de toda a execução do algoritmo.

3.6.10 Formação da Nova População e Critério Evolutivo

Após a aplicação dos operadores de seleção, *crossover*, mutação e elitismo, é formada a nova população do Algoritmo Genético. Essa população mantém o mesmo tamanho da população inicial, garantindo que o processo evolutivo preserve uma quantidade constante de indivíduos ao longo das gerações.

Cada nova geração é composta por indivíduos preservados pelo elitismo e por descendentes produzidos pelos operadores de *crossover* e mutação. Após sua formação, todos os indivíduos são novamente convertidos em arquiteturas LSTM, treinados e avaliados utilizando o procedimento descrito na Seção 3.6.5.

Ao término da avaliação de cada geração, o algoritmo verifica se algum indivíduo apresentou desempenho superior ao melhor resultado obtido até aquele momento. Quando isso ocorre, seus hiperparâmetros são armazenados como a melhor solução global encontrada durante a busca evolutiva.

Esse procedimento é repetido durante cinco gerações. Ao final do processo evolutivo, o cromossomo correspondente ao menor valor de MSE médio é selecionado como a melhor configuração encontrada pelo Algoritmo Genético. Os hiperparâmetros desse indivíduo são posteriormente utilizados para reconstruir a arquitetura da rede LSTM e realizar o treinamento final do modelo.

3.6.11 Construção Dinâmica da Rede LSTM

Após a definição dos valores armazenados no cromossomo, cada indivíduo é convertido automaticamente em uma arquitetura LSTM para que seu desempenho possa ser avaliado. Esse procedimento é realizado dinamicamente, permitindo que cada configuração genética origine uma rede neural distinta.

O número de camadas recorrentes é determinado pelo gene correspondente à profundidade da rede. Para arquiteturas compostas por mais de uma camada LSTM, as camadas intermediárias são configuradas com o parâmetro `return_sequences=True`, permitindo que toda a sequência produzida por uma camada seja utilizada como entrada da camada subsequente. A última camada recorrente retorna apenas o estado oculto final, que é encaminhado para a camada de saída da rede.

Nas arquiteturas com três ou quatro camadas recorrentes, uma camada de *dropout* com taxa fixa de 20% é inserida após cada camada LSTM intermediária. Não é aplicado *dropout* após a primeira camada, após a última camada recorrente ou nas arquiteturas compostas por apenas uma ou duas camadas LSTM.

O número de neurônios em cada camada LSTM é definido pelo segundo gene do cromossomo. A taxa de aprendizado utilizada pelo otimizador Adam, o tamanho do lote de treinamento (*batch size*) e o número de épocas também são determinados diretamente pelos respectivos genes.

A entrada da rede é composta por sequências temporais univariadas contendo 30 observações consecutivas do preço de fechamento, organizadas no formato $(30, 1)$. Após a construção da arquitetura, a rede é compilada, treinada e avaliada utilizando o procedimento descrito nas seções anteriores.

Dessa forma, cada indivíduo do Algoritmo Genético representa simultaneamente uma configuração de hiperparâmetros e uma arquitetura LSTM completa, possibilitando que diferentes estruturas sejam avaliadas automaticamente durante o processo evolutivo.

3.6.12 LSTM Otimizada por Algoritmo Genético Transgênico (LSTM-AGT)

O modelo LSTM-AGT foi desenvolvido a partir da estrutura evolutiva apresentada para o Algoritmo Genético convencional, incorporando um operador transgênico inspirado na proposta de Amaral e Jr (2014). Enquanto o AG realiza a exploração do espaço de busca por meio dos operadores de seleção, *crossover*, mutação e elitismo, o AGT acrescenta um mecanismo capaz de introduzir, de forma direcionada, informações genéticas obtidas a partir do histórico evolutivo da população.

Na implementação desenvolvida nesta dissertação, o operador transgênico não substitui os operadores tradicionais do Algoritmo Genético, mas atua de forma complementar. Inicialmente, cada geração é produzida pelo AG convencional. Em seguida, é construído um histórico genético utilizando os indivíduos preservados pelo elitismo, a partir do qual são identificadas as frequências de ocorrência dos valores observados em cada gene do cromossomo.

Com base nesse histórico, são gerados indivíduos transgênicos por meio de quatro módulos independentes, denominados M1, M2, M3 e M4, responsáveis por modificar, respectivamente, um, dois, três ou quatro genes do indivíduo-base. Os valores inseridos em cada posição gênica são selecionados probabilisticamente de acordo com as frequências observadas no histórico transgênico.

Diferentemente da implementação original proposta por Amaral e Jr (2014), em que

cada módulo produz uma população completa e independente, nesta dissertação os módulos geram apenas indivíduos transgênicos, os quais substituem indivíduos de pior desempenho da população provisória produzida pelo Algoritmo Genético. Essa adaptação foi proposta para reduzir o custo computacional associado ao treinamento repetido de redes LSTM, preservando a lógica fundamental do operador transgênico e mantendo constante o tamanho da população ao longo do processo evolutivo.

As subseções seguintes descrevem detalhadamente a construção do histórico transgênico, o processo de seleção probabilística dos genes, a implementação dos módulos M1, M2, M3 e M4, a estratégia de aplicação das taxas transgênicas e as adaptações realizadas em relação ao operador originalmente proposto por Amaral e Jr (2014).

3.6.13 Construção do Histórico Transgênico

O histórico transgênico constitui o principal mecanismo responsável por orientar a atuação do operador transgênico durante o processo evolutivo. Sua finalidade é registrar, a cada geração, quais valores gênicos estão associados aos indivíduos de melhor desempenho, permitindo que essas informações sejam posteriormente utilizadas na construção dos indivíduos transgênicos.

Na implementação desenvolvida nesta dissertação, o histórico transgênico é reconstruído ao final de cada geração utilizando exclusivamente os indivíduos preservados pelo elitismo. Dessa forma, apenas as soluções com menor (MSE) contribuem para a composição do histórico genético empregado pelo operador transgênico.

Cada indivíduo é representado por um cromossomo formado por cinco genes, correspondentes aos hiperparâmetros otimizados pelo algoritmo: número de camadas LSTM, número de neurônios por camada, taxa de aprendizado, tamanho do lote de treinamento (*batch size*) e número de épocas. Para cada posição do cromossomo, o histórico contabiliza a frequência de ocorrência dos valores observados entre os indivíduos elitistas da geração.

Assim, para cada um dos cinco genes, é construída uma distribuição de frequências contendo apenas os valores presentes nos indivíduos selecionados pelo elitismo. Essas distribuições representam o conhecimento acumulado pelo processo evolutivo naquela geração e servem como base para a seleção probabilística dos genes que serão inseridos nos indivíduos transgênicos.

A Figura 7 ilustra o processo de construção do histórico transgênico utilizado nesta dissertação.

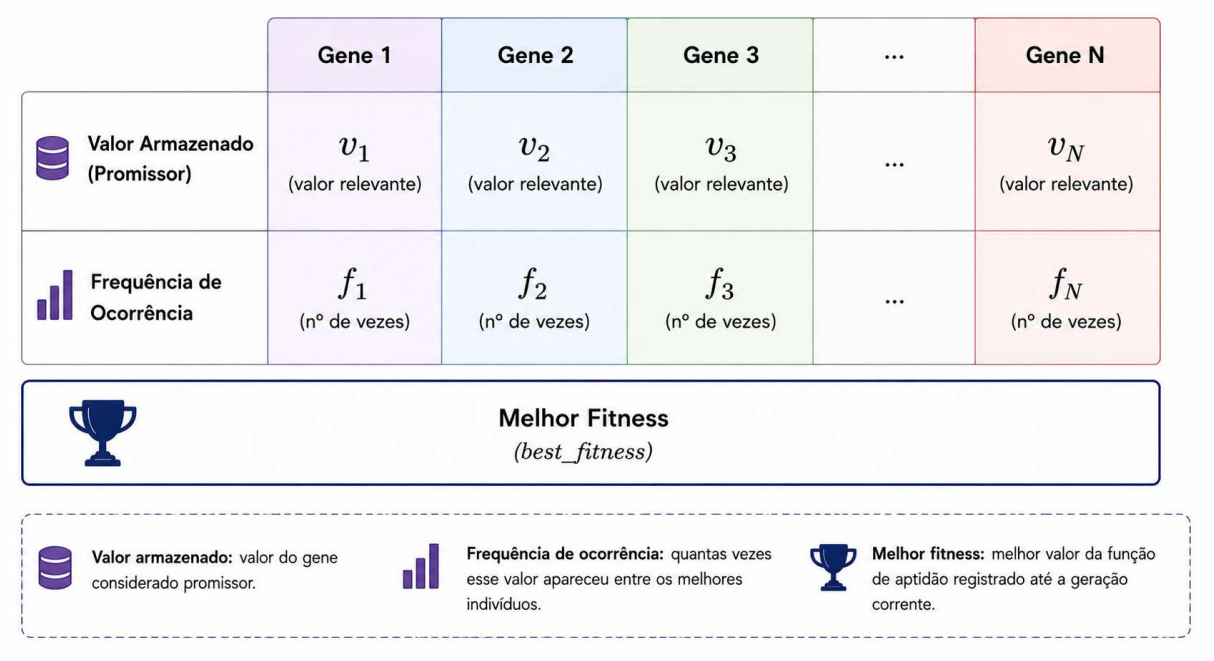

Figura 7 – Construção do histórico transgênico a partir dos indivíduos preservados pelo elitismo. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

Diferentemente do operador original proposto por Amaral e Jr (2014), cujo histórico armazena informações referentes aos indivíduos que apresentam o melhor valor de aptidão encontrado durante a evolução, nesta dissertação o histórico é reconstruído a cada geração utilizando os indivíduos preservados pelo elitismo. Essa adaptação foi adotada para tornar o operador mais adequado ao problema de otimização contínua dos hiperparâmetros da rede LSTM, permitindo que o histórico acompanhe dinamicamente a evolução da população ao longo das gerações.

3.6.13.1 Histórico Transgênico

No modelo LSTM-AGT desenvolvido neste trabalho, o histórico transgênico é construído a partir dos melhores indivíduos selecionados por elitismo em cada geração. Após a avaliação da população por meio da função de aptidão, baseada no (MSE), os indivíduos com menor erro são preservados e utilizados como referência para a atualização do histórico genético.

Cada indivíduo é representado por um cromossomo composto por cinco genes: número de camadas LSTM, número de neurônios, taxa de aprendizado, tamanho do lote (*batch size*) e número de épocas. Para cada posição genética, o histórico contabiliza a frequência com que cada valor aparece entre os melhores indivíduos da geração.

Dessa forma, diferentemente de uma escolha aleatória totalmente uniforme, o opera-

dor transgênico passa a utilizar informações do processo evolutivo para identificar padrões associados às melhores soluções encontradas. Os genes mais frequentes entre os indivíduos elitistas são interpretados como características promissoras e passam a ter maior probabilidade de serem transferidos para novos indivíduos transgênicos.

3.6.13.2 Seleção Probabilística dos Genes Frequentes

A seleção dos genes utilizados pelo operador transgênico é realizada de forma probabilística, considerando a frequência observada no histórico genético. Para cada posição do cromossomo, são levantados os valores presentes entre os melhores indivíduos da geração e suas respectivas ocorrências.

A probabilidade de seleção de um valor genético é proporcional à sua frequência. Assim, valores que aparecem mais vezes entre os indivíduos de melhor desempenho possuem maior chance de serem escolhidos. No entanto, a seleção não é determinística, pois valores menos frequentes ainda podem ser selecionados, preservando certo grau de diversidade no processo evolutivo.

Essa estratégia segue a lógica do operador transgênico original proposto por Amaral e Jr (2014), no qual características consideradas relevantes são identificadas a partir do histórico evolutivo e inseridas em novos indivíduos. No presente trabalho, essa lógica foi adaptada para o contexto de otimização de hiperparâmetros de redes LSTM.

3.6.14 Operador Transgênico Modular

O operador transgênico implementado nesta dissertação é composto por quatro módulos independentes, denominados M1, M2, M3 e M4. Todos os módulos utilizam o mesmo histórico transgênico construído a partir dos indivíduos preservados pelo elitismo, diferindo apenas pela quantidade de genes modificados em cada indivíduo.

A geração de um indivíduo transgênico inicia-se pela seleção aleatória de um indivíduo-base pertencente à população corrente. Em seguida, são escolhidas as posições do cromossomo que serão modificadas de acordo com o módulo utilizado. Para cada posição selecionada, um novo valor é obtido por meio da seleção probabilística baseada nas frequências registradas no histórico transgênico, substituindo o valor originalmente presente naquele gene.

Dessa forma, cada módulo produz novos indivíduos preservando parte da estrutura genética do indivíduo-base e alterando apenas o número de genes correspondente ao módulo aplicado. Esse procedimento permite diferentes níveis de intervenção genética, equilibrando exploração de novas configurações de hiperparâmetros e preservação de características promissoras identificadas durante o processo evolutivo.

3.6.14.1 Módulo M1

O módulo M1 realiza a modificação de apenas um gene do indivíduo-base. Após a seleção da posição genética a ser alterada, um novo valor é sorteado probabilisticamente a partir do histórico transgênico correspondente àquela posição do cromossomo. Os demais genes permanecem inalterados.

Por promover a menor intervenção genética entre os quatro módulos, o M1 preserva grande parte da configuração original do indivíduo, realizando pequenas perturbações direcionadas na busca por soluções com melhor desempenho.

3.6.14.2 Módulo M2

No módulo M2 são modificados dois genes do indivíduo-base. As posições são selecionadas aleatoriamente e cada uma delas recebe um novo valor obtido por seleção probabilística no respectivo histórico transgênico.

Esse módulo amplia o grau de exploração do espaço de busca em relação ao M1, permitindo a combinação simultânea de duas novas configurações de hiperparâmetros sem alterar completamente a estrutura genética original do indivíduo.

3.6.14.3 Módulo M3

O módulo M3 modifica três genes do cromossomo do indivíduo-base. Para cada posição selecionada, um novo valor é obtido utilizando a distribuição de frequências correspondente ao gene, construída a partir do histórico transgênico.

Ao alterar simultaneamente três hiperparâmetros, o módulo M3 aumenta significativamente a capacidade exploratória do operador transgênico, permitindo gerar arquiteturas LSTM mais distintas da configuração original.

3.6.14.4 Módulo M4

O módulo M4 promove a maior intervenção genética entre os módulos implementados, modificando quatro dos cinco genes que compõem o cromossomo do indivíduo-base.

Embora preserve apenas uma característica genética do indivíduo original, os novos valores continuam sendo selecionados com base nas distribuições de frequência do histórico transgênico. Dessa forma, mesmo realizando modificações mais intensas, o módulo mantém o direcionamento da busca para regiões do espaço de soluções associadas aos indivíduos de melhor desempenho.

3.6.15 Taxas Transgênicas

As taxas transgênicas controlam a intensidade de atuação do operador transgênico em cada geração do processo evolutivo. Nesta dissertação foram consideradas cinco intensidades distintas de aplicação: 10%, 30%, 50%, 70% e 90%.

Cada taxa determina a quantidade de indivíduos transgênicos gerados em uma determinada execução do algoritmo. Após definida a taxa, calcula-se o número total de indivíduos que serão produzidos pelo operador transgênico. Essa quantidade é então distribuída igualmente entre os quatro módulos M1, M2, M3 e M4, de modo que todos participem da geração dos indivíduos transgênicos.

Assim, considerando N_T como o número total de indivíduos transgênicos a serem produzidos, cada módulo gera aproximadamente

$$N_1 = N_2 = N_3 = N_4 = \frac{N_T}{4},$$

Essa distribuição uniforme garante que todos os módulos contribuam igualmente para a geração dos indivíduos transgênicos, permitindo comparar o efeito do número de genes modificados sem introduzir diferenças decorrentes da quantidade de indivíduos produzidos por cada módulo.

A utilização de diferentes taxas transgênicas permite avaliar o efeito da intensidade de inserção genética sobre o desempenho do processo evolutivo. Taxas menores promovem alterações mais discretas na população, enquanto taxas mais elevadas ampliam a influência do operador transgênico sobre a composição da geração seguinte.

3.6.16 Inserção dos Indivíduos Transgênicos na População

Após a aplicação dos operadores de seleção, *crossover* e mutação, forma-se uma população provisória composta pelos indivíduos preservados pelo elitismo e pelos descendentes gerados durante o processo evolutivo. É sobre essa população que o operador transgênico atua.

Inicialmente, os módulos M1, M2, M3 e M4 produzem os indivíduos transgênicos de acordo com a taxa transgênica definida para a execução. Esses indivíduos não são adicionados diretamente à população. Antes disso, a população provisória é ordenada de acordo com os valores da função de aptidão.

Em seguida, os indivíduos transgênicos substituem indivíduos de pior desempenho da população provisória. Como a função de aptidão é baseada no (MSE), são substituídos aqueles que apresentam os maiores valores dessa métrica.

Essa estratégia preserva as melhores soluções obtidas pelo processo evolutivo e, simul-

taneamente, introduz novos indivíduos contendo características genéticas selecionadas a partir do histórico transgênico. Dessa forma, o operador transgênico atua como um mecanismo complementar aos operadores tradicionais do algoritmo genético, promovendo diversidade direcionada sem comprometer a qualidade das melhores soluções encontradas até a geração corrente.

A substituição apenas dos indivíduos de pior desempenho reduz o risco de perda de soluções promissoras e mantém constante o tamanho da população ao longo de toda a execução do algoritmo, preservando a estabilidade do processo evolutivo.

3.6.17 Fluxo Geral do Algoritmo Genético Transgênico

A Figura 8 apresenta uma visão geral do Algoritmo Genético Transgênico implementado nesta dissertação. Inicialmente, a população é evoluída pelos operadores tradicionais do algoritmo genético. Em seguida, os indivíduos preservados pelo elitismo são utilizados para construir o histórico transgênico, que orienta a geração de novos indivíduos pelos módulos M1, M2, M3 e M4. Os indivíduos transgênicos produzidos substituem indivíduos de pior desempenho da população provisória, formando a população da geração seguinte.

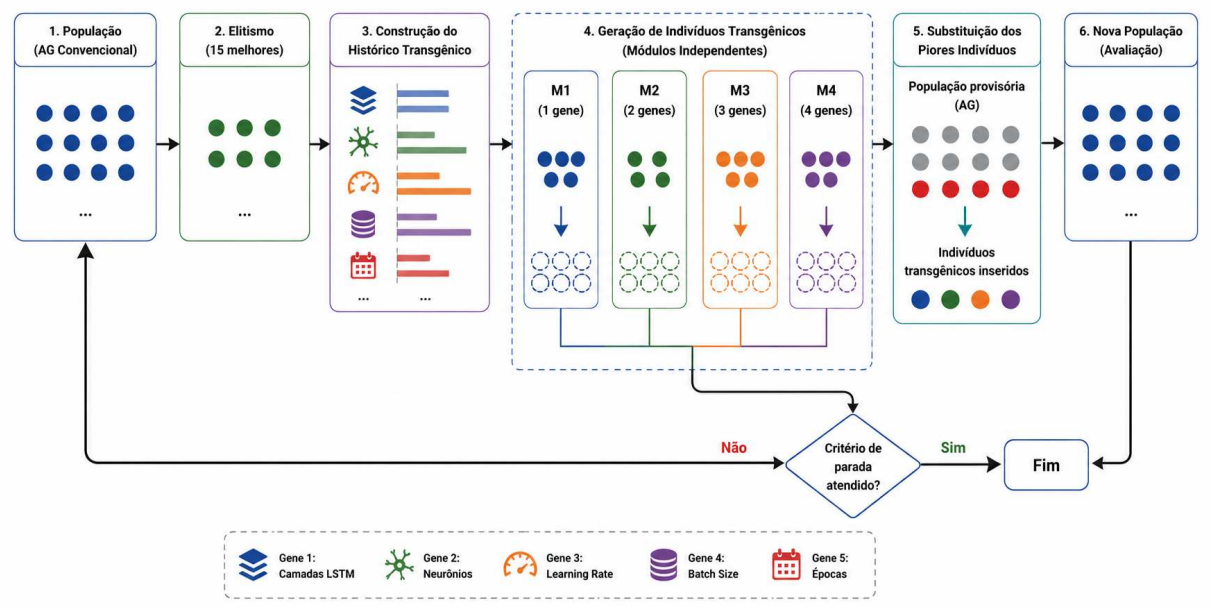

Figura 8 – Fluxo geral do Algoritmo Genético Transgênico implementado nesta dissertação. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

3.6.18 Adaptações em Relação ao Operador Transgênico Original

O operador transgênico implementado nesta dissertação foi inspirado na proposta de Amaral e Jr (2014), preservando seus princípios fundamentais de utilização de um histórico genético, organização modular em quatro operadores (M1, M2, M3 e M4) e inserção direcionada de informações genéticas na população evolutiva.

Entretanto, algumas adaptações foram realizadas para adequar o operador ao problema de otimização de hiperparâmetros de redes LSTM para previsão de séries temporais financeiras. Essas adaptações foram motivadas principalmente pelo elevado custo computacional associado ao treinamento de cada rede neural durante a avaliação da função de aptidão.

A principal diferença consiste na forma de aplicação do operador transgênico. Enquanto a proposta original gera populações independentes por meio dos módulos transgênicos, nesta dissertação cada módulo produz apenas indivíduos transgênicos, que posteriormente substituem indivíduos de pior desempenho da população provisória formada após a aplicação dos operadores tradicionais do algoritmo genético.

Outra adaptação refere-se à representação cromossômica. No trabalho original, os genes representam atributos relacionados ao problema de classificação estudado pelos autores. Nesta pesquisa, cada gene corresponde a um hiperparâmetro da arquitetura da rede LSTM, permitindo que o operador transgênico atue diretamente sobre a configuração estrutural do modelo.

Apesar dessas adaptações, foram preservados os princípios centrais do operador proposto por Amaral et al. Amaral e Jr (2014), especialmente a utilização do histórico transgênico como mecanismo para orientar a inserção probabilística de características genéticas observadas entre os indivíduos de melhor desempenho.

Tabela 5 – Comparação entre o operador transgênico original e a implementação proposta nesta dissertação.

Operador original Amaral e Jr (2014)	**Implementação desta dissertação**
Genes representam atributos do problema	Genes representam hiperparâmetros da rede LSTM
Módulos geram populações independentes	Módulos geram indivíduos transgênicos
Operador aplicado em problemas de classificação	Operador aplicado à otimização de hiperparâmetros
Histórico utilizado para orientar a inserção gênica	Histórico utilizado para selecionar probabilisticamente hiperparâmetros
Inserção baseada no histórico	Inserção baseada no histórico preservando o tamanho da população

Assim, a implementação desenvolvida preserva os princípios fundamentais do operador transgênico proposto por Amaral e Jr (2014), ao mesmo tempo em que adapta sua execução às características específicas do problema de otimização de hiperparâmetros de redes neurais LSTM.

3.6.19 Treinamento Final da Rede

Concluída a etapa de otimização dos hiperparâmetros, a melhor configuração obtida por cada abordagem foi utilizada para o treinamento definitivo da respectiva rede LSTM. Para o modelo LSTM Clássica foram empregados os hiperparâmetros previamente definidos, enquanto para os modelos LSTM-AG e LSTM-AGT foi utilizada a configuração correspondente ao melhor indivíduo global obtido ao final do processo evolutivo.

O treinamento final foi realizado utilizando todos os dados destinados ao conjunto de treinamento, mantendo a mesma estratégia de pré-processamento adotada durante a etapa de otimização, incluindo a normalização dos dados e a construção das janelas temporais.

Para evitar vazamento de informação (*data leakage*), os dados reservados para o *backtesting* permaneceram completamente separados durante todo o processo de otimização e treinamento. Dessa forma, as observações utilizadas na avaliação final não participaram da busca dos hiperparâmetros nem do treinamento definitivo dos modelos.

Os modelos treinados foram então armazenados para utilização nas etapas de avaliação quantitativa e de *backtesting* descritas nas seções seguintes.

3.6.20 Avaliação de Desempenho

Após o treinamento final, o desempenho dos modelos foi avaliado utilizando dados previamente separados para teste. As previsões geradas por cada modelo foram convertidas para a escala original da série temporal por meio da transformação inversa do processo de normalização, permitindo que todas as métricas fossem calculadas diretamente sobre os valores reais do ativo financeiro.

A avaliação quantitativa foi realizada por meio de métricas amplamente empregadas em problemas de regressão e previsão de séries temporais, permitindo analisar diferentes aspectos da qualidade preditiva dos modelos.

As métricas consideradas neste trabalho foram:

- ❏ Erro Quadrático Médio (Mean Squared Error – MSE);
- ❏ Raiz do Erro Quadrático Médio (Root Mean Squared Error – RMSE);
- ❏ Erro Percentual Absoluto Médio (Mean Absolute Percentage Error – MAPE);

- ❏ Coeficiente de Determinação (R^2).

Além das métricas preditivas, também foram analisados o tempo de treinamento e o tempo de inferência dos modelos, possibilitando uma avaliação conjunta da qualidade das previsões e do custo computacional associado a cada abordagem.

3.6.21 Metodologia de *Backtesting*

A etapa de *backtesting* foi realizada com o objetivo de avaliar a capacidade de generalização dos modelos em dados não utilizados durante as etapas de otimização dos hiperparâmetros e treinamento final. Dessa forma, buscou-se reproduzir um cenário mais próximo de uma aplicação prática, no qual o modelo realiza previsões sobre observações futuras completamente desconhecidas durante o processo de aprendizagem.

Foram avaliadas três abordagens de modelagem:

- ❏ LSTM Clássica;
- ❏ LSTM otimizada por Algoritmo Genético (LSTM-AG);
- ❏ LSTM otimizada por Algoritmo Genético Transgênico (LSTM-AGT).

Cada abordagem foi aplicada independentemente a três séries temporais com diferentes granularidades, representando distintos horizontes de previsão no mercado financeiro:

- ❏ Série intradiária com intervalo de 4 horas;
- ❏ Série diária;
- ❏ Série semanal.

Essa estratégia resultou em nove configurações experimentais, permitindo analisar o comportamento dos três modelos em diferentes escalas temporais e níveis de volatilidade do mercado.

Os conjuntos diário e semanal compreendem observações entre janeiro de 2000 e dezembro de 2024. Para a série intradiária de quatro horas foi utilizada a maior janela histórica disponibilizada pela fonte de dados, correspondente aos 730 dias anteriores ao momento da coleta.

Cada série temporal foi tratada como um conjunto de dados independente, possuindo seu próprio processo de normalização, construção das janelas temporais, treinamento e avaliação. Essa estratégia evita o compartilhamento de informações entre diferentes granularidades temporais e assegura a independência entre os experimentos realizados.

Durante o *backtesting* foram utilizados exclusivamente os modelos treinados ao final da etapa de otimização. Em nenhum momento houve atualização dos parâmetros da rede ou reajuste dos normalizadores utilizando dados pertencentes ao período de teste.

As previsões produzidas pelos modelos foram convertidas para a escala original da série temporal por meio da transformação inversa da normalização. Em seguida, foram calculadas as métricas quantitativas descritas na Seção 3.6.20, permitindo comparar o desempenho preditivo das diferentes abordagens em condições equivalentes.

Tabela 6 – Configurações experimentais utilizadas no *backtesting*.

Modelo	Granularidade	Horizonte
LSTM Clássica	4 h	Curto prazo
LSTM Clássica	Diário	Médio prazo
LSTM Clássica	Semanal	Longo prazo
LSTM-AG	4 h	Curto prazo
LSTM-AG	Diário	Médio prazo
LSTM-AG	Semanal	Longo prazo
LSTM-AGT	4 h	Curto prazo
LSTM-AGT	Diário	Médio prazo
LSTM-AGT	Semanal	Longo prazo

A comparação entre as nove configurações experimentais permitiu avaliar o impacto da otimização por Algoritmo Genético e Algoritmo Genético Transgênico em diferentes horizontes temporais de previsão, fornecendo uma análise abrangente da robustez e da capacidade de generalização dos modelos propostos.

Capítulo 4

Experimentos e Análise dos Resultados

Este capítulo apresenta os experimentos desenvolvidos para investigar os efeitos da otimização evolutiva de hiperparâmetros sobre o desempenho de redes neurais LSTM aplicadas à previsão de séries temporais financeiras. Para essa finalidade, são comparadas três abordagens distintas: uma arquitetura de referência com hiperparâmetros previamente definidos (LSTM Clássica), uma arquitetura otimizada por Algoritmo Genético (AG) e uma arquitetura otimizada por Algoritmo Genético Transgênico (AGT).

A análise experimental foi estruturada de forma progressiva. Inicialmente, é apresentada a arquitetura de referência, utilizada como linha de base para todas as comparações posteriores. Em seguida, são analisados os resultados obtidos pelos processos de otimização conduzidos pelo AG e pelo AGT, contemplando a evolução do processo evolutivo, os hiperparâmetros selecionados, o desempenho preditivo e o custo computacional das arquiteturas encontradas. Posteriormente, realiza-se uma comparação integrada entre as três abordagens, permitindo identificar os impactos da otimização evolutiva sobre diferentes métricas de desempenho.

Por fim, os modelos são avaliados por meio de *backtesting* utilizando dados fora da amostra. Essa etapa complementa a análise preditiva ao investigar se melhorias obtidas durante o treinamento das redes neurais também se refletem em desempenho financeiro quando as previsões são empregadas em um cenário operacional. Dessa forma, a avaliação realizada neste capítulo permite analisar, de maneira integrada, os efeitos da otimização de hiperparâmetros desde a seleção das arquiteturas até sua aplicação em operações financeiras simuladas.

4.1 Resultados da LSTM Clássica

Antes da aplicação dos algoritmos evolutivos, foi estabelecida uma arquitetura de referência (*baseline*) utilizando uma configuração fixa de hiperparâmetros para a rede LSTM. Essa arquitetura foi construída a partir de valores amplamente empregados na

literatura para problemas de previsão de séries temporais e tem como finalidade fornecer uma base de comparação para todas as análises desenvolvidas neste capítulo.

A utilização de um modelo de referência permite avaliar objetivamente se as arquiteturas obtidas pelos processos de otimização evolutiva produzem melhorias efetivas em relação a uma configuração previamente definida. Dessa forma, os resultados apresentados nas seções seguintes poderão ser interpretados à luz dos ganhos obtidos exclusivamente pela otimização automática dos hiperparâmetros.

Os experimentos foram realizados utilizando exatamente a mesma arquitetura para as três granularidades temporais analisadas (4 horas, diária e semanal), alterando-se apenas o conjunto de dados correspondente a cada série temporal. Dessa forma, eventuais diferenças observadas entre os resultados podem ser atribuídas principalmente às características inerentes de cada granularidade, e não a modificações na arquitetura da rede.

A configuração utilizada na LSTM Clássica é apresentada na Tabela 7.

Tabela 7 – Hiperparâmetros utilizados na LSTM Clássica.

Hiperparâmetro	Valor
Número de camadas LSTM	2
Número de neurônios	50
Taxa de aprendizagem	0,001
Batch Size	32
Número de épocas	50
Dropout	0,2
Janela temporal	30

Optou-se por manter a mesma configuração da LSTM Clássica para as três granularidades temporais, modificando-se apenas o conjunto de dados utilizado em cada experimento. Essa decisão metodológica reduz a influência de fatores externos na comparação entre os modelos, permitindo que diferenças observadas no desempenho sejam atribuídas predominantemente às características das séries temporais e, posteriormente, aos efeitos da otimização automática de hiperparâmetros.

Em conjunto, a arquitetura de referência estabelece uma linha de base consistente para todas as comparações realizadas neste capítulo. A partir dela, torna-se possível analisar de forma objetiva se os processos de otimização conduzidos pelo Algoritmo Genético e pelo Algoritmo Genético Transgênico são capazes de identificar configurações de hiperparâmetros que proporcionem melhorias no desempenho preditivo, no custo computacional e, posteriormente, no desempenho financeiro observado durante o *backtesting*.

4.1.1 Desempenho preditivo da LSTM Clássica

Após o treinamento da arquitetura de referência, seu desempenho foi avaliado nas três granularidades temporais consideradas neste estudo. A análise contemplou métricas de precisão preditiva (MAPE, MSE, RMSE e coeficiente de determinação R^2), bem como os tempos de treinamento e de inferência, permitindo caracterizar simultaneamente a qualidade das previsões produzidas e o custo computacional associado à arquitetura adotada.

Essa avaliação estabelece o desempenho de referência que será utilizado nas comparações com os modelos otimizados por Algoritmo Genético e Algoritmo Genético Transgênico nas seções seguintes.

Os resultados obtidos são apresentados na Tabela 8.

Tabela 8 – Resultados obtidos pela LSTM Clássica nas diferentes granularidades temporais.

Granularidade	MAPE (%)	MSE	RMSE	R^2	Treinamento (s)	Inferência (s)
4 horas	1,7304	49,3383	7,0241	0,9883	104,94	1,01
Diária	2,1067	10,8959	3,3009	0,9957	254,01	2,56
Semanal	4,1328	52,8440	7,2694	0,9795	69,33	0,99

Os resultados apresentados na Tabela 8 mostram que a arquitetura de referência apresentou elevado desempenho preditivo nas três granularidades temporais avaliadas, alcançando coeficientes de determinação superiores a 0,97 em todos os experimentos. Esses valores indicam que a configuração adotada foi capaz de representar adequadamente a dinâmica das séries temporais utilizadas neste estudo.

Observa-se, entretanto, que o comportamento da arquitetura variou conforme a granularidade temporal analisada. A série diária apresentou os menores valores de MSE e RMSE, bem como o maior coeficiente de determinação, indicando maior precisão sob métricas baseadas na magnitude absoluta dos erros. Por outro lado, a série de 4 horas apresentou o menor MAPE, evidenciando melhor desempenho quando o erro é analisado de forma relativa.

Esses resultados mostram que diferentes métricas enfatizam aspectos distintos da qualidade preditiva. Enquanto MSE e RMSE atribuem maior peso a erros de maior magnitude, o MAPE expressa o erro em termos percentuais, permitindo uma avaliação complementar do desempenho do modelo. Assim, a interpretação dos resultados não deve ser baseada em uma única métrica, mas no conjunto dos indicadores considerados.

Entre as três granularidades, a série semanal apresentou os maiores erros preditivos e o menor coeficiente de determinação. Ainda assim, o valor de $R^2 = 0,9795$ indica que a arquitetura foi capaz de capturar adequadamente o comportamento global da série,

fornecendo uma referência consistente para as comparações com os modelos otimizados apresentadas nas próximas seções.

Em síntese, os resultados mostram que a LSTM Clássica constitui uma arquitetura de referência robusta para os experimentos desenvolvidos nesta pesquisa. Embora diferenças de desempenho tenham sido observadas entre as granularidades temporais, a configuração adotada apresentou elevada capacidade preditiva em todos os cenários analisados, estabelecendo uma base consistente para avaliar os ganhos decorrentes da otimização automática de hiperparâmetros.

4.1.2 Análise da convergência do treinamento

Além das métricas de desempenho preditivo, foi analisada a evolução da função perda durante o treinamento da LSTM Clássica. Essa análise tem como objetivo investigar a dinâmica de aprendizagem da rede ao longo das épocas de treinamento, permitindo verificar se o processo de otimização apresentou comportamento convergente e estabilidade durante o ajuste dos parâmetros da rede.

A Figura 9 apresenta a evolução da função perda ao longo das 50 épocas de treinamento para as três granularidades temporais analisadas, permitindo comparar o comportamento da convergência entre as diferentes séries temporais.

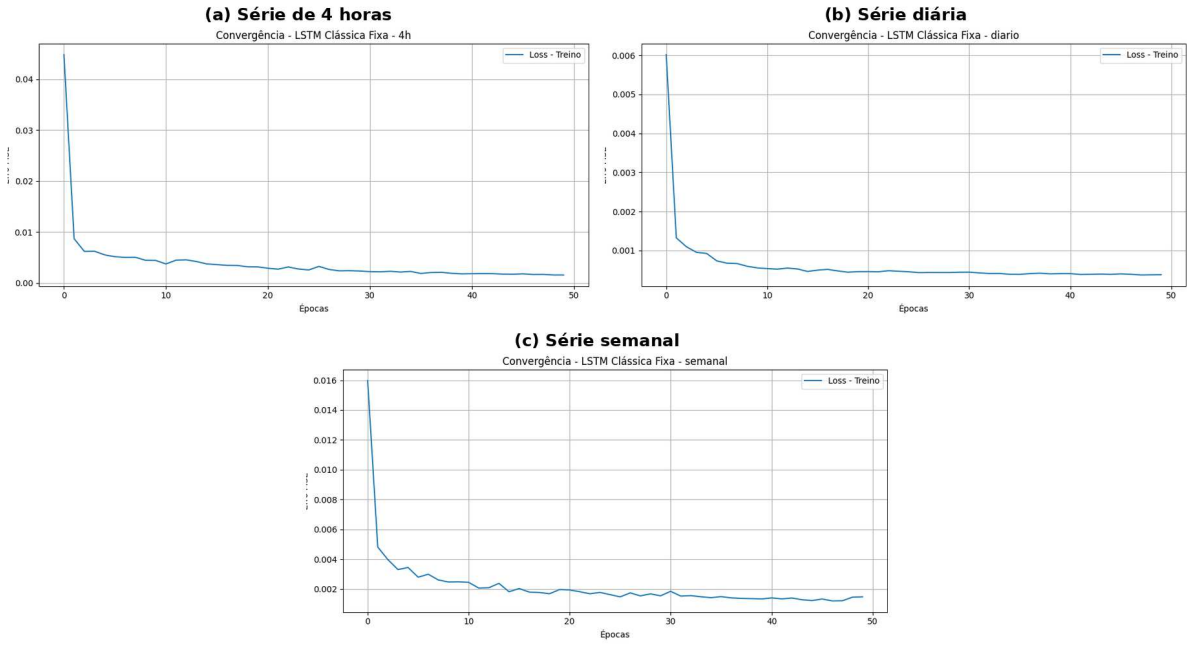

Figura 9 – Evolução da função perda da LSTM Clássica durante o treinamento nas granularidades de 4 horas, diária e semanal.

Fonte: Elaborada pelo autor.

Em todas as granularidades analisadas observa-se uma redução acentuada da função perda nas primeiras épocas de treinamento, seguida por uma diminuição progressiva da taxa de redução até a estabilização das curvas. Esse comportamento é característico de processos de treinamento em que a rede neural realiza ajustes mais significativos dos pesos nas etapas iniciais e refinamentos graduais nas épocas subsequentes.

Embora o comportamento geral de convergência tenha sido semelhante nas três séries temporais, diferenças na velocidade de redução da função perda podem ser observadas. A série diária apresentou estabilização mais rápida e menores valores finais de perda, enquanto a série semanal exibiu redução mais gradual ao longo do treinamento, indicando maior dificuldade de ajuste da arquitetura aos padrões presentes nessa série temporal.

A série de 4 horas apresentou comportamento intermediário entre as demais granularidades. Embora pequenas oscilações tenham sido observadas ao longo do treinamento, apesar de pequenas oscilações, observou-se tendência geral de redução da função de perda. Pequenas oscilações observadas ao longo do treinamento são compatíveis com procedimentos de otimização baseados em mini-lotes *mini-batch*, nos quais pequenas variações da função perda entre épocas consecutivas são esperadas e não caracterizam instabilidade do processo de aprendizagem.

Além da análise qualitativa das curvas de treinamento, foi avaliada a redução relativa da função perda entre a primeira e a última época de treinamento, permitindo quantificar a intensidade da convergência obtida em cada granularidade temporal.

Comparando-se os valores inicial e final da função perda, observou-se uma redução aproximada de 95,7% para a série de 4 horas, 96,1% para a série diária e 91,0% para a série semanal.

Os resultados evidenciam que todas as redes apresentaram convergência consistente durante o treinamento. Entretanto, a velocidade e a intensidade da aprendizagem diferiram entre as granularidades temporais. A série diária apresentou a maior redução relativa da função perda, enquanto a série semanal demandou maior número de épocas para atingir níveis semelhantes de estabilização, corroborando a percepção visual obtida na Figura 9.

Em síntese, a análise das curvas de treinamento evidencia que a arquitetura de referência apresentou comportamento convergente nas três granularidades temporais, sem indícios de instabilidade durante o processo de aprendizagem. Esses resultados reforçam que a configuração adotada foi suficiente para capturar os padrões presentes nas séries temporais, constituindo uma base experimental consistente para avaliar os efeitos da otimização automática de hiperparâmetros nas etapas seguintes.

4.1.3 Análise do custo computacional

Além da qualidade das previsões, foi avaliado o custo computacional associado ao treinamento da LSTM Clássica.

Observou-se que a série diária apresentou o maior tempo de treinamento (254,01 segundos), seguida pelas séries de 4 horas (104,94 segundos) e semanal (69,33 segundos). Esse comportamento é consistente com a quantidade de amostras utilizada em cada experimento, uma vez que a série diária possui aproximadamente três vezes mais observações do que a série de 4 horas e quase cinco vezes mais do que a série semanal.

Apesar do maior tempo absoluto de treinamento, não foram observadas evidências de instabilidade computacional ou dificuldades de convergência. Os tempos de inferência permaneceram inferiores a três segundos em todas as granularidades, indicando que, após treinado, o modelo realiza previsões de forma computacionalmente eficiente.

Tabela 9 – Síntese dos principais resultados obtidos pela LSTM Clássica.

Aspecto	**Principal evidência observada**
Maior R^2	Série diária
Menor RMSE	Série diária
Menor MAPE	Série de 4 horas
Maior redução relativa da perda	Série diária
Convergência mais lenta	Série semanal
Maior tempo de treinamento	Série diária
Menor tempo de inferência	Série semanal

4.2 Otimização dos hiperparâmetros utilizando Algoritmo Genético

Após a definição da arquitetura de referência apresentada na Seção 3.6.1, foi conduzido o processo de otimização automática dos hiperparâmetros por meio do Algoritmo Genético (AG). O objetivo dessa etapa consiste em investigar se a busca evolutiva é capaz de identificar configurações mais adequadas para cada granularidade temporal, proporcionando melhorias no desempenho preditivo da rede LSTM em relação ao modelo de referência.

Diferentemente da LSTM Clássica, cuja configuração permaneceu fixa durante todos os experimentos, o AG realiza sucessivas gerações de soluções candidatas, promovendo a seleção e a recombinação dos indivíduos mais aptos segundo a função de aptidão definida na metodologia. Ao final do processo evolutivo, o melhor indivíduo encontrado é utilizado para treinar novamente a rede LSTM, permitindo avaliar os impactos da otimização sobre o desempenho final do modelo.

4.2.1 Configuração do processo de otimização

O processo de otimização foi realizado separadamente para as três granularidades temporais consideradas neste estudo: 4 horas, diária e semanal. Em todos os experimentos foi utilizada uma população composta por 30 indivíduos, evoluída durante cinco gerações.

Cada indivíduo foi constituído pelos hiperparâmetros número de camadas LSTM, número de neurônios, taxa de aprendizagem, tamanho do lote e número de épocas. A janela temporal foi mantida fixa em 30 observações e, portanto, não integrou o cromossomo submetido ao processo evolutivo.

A avaliação de cada indivíduo foi realizada por meio de validação cruzada temporal com dois particionamentos. Em cada divisão, uma rede LSTM foi construída e treinada utilizando os hiperparâmetros representados pelo cromossomo. O valor de *fitness* correspondeu à média dos valores de MSE obtidos nos dois particionamentos.

Como o problema foi formulado como uma tarefa de minimização, menores valores de *fitness* representaram indivíduos com melhor desempenho.

A Tabela 10 apresenta os parâmetros empregados no processo de otimização.

Tabela 10 – Configuração utilizada no processo de otimização com o Algoritmo Genético.

Parâmetro	Valor
Tamanho da população	30 indivíduos
Número de gerações	5
Taxa de mutação	10%
Seleção	Torneio com $k = 3$
Cruzamento	Um ponto
Elitismo	50% da população
Validação temporal	*TimeSeriesSplit* com 2 divisões
Função de *fitness*	Média do MSE
Janela temporal	30 observações

A manutenção da mesma configuração evolutiva nas três execuções permitiu comparar o comportamento do Algoritmo Genético em diferentes granularidades temporais. Dessa forma, as diferenças observadas no processo de otimização decorreram das características das séries e das combinações de hiperparâmetros avaliadas em cada execução.

4.2.2 Espaço de busca dos hiperparâmetros

Embora a configuração geral do Algoritmo Genético permaneça constante durante toda a otimização, o espaço de busca explorado pelo algoritmo é relativamente amplo, sendo composto por diferentes combinações dos hiperparâmetros que definem a arquitetura e o processo de treinamento da rede LSTM.

Cada indivíduo da população representa uma solução candidata nesse espaço multidimensional, sendo caracterizado pelo número de camadas LSTM, quantidade de neurônios, taxa de aprendizagem, tamanho do lote e número de épocas. Durante o processo evolutivo, diferentes combinações desses hiperparâmetros são avaliadas e sucessivamente refinadas pelos operadores genéticos, permitindo que regiões promissoras do espaço de busca sejam progressivamente exploradas.

Antes de analisar o desempenho preditivo da arquitetura otimizada, torna-se necessário compreender o comportamento do processo evolutivo responsável pela seleção dos hiperparâmetros. Para isso, inicialmente são analisadas a evolução da aptidão da população ao longo das gerações e a convergência do algoritmo durante a busca pelas melhores configurações.

A Figura 10 sintetiza os limites definidos para cada hiperparâmetro durante o processo de otimização. Em vez de restringir previamente a arquitetura da rede, o Algoritmo Genético explora diferentes regiões desse espaço de busca, permitindo que a seleção evolutiva identifique automaticamente configurações mais adequadas às características de cada série temporal.

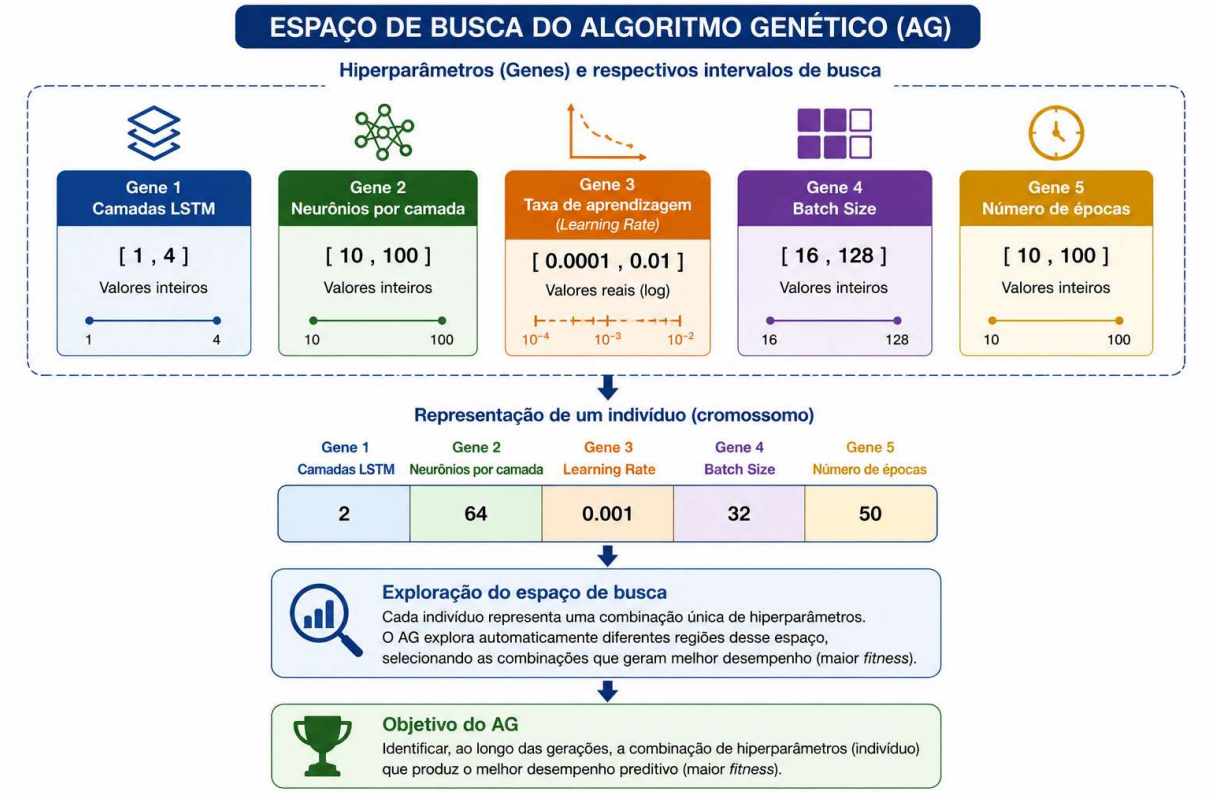

Figura 10 – Espaço de busca definido para a otimização dos hiperparâmetros da rede LSTM utilizando Algoritmo Genético. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

Dessa forma, o espaço de busca definido nesta pesquisa permite ao Algoritmo Genético explorar diferentes combinações arquiteturais sem impor previamente uma configuração considerada ideal. A qualidade dessas soluções será analisada na próxima subseção por meio da evolução da função de *fitness* ao longo das gerações.

4.2.3 Evolução do *fitness* ao longo das gerações

A evolução da função de *fitness* foi analisada com o objetivo de verificar como o Algoritmo Genético modificou a qualidade das soluções candidatas ao longo das gerações. Como o valor de aptidão corresponde à média do MSE obtido nos dois particionamentos da validação cruzada temporal, menores valores de *fitness* representam configurações de hiperparâmetros com melhor desempenho durante a etapa de busca.

A Figura 11 apresenta, para cada granularidade temporal, o melhor valor de *fitness* obtido em cada geração e o melhor resultado global preservado até aquela etapa. A avaliação da população resultante após a quinta geração também foi considerada, pois os indivíduos produzidos ao final do último ciclo evolutivo ainda precisaram ser avaliados antes da seleção definitiva da melhor configuração.

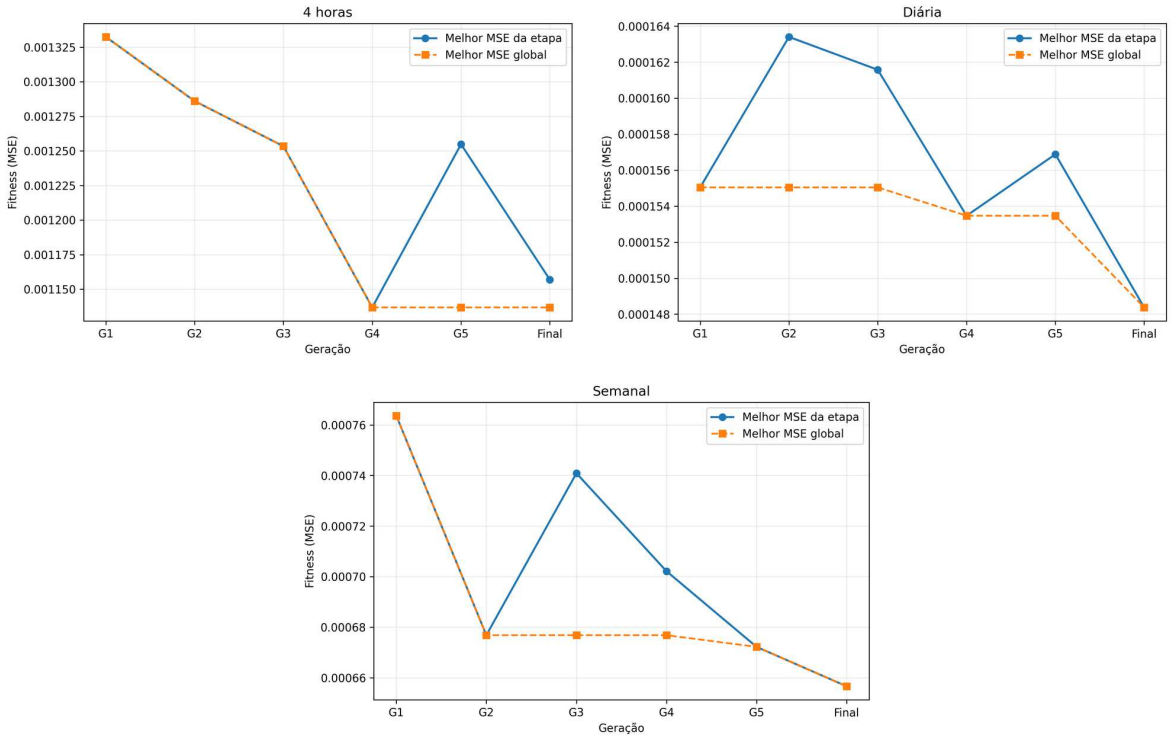

Figura 11 – Evolução do melhor valor de *fitness* do Algoritmo Genético nas granularidades de 4 horas, diária e semanal. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)),

A análise da Figura 11 evidencia que o comportamento evolutivo do Algoritmo Gené-

tico apresentou características distintas entre as três granularidades temporais, refletindo diferenças na complexidade dos respectivos espaços de busca. Em todas as configurações observou-se que o processo evolutivo foi capaz de identificar indivíduos com menor valor de *fitness* ao longo das gerações, indicando que a estratégia de seleção, cruzamento e mutação promoveu melhoria gradual da qualidade das soluções candidatas.

Na série de 4 horas, verifica-se uma redução consistente do melhor MSE entre a primeira e a quarta geração, indicando que as primeiras iterações concentraram a maior parte dos ganhos obtidos durante a otimização. Embora a quinta geração tenha produzido um indivíduo ligeiramente inferior ao melhor já encontrado, o processo preservou a melhor solução global por meio do mecanismo de elitismo, sendo a população final novamente avaliada antes da seleção definitiva dos hiperparâmetros. Esse comportamento evidencia que pequenas oscilações entre gerações são esperadas em algoritmos evolutivos e não representam perda da convergência do processo.

Para a série diária observa-se um comportamento semelhante, porém com menor amplitude de variação entre as gerações. Após uma pequena oscilação inicial, o algoritmo apresentou redução gradual do valor de *fitness*, alcançando seu melhor resultado na avaliação final da população. Esse comportamento sugere que, embora o melhor indivíduo não tenha surgido imediatamente durante a evolução, a recombinação dos cromossomos nas últimas etapas produziu soluções mais adequadas para essa granularidade temporal.

Na série semanal, o processo evolutivo apresentou maior variabilidade nas primeiras gerações, seguida por redução progressiva do *fitness* até a avaliação final. Esse comportamento é compatível com a natureza estocástica do Algoritmo Genético, na qual diferentes regiões do espaço de busca são exploradas antes da convergência para soluções de melhor desempenho. A ausência de oscilações abruptas nas etapas finais indica estabilização do processo evolutivo e reforça a consistência da configuração selecionada para o treinamento definitivo da rede.

Embora a evolução do melhor indivíduo permita verificar a convergência da busca, ela não evidencia como o conjunto da população evoluiu durante o processo de otimização. Para complementar essa análise, também foram avaliadas duas medidas populacionais: a proporção de cromossomos distintos presentes em cada geração e a entropia média dos genes. Enquanto a diversidade cromossômica indica o grau de variedade das soluções candidatas, a entropia quantifica a variabilidade genética existente na população ao longo do processo evolutivo.

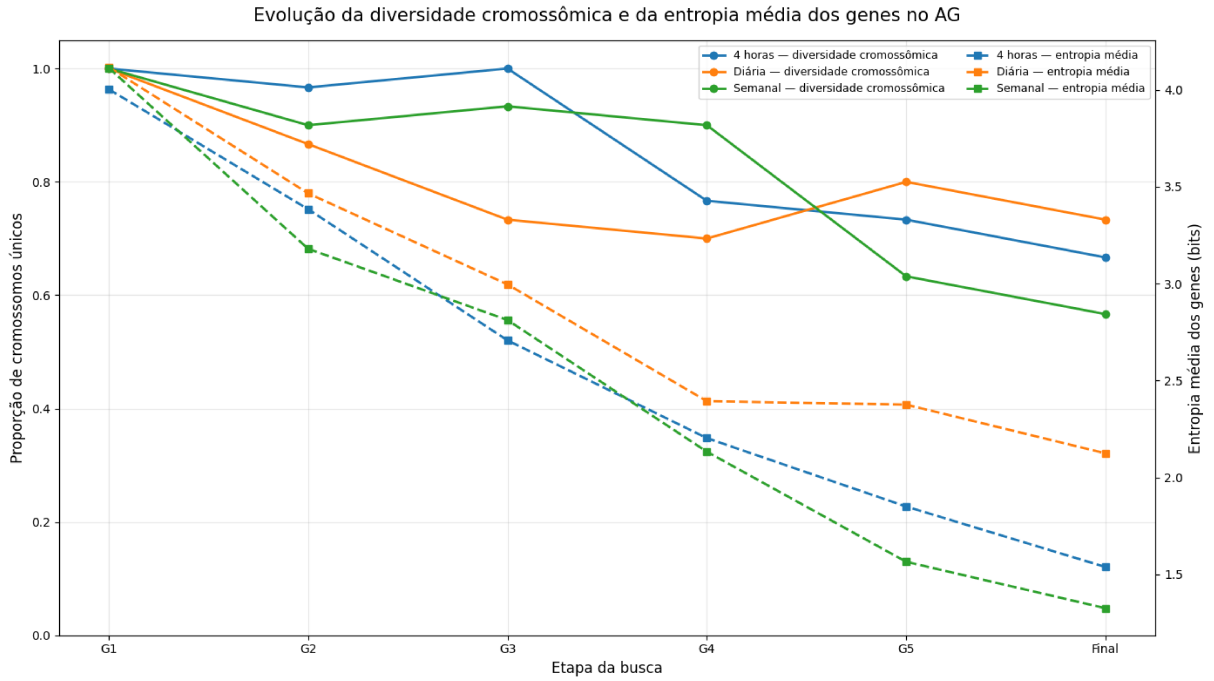

Figura 12 – Evolução da diversidade cromossômica e da entropia média dos genes durante a otimização pelo Algoritmo Genético.Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

A Figura 12 evidencia que, nas três granularidades temporais, a diversidade cromossômica e a entropia genética apresentaram tendência geral de redução ao longo das gerações. Esse comportamento é esperado em algoritmos evolutivos, uma vez que os operadores de seleção favorecem progressivamente indivíduos mais aptos, reduzindo gradualmente a variabilidade da população.

Na série de 4 horas, observa-se que a diversidade permaneceu elevada durante as primeiras gerações, indicando maior exploração do espaço de busca antes da convergência. Nas séries diária e semanal, a redução da diversidade ocorreu de forma mais acentuada, sugerindo que boas configurações foram identificadas mais rapidamente e passaram a predominar na população.

A entropia média dos genes apresentou comportamento semelhante ao observado para a diversidade cromossômica, evidenciando diminuição gradual da variabilidade genética ao longo da otimização. Esse resultado indica que os operadores evolutivos conduziram a população para regiões progressivamente mais promissoras do espaço de busca, preservando simultaneamente a capacidade de exploração necessária para evitar convergência prematura nas etapas iniciais.

Em conjunto, a análise do melhor *fitness*, da diversidade cromossômica e da entropia populacional demonstra que o Algoritmo Genético apresentou um processo evolutivo consistente, combinando exploração inicial do espaço de busca com convergência gradual

para soluções de melhor desempenho.

As evidências apresentadas nesta subseção mostram que o Algoritmo Genético não apenas identificou indivíduos com menor valor de *fitness*, mas também conduziu toda a população para configurações progressivamente mais adequadas ao problema estudado. A partir dessas soluções, procede-se à análise dos hiperparâmetros selecionados para o treinamento definitivo da rede LSTM.

4.2.4 Hiperparâmetros selecionados pelo Algoritmo Genético

Após a conclusão das cinco gerações, a população final foi novamente avaliada e comparada com o melhor indivíduo global registrado durante o processo evolutivo. Para cada granularidade temporal foi selecionada a configuração que apresentou o menor valor de *fitness* entre todas as soluções avaliadas.

Tabela 11 – Hiperparâmetros utilizados no treinamento final da LSTM otimizada pelo Algoritmo Genético.

Granularidade	Camadas LSTM	Neurônios	Taxa de aprendizagem	*Batch size*	Épocas	*Dropout*
4 horas	2	52	0,0088	70	99	0,2
Diária	1	54	0,0032	32	41	0,2
Semanal	1	48	0,0096	38	84	0,2

Os hiperparâmetros apresentados na Tabela 11 correspondem às configurações selecionadas automaticamente pelo Algoritmo Genético ao término do processo evolutivo e posteriormente empregadas no treinamento final da rede LSTM. Observa-se que cada granularidade temporal resultou em uma combinação distinta de número de camadas, quantidade de neurônios, taxa de aprendizagem, tamanho do lote e número de épocas, indicando que as características estatísticas das séries temporais influenciaram diretamente a configuração considerada mais adequada pelo algoritmo.

Esse resultado evidencia uma das principais vantagens da otimização evolutiva: a adaptação automática da arquitetura da rede às características específicas de cada conjunto de dados, dispensando a definição manual de uma configuração única para todas as granularidades analisadas.

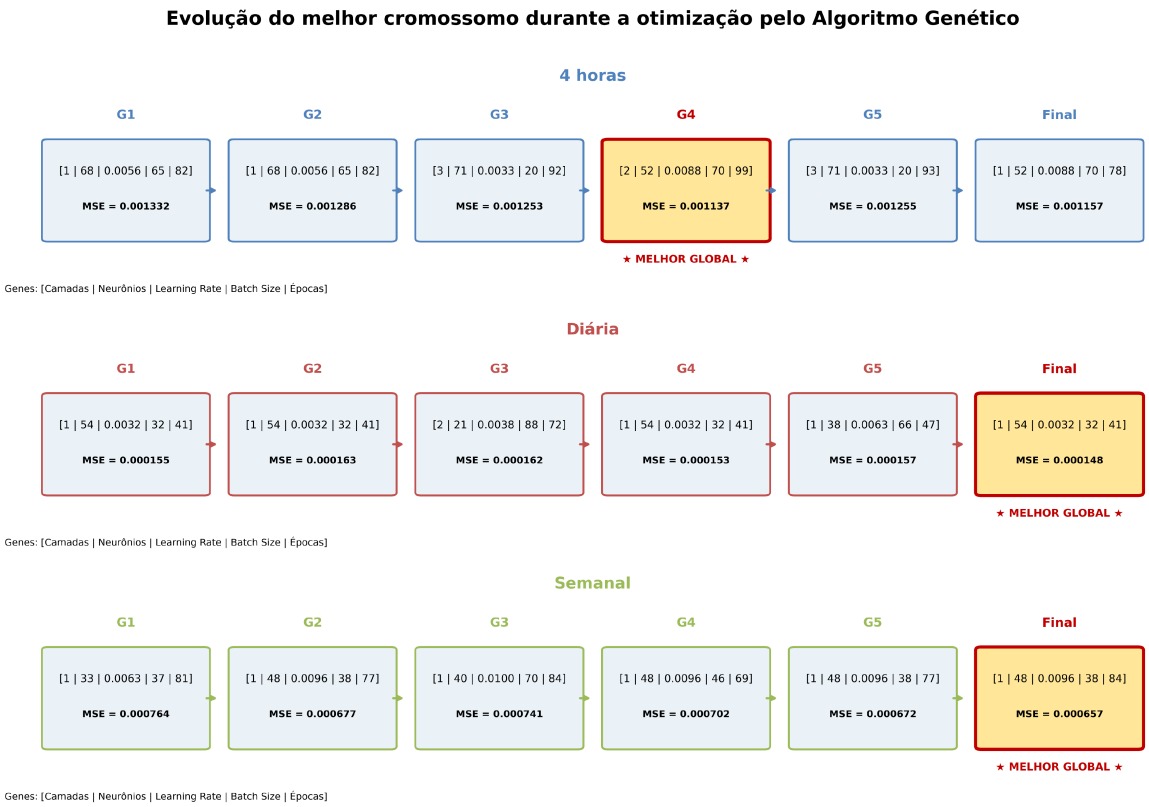

Figura 13 – Evolução do melhor cromossomo identificado em cada geração do Algoritmo Genético para as granularidades de 4 horas, diária e semanal. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

A Figura 13 complementa as informações apresentadas na tabela de hiperparâmetros ao mostrar como o melhor cromossomo evoluiu ao longo das gerações. Enquanto a tabela apresenta apenas a configuração final selecionada para o treinamento da rede, a figura evidencia que diferentes combinações de hiperparâmetros competiram entre si antes da convergência da busca.

Observa-se que os melhores cromossomos sofreram modificações sucessivas ao longo das gerações, indicando que o processo evolutivo refinou progressivamente os valores dos hiperparâmetros. Além disso, as trajetórias observadas diferem entre as três granularidades temporais, evidenciando que cada frequência de amostragem demandou um processo de adaptação próprio até a obtenção da configuração final.

Esses resultados reforçam que a utilização do Algoritmo Genético permitiu selecionar automaticamente configurações competitivas às características de cada série temporal, justificando as diferenças observadas entre os hiperparâmetros finais utilizados no treinamento das redes LSTM.

4.2.5 Desempenho preditivo da LSTM otimizada pelo Algoritmo Genético

Após a seleção dos hiperparâmetros, as configurações identificadas pelo Algoritmo Genético foram utilizadas no treinamento final das redes LSTM. O desempenho foi avaliado sobre o conjunto de teste por meio das métricas MAPE, MSE, RMSE e coeficiente de determinação, além dos tempos de treinamento e inferência.

Tabela 12 – Resultados obtidos pela LSTM otimizada pelo Algoritmo Genético.

Granularidade	MAPE (%)	MSE	RMSE	R^2	Treinamento (s)	Inferência (s)
4 horas	1,1803	27,0210	5,1982	0,9933	109,31	2,62
Diária	1,9093	10,9286	3,3058	0,9957	93,70	1,31
Semanal	3,6030	39,1101	6,2538	0,9849	46,21	0,50

Os resultados da Tabela 12 mostram que as três configurações selecionadas pelo Algoritmo Genético apresentaram coeficientes de determinação superiores a 0,98. A granularidade diária alcançou os menores valores absolutos de MSE e RMSE e o maior R^2, enquanto a série de 4 horas apresentou o menor erro percentual, com MAPE de 1,1803%.

Em comparação com a LSTM Clássica, o MAPE foi reduzido nas três granularidades. A maior redução ocorreu na série de 4 horas, passando de 1,7304% para 1,1803%, o que corresponde a aproximadamente 31,8%. Nas séries diária e semanal, as reduções foram de aproximadamente 9,4% e 12,8%, respectivamente.

Na granularidade de 4 horas, a otimização também reduziu o MSE de 49,3383 para 27,0210 e o RMSE de 7,0241 para 5,1982, acompanhados pelo aumento do R^2 de 0,9883 para 0,9933. Na série semanal, o MSE diminuiu de 52,8440 para 39,1101, o RMSE passou de 7,2694 para 6,2538 e o R^2 aumentou de 0,9795 para 0,9849.

Na série diária, entretanto, o MSE e o RMSE permaneceram praticamente estáveis: o MSE passou de 10,8959 para 10,9286 e o RMSE de 3,3009 para 3,3058. Nesse caso, o principal ganho ocorreu no MAPE, indicando redução do erro relativo, sem melhoria correspondente nas métricas baseadas na magnitude absoluta dos erros.

De maneira geral, os resultados indicam que os efeitos da otimização variaram conforme a granularidade temporal. Os ganhos mais consistentes no conjunto das métricas ocorreram nas séries de 4 horas e semanal, enquanto, na série diária, a melhoria concentrou-se no erro percentual.

4.2.6 Análise do custo computacional

Além do desempenho preditivo, foram analisados os tempos de treinamento e de inferência das arquiteturas selecionadas pelo Algoritmo Genético.

A série de 4 horas apresentou o maior tempo de treinamento entre os modelos otimizados, totalizando 109,31 segundos. Esse resultado está relacionado à configuração selecionada para essa granularidade, composta por duas camadas LSTM, 52 neurônios e 99 épocas de treinamento.

A série diária apresentou tempo de treinamento de 93,70 segundos, enquanto a série semanal demandou 46,21 segundos. Os tempos de inferência permaneceram reduzidos em todos os experimentos, variando entre 0,50 e 2,62 segundos.

Em comparação com a LSTM Clássica, o tempo de treinamento da série diária foi reduzido de 254,01 para 93,70 segundos, correspondendo a uma diminuição aproximada de 63,1%. Essa redução é compatível com a arquitetura selecionada pelo AG, composta por uma única camada LSTM e 41 épocas de treinamento, em contraste com as duas camadas e 50 épocas da configuração de referência.

Na série semanal, o tempo de treinamento diminuiu de 69,33 para 46,21 segundos, representando redução aproximada de 33,3%. Embora a configuração otimizada tenha utilizado 84 épocas, sua arquitetura possuía apenas uma camada LSTM com 48 neurônios.

Na série de 4 horas, o tempo de treinamento aumentou de 104,94 para 109,31 segundos, acréscimo aproximado de 4,2%. Esse aumento foi limitado, apesar de a configuração otimizada utilizar 99 épocas, e foi acompanhado por reduções expressivas no MAPE, MSE e RMSE.

Os resultados mostram que a otimização modificou simultaneamente a precisão preditiva e o custo computacional. Nas séries diária e semanal, foram obtidas reduções no tempo de treinamento, enquanto, na série de 4 horas, o pequeno aumento do custo computacional foi acompanhado pelos maiores ganhos preditivos observados entre as três granularidades.

4.2.7 Síntese dos resultados obtidos pelo Algoritmo Genético

Os experimentos mostram que o AG selecionou configurações distintas para cada granularidade temporal e produziu efeitos diferentes sobre a precisão preditiva e o custo computacional. A série de 4 horas apresentou os maiores ganhos preditivos em relação à LSTM Clássica, enquanto as séries diária e semanal obtiveram as maiores reduções no tempo de treinamento.

A análise do processo evolutivo também evidenciou redução do *fitness*, da diversidade cromossômica e da entropia dos genes ao longo das gerações, indicando transição de uma população inicialmente mais diversificada para conjuntos de soluções mais concentrados. Esses resultados estabelecem a referência evolutiva necessária para analisar, na seção seguinte, os efeitos da introdução do operador transgênico no processo de otimização.

4.3 Otimização dos hiperparâmetros utilizando Algoritmo Genético Transgênico

Nesta etapa foi avaliado o AGT, proposto como uma extensão do AG por meio da incorporação do operador transgênico descrito no Capítulo 3. Enquanto o AG realiza a busca evolutiva utilizando exclusivamente os operadores de seleção, cruzamento, mutação e elitismo, o AGT incorpora informações provenientes do histórico evolutivo da população para produzir novos indivíduos durante o processo de otimização.

O objetivo dessa estratégia consiste em ampliar a exploração do espaço de busca e reduzir a probabilidade de convergência prematura, permitindo que características consideradas promissoras em gerações anteriores sejam reutilizadas na construção de novas soluções. Dessa forma, espera-se favorecer a obtenção de arquiteturas mais adequadas para o treinamento das redes LSTM.

Assim como no AG, cada indivíduo representa uma configuração completa da rede LSTM, composta pelos hiperparâmetros número de camadas, quantidade de neurônios, taxa de aprendizagem, tamanho do lote e número de épocas de treinamento. A avaliação de cada cromossomo foi realizada utilizando validação cruzada temporal por meio do método *TimeSeriesSplit*, sendo o valor de *fitness* definido pela média do erro quadrático médio obtido nas duas divisões temporais.

Nesta seção são analisados o comportamento do processo evolutivo, a evolução do *fitness*, os hiperparâmetros selecionados, o desempenho preditivo das arquiteturas encontradas e o custo computacional associado ao treinamento final dos modelos.

4.3.1 Configuração do processo de otimização

O processo de otimização utilizando o AGT foi realizado separadamente para as séries temporais de 4 horas, diária e semanal. A configuração básica do algoritmo permaneceu equivalente à utilizada no AG, permitindo que as diferenças observadas entre os resultados fossem atribuídas exclusivamente à inclusão do operador transgênico.

Cada execução foi realizada utilizando população composta por 30 indivíduos evoluída durante cinco gerações. Os operadores de seleção por torneio, cruzamento de um ponto, mutação e elitismo permaneceram inalterados. A diferença em relação ao AG consiste na aplicação do operador transgênico após a etapa de mutação, utilizando o histórico dos melhores indivíduos para gerar novas combinações gênicas.

Os genes transgênicos foram construídos a partir da memória evolutiva acumulada durante a execução do algoritmo. Em cada geração foram atualizadas as frequências de ocorrência dos alelos presentes nos indivíduos de melhor desempenho, possibilitando a criação de cromossomos sintéticos contendo combinações consideradas promissoras.

A avaliação dos indivíduos foi realizada por meio da média do MSE obtido utilizando validação cruzada temporal com duas divisões do *TimeSeriesSplit*. Assim como no AG, menores valores de *fitness* representam melhores soluções.

A Tabela 13 apresenta os parâmetros utilizados durante o processo de otimização.

Tabela 13 – Configuração utilizada no processo de otimização com o Algoritmo Genético Transgênico.

Parâmetro	Valor
Tamanho da população	30 indivíduos
Número de gerações	5
Taxa de mutação	10%
Seleção	Torneio com $k = 3$
Cruzamento	Um ponto
Elitismo	50% da população
Operador transgênico	Ativado
Validação temporal	*TimeSeriesSplit* com 2 divisões
Função de *fitness*	Média do MSE
Janela temporal	30 observações

A utilização da mesma configuração empregada no AG garante que a comparação entre as duas abordagens seja realizada sob condições equivalentes. Dessa forma, eventuais diferenças observadas na evolução do *fitness*, nos hiperparâmetros selecionados e no desempenho final das redes podem ser atribuídas à atuação do operador transgênico incorporado ao processo evolutivo.

4.3.2 Evolução do *fitness* ao longo das gerações

Assim como realizado para o AG, foi analisada a evolução do *fitness* durante as cinco gerações do AGT. Como a função de avaliação correspondeu à média do erro quadrático médio (MSE) obtido por validação cruzada temporal, menores valores de *fitness* indicam indivíduos com melhor desempenho.

A Figura 14 apresenta a evolução do melhor valor de *fitness* obtido em cada geração, bem como o melhor valor global acumulado durante todo o processo evolutivo. Também foi considerada a avaliação da população final produzida após a quinta geração.

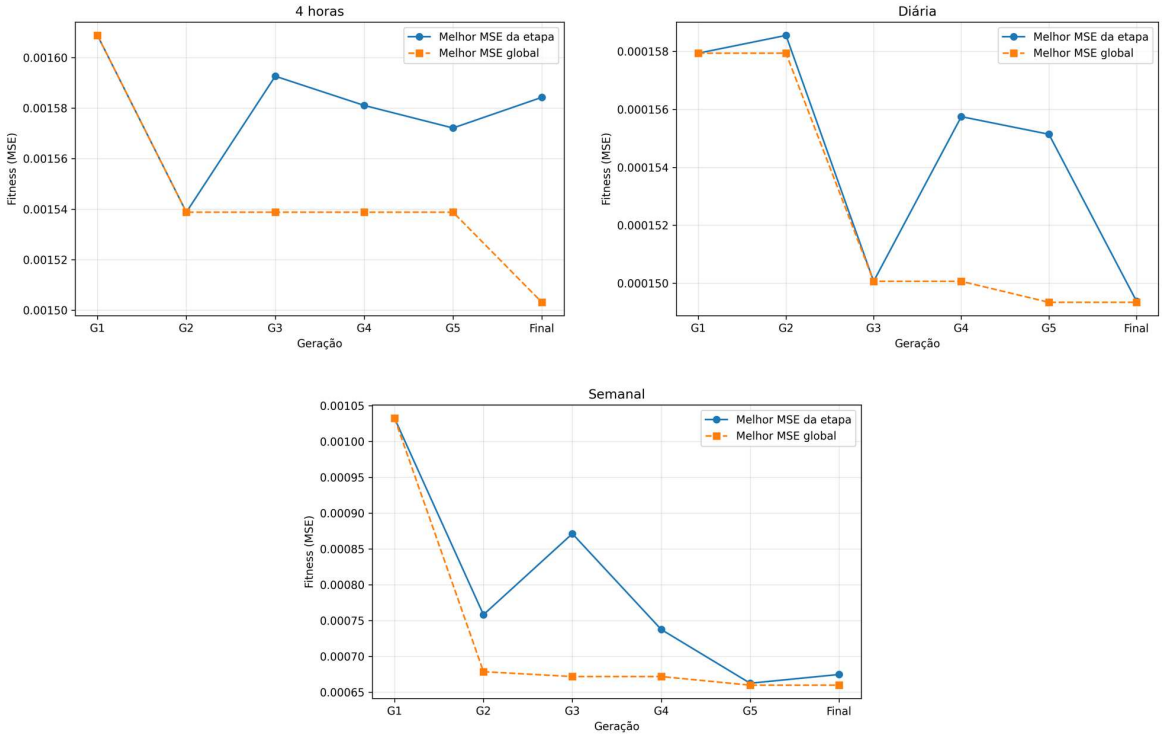

Figura 14 – Evolução do melhor valor de *fitness* do Algoritmo Genético Transgênico nas granularidades de 4 horas, diária e semanal.Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

Na série de 4 horas, o melhor valor de *fitness* obtido na primeira geração foi aproximadamente 0,001609. Nas gerações seguintes ocorreram pequenas oscilações, sendo identificado o melhor indivíduo global com valor próximo de 0,001503. Embora a população final não tenha produzido um indivíduo superior ao melhor já encontrado, a preservação desse cromossomo garantiu a manutenção da melhor solução obtida durante toda a execução.

Na série diária, o comportamento do processo evolutivo apresentou pequenas variações entre as gerações. O melhor valor observado na primeira geração foi aproximadamente 0,000158, enquanto a melhor solução global apresentou valor próximo de 0,000149. As diferenças entre as gerações permaneceram reduzidas, indicando rápida estabilização do processo de busca.

Na série semanal observou-se maior variação durante a evolução da população. O melhor valor de *fitness* reduziu-se de aproximadamente 0,001033 na primeira geração para aproximadamente 0,000660 ao final do processo evolutivo, representando a maior redução relativa entre as três granularidades avaliadas.

De maneira semelhante ao observado no AG, o melhor indivíduo de cada geração não apresentou comportamento estritamente monotônico. Em determinadas gerações ocorre-

ram pequenas oscilações no melhor valor de *fitness*; entretanto, a estratégia de preservação do melhor indivíduo global garantiu que a melhor solução encontrada permanecesse disponível até o encerramento da execução.

Os resultados indicam que o AGT foi capaz de promover melhoria progressiva das soluções avaliadas ao longo das gerações, preservando indivíduos de elevado desempenho e produzindo novas combinações de hiperparâmetros por meio do operador transgênico.

Embora a evolução do melhor indivíduo permita verificar a convergência da busca, ela não evidencia como a população evoluiu durante o processo de otimização. Para complementar essa análise, também foram avaliadas duas medidas populacionais: a proporção de cromossomos distintos presentes em cada geração e a entropia média dos genes. Enquanto a diversidade cromossômica representa a variedade de soluções candidatas existentes na população, a entropia mede a variabilidade genética dos genes ao longo do processo evolutivo.

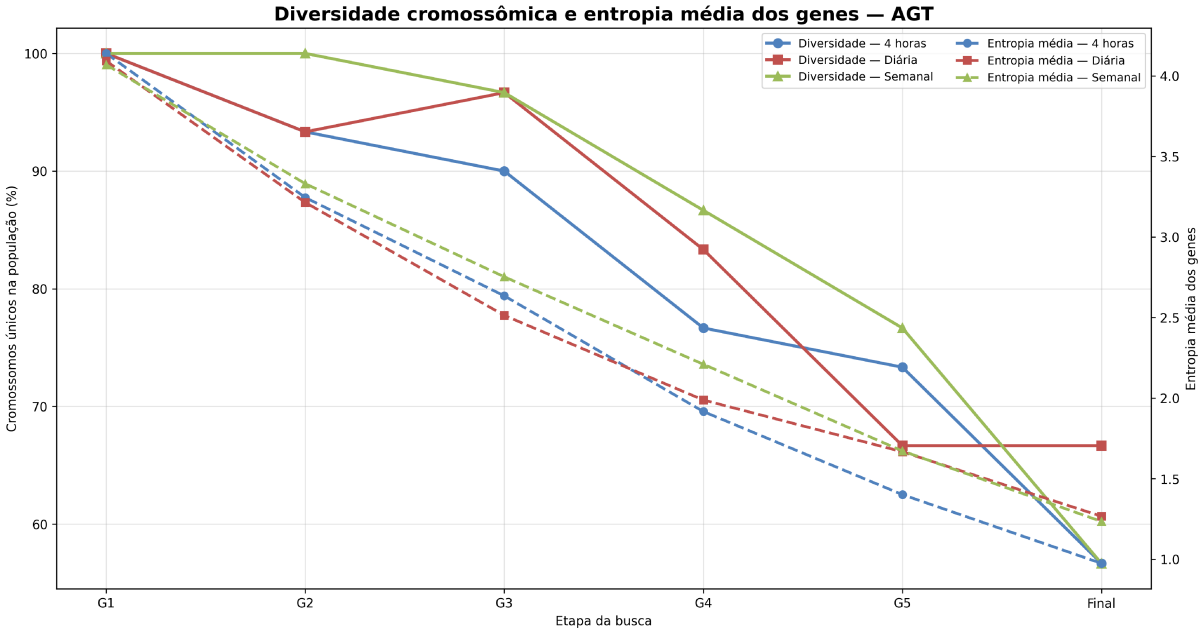

Figura 15 – Evolução da diversidade cromossômica e da entropia média dos genes durante a otimização pelo Algoritmo Genético Transgênico. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

A Figura 15 mostra que, nas três granularidades temporais, a diversidade cromossômica apresentou redução gradual ao longo das gerações, comportamento esperado em algoritmos evolutivos à medida que indivíduos mais aptos passam a predominar na população.

A entropia média dos genes apresentou tendência semelhante, indicando diminuição progressiva da variabilidade genética durante a otimização. Entretanto, diferentemente

de uma convergência abrupta, observa-se que ambas as medidas permaneceram em níveis suficientes para preservar a exploração do espaço de busca durante todo o processo evolutivo.

Esses resultados indicam que a introdução do operador transgênico não comprometeu a diversidade populacional, permitindo que o algoritmo mantivesse simultaneamente mecanismos de exploração e intensificação da busca até a seleção da configuração final.

4.3.3 Hiperparâmetros selecionados pelo Algoritmo Genético Transgênico

Após a conclusão do processo evolutivo, a população final foi novamente avaliada e comparada com o melhor indivíduo global registrado durante toda a execução. Para cada granularidade temporal foi selecionada a configuração que apresentou o menor valor de *fitness* entre todas as soluções avaliadas.

A Tabela 14 apresenta os hiperparâmetros utilizados no treinamento final das redes LSTM otimizadas pelo AGT.

Tabela 14 – Hiperparâmetros selecionados pelo AGT para o treinamento final.

Granularidade	Camadas LSTM	Neurônios	Taxa de aprendizagem	*Batch size*	Épocas	*Dropout*
4 horas	1	45	0,0073	47	79	0,2
Diária	1	42	0,0029	64	83	0,2
Semanal	1	43	0,0088	41	68	0,2

Os hiperparâmetros apresentados na Tabela 14 correspondem às configurações selecionadas automaticamente pelo AGT ao término do processo evolutivo e posteriormente utilizadas no treinamento final das redes LSTM. Observa-se que as três granularidades convergiram para arquiteturas compostas por uma única camada LSTM, porém com diferentes combinações de neurônios, taxa de aprendizagem, tamanho do lote e número de épocas, evidenciando que a otimização preservou características específicas de cada série temporal.

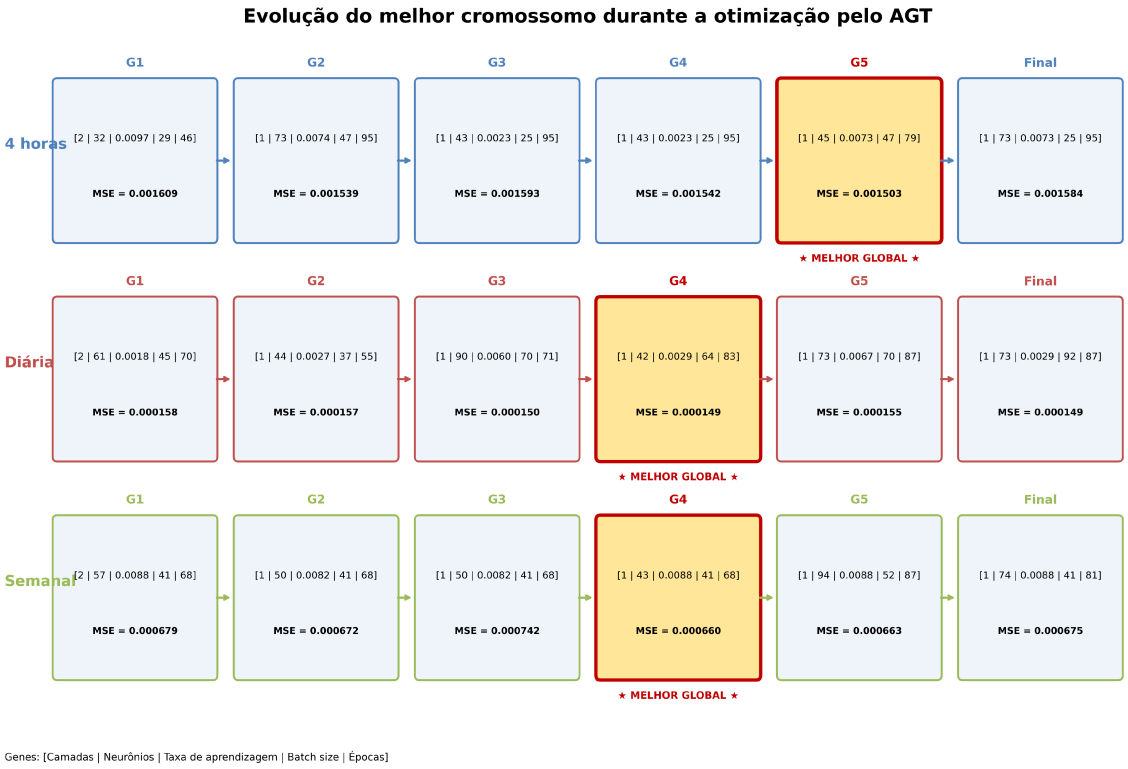

Figura 16 – Evolução do melhor cromossomo identificado em cada geração do Algoritmo Genético Transgênico para as granularidades de 4 horas, diária e semanal.Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

A Figura 16 complementa as informações apresentadas na Tabela 14 ao evidenciar que diferentes combinações de hiperparâmetros competiram entre si antes da convergência da busca. Observa-se que os melhores cromossomos sofreram modificações sucessivas ao longo das gerações, indicando refinamento progressivo das soluções candidatas.

Embora as três granularidades tenham convergido para arquiteturas distintas, verifica-se que a evolução ocorreu de forma gradual, evidenciando que o operador transgênico contribuiu para o refinamento contínuo dos cromossomos até a obtenção da configuração final utilizada no treinamento da rede.

4.3.4 Desempenho preditivo da LSTM otimizada pelo Algoritmo Genético Transgênico

Após a seleção dos melhores hiperparâmetros, foi realizado o treinamento final das redes LSTM utilizando as configurações obtidas pelo AGT para cada granularidade temporal. O desempenho preditivo foi avaliado por meio das métricas MAPE, MSE, RMSE e coeficiente de determinação (R^2), além dos tempos de treinamento e de inferência.

Os resultados obtidos são apresentados na Tabela 15.

Tabela 15 – Resultados obtidos pela LSTM otimizada pelo Algoritmo Genético Transgênico.

Granularidade	MAPE (%)	MSE	RMSE	R^2	Treinamento (s)	Inferência (s)
4 horas	1,3200	29,1575	5,3998	0,9928	52,11	0,68
Diária	2,0744	10,3030	3,2098	0,9960	121,12	0,90
Semanal	3,8395	38,6998	6,2209	0,9850	35,38	0,51

Os resultados apresentados na Tabela 15 evidenciam que os modelos otimizados pelo AGT apresentaram elevado desempenho preditivo nas três granularidades temporais, alcançando coeficientes de determinação superiores a 0,98 em todos os experimentos.

Na série de 4 horas, o modelo obteve MAPE de 1,3200%, MSE de 29,1575, RMSE de 5,3998 e coeficiente de determinação de 0,9928. Entre as três granularidades, essa série apresentou o menor erro percentual médio.

Na série diária, foram obtidos MAPE de 2,0744%, MSE de 10,3030, RMSE de 3,2098 e R^2 igual a 0,9960. Essa granularidade apresentou simultaneamente o maior coeficiente de determinação e os menores valores absolutos de MSE e RMSE.

Para a série semanal, o modelo apresentou MAPE de 3,8395%, MSE de 38,6998, RMSE de 6,2209 e coeficiente de determinação de 0,9850. Embora essa granularidade tenha apresentado os maiores erros relativos entre os três experimentos, o valor de R^2 permaneceu elevado, indicando boa capacidade de representação da dinâmica da série temporal.

Em comparação com a LSTM Clássica, o AGT reduziu o MAPE nas três granularidades temporais. Na série de 4 horas, o erro percentual médio diminuiu de 1,7304% para 1,3200%, correspondendo a uma redução aproximada de 23,7%. Na série diária, o MAPE foi reduzido de 2,1067% para 2,0744%, enquanto na série semanal a redução ocorreu de 4,1328% para 3,8395%.

Na série de 4 horas também foram observadas reduções expressivas no MSE e no RMSE, que passaram de 49,3383 para 29,1575 e de 7,0241 para 5,3998, respectivamente. O coeficiente de determinação aumentou de 0,9883 para 0,9928.

Na série diária, além da redução do MSE de 10,8959 para 10,3030 e do RMSE de 3,3009 para 3,2098, observou-se um pequeno aumento do coeficiente de determinação, que passou de 0,9957 para 0,9960.

Na série semanal, o MSE diminuiu de 52,8440 para 38,6998, enquanto o RMSE foi reduzido de 7,2694 para 6,2209. O coeficiente de determinação aumentou de 0,9795 para 0,9850.

De maneira geral, a otimização realizada pelo AGT produziu melhorias nas métricas

preditivas em relação à arquitetura de referência nas três granularidades temporais analisadas, embora a magnitude desses ganhos tenha variado entre os diferentes conjuntos de dados.

4.3.5 Análise da dinâmica do operador transgênico

Além da análise das métricas preditivas, foram avaliados os registros produzidos durante a execução do operador transgênico com o objetivo de compreender sua atuação ao longo do processo evolutivo. Diferentemente do AG, o AGT registra informações sobre os módulos responsáveis pela geração dos indivíduos transgênicos e sobre a efetiva incorporação de genes provenientes da memória evolutiva, permitindo analisar o comportamento interno do algoritmo durante a otimização.

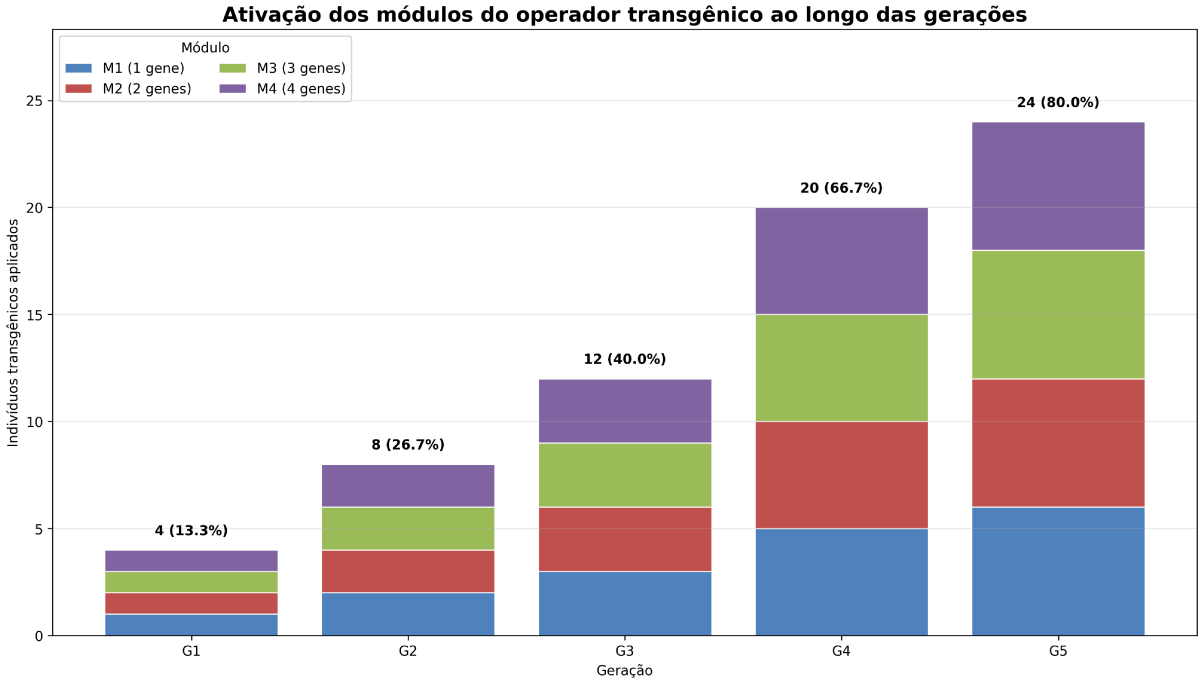

Figura 17 – Ativação dos módulos do operador transgênico ao longo das gerações do processo evolutivo.Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

A Figura 17 apresenta que o número de indivíduos transgênicos aumentou progressivamente ao longo das gerações, acompanhando a estratégia de incremento da taxa transgênica adotada no algoritmo. Observa-se ainda que os quatro módulos participaram de forma equilibrada da geração dos indivíduos transgênicos, indicando que diferentes níveis de modificação genética permaneceram ativos durante toda a execução, sem predominância exclusiva de um único operador.

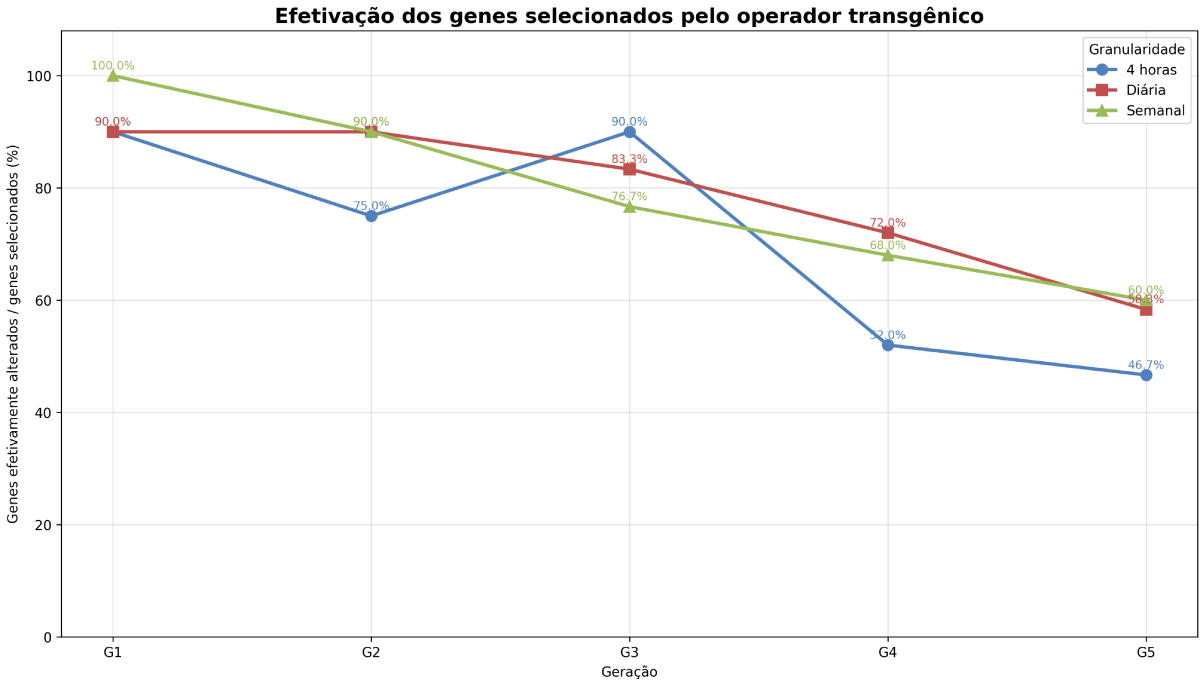

Figura 18 – Proporção de genes efetivamente alterados em relação aos genes selecionados pelo operador transgênico ao longo das gerações. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

A Figura 18 mostra que a proporção de genes efetivamente modificados permaneceu elevada nas gerações iniciais e apresentou redução gradual nas etapas finais da otimização. Esse comportamento indica que, à medida que a população evoluiu, parte dos genes selecionados pelo operador transgênico já coincidia com os valores presentes nos cromossomos-base, reduzindo naturalmente a quantidade de alterações efetivamente realizadas.

Esse resultado sugere que a memória evolutiva permaneceu ativa durante todo o processo de otimização, contribuindo para o refinamento das soluções sem provocar alterações excessivas em populações que já apresentavam elevado nível de adaptação. Dessa forma, os registros produzidos pelo AGT evidenciam que o operador transgênico atuou de maneira progressivamente mais conservadora à medida que a busca se aproximou da convergência.

4.3.6 Análise do custo computacional

Além da qualidade das previsões, foi analisado o custo computacional associado às arquiteturas selecionadas pelo AGT.

A série diária apresentou o maior tempo de treinamento entre os três modelos, totalizando 121,12 segundos. Em seguida, aparecem a série de 4 horas, com 52,11 segundos, e

a série semanal, cujo treinamento foi concluído em 35,38 segundos.

Os tempos de inferência permaneceram reduzidos em todas as granularidades, variando entre 0,51 e 0,90 segundos. Esses resultados indicam que, após concluído o treinamento, as redes realizam previsões de forma computacionalmente eficiente.

Em comparação com a LSTM Clássica, o tempo de treinamento foi reduzido nas três granularidades. Na série de 4 horas, o tempo passou de 104,94 para 52,11 segundos, correspondendo a uma redução aproximada de 50,3%. Na série diária, diminuiu de 254,01 para 121,12 segundos, representando redução de aproximadamente 52,3%. Na série semanal, o tempo passou de 69,33 para 35,38 segundos, resultando em uma redução aproximada de 49,0%.

Essas reduções são compatíveis com as arquiteturas selecionadas pelo AGT, compostas por uma única camada LSTM nas três granularidades. Embora os modelos diário e semanal tenham utilizado 83 e 68 épocas, respectivamente, a menor profundidade da rede contribuiu para limitar o custo computacional do treinamento.

Os resultados mostram que o AGT produziu configurações com menor tempo de treinamento do que a LSTM Clássica em todos os experimentos, mantendo simultaneamente coeficientes de determinação superiores a 0,98. Os tempos de inferência permaneceram inferiores a um segundo, indicando baixo custo computacional para a geração das previsões após o treinamento.

4.3.7 Síntese dos resultados obtidos pelo Algoritmo Genético Transgênico

Os resultados obtidos mostram que o AGT selecionou configurações específicas para cada granularidade temporal e produziu melhorias preditivas em relação à LSTM Clássica. Os ganhos foram mais expressivos nas séries de 4 horas e semanal, enquanto a série diária, que já apresentava elevado desempenho na arquitetura de referência, apresentou melhorias mais discretas.

Além dos resultados preditivos, o AGT reduziu o tempo de treinamento nas três granularidades e manteve tempos de inferência inferiores a um segundo. A análise dos registros internos também mostrou que os módulos transgênicos permaneceram ativos durante toda a execução e que a proporção de genes efetivamente modificados diminuiu nas gerações finais, indicando atuação progressivamente mais conservadora à medida que a população se aproximava da convergência.

Na próxima seção é apresentada uma análise comparativa entre as três abordagens investigadas — LSTM Clássica, AG e AGT — permitindo avaliar conjuntamente o impacto da otimização evolutiva e da incorporação do operador transgênico sobre o desempenho

preditivo e o custo computacional dos modelos.

4.4 Análise comparativa entre as abordagens

Após a análise individual da LSTM Clássica, AG e AGT, esta seção apresenta uma comparação integrada entre as três abordagens. O objetivo é avaliar, de forma conjunta, os efeitos da otimização evolutiva sobre a qualidade das previsões e sobre o custo computacional das redes LSTM, considerando as três granularidades temporais investigadas.

A comparação considera simultaneamente as métricas MAPE, MSE, RMSE, coeficiente de determinação (R^2), tempo de treinamento e tempo de inferência, permitindo analisar os compromissos existentes entre precisão preditiva e eficiência computacional.

Tabela 16 – Comparação entre LSTM Clássica, AG e AGT.

Granularidade	Modelo	MAPE	MSE	RMSE	R^2	Treino (s)	Inferência (s)
4 horas	LSTM Clássica	1,7304	49,3383	7,0241	0,9883	104,94	1,01
	AG	1,1803	27,0210	5,1982	0,9933	109,31	2,62
	AGT	1,3200	29,1575	5,3998	0,9928	52,11	0,68
Diária	LSTM Clássica	2,1067	10,8959	3,3009	0,9957	254,01	2,56
	AG	1,9093	10,9286	3,3058	0,9957	93,70	1,31
	AGT	2,0744	10,3030	3,2098	0,9960	121,12	0,90
Semanal	LSTM Clássica	4,1328	52,8440	7,2694	0,9795	69,33	0,99
	AG	3,6030	39,1101	6,2538	0,9849	46,21	0,50
	AGT	3,8395	38,6998	6,2209	0,9850	35,38	0,51

A Tabela 16 mostra que a utilização de algoritmos evolutivos promoveu melhorias em relação à LSTM Clássica nas três granularidades analisadas. Entretanto, os ganhos obtidos pelo AG e AGT não foram uniformes entre as séries temporais, evidenciando que o comportamento da otimização depende das características específicas dos dados.

Na série de 4 horas, ambos os algoritmos evolutivos reduziram substancialmente os erros de previsão em relação à arquitetura de referência. O AG apresentou os menores valores de MAPE, MSE e RMSE, bem como o maior coeficiente de determinação, enquanto o AGT também produziu melhorias expressivas, embora ligeiramente inferiores às obtidas pelo AG.

Na série diária, observou-se comportamento distinto. Enquanto o AG apresentou o menor erro percentual médio (MAPE), o AGT obteve os menores valores de MSE e RMSE e o maior coeficiente de determinação, indicando melhor capacidade de aproximação dos valores previstos em relação aos valores observados.

Na série semanal, ambos os algoritmos evolutivos novamente superaram a LSTM Clássica. O AG manteve vantagem quanto ao MAPE, enquanto o AGT apresentou menores valores de MSE e RMSE e o maior coeficiente de determinação. Embora as diferenças

entre AG e AGT sejam reduzidas nessa granularidade, os resultados mostram que ambos produziram melhorias consistentes em relação ao modelo de referência.

4.4.1 Comparação do custo computacional

A comparação do custo computacional evidencia que as duas estratégias evolutivas reduziram significativamente o tempo necessário para o treinamento das redes LSTM quando comparadas à arquitetura clássica. Embora o AG tenha apresentado menor tempo de treinamento na série diária, o AGT obteve os menores tempos nas séries de 4 horas e semanal.

Os tempos de inferência permaneceram inferiores a alguns segundos para todas as abordagens, indicando que as diferenças computacionais concentram-se predominantemente na etapa de treinamento dos modelos.

Esses resultados mostram que a otimização evolutiva não apenas melhora a qualidade das previsões, mas também pode reduzir o custo computacional associado ao ajuste dos hiperparâmetros, especialmente quando empregada em conjunto com o operador transgênico.

4.4.2 Síntese comparativa das abordagens

Os resultados obtidos demonstram que os algoritmos evolutivos constituem uma estratégia eficaz para o ajuste automático de hiperparâmetros em redes LSTM aplicadas à previsão de séries temporais financeiras. Tanto o AG quanto o AGT produziram melhorias em relação à arquitetura de referência, embora com comportamentos distintos entre as granularidades analisadas.

De maneira geral, o AG apresentou os menores valores de MAPE nas três séries temporais e obteve o melhor desempenho global na série de 4 horas. O AGT, por sua vez, destacou-se principalmente nas séries diária e semanal, nas quais apresentou menores valores de MSE e RMSE e maiores coeficientes de determinação.

Sob o ponto de vista computacional, ambas as abordagens reduziram o tempo de treinamento em relação à LSTM Clássica. O AGT apresentou os menores tempos de treinamento nas séries de 4 horas e semanal, enquanto o AG foi mais eficiente na série diária.

Além das melhorias preditivas e computacionais, o AGT forneceu informações adicionais sobre a dinâmica do processo evolutivo por meio dos registros produzidos pelo operador transgênico, permitindo analisar aspectos internos da otimização que não estão disponíveis na abordagem baseada exclusivamente no AG. Esse conjunto de evidências reforça o potencial do operador transgênico como mecanismo complementar para o ajuste automático de hiperparâmetros em redes neurais recorrentes.

4.5 Avaliação financeira por meio de *backtesting*

Após a avaliação da capacidade preditiva dos modelos, realizou-se uma análise de seu desempenho financeiro por meio de *backtesting*. Essa etapa permite verificar se a redução das métricas de erro se converteu em resultados economicamente superiores quando as previsões foram utilizadas para orientar operações de compra e venda.

A avaliação considerou o capital final, o lucro líquido, o número de operações encerradas, a taxa de acerto, o fator de lucro e o *drawdown* máximo. Em todos os experimentos, o capital inicial foi fixado em 100.000 unidades monetárias.

Antes da comparação, foi verificada a equivalência dos períodos empregados em cada experimento. Para as granularidades diária e semanal, os três modelos foram avaliados sobre períodos temporais equivalentes. Entretanto, na granularidade de 4 horas, o *backtesting* da LSTM Clássica compreendeu o intervalo entre 12 de março e 26 de junho de 2026, enquanto os modelos otimizados por AG e AGT foram avaliados entre 24 de março e 10 de julho de 2026. Isso aconteceu porque nao foi possivel rodar todos os *backtesting* no mesmo dia.

Em razão dessa diferença, o capital final da LSTM Clássica em 4 horas não foi utilizado em uma comparação financeira direta com os modelos evolutivos. Nessa granularidade, a comparação direta foi realizada apenas entre AG e AGT, que compartilharam o mesmo período de avaliação. Essa precaução evita atribuir diferenças de rentabilidade exclusivamente aos modelos quando parte do resultado também pode decorrer das condições distintas de mercado.

Tabela 17 – Resultados financeiros obtidos no *backtesting*.

Granularidade	Modelo	Capital final	Lucro líquido	Operações	*Win rate* (%)	*Drawdown* (%)	Fator de lucro
4 horas	LSTM Clássica*	371.687,76	271.687,76	154	54,55	-5,86	4,51
	AG	352.880,19	252.880,19	155	54,19	-3,95	4,02
	AGT	326.365,27	226.365,27	154	51,95	-6,31	3,68
Diária	LSTM Clássica	304.805,01	204.805,01	390	33,08	-11,67	1,82
	AG	273.273,55	173.273,55	363	33,61	-6,78	1,71
	AGT	234.866,81	134.866,81	390	30,77	-10,33	1,56
Semanal	LSTM Clássica	112.818,52	12.818,52	70	27,14	-5,91	1,48
	AG	121.537,88	21.537,88	70	31,43	-5,91	1,86
	AGT	118.813,43	18.813,43	71	30,99	-6,38	1,73

*O experimento da LSTM Clássica em 4 horas utilizou período diferente daquele empregado por AG e AGT e, portanto, seu capital final não deve ser comparado diretamente com os demais nessa granularidade.

4.5.1 Resultados do *backtesting* na granularidade de 4 horas

Na granularidade de 4 horas, a comparação financeira direta foi restrita aos modelos AG e AGT, pois ambos foram avaliados no mesmo intervalo, compreendido entre 24 de março e 10 de julho de 2026.

O modelo otimizado pelo AG encerrou o período com capital de 352.880,19, correspondente a um lucro líquido de 252.880,19. O AGT alcançou capital final de 326.365,27 e lucro líquido de 226.365,27. Assim, no mesmo período de mercado, o AG apresentou capital final aproximadamente 7,51% superior ao AGT. Quando se considera apenas o lucro líquido, a vantagem do AG foi de aproximadamente 10,49%.

O AG também apresentou maior taxa de acerto, com 54,19%, contra 51,95% do AGT, e maior fator de lucro, com 4,02 contra 3,68. Além disso, o *drawdown* máximo do AG foi de -3,95%, enquanto o AGT apresentou -6,31%. Dessa forma, nessa granularidade, o AG não apenas gerou maior capital, mas também apresentou menor retração máxima do patrimônio.

Esses resultados são coerentes com a avaliação preditiva em 4 horas, na qual o AG havia obtido os menores valores de MAPE, MSE e RMSE. Portanto, para essa granularidade e nesse período específico, o menor erro preditivo foi acompanhado por melhor desempenho financeiro.

4.5.2 Resultados do *backtesting* na granularidade diária

Na série diária, os três modelos foram avaliados entre 2 de janeiro de 2024 e 30 de dezembro de 2025, permitindo sua comparação direta.

A LSTM Clássica apresentou o maior capital final, com 304.805,01, seguida pelo AG, com 273.273,55, e pelo AGT, com 234.866,81. Em relação à arquitetura clássica, o capital final do AG foi aproximadamente 10,34% menor, enquanto o capital final do AGT foi aproximadamente 22,95% menor.

Quando considerado o lucro líquido, a LSTM Clássica obteve 204.805,01, contra 173.273,55 do AG e 134.866,81 do AGT. Isso representa lucros aproximadamente 15,40% e 34,15% inferiores ao da LSTM Clássica, respectivamente.

Esse resultado contrasta com as métricas preditivas. Na série diária, o AGT apresentou o menor MSE, o menor RMSE e o maior coeficiente de determinação entre os três modelos. Apesar disso, foi o modelo com menor capital final e menor fator de lucro no *backtesting*. Portanto, nessa granularidade, a melhoria das métricas preditivas não se converteu em melhor desempenho financeiro.

O AG apresentou taxa de acerto de 33,61%, ligeiramente superior aos 33,08% da LSTM Clássica. Entretanto, a arquitetura clássica obteve maior fator de lucro, com 1,82, contra 1,71 do AG. Isso indica que a taxa de acerto, isoladamente, também não foi suficiente para explicar o capital acumulado.

Sob a perspectiva do risco, o AG apresentou o menor *drawdown* máximo, de -6,78%, frente a -11,67% da LSTM Clássica e -10,33% do AGT. Assim, embora tenha obtido

capital inferior ao da arquitetura clássica, o AG apresentou a menor retração patrimonial durante o período analisado.

4.5.3 Resultados do *backtesting* na granularidade semanal

Na granularidade semanal, os modelos foram avaliados entre 1º de janeiro de 2024 e 29 de dezembro de 2025. O AG apresentou o maior capital final, com 121.537,88, seguido pelo AGT, com 118.813,43, e pela LSTM Clássica, com 112.818,52.

Em relação à LSTM Clássica, o AG produziu capital final 7,73% superior, enquanto o AGT apresentou aumento de 5,31%. Considerando apenas o lucro líquido, os ganhos relativos tornam-se mais expressivos: o lucro do AG foi aproximadamente 68,02% superior ao da arquitetura clássica, enquanto o lucro do AGT foi 46,77% superior.

O AG também obteve a maior taxa de acerto, com 31,43%, e o maior fator de lucro, com 1,86. O AGT apresentou taxa de acerto de 30,99% e fator de lucro de 1,73, ambos superiores aos valores da LSTM Clássica, de 27,14% e 1,48, respectivamente.

O *drawdown* do AG foi igual ao observado na arquitetura clássica, aproximadamente -5,91%, enquanto o AGT apresentou *drawdown* ligeiramente maior, de -6,38%. Assim, o AG conseguiu elevar o retorno sem aumentar a retração máxima observada no experimento.

Na avaliação preditiva, o AGT havia apresentado MSE e RMSE ligeiramente inferiores aos do AG. Entretanto, o AG produziu maior capital final e maior fator de lucro. Portanto, assim como observado na série diária, o modelo com o menor erro quadrático não foi necessariamente aquele com melhor resultado financeiro.

4.5.4 Relação inicial entre desempenho preditivo e desempenho financeiro

Os resultados mostram que a relação entre as métricas de previsão e o desempenho financeiro não foi uniforme entre as granularidades.

Na série de 4 horas, considerando a comparação direta entre AG e AGT, o modelo com menores erros preditivos também apresentou maior capital final, maior fator de lucro e menor *drawdown*. Entretanto, esse comportamento não se repetiu nas séries diária e semanal.

Na granularidade diária, o AGT apresentou os menores valores de MSE e RMSE, mas obteve o menor capital final entre os três modelos. Na granularidade semanal, o AGT também apresentou o menor MSE e o menor RMSE, porém o AG alcançou o maior capital final e o maior fator de lucro.

Esses resultados indicam que a minimização do erro de magnitude das previsões não garante, isoladamente, melhor resultado financeiro. O *backtesting* depende também da capacidade do modelo de gerar sinais úteis para a estratégia de negociação, particularmente em relação à direção dos movimentos, ao momento das entradas e saídas e à distribuição dos erros ao longo da série.

Dessa forma, a avaliação de modelos destinados a aplicações financeiras não deve ser limitada às métricas tradicionais de regressão. A incorporação de métricas de retorno e risco fornece uma perspectiva complementar e necessária para verificar a utilidade prática das previsões produzidas.

4.6 Análise da dinâmica do operador transgênico

Além da comparação entre os indicadores preditivos e do custo computacional, esta seção analisa a dinâmica interna do Algoritmo Genético Transgênico (AGT) a partir dos registros produzidos durante sua execução. Diferentemente da maioria das implementações de algoritmos evolutivos, nas quais apenas a solução final é preservada, o AGT desenvolvido nesta pesquisa armazenou informações detalhadas sobre a população, os indivíduos transgênicos produzidos, os módulos responsáveis pelas modificações genéticas e a evolução do processo de otimização ao longo das gerações.

Esses registros permitem investigar não apenas a qualidade da solução obtida, mas também compreender como o operador transgênico influenciou a evolução da população durante a busca pelos melhores hiperparâmetros. Dessa forma, a análise deixa de se concentrar exclusivamente nas métricas finais de desempenho e passa a considerar evidências experimentais sobre o comportamento interno do algoritmo.

Ao longo desta seção são discutidas a evolução da diversidade populacional, a atuação dos módulos transgênicos, a efetivação das modificações genéticas e o impacto do operador sobre o processo de convergência da busca, permitindo caracterizar experimentalmente o funcionamento do AGT proposto nesta pesquisa.

Essa estratégia permite alternar períodos de menor intervenção, nos quais a população evolui predominantemente pelos operadores evolutivos convencionais, com períodos de maior intensidade transgênica, destinados a ampliar a exploração do espaço de busca por meio da geração de novos indivíduos.

4.6.1 Diversidade populacional e evolução dos cromossomos

Os registros produzidos durante a execução do AGT permitiram acompanhar a evolução da população ao longo de todas as gerações. Além do melhor indivíduo, foram armazenadas informações referentes à diversidade cromossômica, à distribuição dos hiper-

parâmetros e à variabilidade genética da população, permitindo analisar como o processo evolutivo se desenvolveu durante a otimização.

Enquanto a avaliação tradicional de algoritmos evolutivos normalmente se restringe ao melhor indivíduo encontrado, a análise da população fornece evidências sobre a forma como as soluções evoluíram até a convergência, possibilitando identificar o equilíbrio entre exploração e intensificação da busca.

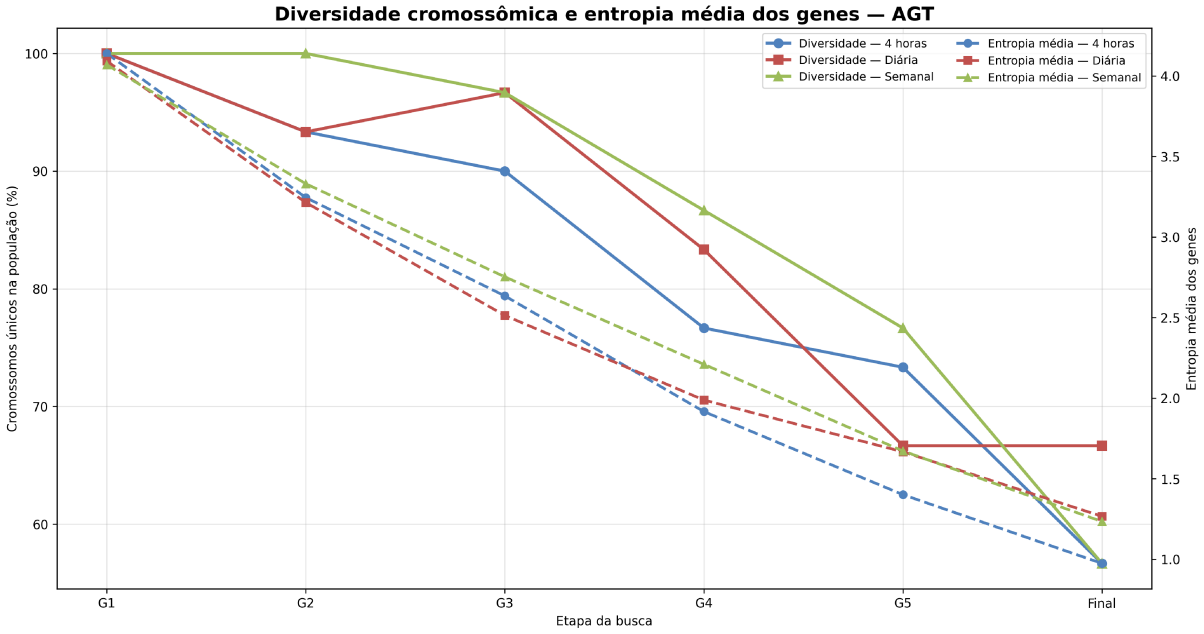

Figura 19 – Evolução da diversidade cromossômica e da entropia média dos genes durante a otimização pelo AGT. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

A Figura 19 mostra que, nas três granularidades temporais, a diversidade cromossômica apresentou redução gradual ao longo das gerações, comportamento esperado em algoritmos evolutivos à medida que indivíduos mais aptos passam a predominar na população.

A entropia média dos genes apresentou tendência semelhante, indicando diminuição progressiva da variabilidade genética sem ocorrência de perda abrupta de diversidade. Dessa forma, observa-se que a população evoluiu de maneira gradual, preservando capacidade exploratória durante todo o processo de otimização.

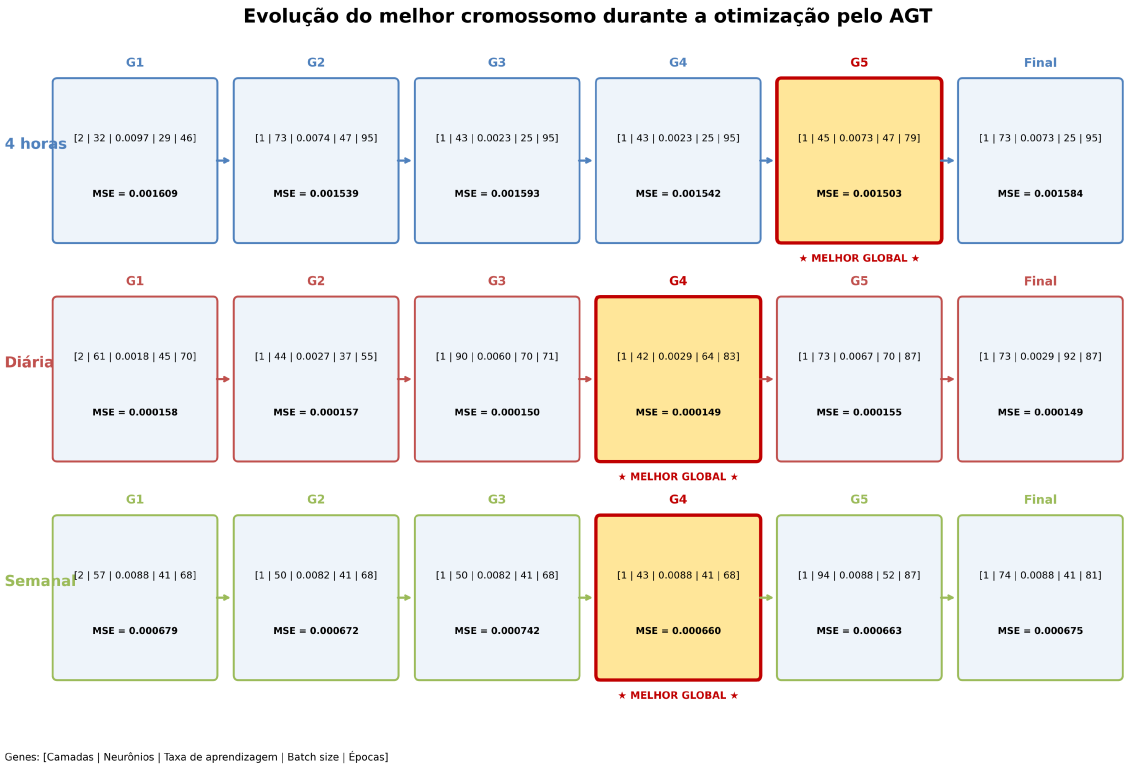

Figura 20 – Evolução do melhor cromossomo identificado em cada geração do AGT para as granularidades de 4 horas, diária e semanal. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

A Figura 20 complementa essa análise ao apresentar a evolução dos melhores cromossomos encontrados em cada geração. Observa-se que diferentes combinações de hiperparâmetros competiram entre si ao longo do processo evolutivo, evidenciando refinamento progressivo das soluções candidatas até a obtenção da configuração final utilizada no treinamento da rede LSTM.

Esse comportamento reforça que a convergência ocorreu de forma gradual, sem mudanças abruptas na estrutura dos melhores indivíduos, indicando estabilidade no processo de otimização.

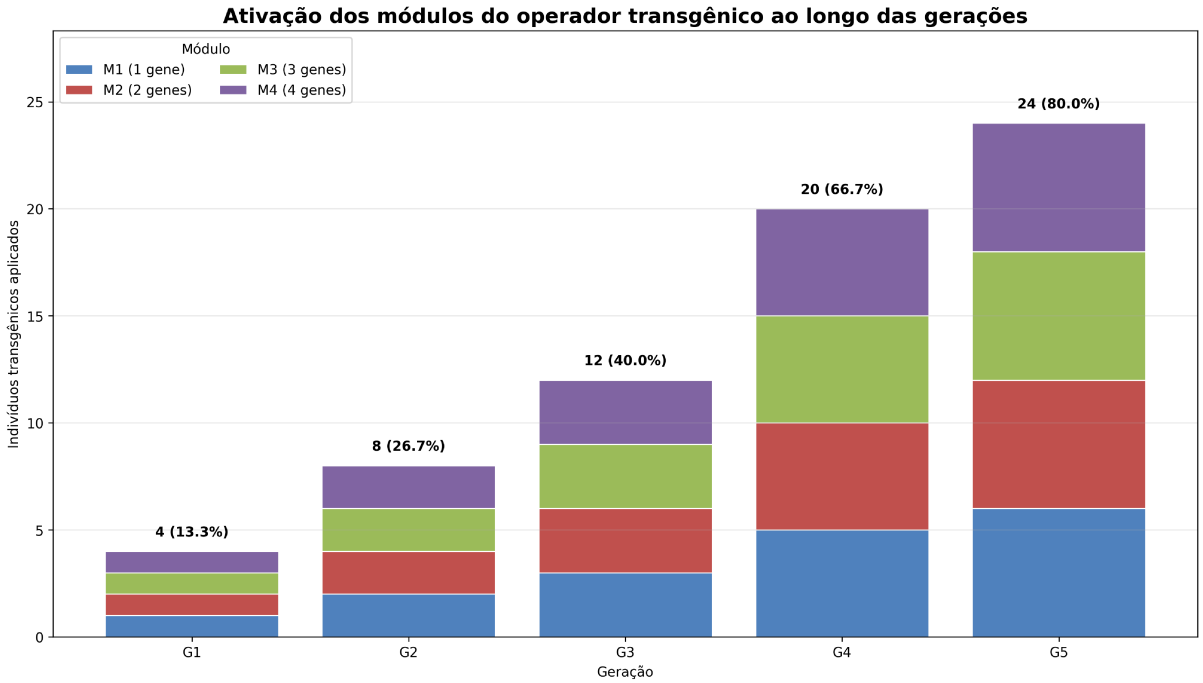

Figura 21 – Ativação dos módulos do operador transgênico ao longo das gerações do processo evolutivo. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

A Figura 21 apresenta a distribuição dos quatro módulos transgênicos durante as cinco gerações do processo evolutivo. Observa-se que o número de indivíduos transgênicos aumentou progressivamente, acompanhando a estratégia de incremento da taxa transgênica adotada no algoritmo.

Verifica-se ainda que os módulos M1, M2, M3 e M4 participaram de forma equilibrada da geração dos indivíduos transgênicos. Dessa forma, o algoritmo manteve simultaneamente mecanismos de baixa, média e alta intensidade de modificação genética, permitindo explorar diferentes níveis de intervenção sobre os cromossomos ao longo da otimização.

Essa distribuição evidencia que o AGT não depende exclusivamente de pequenas alterações locais nem de modificações intensas, mas combina diferentes níveis de intervenção genética durante toda a evolução da população.

Além da distribuição dos módulos, os registros armazenados pelo AGT permitiram acompanhar detalhadamente todas as intervenções realizadas durante o processo evolutivo. Para cada indivíduo transgênico foram registrados os genes selecionados, os genes efetivamente modificados, os valores anteriores, os novos valores atribuídos e o cromossomo resultante após a intervenção.

Esse conjunto de informações fornece elevado nível de rastreabilidade do processo evolutivo, permitindo reconstruir posteriormente cada modificação realizada pelo operador

transgênico e investigar sua contribuição para a evolução da população.

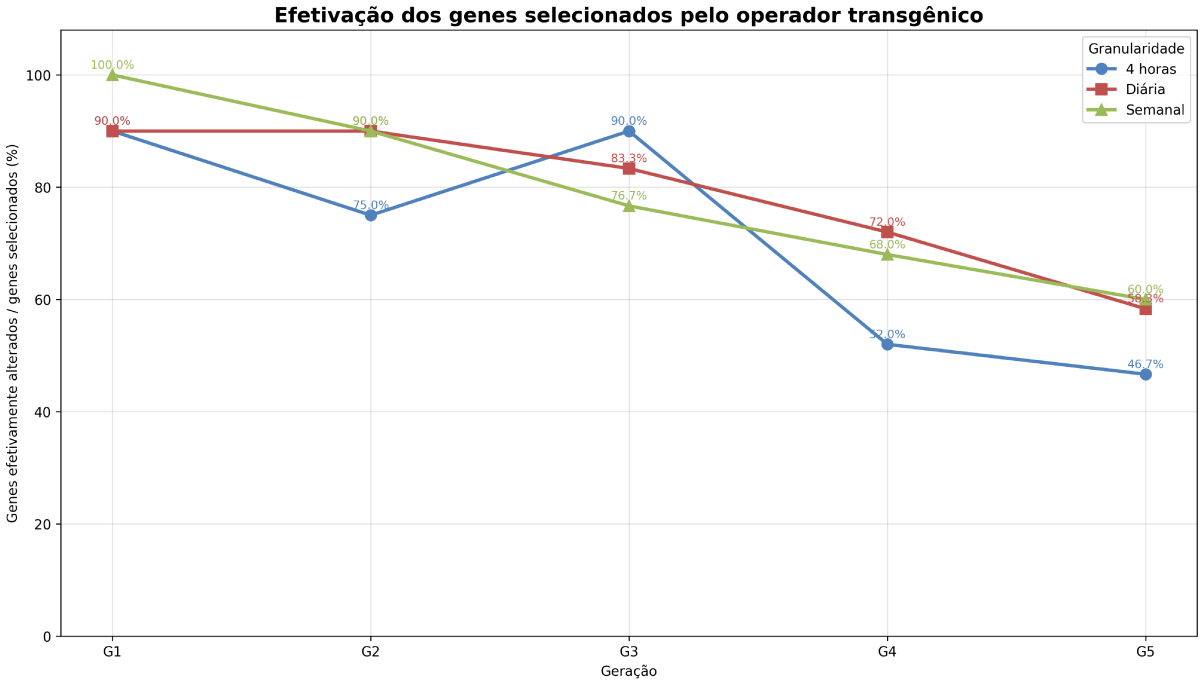

Figura 22 – Proporção de genes efetivamente alterados em relação aos genes selecionados pelo operador transgênico ao longo das gerações. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

A Figura 22 mostra que a proporção de genes efetivamente modificados permaneceu elevada nas primeiras gerações, reduzindo-se gradualmente nas etapas finais da otimização.

Esse comportamento indica que, à medida que a população evoluiu, parte dos genes selecionados pelo operador transgênico já coincidia com os valores presentes nos cromossomos-base. Consequentemente, o operador passou a realizar menos modificações efetivas, comportamento compatível com a aproximação da população em direção à convergência.

Esse resultado mostra que o operador transgênico permaneceu ativo durante todo o processo evolutivo, porém sua atuação tornou-se progressivamente mais conservadora conforme a população se estabilizou.

Outra característica observada nos registros experimentais foi a avaliação explícita do efeito imediato do operador transgênico sobre a população. Para cada geração foram armazenados três indicadores distintos: o melhor *fitness* obtido antes da aplicação do operador, o melhor *fitness* da população provisória produzida pelos operadores evolutivos convencionais e o melhor *fitness* global após a inserção dos indivíduos transgênicos.

Essa estrutura permite analisar separadamente o comportamento dos operadores genéticos tradicionais e a contribuição específica do operador transgênico, tornando possível identificar situações nas quais a população provisória já continha boas soluções e situações em que a introdução dos indivíduos transgênicos produziu melhorias adicionais.

A comparação entre o melhor *fitness* da população provisória e o melhor *fitness* global após a aplicação do operador revelou que a contribuição do mecanismo transgênico não ocorreu de forma uniforme ao longo de todas as gerações.

Nos experimentos realizados foram observadas situações em que a população provisória já continha a melhor solução disponível, não sendo registrada melhoria adicional após a aplicação do operador. Em contrapartida, também foram identificadas gerações nas quais a introdução dos indivíduos transgênicos reduziu imediatamente o valor do MSE, indicando que o operador foi capaz de produzir soluções superiores às obtidas exclusivamente pelos operadores evolutivos convencionais.

Esse comportamento apresenta a contribuição do operador transgênico é dependente do estado evolutivo da população. Em determinadas gerações sua atuação apenas preserva a melhor solução existente; em outras, amplia a exploração do espaço de busca e produz indivíduos capazes de melhorar o melhor *fitness* global do algoritmo.

4.6.2 Convergência da população durante a otimização

Os resultados apresentados anteriormente mostram que a convergência da população ocorreu de maneira gradual ao longo das gerações. A redução progressiva da diversidade cromossômica foi acompanhada pela diminuição da entropia média dos genes, indicando predominância crescente de soluções mais adaptadas sem perda abrupta da capacidade exploratória.

Paralelamente, a evolução dos melhores cromossomos evidenciou refinamento contínuo dos hiperparâmetros, enquanto os registros do operador transgênico mostraram que novas combinações continuaram sendo introduzidas durante toda a execução. Dessa forma, o processo evolutivo combinou exploração de novas regiões do espaço de busca com intensificação sobre soluções promissoras.

Em conjunto, essas evidências indicam que o AGT preservou um equilíbrio entre exploração e convergência, reduzindo a probabilidade de convergência prematura e favorecendo o refinamento progressivo das soluções candidatas.

4.6.3 Impacto do operador transgênico sobre a convergência da busca

A análise conjunta dos registros produzidos durante a execução do AGT mostra que a convergência da população ocorreu de maneira progressiva, resultado da atuação combinada dos operadores evolutivos convencionais e do operador transgênico.

Nas gerações iniciais observou-se elevada diversidade cromossômica e maior quantidade de modificações efetivas realizadas pelo operador transgênico. À medida que o processo evolutivo avançou, a população tornou-se mais homogênea, reduzindo naturalmente o número de alterações efetivas sobre os cromossomos, comportamento compatível com a aproximação da convergência.

Entretanto, mesmo nas gerações finais, os registros mostram que o operador transgênico permaneceu ativo, produzindo novos indivíduos e preservando a capacidade exploratória do algoritmo. Esse comportamento indica que o mecanismo transgênico não atua apenas como uma mutação de maior intensidade, mas como um operador complementar capaz de introduzir novas combinações de hiperparâmetros durante toda a otimização.

Essa estratégia permite alternar períodos de menor intervenção, favorecendo a exploração realizada pelos operadores genéticos convencionais, com períodos de maior intensidade transgênica, destinados a ampliar a geração de novas combinações de hiperparâmetros quando necessário.

Do ponto de vista computacional, o operador transgênico amplia o conjunto de mecanismos disponíveis para exploração do espaço de busca. Enquanto os operadores clássicos de seleção, cruzamento e mutação produzem novas soluções por recombinação e pequenas perturbações genéticas, o mecanismo transgênico permite modificar diretamente subconjuntos específicos do cromossomo, produzindo novas configurações de hiperparâmetros de forma controlada.

Essa estratégia aumenta a flexibilidade do processo evolutivo e fornece uma alternativa complementar para geração de diversidade genética, especialmente em fases nas quais a população já apresenta tendência de convergência.

Consequentemente, o processo evolutivo passa a combinar duas formas complementares de exploração: aquela promovida pelos operadores tradicionais de cruzamento e mutação e aquela produzida pelas intervenções transgênicas realizadas sobre indivíduos previamente selecionados.

Os resultados experimentais apresentados ao longo desta dissertação mostram que o AGT produziu desempenho competitivo em todas as granularidades temporais analisadas, obtendo, em diferentes cenários, reduções nas métricas de erro e no tempo de treinamento quando comparado à arquitetura LSTM clássica.

Entretanto, a principal contribuição desta pesquisa não está restrita às métricas finais de desempenho. Os registros produzidos durante a execução do algoritmo permitiram reconstruir detalhadamente a dinâmica interna da otimização, possibilitando analisar a evolução da população, a atuação dos módulos transgênicos, a efetivação das modificações genéticas e o comportamento da convergência ao longo das gerações.

Esse conjunto de evidências amplia a compreensão sobre o funcionamento do operador transgênico e demonstra que sua contribuição vai além da obtenção de uma solução final de melhor qualidade. Ao disponibilizar informações detalhadas sobre todas as etapas do processo evolutivo, a implementação desenvolvida nesta pesquisa fornece uma base experimental que pode subsidiar futuras investigações sobre mecanismos de otimização evolutiva aplicados ao ajuste automático de hiperparâmetros de redes neurais profundas.

Em síntese, a análise da dinâmica do operador transgênico evidencia que a combinação entre operadores evolutivos tradicionais e intervenções transgênicas controladas favoreceu um processo de busca equilibrado entre exploração e intensificação. Além dos ganhos observados nas métricas preditivas e no custo computacional, os resultados demonstram que o armazenamento detalhado dos registros evolutivos constitui uma contribuição metodológica da implementação proposta, permitindo analisar aspectos internos da otimização que normalmente permanecem ocultos em aplicações convencionais de AG.

4.7 Integração entre desempenho preditivo, desempenho financeiro e complexidade das arquiteturas

As análises apresentadas nas seções anteriores permitiram avaliar individualmente o desempenho preditivo das arquiteturas LSTM, os resultados obtidos pelos processos de otimização evolutiva e o comportamento financeiro das estratégias desenvolvidas a partir das previsões geradas pelos modelos. Entretanto, a análise isolada desses resultados não permite compreender de forma completa as relações existentes entre precisão preditiva, complexidade arquitetural e desempenho operacional.

Nesta seção realiza-se uma análise integrada dos resultados obtidos para as granularidades de 4 horas, diária e semanal, considerando simultaneamente os indicadores estatísticos de previsão, os resultados financeiros do *backtesting* e as características das arquiteturas finais selecionadas para cada modelo. Essa abordagem permite avaliar não apenas qual arquitetura apresentou melhor desempenho em determinado critério, mas também compreender os compromissos estabelecidos entre capacidade preditiva, rentabilidade, controle de risco e simplicidade estrutural.

Diferentemente de avaliações baseadas exclusivamente em métricas de erro, esta pes-

quisa busca analisar se melhorias obtidas durante a etapa de otimização realmente se traduziram em ganhos financeiros consistentes, permitindo identificar situações em que maior precisão estatística não necessariamente implicou melhor desempenho operacional.

4.7.1 Panorama geral dos resultados experimentais

A Tabela 18 apresenta uma visão consolidada dos principais indicadores obtidos pelas arquiteturas LSTM Clássica, LSTM+AG e LSTM+AGT nas três frequências analisadas. Para a comparação entre os modelos foram consideradas exclusivamente as métricas obtidas sobre o conjunto de teste, uma vez que representam a capacidade de generalização das redes neurais em dados não utilizados durante o treinamento. Além das métricas preditivas, também são apresentados os principais indicadores provenientes do *backtesting*, permitindo uma análise conjunta entre qualidade das previsões e desempenho financeiro das estratégias.

Tabela 18 – Síntese integrada do desempenho preditivo e financeiro dos modelos.

Frequência	**Modelo**	**MSE**	**RMSE**	**MAPE (%)**	R^2	**Capital final**	**Fator de lucro**	***Drawdown* (%)**
4 horas	LSTM Clássica	**33,481148**	**5,786290**	**1,644236**	0,884990	**371.687,76**	**4,508539**	5,861025
	LSTM+AG	38,937185	6,239967	1,731598	0,868300	352.880,19	4,019907	**3,954474**
	LSTM+AGT	40,664585	6,376879	1,694796	**0,892418**	326.365,27	3,683492	6,308531
Diária	LSTM Clássica	6,414597	2,532705	1,718472	0,975383	**304.805,01**	**1,821639**	11,668131
	LSTM+AG	**5,106732**	**2,259808**	**1,483605**	**0,980402**	273.273,55	1,711868	**6,776988**
	LSTM+AGT	7,951633	2,819864	1,964696	0,969484	234.866,81	1,556325	10,329792
Semanal	LSTM Clássica	42,716534	6,535789	4,634922	0,811265	112.818,52	1,476210	**5,907620**
	LSTM+AG	**21,385865**	**4,624485**	**3,101156**	**0,905511**	**121.537,88**	**1,856676**	**5,907620**
	LSTM+AGT	23,882849	4,887008	3,372777	0,894478	118.813,43	1,730634	6,378082

Nota: as métricas MSE, RMSE, MAPE e R^2 correspondem ao conjunto de teste das arquiteturas finais. Capital final, fator de lucro e *drawdown* máximo foram extraídos dos resultados do *backtesting*. O *drawdown* é apresentado em valor absoluto, de modo que valores menores representam menor rebaixamento máximo do capital. Os destaques em verde indicam os casos em que um desempenho preditivo favorável esteve associado ao maior capital final dentro da mesma frequência temporal. Os valores em negrito representam o melhor resultado de cada indicador em cada frequência.

A Tabela 18 evidencia que o desempenho dos modelos variou de acordo com a frequência temporal considerada, não sendo possível identificar uma arquitetura dominante em todos os cenários avaliados.

Na frequência de 4 horas, a LSTM Clássica apresentou os menores valores de MSE e RMSE, enquanto a LSTM+AGT obteve o maior coeficiente de determinação, indicando ligeira superioridade na explicação da variabilidade da série. Entretanto, as diferenças observadas entre os três modelos foram relativamente pequenas, sugerindo que as arquiteturas evolutivas mantiveram desempenho preditivo semelhante ao modelo de referência.

Para a série diária, a LSTM+AG apresentou os melhores resultados em todas as principais métricas de generalização, obtendo simultaneamente o menor MSE, menor RMSE, menor MAPE e maior coeficiente de determinação. Esses resultados indicam que o pro-

cesso de otimização evolutiva foi capaz de identificar uma configuração arquitetural mais adequada para representar a dinâmica da série temporal diária.

Na frequência semanal, a superioridade da LSTM+AG tornou-se ainda mais evidente, apresentando os menores erros preditivos e o maior valor de R^2. Esse comportamento demonstra que a otimização evolutiva produziu ganhos consistentes de generalização para séries temporais de maior horizonte de previsão.

Sob a perspectiva financeira, entretanto, observa-se que a superioridade estatística não implica necessariamente maior retorno operacional. Os indicadores de capital final, fator de lucro e *drawdown*, apresentados na Tabela 18, serão discutidos conjuntamente nas próximas subseções, buscando identificar em quais situações a melhoria da qualidade preditiva foi efetivamente convertida em ganhos financeiros.

4.7.2 Arquiteturas finais selecionadas pelos algoritmos evolutivos

Os resultados apresentados anteriormente evidenciam o desempenho obtido pelas arquiteturas finais. Entretanto, para compreender as diferenças observadas entre os modelos, torna-se necessário analisar as configurações arquiteturais selecionadas durante o processo de otimização.

Enquanto a LSTM Clássica utilizou uma arquitetura fixa previamente definida, os modelos LSTM+AG e LSTM+AGT tiveram seus hiperparâmetros ajustados automaticamente por meio de algoritmos evolutivos. Dessa forma, cada granularidade temporal passou a possuir uma arquitetura específica, selecionada em função do desempenho obtido durante o processo de busca.

A Tabela 19 apresenta as melhores soluções identificadas pelos algoritmos evolutivos para cada frequência temporal, incluindo número de camadas ocultas, quantidade de neurônios, taxa de aprendizagem, tamanho do lote, número de épocas e o melhor valor global da função objetivo durante a otimização.

Tabela 19 – Arquiteturas finais selecionadas pelos algoritmos evolutivos.

Freq.	Modelo	Camadas	Neurônios	Learning Rate	Batch	Épocas	Melhor MSE Global
4 horas	LSTM Clássica	2	50	0,0010	32	50	–
	LSTM+AG	2	52	0,0088	70	99	0,001137
	LSTM+AGT	1	45	0,0073	47	79	0,001503
Diário	LSTM Clássica	2	50	0,0010	32	50	–
	LSTM+AG	1	54	0,0032	32	41	0,000148
	LSTM+AGT	1	42	0,0029	64	83	0,000149
Semanal	LSTM Clássica	2	50	0,0010	32	50	–
	LSTM+AG	1	48	0,0096	38	84	0,000657
	LSTM+AGT	1	43	0,0088	41	68	0,000660

A análise da Tabela 19 evidencia que os dois algoritmos evolutivos modificaram substancialmente a arquitetura originalmente adotada pela LSTM Clássica. Em todas as frequências temporais foram selecionadas configurações distintas da arquitetura de referência, demonstrando que uma configuração fixa não representa, necessariamente, a melhor alternativa para diferentes escalas temporais.

Na frequência de 4 horas, o AG manteve duas camadas ocultas, porém aumentou ligeiramente o número de neurônios e promoveu alterações expressivas na taxa de aprendizagem, tamanho do lote e número de épocas. Por outro lado, o AGT selecionou uma arquitetura mais compacta, reduzindo a rede para apenas uma camada oculta com 45 neurônios. Essa redução estrutural foi acompanhada por desempenho preditivo competitivo, culminando no maior coeficiente de determinação entre os três modelos avaliados nessa frequência.

Para a série diária, ambos os algoritmos convergiram para arquiteturas com apenas uma camada oculta, indicando que a segunda camada utilizada pela arquitetura clássica não era necessária para representar adequadamente a dinâmica dessa série temporal. Embora o AG tenha obtido desempenho ligeiramente superior nas métricas de erro, o AGT alcançou resultados muito próximos empregando uma arquitetura ainda mais compacta, com apenas 42 neurônios.

No horizonte semanal observa-se comportamento semelhante. Tanto o AG quanto o AGT reduziram a profundidade da rede para uma única camada, entretanto o AGT novamente selecionou uma arquitetura com menor quantidade de neurônios. Essa redução da complexidade foi obtida preservando desempenho preditivo bastante próximo ao melhor modelo, indicando que soluções estruturalmente mais simples podem apresentar elevada capacidade de generalização.

De maneira geral, observa-se que o AGT demonstrou uma tendência consistente em selecionar arquiteturas menos complexas que aquelas produzidas pelo AG. Esse resultado constitui uma evidência importante da eficiência do mecanismo transgenético, uma vez que redes neurais menores demandam menor custo computacional, reduzem a quantidade de parâmetros ajustáveis e tendem a apresentar menor risco de sobreajuste, preservando simultaneamente níveis de desempenho compatíveis com os melhores modelos obtidos durante a pesquisa.

4.7.3 Relação entre desempenho preditivo e desempenho financeiro

Os resultados apresentados nas Tabelas 18 e 19 permitem investigar uma questão central desta pesquisa: melhorias obtidas nas métricas tradicionais de previsão resultam, necessariamente, em maior desempenho financeiro durante a execução das estratégias de

negociação?

Embora seja intuitivo supor que modelos mais precisos conduzam automaticamente a melhores resultados operacionais, essa relação nem sempre é observada em aplicações financeiras. Pequenas diferenças entre valores previstos e observados podem produzir impactos distintos sobre a decisão de compra ou venda, influenciando diretamente o retorno acumulado da estratégia. Assim, a avaliação exclusiva de métricas estatísticas pode conduzir a interpretações incompletas acerca da efetiva utilidade prática de um modelo preditivo.

Com base nessa perspectiva, esta subseção analisa conjuntamente os indicadores de previsão e os resultados obtidos durante o *backtesting*, buscando identificar em quais situações a melhoria da qualidade preditiva foi efetivamente convertida em maior rentabilidade financeira.

Na frequência de 4 horas observa-se uma relação parcialmente consistente entre desempenho preditivo e desempenho financeiro. A LSTM Clássica apresentou os menores valores de MSE, RMSE e MAPE, alcançando também o maior capital final e o maior fator de lucro entre os três modelos avaliados. Entretanto, o maior coeficiente de determinação foi obtido pela arquitetura LSTM+AGT, evidenciando maior capacidade de explicar a variabilidade da série temporal. Esse resultado demonstra que diferentes métricas estatísticas podem favorecer arquiteturas distintas, dependendo do aspecto do desempenho que se pretende analisar.

Além disso, observa-se que a LSTM-AG apresentou o menor *drawdown* máximo, indicando menor exposição ao risco durante a execução da estratégia. Portanto, embora a arquitetura clássica tenha produzido maior retorno financeiro acumulado, a solução encontrada pelo AG mostrou-se mais conservadora sob a perspectiva de gerenciamento de risco, enquanto o AGT apresentou a melhor capacidade explicativa da série temporal.

Na série diária observa-se um comportamento distinto daquele verificado na granularidade de 4 horas. O AG apresentou superioridade em todas as principais métricas de previsão, obtendo simultaneamente o menor MSE, o menor RMSE, o menor MAPE e o maior coeficiente de determinação. Entretanto, essa superioridade estatística não se refletiu diretamente no maior capital final da estratégia, que permaneceu sendo obtido pela LSTM Clássica.

Esse resultado evidencia que melhorias na precisão das previsões não são, por si só, suficientes para garantir maior retorno financeiro. Em mercados financeiros, diferenças reduzidas entre valores previstos podem alterar o instante de entrada ou saída das operações, produzindo impactos acumulativos sobre o desempenho final da estratégia. Dessa forma, pequenas variações nos erros estatísticos podem resultar em diferenças relevantes de rentabilidade, mesmo quando os modelos apresentam níveis elevados de capacidade

preditiva.

A arquitetura obtida pelo AGT apresentou desempenho estatístico intermediário, permanecendo próxima da solução encontrada pelo AG, porém utilizando uma configuração arquitetural mais compacta. Esse comportamento reforça a capacidade do AGT de produzir soluções equilibradas, conciliando desempenho preditivo e menor complexidade estrutural.

No horizonte semanal observa-se um cenário particularmente interessante. Embora o AG tenha apresentado os menores erros de previsão e o maior capital final, as diferenças observadas em relação ao AGT foram relativamente pequenas. A arquitetura selecionada pelo AGT apresentou valores de MSE, RMSE, MAPE e coeficiente de determinação bastante próximos daqueles obtidos pelo AG, preservando elevada capacidade de generalização mesmo empregando uma configuração estruturalmente mais simples.

Sob a perspectiva financeira, o AGT também apresentou desempenho competitivo, obtendo capital final e fator de lucro próximos aos melhores resultados da pesquisa. Essa proximidade torna-se particularmente relevante quando considerada em conjunto com a redução da complexidade arquitetural observada na seção anterior, indicando que o mecanismo transgenético foi capaz de identificar soluções eficientes sem depender do aumento da capacidade da rede neural.

De maneira geral, os resultados demonstram que a relação entre desempenho preditivo e desempenho financeiro não é linear. Em determinadas situações, como observado na frequência de 4 horas, menores erros estatísticos foram acompanhados por maior rentabilidade. Em outras, como na série diária, arquiteturas com desempenho preditivo superior não produziram, necessariamente, maior capital final.

Esses resultados reforçam que a avaliação de modelos destinados à previsão de séries temporais financeiras deve considerar simultaneamente métricas estatísticas, indicadores financeiros e características arquiteturais. Sob essa perspectiva, o AGT destacou-se por selecionar arquiteturas mais compactas e manter desempenho competitivo em todas as granularidades analisadas, demonstrando que a eficiência de um modelo não deve ser avaliada apenas pela minimização dos erros de previsão, mas também pelo equilíbrio entre capacidade de generalização, simplicidade estrutural e desempenho operacional.

4.7.4 Influência da otimização evolutiva na complexidade das arquiteturas

Os resultados obtidos nesta pesquisa permitem uma análise que vai além da comparação entre métricas de previsão e desempenho financeiro. A própria estrutura das arquiteturas selecionadas pelos algoritmos evolutivos fornece evidências importantes acerca da influência dos processos de otimização sobre a capacidade de representação das séries

temporais analisadas.

A arquitetura LSTM Clássica foi construída utilizando uma configuração fixa, composta por duas camadas ocultas, cinquenta neurônios por camada, taxa de aprendizagem constante, tamanho de lote fixo e cinquenta épocas de treinamento. Embora essa configuração tenha produzido resultados competitivos, ela representa apenas uma escolha empírica realizada antes do início do treinamento, sem considerar as características específicas de cada base temporal.

Em contraste, os algoritmos evolutivos exploraram automaticamente diferentes combinações de hiperparâmetros, permitindo que a arquitetura fosse adaptada às particularidades de cada frequência analisada. Como consequência, foram obtidas configurações distintas para as séries de 4 horas, diária e semanal, evidenciando que não existe uma arquitetura universalmente ótima para todos os horizontes de previsão.

A primeira evidência observada refere-se à redução da profundidade das redes. Enquanto a arquitetura clássica manteve duas camadas ocultas em todas as simulações, tanto o AG quanto o AGT convergiram predominantemente para arquiteturas compostas por apenas uma camada LSTM nas séries diária e semanal.

Esse comportamento sugere que o aumento da profundidade da rede não implicou, necessariamente, melhoria da capacidade de generalização. Pelo contrário, os algoritmos evolutivos identificaram que arquiteturas menos profundas eram suficientes para representar a dinâmica temporal dessas séries, reduzindo a quantidade de parâmetros treináveis sem comprometer a qualidade das previsões.

Entre os dois métodos de otimização, observa-se um comportamento particularmente interessante do AGT. Os algoritmos evolutivos modificaram a complexidade das arquiteturas de maneiras distintas. Enquanto o AG reduziu a profundidade nas frequências diária e semanal, na série de 4 horas manteve duas camadas e produziu uma arquitetura ligeiramente maior que a configuração clássica. O AGT, por sua vez, reduziu a quantidade de parâmetros em todas as frequências analisadas.

Na frequência de 4 horas, por exemplo, o AG selecionou uma arquitetura com 52 neurônios distribuídos em duas camadas, enquanto o AGT convergiu para uma rede composta por apenas uma camada contendo 45 neurônios. Situação semelhante ocorreu nas frequências diária e semanal, nas quais o AGT novamente produziu arquiteturas estruturalmente menores.

Esse comportamento evidencia que o mecanismo transgenético não direcionou a busca apenas para a redução do erro de treinamento, mas também favoreceu configurações capazes de manter elevado desempenho utilizando menor complexidade estrutural.

Sob a perspectiva da aprendizagem de máquina, arquiteturas mais compactas apresentam vantagens relevantes. Redes neurais com menor quantidade de parâmetros de-

mandam menor esforço computacional durante o treinamento e a inferência, reduzem o consumo de memória e tendem a apresentar menor propensão ao sobreajuste (*overfitting*), especialmente quando o volume de dados disponíveis é limitado.

Embora arquiteturas maiores possuam maior capacidade de representação, essa capacidade adicional nem sempre se traduz em melhoria efetiva da generalização. Em diversos problemas de previsão de séries temporais, estruturas excessivamente complexas podem aprender padrões específicos do conjunto de treinamento, reduzindo sua capacidade de adaptação a dados não observados.

Os resultados obtidos nesta pesquisa reforçam essa interpretação. Mesmo utilizando arquiteturas mais compactas, o AGT apresentou desempenho estatístico competitivo em todas as frequências analisadas, alcançando, inclusive, o maior coeficiente de determinação na série de 4 horas e mantendo diferenças reduzidas em relação ao AG nas séries diária e semanal.

Outro aspecto relevante refere-se à eficiência do processo de busca. Embora o AG tenha obtido os menores erros em parte dos experimentos, as soluções encontradas pelo AGT permaneceram muito próximas desses valores, porém utilizando arquiteturas sistematicamente menores. Sob essa perspectiva, o AGT apresentou uma relação favorável entre complexidade estrutural e desempenho preditivo.

Essa característica torna-se particularmente importante em aplicações reais, nas quais modelos mais compactos tendem a apresentar menor custo de implantação, menor tempo de inferência e maior facilidade de manutenção, especialmente quando utilizados em sistemas de negociação automatizada ou ambientes com restrições computacionais.

De maneira geral, os resultados obtidos demonstram que a utilização de algoritmos evolutivos ultrapassa a simples otimização de hiperparâmetros. Os processos evolutivos atuaram como mecanismos automáticos de seleção de arquiteturas, adaptando a complexidade das redes neurais às características de cada frequência temporal analisada.

Nesse contexto, o AGT destacou-se por produzir soluções estruturalmente mais simples, mantendo desempenho estatístico e financeiro competitivo em relação ao AG. Essa evidência reforça a principal contribuição desta pesquisa: demonstrar que a incorporação de operadores transgenéticos pode favorecer a obtenção de arquiteturas mais eficientes, conciliando capacidade preditiva, simplicidade estrutural e potencial de aplicação em cenários reais de previsão de séries temporais financeiras.

4.7.5 Análise quantitativa da complexidade das arquiteturas

Embora o número de camadas e de neurônios forneça uma indicação inicial da complexidade estrutural das redes, uma comparação mais objetiva pode ser realizada por meio da quantidade de parâmetros treináveis de cada arquitetura. Essa medida representa o

número de pesos e vieses ajustados durante o treinamento, estando diretamente relacionada ao consumo de memória, ao custo computacional e à capacidade de representação do modelo.

Em todos os experimentos, a entrada da rede foi constituída por uma única variável, correspondente ao preço de fechamento, organizada em janelas temporais com os 30 registros anteriores. A quantidade de observações da janela determina o comprimento da sequência apresentada à rede, mas não altera diretamente o número de parâmetros treináveis. Assim, as diferenças de complexidade decorreram do número de camadas e de neurônios selecionados em cada arquitetura.

4.7.5.1 Quantificação da complexidade das arquiteturas

A complexidade estrutural das arquiteturas não depende apenas do número de camadas e de neurônios, mas também da quantidade de parâmetros treináveis, que representa o número total de pesos e vieses ajustados durante o processo de treinamento.

Na implementação padrão da camada `LSTM` disponibilizada pelo *TensorFlow/Keras*, cada célula é composta por quatro componentes internos *(forget gate, input gate, candidate gate e output gate)*, sendo que cada componente possui pesos associados às entradas, pesos recorrentes e um vetor de vieses.

$$P_{\text{LSTM}} = 4\left[(d+u)u + u\right], \tag{12}$$

em que P_{LSTM} representa a quantidade de parâmetros treináveis da camada, u corresponde ao número de unidades (neurônios) da camada LSTM e d representa a quantidade de atributos de entrada.

Observa-se que o primeiro termo, $(d+u)u$, representa os pesos associados às conexões de entrada e às conexões recorrentes, enquanto o segundo termo (u) corresponde aos vieses da camada. Como existem quatro componentes internos na célula LSTM, esse conjunto de parâmetros é multiplicado por quatro.

Desenvolvendo a Equação (12), obtém-se

$$\begin{aligned} P_{\text{LSTM}} &= 4\left[(d+u)u + u\right] \\ &= 4\left(du + u^2 + u\right) \\ &= 4u(d+u+1). \end{aligned} \tag{13}$$

Como neste trabalho foi utilizada apenas uma variável de entrada, correspondente ao preço de fechamento do ativo financeiro, tem-se

$$d = 1.$$

Assim, a Equação (13) pode ser simplificada para

$$P_{\text{LSTM}} = 4u(u+2). \tag{14}$$

A fim de ilustrar a aplicação da Equação (13), considere a primeira camada da arquitetura LSTM Clássica.

$$P_1 = 4 \times 50 \times (50 + 1 + 1) = 10\,400.$$

A segunda camada recebe como entrada as 50 saídas produzidas pela primeira camada, resultando em

$$P_2 = 4 \times 50 \times (50 + 50 + 1) = 20\,200.$$

Por fim, a camada densa responsável pela previsão adiciona

$$P_{\text{Dense}} = 50 + 1 = 51$$

parâmetros.

Assim, o número total de parâmetros treináveis da arquitetura LSTM Clássica é dado por

$$P_{\text{Total}} = 10\,400 + 20\,200 + 51 = 30\,651.$$

O mesmo procedimento foi aplicado às demais arquiteturas, alterando apenas o número de camadas e de neurônios selecionados pelos algoritmos evolutivos. Os resultados obtidos são apresentados na Tabela 20.

Tabela 20 – Complexidade das arquiteturas finais em quantidade de parâmetros treináveis.

Frequência	**Modelo**	**Camadas LSTM**	**Neurônios por camada**	**Parâmetros treináveis**	**Variação (%)**
4 horas	LSTM Clássica	2	50	30.651	–
	LSTM+AG	2	52	33.125	+8,07
	LSTM+AGT	1	45	8.506	-72,25
Diária	LSTM Clássica	2	50	30.651	–
	LSTM+AG	1	54	12.151	-60,36
	LSTM+AGT	1	42	7.435	-75,74
Semanal	LSTM Clássica	2	50	30.651	–
	LSTM+AG	1	48	9.649	-68,52
	LSTM+AGT	1	43	7.784	-74,60

Nota: valores negativos na coluna de variação indicam redução da quantidade de parâmetros em relação à LSTM Clássica. O valor positivo representa aumento da complexidade arquitetural.

Os resultados apresentados na Tabela 20 evidenciam que a redução do número de camadas produziu impacto expressivo sobre a complexidade das redes. A arquitetura LSTM Clássica apresentou 30.651 parâmetros treináveis em todas as frequências, uma vez que sua configuração foi mantida fixa ao longo dos experimentos.

Na frequência de 4 horas, o AG manteve duas camadas LSTM e ampliou o número de neurônios de 50 para 52, resultando em 33.125 parâmetros treináveis. Esse valor representa aumento de aproximadamente 8,07% em relação à arquitetura clássica. Portanto, nesse horizonte, o AG não promoveu redução da complexidade, mas selecionou uma rede ligeiramente maior.

Em contraste, o AGT reduziu a arquitetura de 4 horas para uma única camada com 45 neurônios, totalizando 8.506 parâmetros. Essa configuração corresponde a uma redução de aproximadamente 72,25% em relação à LSTM Clássica, mantendo, ainda assim, desempenho preditivo competitivo.

Nas frequências diária e semanal, tanto o AG quanto o AGT selecionaram arquiteturas com apenas uma camada LSTM. No horizonte diário, o AG reduziu a quantidade de parâmetros em 60,36%, enquanto o AGT alcançou redução de 75,74%. Na frequência semanal, as reduções foram de 68,52% para o AG e 74,60% para o AGT.

De modo geral, o AGT apresentou a menor arquitetura em todas as granularidades analisadas. Esse resultado demonstra que o mecanismo transgenético favoreceu soluções estruturalmente mais compactas, especialmente nas frequências diária e de 4 horas. Entretanto, a menor quantidade de parâmetros deve ser analisada conjuntamente com as métricas preditivas e financeiras, uma vez que a redução da complexidade, por si só, não determina a superioridade global de um modelo.

A Figura 23 evidencia visualmente o efeito das configurações selecionadas pelos algoritmos evolutivos sobre a complexidade das redes. Na frequência de 4 horas, o AG produziu uma arquitetura ligeiramente maior que a LSTM Clássica, enquanto o AGT reduziu substancialmente a quantidade de parâmetros. Nas frequências diária e semanal, ambos os métodos evolutivos selecionaram redes menores que a arquitetura de referência, com destaque para o AGT, que apresentou as configurações mais compactas nas três granularidades analisadas.

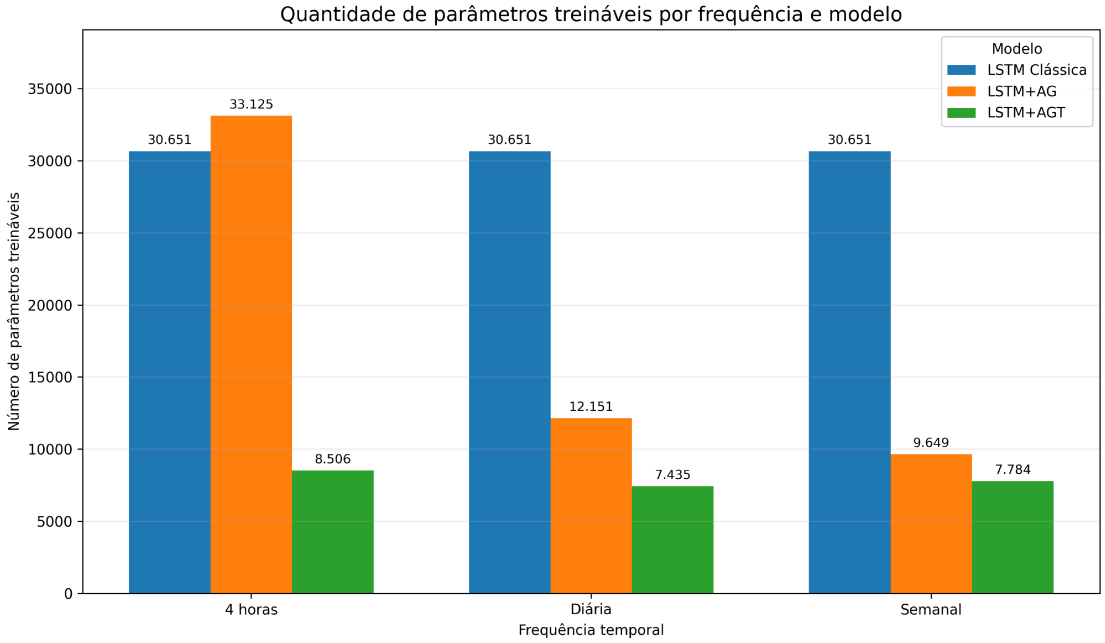

Figura 23 – Quantidade de parâmetros treináveis das arquiteturas finais por frequência temporal e modelo. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

4.7.6 Relação entre complexidade arquitetural, desempenho preditivo e desempenho financeiro

Os resultados obtidos demonstram que a avaliação de modelos de previsão financeira não deve ser baseada exclusivamente em métricas estatísticas de erro ou em indicadores financeiros isolados. Embora essas medidas sejam fundamentais para caracterizar o desempenho dos modelos, elas representam perspectivas distintas do problema e podem conduzir a conclusões diferentes quando analisadas separadamente.

Neste trabalho foram considerados três critérios complementares de avaliação: (i) desempenho preditivo, representado pelas métricas MSE, RMSE, MAPE e R^2; (ii) desempenho financeiro, avaliado por meio do *backtesting*; e (iii) complexidade arquitetural, quantificada pela quantidade de parâmetros treináveis das redes neurais. A análise conjunta desses três aspectos permite compreender os benefícios e limitações das arquiteturas obtidas pelos algoritmos evolutivos de forma mais abrangente do que a utilização de apenas um único indicador.

Observou-se que reduções expressivas na complexidade arquitetural não implicaram necessariamente perdas significativas de desempenho preditivo. De forma semelhante, a obtenção do menor erro estatístico não resultou, obrigatoriamente, no maior retorno financeiro durante o *backtesting*. Esses resultados evidenciam que existe um compromisso (*trade-off*) entre capacidade de representação da rede, qualidade das previsões e desem-

penho operacional das estratégias de negociação.

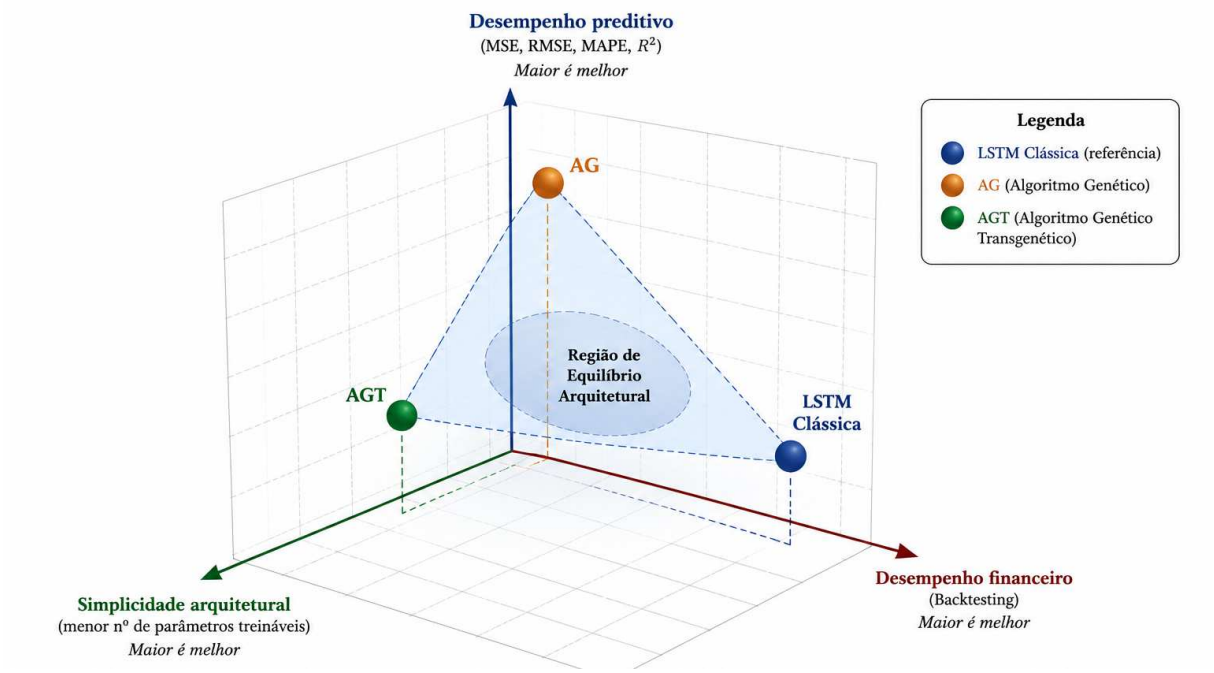

Figura 24 – Relação tridimensional entre simplicidade arquitetural, desempenho preditivo e desempenho financeiro. O posicionamento dos modelos representa qualitativamente o comportamento predominante observado nos experimentos, enquanto a região central indica o conceito de Equilíbrio Arquitetural. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

A Figura 24 sintetiza os três critérios utilizados na avaliação das arquiteturas desenvolvidas neste estudo: desempenho preditivo, simplicidade arquitetural e desempenho financeiro. Enquanto as métricas estatísticas avaliam a capacidade de previsão dos modelos e o *backtesting* mensura sua efetividade em um cenário de negociação, a simplicidade arquitetural representa o custo estrutural da rede, quantificado pela quantidade de parâmetros treináveis.

Observa-se que esses critérios não são necessariamente convergentes. Uma arquitetura pode apresentar excelente desempenho preditivo, porém exigir maior complexidade computacional, enquanto outra pode possuir estrutura mais simples sem comprometer significativamente os resultados obtidos. Da mesma forma, menores erros estatísticos não implicam, obrigatoriamente, maior retorno financeiro, evidenciando que diferentes critérios de avaliação podem conduzir a interpretações distintas sobre a qualidade de um modelo.

Diante dessa característica, este trabalho propõe o conceito de **Equilíbrio Arquitetural**, entendido como o compromisso entre precisão preditiva, simplicidade arquitetural e desempenho financeiro. Sob essa perspectiva, a seleção de uma arquitetura não deve

considerar apenas um único indicador de desempenho, mas o equilíbrio entre esses três aspectos, buscando modelos que conciliem boa capacidade de generalização, menor complexidade estrutural e resultados financeiros consistentes.

Os resultados obtidos indicam que os algoritmos evolutivos favoreceram esse equilíbrio de maneiras distintas. O AG priorizou a melhoria do desempenho preditivo, enquanto o AGT produziu arquiteturas significativamente mais compactas, preservando desempenho competitivo nas métricas estatísticas e nos resultados financeiros. Dessa forma, nenhuma arquitetura pode ser considerada superior em todos os critérios simultaneamente, reforçando a importância de uma análise multicritério para a seleção de modelos de previsão de séries temporais financeiras.

4.7.7 Eficiência arquitetural

Embora a quantidade de parâmetros treináveis constitua uma medida objetiva da complexidade estrutural das arquiteturas, essa informação, isoladamente, não é suficiente para avaliar a eficiência de um modelo. Arquiteturas maiores podem apresentar elevado poder de representação, porém demandam maior custo computacional durante o treinamento e a inferência. Em contrapartida, arquiteturas mais compactas podem produzir desempenho semelhante utilizando quantidade significativamente menor de parâmetros.

Sob essa perspectiva, torna-se relevante analisar não apenas a complexidade absoluta das arquiteturas, mas também o desempenho obtido em relação ao custo computacional necessário para alcançá-lo. Essa abordagem permite identificar modelos capazes de oferecer melhor equilíbrio entre simplicidade estrutural e resultados práticos, complementando as análises realizadas nas seções anteriores.

Neste trabalho, a eficiência arquitetural é interpretada como a relação entre os resultados obtidos durante o *backtesting* e a quantidade de parâmetros treináveis utilizada por cada arquitetura. Dessa forma, modelos que produzem retornos financeiros competitivos empregando menor quantidade de parâmetros podem ser considerados estruturalmente mais eficientes do que arquiteturas que necessitam de maior capacidade para atingir desempenho semelhante.

O capital final por mil parâmetros foi calculado por meio da Equação (15):

$$E_{\text{capital}} = \frac{C_f}{P/1000}, \tag{15}$$

em que C_f representa o capital final obtido no *backtesting* e P corresponde à quantidade de parâmetros treináveis da arquitetura.

De forma análoga, o fator de lucro por mil parâmetros foi determinado por

$$E_{\mathrm{FL}} = \frac{FL}{P/1000}, \tag{16}$$

em que FL representa o fator de lucro da estratégia.

Tabela 21 – Eficiência arquitetural das arquiteturas finais em relação ao desempenho financeiro.

Frequência	Modelo	Parâmetros	Capital final	Capital/1.000 parâmetros	Fator de lucro	Fator de lucro/1.000 parâmetros
4 horas	LSTM Clássica	30.651	371.687,76	12.126,45	4,508539	0,147093
	LSTM+AG	33.125	352.880,19	10.652,99	4,019907	0,121356
	LSTM+AGT	8.506	326.365,27	**38.368,83**	3,683492	**0,433046**
Diária	LSTM Clássica	30.651	304.805,01	9.944,37	1,821639	0,059432
	LSTM+AG	12.151	273.273,55	22.489,80	1,711868	0,140883
	LSTM+AGT	7.435	234.866,81	**31.589,35**	1,556325	**0,209324**
Semanal	LSTM Clássica	30.651	112.818,52	3.680,75	1,476210	0,048162
	LSTM+AG	9.649	121.537,88	12.595,90	1,856676	0,192422
	LSTM+AGT	7.784	118.813,43	**15.263,80**	1,730634	**0,222332**

Nota: os indicadores relativos foram calculados dividindo-se o capital final e o fator de lucro pela quantidade de parâmetros treináveis, expressa em milhares. Valores mais elevados indicam maior desempenho financeiro relativo por unidade de complexidade arquitetural. O negrito destaca o melhor resultado relativo em cada frequência temporal.

A Tabela 21 evidencia que a análise da eficiência arquitetural conduz a interpretações distintas daquelas obtidas quando se considera apenas o capital final ou as métricas tradicionais de desempenho financeiro. Ao relacionar os resultados do *backtesting* com a quantidade de parâmetros treináveis, torna-se possível avaliar o retorno obtido em função da complexidade estrutural necessária para produzi-lo.

Na frequência de 4 horas, embora a LSTM Clássica tenha apresentado o maior capital final absoluto, a arquitetura obtida pelo AGT destacou-se por alcançar o maior capital final por mil parâmetros treináveis. Esse comportamento decorre da expressiva redução da complexidade estrutural promovida pelo AGT, que utilizou aproximadamente um quarto da quantidade de parâmetros da arquitetura clássica, preservando desempenho financeiro competitivo.

Comportamento semelhante foi observado nas frequências diária e semanal. Em ambos os casos, a arquitetura produzida pelo AGT apresentou os maiores valores de capital final por mil parâmetros e de fator de lucro por mil parâmetros, superando tanto a LSTM Clássica quanto a arquitetura otimizada pelo AG. Esses resultados indicam melhor aproveitamento da capacidade computacional disponível, evidenciando que arquiteturas estruturalmente mais compactas podem entregar maior retorno relativo por unidade de complexidade.

Observa-se ainda que o AG ocupou posição intermediária em praticamente todas as comparações. Embora tenha produzido arquiteturas menores que a LSTM Clássica nas frequências diária e semanal, seu ganho de eficiência permaneceu inferior ao obtido pelo AGT, que apresentou desempenho consistente nas três granularidades analisadas.

Esses resultados reforçam que a eficiência arquitetural não depende exclusivamente do desempenho financeiro absoluto, mas da capacidade do modelo em converter recursos computacionais em resultados operacionais. Sob essa perspectiva, as arquiteturas obtidas pelo AGT apresentaram o melhor equilíbrio entre simplicidade estrutural e desempenho financeiro, corroborando a proposta do conceito de Equilíbrio Arquitetural desenvolvido nesta pesquisa.

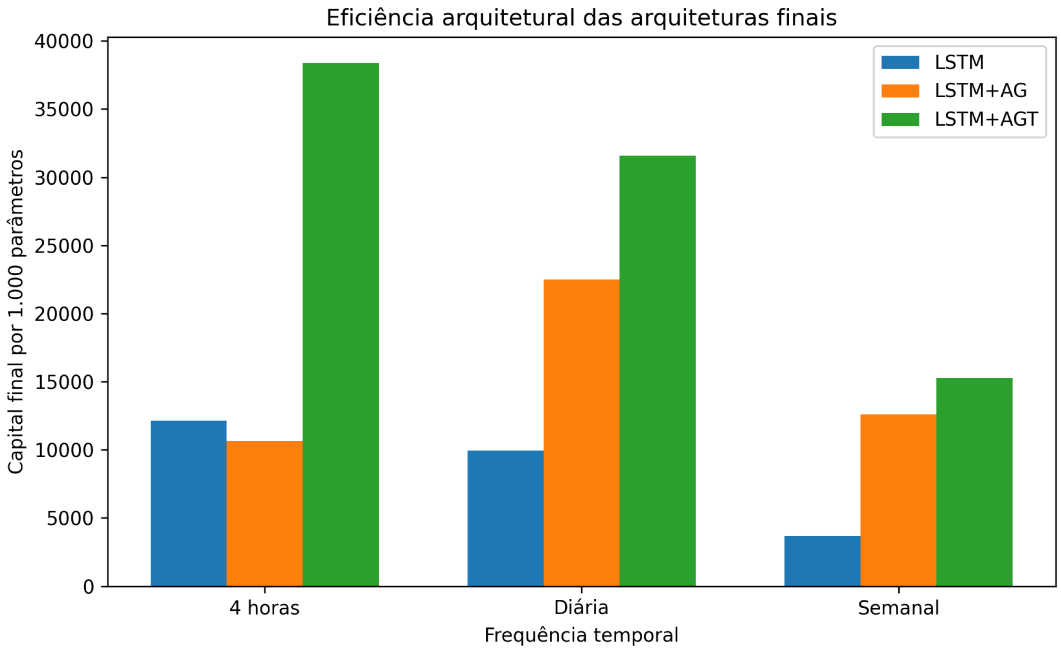

Figura 25 – Eficiência arquitetural das arquiteturas finais, expressa pelo capital final obtido por mil parâmetros treináveis. Valores mais elevados indicam maior retorno financeiro relativo à complexidade estrutural da rede neural. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

A Figura 25 sintetiza a relação entre o capital final obtido pelas estratégias e a complexidade estrutural das arquiteturas. Observa-se que, embora a LSTM Clássica tenha apresentado os maiores valores absolutos de capital final em parte dos experimentos, a normalização em função da quantidade de parâmetros treináveis altera significativamente essa interpretação. Em todas as frequências analisadas, a arquitetura produzida pelo AGT apresentou o maior capital final por mil parâmetros, evidenciando melhor aproveitamento dos recursos computacionais empregados. Esse comportamento reforça que arquiteturas mais compactas podem alcançar desempenho financeiro competitivo com menor custo estrutural, fortalecendo o conceito de Equilíbrio Arquitetural proposto nesta pesquisa.

Os resultados dessa análise complementam o conceito de Equilíbrio Arquitetural proposto nesta pesquisa. Enquanto as seções anteriores demonstraram que desempenho preditivo, desempenho financeiro e complexidade estrutural devem ser avaliados de forma integrada, a eficiência arquitetural acrescenta uma nova perspectiva ao considerar simul-

taneamente o benefício obtido e o custo computacional necessário para alcançá-lo. Assim, arquiteturas menores que mantêm desempenho competitivo podem apresentar maior eficiência global do que modelos estruturalmente mais complexos, reforçando a importância de critérios multidimensionais na seleção de redes neurais para previsão de séries temporais financeiras.

4.7.8 Análise das operações de negociação

Embora o capital final e o fator de lucro constituam importantes indicadores do desempenho financeiro das estratégias, esses resultados não descrevem integralmente o comportamento operacional dos modelos durante o *backtesting*. Dessa forma, torna-se relevante analisar como cada arquitetura conduziu as operações de compra e venda ao longo do período avaliado.

Nesta subseção são investigados indicadores relacionados à dinâmica das operações realizadas, incluindo quantidade de negociações encerradas, taxa de acerto, lucro líquido e lucro médio por operação. Esses indicadores permitem avaliar não apenas quanto cada estratégia ganhou, mas também como esses resultados foram construídos ao longo das negociações.

Tabela 22 – Indicadores operacionais das estratégias de negociação.

Frequência	Modelo	Operações	Win Rate (%)	Lucro líquido	Lucro médio/operação
4 horas	LSTM Clássica	154	54,55	271.687,76	1.764,21
	LSTM+AG	155	54,19	252.880,19	1.631,49
	LSTM+AGT	154	51,95	226.365,27	1.470,16
Diária	LSTM Clássica	390	33,08	204.805,01	525,14
	LSTM+AG	363	33,61	173.273,55	477,34
	LSTM+AGT	390	30,77	134.866,81	345,81
Semanal	LSTM Clássica	70	27,14	12.818,52	183,12
	LSTM+AG	70	31,43	21.537,88	307,68
	LSTM+AGT	71	30,99	18.813,43	264,98

Nota: o lucro médio por operação foi obtido dividindo-se o lucro líquido pelo número de operações encerradas durante o *backtesting*.

A Tabela 22 evidencia que diferenças relativamente pequenas na quantidade de operações realizadas não foram suficientes para explicar as diferenças observadas no lucro líquido entre os modelos. Em todas as frequências temporais, os três modelos executaram número semelhante de negociações, indicando que os resultados financeiros decorreram predominantemente da qualidade dos sinais produzidos e não da intensidade de negociação.

Na frequência de 4 horas, as três arquiteturas realizaram praticamente o mesmo número de operações, apresentando também taxas de acerto superiores a 50%. Entretanto,

a LSTM Clássica obteve o maior lucro médio por operação, seguida pela arquitetura otimizada pelo AG e, posteriormente, pelo AGT. Esses resultados mostram que pequenas diferenças na qualidade dos sinais foram suficientes para produzir diferenças relevantes no retorno acumulado ao final do período.

Na frequência diária, o comportamento foi distinto. Embora o AG tenha apresentado a maior taxa de acerto, a LSTM Clássica obteve o maior lucro médio por operação. Esse resultado reforça que o percentual de operações vencedoras, isoladamente, não determina o desempenho financeiro da estratégia, uma vez que operações lucrativas de maior magnitude podem compensar uma taxa de acerto inferior.

Na granularidade semanal, o AG apresentou simultaneamente a maior taxa de acerto e o maior lucro médio por operação, alcançando o melhor desempenho operacional entre os modelos avaliados. O AGT apresentou comportamento bastante próximo, enquanto a arquitetura clássica obteve os menores valores para ambos os indicadores.

De maneira geral, observa-se que o desempenho operacional depende não apenas da quantidade de operações vencedoras, mas também da capacidade da estratégia de capturar movimentos suficientemente rentáveis para compensar as perdas incorridas ao longo do período analisado. Essa constatação reforça a necessidade de avaliar conjuntamente indicadores de retorno, risco e qualidade operacional durante a comparação de modelos destinados à previsão de séries temporais financeiras.

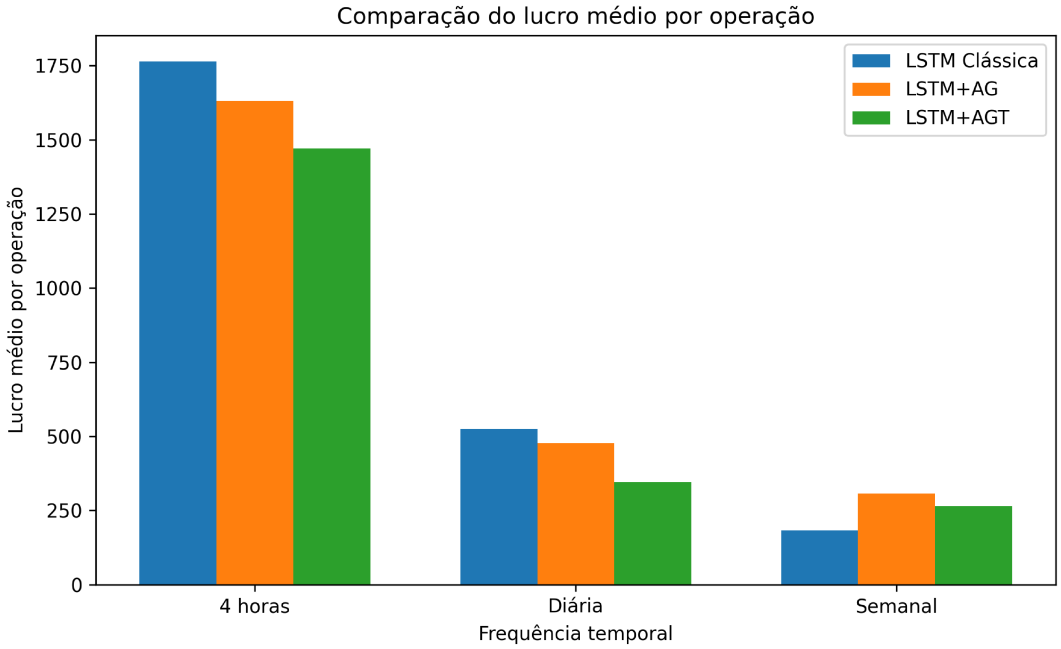

Figura 26 – Comparação do lucro médio por operação obtido pelas arquiteturas LSTM Clássica, LSTM+AG e LSTM+AGT nas frequências de 4 horas, diária e semanal. Fonte: Elaborado pelo autor com auxílio de inteligência artificial generativa (ChatGPT/DALL · E (OpenAI, 2026)).

A Figura 26 complementa os resultados apresentados na Tabela 22, evidenciando que as diferenças no lucro médio por operação variaram conforme a frequência temporal analisada. Na granularidade de 4 horas, a LSTM Clássica apresentou o maior retorno médio por negociação, enquanto, na frequência semanal, a arquitetura otimizada pelo AG obteve o melhor desempenho operacional. Na série diária, observa-se novamente vantagem da LSTM Clássica, indicando que o maior lucro médio por operação não esteve necessariamente associado ao maior número de operações realizadas nem à maior taxa de acerto. Esses resultados reforçam que a qualidade econômica dos sinais gerados pelos modelos depende da interação entre precisão preditiva, momento das entradas e saídas e magnitude dos ganhos e perdas acumulados ao longo da estratégia.

4.7.9 Síntese dos resultados experimentais

As análises desenvolvidas ao longo deste capítulo permitiram avaliar os modelos sob diferentes perspectivas, incluindo desempenho preditivo, desempenho financeiro, complexidade arquitetural, eficiência computacional e comportamento operacional durante o *backtesting*. Considerados isoladamente, esses indicadores conduzem a interpretações parciais, uma vez que cada um evidencia apenas uma dimensão específica do problema investigado.

Com o objetivo de integrar os principais resultados obtidos, esta subseção apresenta uma síntese comparativa das arquiteturas LSTM Clássica, LSTM+AG e LSTM+AGT, destacando os modelos que se sobressaíram em cada critério de avaliação. Essa abordagem permite identificar os pontos fortes e as limitações de cada estratégia, oferecendo uma visão consolidada das contribuições da otimização evolutiva para a previsão de séries temporais financeiras.

A Tabela 23 evidencia que nenhuma arquitetura foi superior em todos os critérios analisados. A LSTM Clássica destacou-se principalmente pelo desempenho financeiro em parte dos experimentos e pelo excelente desempenho preditivo na frequência de 4 horas. O AG, por sua vez, apresentou resultados consistentes nas frequências diária e semanal, obtendo simultaneamente elevada qualidade preditiva e desempenho financeiro competitivo, além de menores níveis de risco em diferentes cenários.

O AGT apresentou comportamento distinto. Embora não tenha obtido, de forma sistemática, os maiores valores absolutos de capital final ou as menores métricas de erro, destacou-se por produzir arquiteturas consideravelmente mais compactas, mantendo desempenho competitivo nas três frequências analisadas. Essa característica tornou-se ainda mais evidente na análise de eficiência arquitetural, na qual o AGT apresentou os maiores valores de capital final e fator de lucro por mil parâmetros treináveis.

Sob essa perspectiva, observa-se que a superioridade de um modelo depende do critério

Tabela 23 – Síntese dos principais resultados experimentais obtidos pelos modelos avaliados.

Critério	**Modelo de destaque**	**Justificativa**
Melhor desempenho preditivo (4 horas)	LSTM Clássica	Menores valores de MSE, RMSE e MAPE.
Melhor desempenho financeiro (4 horas)	Não comparável	Os modelos foram avaliados em períodos distintos.
Menor *drawdown* (4 horas)	LSTM+AG	Menor retração máxima do capital.
Melhor desempenho preditivo (Diária)	LSTM+AG	Melhores métricas de generalização.
Melhor desempenho financeiro (Diária)	LSTM Clássica	Maior capital final e maior fator de lucro.
Menor *drawdown* (Diária)	LSTM+AG	Menor risco financeiro observado.
Melhor desempenho preditivo (Semanal)	AG e AGT	O AG apresentou o menor MAPE, enquanto o AGT obteve menores valores de MSE, RMSE e maior R^2.
Melhor desempenho financeiro (Semanal)	LSTM+AG	Maior capital final e maior fator de lucro.
Arquitetura menos complexa	LSTM+AGT	Menor quantidade de parâmetros treináveis em todas as frequências.
Maior eficiência arquitetural	LSTM+AGT	Maior capital final e fator de lucro por mil parâmetros.
Melhor equilíbrio arquitetural	LSTM+AGT	Melhor relação entre desempenho preditivo, simplicidade estrutural e eficiência financeira.

Nota: (*) O resultado financeiro da LSTM Clássica na frequência de 4 horas deve ser interpretado com cautela, pois foi obtido em período distinto daquele utilizado pelos modelos LSTM+AG e LSTM+AGT.

de avaliação adotado. Enquanto métricas tradicionais privilegiam a precisão estatística ou o retorno financeiro absoluto, a análise integrada proposta nesta pesquisa demonstra que a eficiência de uma arquitetura deve ser avaliada considerando simultaneamente sua capacidade preditiva, sua complexidade estrutural e seu desempenho operacional.

Em conjunto, os resultados obtidos indicam que a utilização de algoritmos evolutivos constitui uma estratégia eficaz para o ajuste automático de hiperparâmetros de redes LSTM aplicadas à previsão de séries temporais financeiras. Em particular, o AGT demonstrou que é possível reduzir significativamente a complexidade arquitetural sem comprometer o desempenho global do modelo, reforçando sua relevância como alternativa para o desenvolvimento de sistemas preditivos mais eficientes e computacionalmente econômicos.

4.7.10 Verificação das hipóteses de pesquisa

A análise conjunta dos resultados experimentais permite retomar as hipóteses formuladas na Seção 1.3 e avaliar sua sustentação à luz das evidências obtidas. Considerando que o desempenho dos modelos variou entre as diferentes granularidades temporais e cri-

térios de avaliação, as hipóteses foram analisadas a partir de uma perspectiva integrada, envolvendo desempenho preditivo, desempenho financeiro, complexidade arquitetural e comportamento da busca evolutiva.

A hipótese central desta pesquisa estabeleceu que a adaptação do operador transgênico ao processo de otimização de hiperparâmetros de redes LSTM poderia constituir uma estratégia competitiva para a construção de modelos de previsão de séries temporais financeiras, conciliando qualidade preditiva, desempenho financeiro e custo computacional. Os resultados obtidos sustentam essa hipótese. Embora a LSTM+AGT não tenha apresentado superioridade absoluta em todas as métricas avaliadas, a abordagem produziu arquiteturas competitivas nas três granularidades temporais, com reduções expressivas na quantidade de parâmetros treináveis e elevada eficiência arquitetural. Dessa forma, a principal contribuição do AGT não esteve associada à obtenção sistemática do menor erro preditivo ou do maior retorno financeiro absoluto, mas à capacidade de produzir soluções estruturalmente mais compactas mantendo desempenho global competitivo.

A primeira hipótese específica estabeleceu que a adaptação do operador transgênico poderia produzir configurações de redes LSTM competitivas quando comparadas às obtidas pelo AG. Essa hipótese foi **confirmada**. Os resultados demonstraram que o AGT encontrou configurações com desempenho preditivo próximo ao obtido pelo AG nas diferentes granularidades, ao mesmo tempo em que selecionou arquiteturas com menor quantidade de parâmetros treináveis. Embora o AG tenha apresentado melhores resultados absolutos em determinados indicadores, especialmente no desempenho financeiro e em algumas métricas preditivas, o AGT permaneceu competitivo e apresentou maior eficiência arquitetural.

A segunda hipótese estabeleceu que a utilização de AGs para a otimização dos hiperparâmetros poderia proporcionar desempenho preditivo superior ao da LSTM Clássica. Essa hipótese foi **parcialmente confirmada**. Na frequência diária, a arquitetura otimizada pelo AG apresentou desempenho preditivo superior ao modelo clássico. Na frequência semanal, os modelos evolutivos também apresentaram resultados superiores em diferentes métricas. Entretanto, na frequência de quatro horas, a LSTM Clássica apresentou os menores valores de MSE, RMSE e MAPE, embora o AGT tenha alcançado o maior coeficiente de determinação (R^2). Portanto, a superioridade da otimização evolutiva mostrou-se dependente da granularidade temporal analisada.

A terceira hipótese propôs que o impacto da otimização evolutiva seria mais significativo na série temporal de maior frequência, correspondente à granularidade de quatro horas. Essa hipótese **não foi confirmada**. Embora os algoritmos evolutivos tenham produzido configurações distintas e o AGT tenha reduzido substancialmente a complexidade arquitetural nessa granularidade, a LSTM Clássica apresentou os menores valores de MSE, RMSE e MAPE. Além disso, os ganhos proporcionados pela otimização evolutiva

mostraram-se mais evidentes nas frequências diária e semanal em relação ao desempenho preditivo. Esse resultado indica que o aumento da frequência temporal não implica necessariamente maior benefício decorrente da otimização dos hiperparâmetros.

A quarta hipótese estabeleceu que modelos LSTM otimizados por AGs poderiam alcançar desempenho preditivo comparável ou superior ao da LSTM Clássica utilizando arquiteturas de menor complexidade. Essa hipótese foi **confirmada**. O AGT selecionou arquiteturas com menor quantidade de parâmetros treináveis nas três granularidades analisadas, apresentando reduções de aproximadamente 72,25% na frequência de quatro horas, 75,74% na frequência diária e 74,60% na frequência semanal em relação à arquitetura clássica. O AG também promoveu reduções expressivas nas frequências diária e semanal. Apesar dessa simplificação estrutural, os modelos mantiveram desempenho preditivo competitivo, demonstrando que arquiteturas maiores não são necessariamente necessárias para alcançar elevada capacidade de generalização.

A quinta hipótese propôs que melhorias no desempenho estatístico das previsões dos modelos otimizados poderiam refletir em melhor desempenho financeiro nas estratégias avaliadas por meio de *backtesting*. Essa hipótese foi **parcialmente confirmada**. Em determinados cenários, melhores métricas preditivas estiveram associadas a melhores resultados financeiros. Entretanto, essa relação não se mostrou consistente entre todas as granularidades. Na frequência diária, por exemplo, a melhoria das métricas preditivas obtida pelos modelos evolutivos não resultou em maior capital final, uma vez que a LSTM Clássica apresentou o melhor desempenho financeiro. Na frequência semanal, por outro lado, o AG apresentou simultaneamente desempenho preditivo competitivo e o maior capital final. Esses resultados demonstram que menor erro estatístico não implica necessariamente maior rentabilidade em uma estratégia de negociação.

A Tabela 24 apresenta uma síntese da verificação das hipóteses formuladas nesta pesquisa.

Tabela 24 – Síntese da verificação das hipóteses da pesquisa.

Hipótese	**Resultado**	**Síntese das evidências**
Central	Sustentada	O AGT produziu arquiteturas competitivas, estruturalmente mais compactas e com elevada eficiência arquitetural, embora não tenha apresentado superioridade absoluta em todos os indicadores.
H1	Confirmada	O AGT apresentou desempenho competitivo em relação ao AG e selecionou arquiteturas mais compactas nas três granularidades temporais.
H2	Parcialmente confirmada	A otimização evolutiva superou a LSTM Clássica em diferentes métricas nas frequências diária e semanal, mas não apresentou superioridade generalizada na frequência de quatro horas.
H3	Não confirmada	O maior impacto da otimização evolutiva não ocorreu de forma consistente na frequência de quatro horas, na qual a LSTM Clássica apresentou os menores valores de MSE, RMSE e MAPE.
H4	Confirmada	Os algoritmos evolutivos, especialmente o AGT, produziram arquiteturas significativamente menores mantendo desempenho preditivo competitivo.
H5	Parcialmente confirmada	A melhoria das métricas preditivas esteve associada ao melhor desempenho financeiro em alguns cenários, mas essa relação não ocorreu de forma consistente entre todas as granularidades.

De forma geral, a verificação das hipóteses demonstra que os efeitos da otimização evolutiva dependem da granularidade temporal e do critério de avaliação considerado. Os resultados não sustentam a existência de uma arquitetura universalmente superior, mas evidenciam que a otimização automática de hiperparâmetros permite encontrar soluções com diferentes compromissos entre precisão preditiva, desempenho financeiro e complexidade estrutural. Nesse contexto, o AGT destacou-se principalmente pela obtenção de arquiteturas mais compactas e eficientes, reforçando a importância de avaliar os modelos de forma integrada, e não exclusivamente por métricas tradicionais de erro.

Capítulo 5

Conclusão

5.1 Considerações finais

A previsão de séries temporais financeiras permanece como um dos principais desafios da Inteligência Artificial aplicada, principalmente devido ao comportamento não linear, à elevada volatilidade e às constantes mudanças de regime observadas nos mercados. Nesse contexto, esta dissertação investigou a utilização de redes neurais *Long Short-Term Memory* (LSTM) otimizadas por AGs e por uma adaptação do AGT para a previsão dos preços de fechamento dos contratos futuros de café arábica.

A proposta desenvolvida teve como principal objetivo investigar se a incorporação do operador transgênico ao processo de otimização de hiperparâmetros seria capaz de produzir arquiteturas competitivas para redes LSTM, conciliando desempenho preditivo, desempenho financeiro, simplicidade estrutural e custo computacional. Para isso, foram comparadas três abordagens distintas: uma LSTM com hiperparâmetros fixos, uma LSTM otimizada por AG e uma LSTM otimizada por uma adaptação do AGT, avaliadas nas granularidades de quatro horas, diária e semanal.

Os resultados experimentais demonstraram que a utilização de algoritmos evolutivos constitui uma estratégia eficaz para automatizar a seleção de hiperparâmetros de redes LSTM. Em relação à arquitetura clássica, os modelos otimizados apresentaram ganhos consistentes na qualidade das previsões em parte dos experimentos, além de evidenciarem que diferentes frequências temporais demandam arquiteturas distintas para representar adequadamente a dinâmica das séries financeiras.

Entretanto, os experimentos também demonstraram que melhorias obtidas nas métricas estatísticas tradicionais não implicam, necessariamente, melhor desempenho financeiro. Em determinadas frequências temporais, arquiteturas com menores valores de erro produziram maior capital acumulado durante o *backtesting*. Em outras situações, modelos estatisticamente superiores não foram aqueles que apresentaram maior rentabilidade

financeira. Esse resultado reforça que a avaliação de modelos destinados ao mercado financeiro deve considerar simultaneamente critérios estatísticos e indicadores operacionais, evitando conclusões baseadas exclusivamente em métricas tradicionais de previsão.

A hipótese central desta pesquisa foi parcialmente confirmada. Os resultados obtidos demonstraram que a adaptação do AGT constitui uma estratégia competitiva para a otimização automática dos hiperparâmetros de redes LSTM, produzindo arquiteturas capazes de manter elevado desempenho preditivo com menor complexidade estrutural. Embora o AGT não tenha obtido os menores erros em todas as frequências analisadas, mostrou desempenho competitivo em todos os experimentos realizados, alcançando, em alguns cenários, o maior coeficiente de determinação e selecionando, sistematicamente, arquiteturas mais compactas que aquelas produzidas pelo AG.

Outro resultado relevante refere-se ao comportamento observado durante o processo evolutivo. A implementação desenvolvida permitiu registrar integralmente a dinâmica da população ao longo das gerações, possibilitando a análise da evolução dos cromossomos, da diversidade populacional, da atuação dos módulos transgênicos e da convergência do processo de busca. Esses registros ampliaram significativamente a interpretação dos resultados, permitindo compreender não apenas quais arquiteturas foram selecionadas, mas também como essas soluções foram construídas ao longo da otimização.

Sob a perspectiva metodológica, esta pesquisa demonstrou que a avaliação de algoritmos evolutivos pode ser enriquecida quando são analisados, simultaneamente, o desempenho preditivo, os resultados financeiros, a complexidade das arquiteturas e a dinâmica interna da busca evolutiva. Essa abordagem integrada amplia a compreensão do comportamento dos algoritmos, fornecendo evidências que dificilmente seriam observadas por meio da análise isolada de métricas de erro.

De maneira geral, os resultados obtidos confirmam a viabilidade da utilização de algoritmos evolutivos na otimização de hiperparâmetros de redes LSTM aplicadas à previsão de séries temporais financeiras. Mais especificamente, os experimentos evidenciaram que a adaptação do AGT representa uma alternativa promissora para esse problema, apresentando capacidade de produzir arquiteturas estruturalmente mais simples, estatisticamente competitivas e potencialmente mais eficientes para aplicações práticas. Dessa forma, esta dissertação contribui para o avanço das pesquisas envolvendo otimização evolutiva aplicada ao aprendizado profundo, oferecendo um método reprodutível, rastreável e passível de extensão para diferentes domínios e problemas de previsão.

A análise das hipóteses formuladas no início desta pesquisa demonstrou que a hipótese central foi sustentada pelos resultados experimentais, uma vez que a adaptação do AGT mostrou-se competitiva na otimização de hiperparâmetros das redes LSTM e apresentou vantagens relevantes em termos de simplicidade e eficiência arquitetural. Das cinco hipóteses específicas, duas foram confirmadas, duas foram parcialmente confirmadas e uma

não foi confirmada. Esses resultados apresentam que os benefícios da otimização evolutiva dependem da granularidade temporal e do critério de avaliação considerado, reforçando a necessidade de analisar conjuntamente precisão preditiva, desempenho financeiro, complexidade estrutural e custo computacional.

5.2 Principais contribuições da pesquisa

Ao longo desta pesquisa foram desenvolvidas contribuições metodológicas, computacionais e experimentais relacionadas à utilização de algoritmos evolutivos para otimização automática de hiperparâmetros de redes neurais LSTM aplicadas à previsão de séries temporais financeiras. Diferentemente de trabalhos que concentram a análise apenas na qualidade das previsões, esta dissertação propôs uma abordagem integrada, contemplando simultaneamente aspectos preditivos, financeiros, arquiteturais e comportamentais do processo evolutivo.

As contribuições apresentadas não devem ser analisadas de forma isolada. Embora parte delas esteja relacionada ao desenvolvimento computacional do AGT, sua principal relevância encontra-se na integração entre diferentes dimensões de avaliação. O método proposto permitiu investigar simultaneamente como a otimização evolutiva influencia a qualidade das previsões, o comportamento financeiro das estratégias, a complexidade das arquiteturas geradas e a dinâmica interna do processo evolutivo.

Nesse contexto, destaca-se que a principal contribuição desta dissertação não consiste na obtenção do menor erro estatístico em todos os experimentos realizados. O diferencial da pesquisa está na demonstração de que a adaptação do AGT foi capaz de produzir arquiteturas mais compactas, mantendo desempenho competitivo e fornecendo mecanismos de rastreabilidade que ampliam significativamente a compreensão do processo de otimização evolutiva. Essa característica diferencia a abordagem proposta de grande parte dos trabalhos encontrados na literatura, que normalmente se limitam à comparação entre métricas finais de desempenho.

5.3 Limitações da Pesquisa

Apesar dos resultados alcançados, algumas limitações devem ser consideradas na interpretação dos experimentos realizados.

A primeira limitação refere-se ao domínio de aplicação utilizado nesta pesquisa. Os experimentos foram conduzidos exclusivamente utilizando preços históricos de fechamento dos contratos futuros de café arábica (KC=F). Embora esse ativo apresente elevada complexidade e seja representativo para o mercado de *commodities* agrícolas, os resultados

Tabela 25 – Principais contribuições da dissertação.

Contribuição	**Descrição**
Adaptação do AGT	Desenvolvimento e adaptação do operador transgênico proposto por Amaral e Jr (2014) para otimização automática de hiperparâmetros de redes neurais LSTM aplicadas à previsão de séries temporais financeiras.
Desenvolvimento do operador modular	Implementação de quatro módulos transgênicos (M1–M4), responsáveis por realizar diferentes intensidades de modificação genética durante o processo evolutivo.
Sistema de rastreabilidade evolutiva	Desenvolvimento de mecanismos capazes de registrar integralmente a evolução da população, permitindo analisar diversidade, convergência, evolução dos cromossomos, atuação dos módulos transgênicos e histórico das soluções.
Avaliação integrada	Proposição de uma metodologia experimental baseada na análise conjunta do desempenho preditivo, desempenho financeiro, complexidade arquitetural e custo computacional das soluções encontradas.
Arquiteturas compactas	Demonstração experimental de que o AGT apresentou tendência consistente à seleção de arquiteturas estruturalmente mais simples, mantendo desempenho estatístico competitivo em todas as granularidades avaliadas.
Validação financeira	Avaliação dos modelos utilizando procedimentos sistemáticos de *backtesting*, permitindo analisar simultaneamente rentabilidade, fator de lucro e risco das estratégias construídas a partir das previsões.
Base experimental reproduzível	Construção de uma estrutura experimental passível de reprodução e adaptação para outros ativos financeiros, diferentes arquiteturas neurais e novos algoritmos evolutivos.

obtidos não permitem afirmar, de forma imediata, que o mesmo comportamento será observado para outros ativos financeiros.

Outra limitação está relacionada ao conjunto de hiperparâmetros considerado durante o processo evolutivo. A otimização concentrou-se em cinco parâmetros da arquitetura LSTM (número de camadas, número de neurônios, taxa de aprendizagem, tamanho do lote e número de épocas). Outros parâmetros potencialmente relevantes, como funções de ativação, taxas de *dropout*, otimizadores e tamanhos das janelas temporais, permaneceram fixos durante os experimentos.

Além disso, as avaliações financeiras foram realizadas por meio de uma única estratégia de negociação baseada nas previsões produzidas pelos modelos. Embora essa estratégia permita comparar consistentemente as abordagens investigadas, diferentes critérios de

entrada e saída poderiam produzir resultados financeiros distintos.

Por fim, deve-se considerar que a implementação proposta do AGT representa apenas uma possível adaptação do operador original ao problema de otimização de hiperparâmetros de redes LSTM. Outras estratégias de construção do histórico genético, seleção probabilística dos genes e mecanismos de inserção dos indivíduos transgênicos poderão ser investigadas em pesquisas futuras.

5.4 Trabalhos Futuros

Os resultados obtidos nesta dissertação abrem diversas possibilidades para a continuidade da pesquisa, tanto sob a perspectiva metodológica quanto sob a perspectiva da aplicação em problemas reais de previsão de séries temporais financeiras.

Uma primeira linha de investigação consiste na ampliação do espaço de busca utilizado durante o processo de otimização evolutiva. Nesta pesquisa foram considerados cinco hiperparâmetros da arquitetura LSTM, entretanto outros elementos, como funções de ativação, diferentes algoritmos de otimização, taxas de *dropout*, tamanho da janela temporal, número de unidades densas e técnicas de regularização, poderão ser incorporados ao cromossomo, permitindo ampliar o potencial exploratório dos algoritmos evolutivos.

Outra possibilidade consiste na adaptação do operador transgênico para outras arquiteturas de aprendizado profundo. Modelos como Gated Recurrent Unit (GRU), Transformers, Temporal Convolutional Networks (TCN), Redes Neurais Híbridas e arquiteturas baseadas em mecanismos de atenção representam alternativas promissoras para investigar a generalidade da abordagem proposta nesta dissertação.

Outra direção de pesquisa refere-se à utilização de estratégias de otimização multiobjetivo. Nesta dissertação, a função de aptidão foi baseada no erro quadrático médio (MSE), privilegiando a qualidade das previsões. Entretanto, aplicações financeiras frequentemente envolvem objetivos conflitantes, como maximização do retorno financeiro, redução do *drawdown*, diminuição da complexidade arquitetural e redução do custo computacional. Nesse contexto, algoritmos evolutivos multiobjetivo poderão produzir conjuntos de soluções capazes de equilibrar simultaneamente esses diferentes critérios.

Outra possibilidade consiste na investigação de operadores transgênicos adaptativos. Na implementação proposta, a intensidade de atuação dos módulos transgênicos permanece previamente definida durante todo o processo evolutivo. Pesquisas futuras poderão desenvolver mecanismos capazes de ajustar automaticamente essa intensidade em função da diversidade populacional, da velocidade de convergência ou do comportamento da função de aptidão ao longo das gerações.

Do ponto de vista experimental, torna-se relevante ampliar a validação da metodo-

logia proposta para outros ativos financeiros, incluindo índices de mercado, ações, criptomoedas, contratos futuros de diferentes *commodities* e mercados internacionais. Essa ampliação permitirá avaliar a capacidade de generalização do AGT em diferentes cenários de volatilidade e dinâmica temporal.

Outra linha promissora consiste no aprofundamento da análise da dinâmica evolutiva. Como a evolução individual dos genes, a influência relativa dos diferentes módulos transgênicos sobre a convergência da população, a formação de padrões genéticos recorrentes e a construção automática de estratégias evolutivas adaptativas baseadas no histórico da população.

Por fim, o método desenvolvido nesta pesquisa poderá ser integrada a ambientes reais de negociação automatizada, permitindo investigar o comportamento do AGT em cenários de atualização contínua dos modelos (*online learning*), reotimização periódica dos hiperparâmetros e adaptação dinâmica às mudanças de regime observadas nos mercados financeiros. Essas investigações poderão ampliar a aplicabilidade prática da abordagem proposta, aproximando-a de sistemas inteligentes de apoio à decisão utilizados em ambientes operacionais.

Referências

AMALIA, S. et al. A temporal fusion transformer for multi-step forecasting of indonesia's strategic food commodity prices. **Results in Engineering**, Elsevier, p. 110131, 2026.

AMARAL, L. R. do; JR, E. R. H. Transgenic: An evolutionary algorithm operator. **Neurocomputing**, Elsevier, v. 127, p. 104–113, 2014.

AVINASH, G. et al. Hidden markov guided deep learning models for forecasting highly volatile agricultural commodity prices. **Applied soft computing**, Elsevier, v. 158, p. 111557, 2024.

BÄCK, T. **Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms**. New York, NY: Oxford University Press, 1996. ISBN 9780195099713.

BENIWAL, M.; SINGH, A.; KUMAR, N. Forecasting long-term stock prices of global indices: A forward-validating genetic algorithm optimization approach for support vector regression. **Applied Soft Computing**, Elsevier, v. 145, p. 110566, 2023.

BU, Y. et al. Soybean futures price forecasting based on ga-lstm model. In: YE, Y.; ZHOU, H. (Ed.). **Artificial Intelligence and Human-Computer Interaction: Proceedings of the 2nd International Conference (ArtInHCI 2024)**. [S.l.]: SAGE, 2025. (Frontiers in Artificial Intelligence and Applications, v. 404), p. 209–217. Conference held in Kunming, China, 25–27 October 2024.

CARRASCO-GUTIERREZ, C. E.; ALMEIDA, F. M. d. M. Modelagem e previsão do preço do café brasileiro. **Revista de Economia**, v. 39, n. 2, p. 7–27, 2013.

CELIK, B. A.; CELIK, S. Hybrid forecasting of agricultural commodity prices: Integrating machine learning, time series, and stochastic simulation models. **Borsa Istanbul Review**, Elsevier, 2025.

CHEN, S.; ZHOU, C. Stock prediction based on genetic algorithm feature selection and long short-term memory neural network. **IEEE Access**, IEEE, v. 9, p. 9066–9072, 2021.

CHENG, H. et al. A hybrid electricity price forecasting model with bayesian optimization for german energy exchange. **International Journal of Electrical Power & Energy Systems**, Elsevier, v. 110, p. 653–666, 2019.

CHOUDHARY, K. et al. A genetic algorithm optimized hybrid model for agricultural price forecasting based on vmd and lstm network. **Scientific Reports**, Nature Publishing Group UK London, v. 15, n. 1, p. 9932, 2025.

CHUNG, H.; SHIN, K.-s. Genetic algorithm-optimized long short-term memory network for stock market prediction. **Sustainability**, MDPI, v. 10, n. 10, p. 3765, 2018.

CORTAZAR, G. et al. Commodity price forecasts, futures prices, and pricing models. **Management science**, INFORMS, v. 65, n. 9, p. 4141–4155, 2019.

COULEAU, A.; TRUJILLO-BARRERA, A.; ETIENNE, X. Intraday market momentum in coffee futures: Dynamics and drivers. **Journal of Commodity Markets**, Elsevier, p. 100537, 2026.

CRUVINEL, H. H. **Previsão de ações com modelo híbrido LSTM e algoritmo genético**. Goiânia, GO, 2024. Trabalho de Conclusão de Curso (Ciência da Computação). Disponível em: <https://repositorio.pucgoias.edu.br/jspui/handle/123456789/7900>.

DANTAS, L. C. **Otimização de redes neurais artificiais de múltiplas camadas utilizando algoritmos genéticos e enxame de partículas**. Dissertação (Dissertação (Mestrado em Ciência da Computação)) — Universidade Estadual Paulista (Unesp), São José do Rio Preto, SP, 2017. Disponível em: <http://hdl.handle.net/11449/194489>.

DEINA, C. et al. A methodology for coffee price forecasting based on extreme learning machines. **Information Processing in Agriculture**, Elsevier, v. 9, n. 4, p. 556–565, 2022.

EIBEN, A. E.; SMITH, J. E. **Introduction to Evolutionary Computing**. Berlin, Heidelberg: Springer-Verlag, 2003.

ESANGBEDO, M. O. et al. Enhancing the exploitation of natural resources for green energy: An application of lstm-based meta-model for aluminum prices forecasting. **Resources Policy**, Elsevier, v. 92, p. 105014, 2024.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization, and Machine Learning**. Reading, MA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 9780201157673.

GU, Y. H. et al. Forecasting agricultural commodity prices using dual input attention lstm. **Agriculture**, MDPI, v. 12, n. 2, p. 256, 2022.

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

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence**. Ann Arbor, MI: University of Michigan Press, 1975.

HUANG, Y. et al. A new financial data forecasting model using genetic algorithm and long short-term memory network. **Neurocomputing**, Elsevier, v. 425, p. 207–218, 2021.

HWASE, T. K.; FOFANAH, A. J. Machine learning model approaches for price prediction in coffee market using linear regression, xgb, and lstm techniques. **International Journal of Scientific Research in Science and Technology**, n. 6, p. 10–48, 2021.

JIN, B.; XU, X. Machine learning coffee price predictions. **Journal of Uncertain Systems**, World Scientific, v. 17, n. 04, p. 2450023, 2024.

KAMOCSAI, L.; ORMOS, M. Modeling and forecasting commodity price volatility using a common leverage factor. **The North American Journal of Economics and Finance**, Elsevier, p. 102570, 2026.

KARPATHY, A. **The Unreasonable Effectiveness of Recurrent Neural Networks**. 2015. Publicação online. Acesso em: 11 out. 2024. Disponível em: <https://karpathy.github.io/2015/05/21/rnn-effectiveness/>.

KIM, K.-J.; AHN, H. Simultaneous optimization of artificial neural networks for financial forecasting. **Applied Intelligence**, Springer, v. 36, p. 887–898, 2012.

LAHMIRI, S. A hybrid cnn-lstm-gru system tuned by whale optimization algorithm for enhanced gold and brent price prediction. **Finance Research Open**, Elsevier, p. 100100, 2026.

LEANDRO, J. C. **Aplicação de redes neurais LSTM para previsão de séries temporais financeiras**. Dourados, MS, 2021. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Computação). Disponível em: <http://repositorio.ufgd.edu.br/jspui/handle/prefix/4806>.

LIANG, J.; JIA, G. China futures price forecasting based on online search and information transfer. **Data Science and Management**, Elsevier, v. 5, n. 4, p. 187–198, 2022.

LIN, M.; CHEN, C. Short-term prediction of stock market price based on ga optimization lstm neurons. In: **Proceedings of the 2018 2nd International Conference on Deep Learning Technologies**. New York, NY, USA: Association for Computing Machinery, 2018. (ICDLT '18), p. 66–70.

MANOGNA, R.; DHARMAJI, V.; SARANG, S. Enhancing agricultural commodity price forecasting with deep learning. **Scientific Reports**, Nature Publishing Group UK London, v. 15, n. 1, p. 20903, 2025.

MATSUMOTO, D. K. F. **Estudo em séries temporais financeiras utilizando redes neurais recorrentes**. Dissertação (Dissertação (Mestrado em Modelagem Computacional do Conhecimento)) — Universidade Federal de Alagoas, Maceió, AL, 2019. Disponível em: <https://www.repositorio.ufal.br/handle/riufal/6813>.

MELO-VELANDIA, L. F.; OTERO, J.; RAMÍREZ-GONZÁLEZ, M. S. Quality differences, location, and coffee price returns networks: Insights from a high-dimensional covar-copula analysis. **International Review of Financial Analysis**, Elsevier, p. 104537, 2025.

MENDES, R. d. L. **Abordagens evolutivas para otimização de redes neurais convolucionais baseadas em algoritmos genéticos**. 86 p. Dissertação (Dissertação (Mestrado em Ciência da Computação)) — Universidade Federal de Uberlândia, Uberlândia, MG, 2021.

MITCHELL, M. **An Introduction to Genetic Algorithms**. Cambridge, MA: MIT Press, 1998. ISBN 9780262631853.

MORAIS, M. A. d. S. **Previsão de preços de ações no setor bancário brasileiro por meio do uso de redes neurais recorrentes de tipo LSTM**. Uberlândia, MG, 2023. 48 p. Trabalho de Conclusão de Curso (Graduação em Gestão da Informação). Disponível em: <https://repositorio.ufu.br/handle/123456789/39852>.

MURUGESAN, R.; MISHRA, E.; KRISHNAN, A. H. Forecasting agricultural commodities prices using deep learning-based models: Basic lstm, bi-lstm, stacked lstm, cnn lstm, and convolutional lstm. **International Journal of Sustainable Agricultural Management and Informatics**, v. 8, n. 3, p. 242–277, 2022.

NABIPOUR, M. et al. Predicting stock market trends using machine learning and deep learning algorithms via continuous and binary data; a comparative analysis. **Ieee Access**, IEEE, v. 8, p. 150199–150212, 2020.

NAVEENA, K. et al. Hybrid time series modelling for forecasting the price of washed coffee (arabica plantation coffee) in india. **International Journal of Agriculture Sciences**, v. 9, p. 4004–4007, 2017.

NELSON, D. M. Q. **Uso de redes neurais recorrentes para previsão de séries temporais financeiras**. Dissertação (Dissertação (Mestrado em Ciência da Computação)) — Universidade Federal de Minas Gerais, Belo Horizonte, MG, 2017. Disponível em: <https://hdl.handle.net/1843/ESBF-AM2NTS>.

OLAH, C. **Understanding LSTM Networks**. 2015. <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>. Acessado em: 20 jan. 2026. Disponível em: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.

OpenAI. **ChatGPT e DALL · E: modelos de inteligência artificial generativa**. 2026. <https://chatgpt.com>. Utilizados como ferramenta de apoio na elaboração de ilustrações, diagramas e elementos gráficos desta dissertação.

PAIVA, F. D. **Redes neurais para decisões no mercado de ações brasileiro**. 118 p. Tese (Tese (Doutorado em Administração)) — Universidade Federal de Lavras, Lavras, MG, 2014.

PANG, Q.; SUN, Y. Stock price prediction based on genetic algorithm optimized bp neural networks. In: **Proceedings of the 2025 8th International Conference on Computer Information Science and Artificial Intelligence**. New York, NY, USA: Association for Computing Machinery, 2025. (CISAI '25), p. 88–92.

PENG, L. et al. Effective long short-term memory with differential evolution algorithm for electricity price prediction. **Energy**, Elsevier, v. 162, p. 1301–1314, 2018.

RODRIGUES, N.; REIS, E.; TAVARES, M. Influências dos fatores climáticos no custo de produção do café arábica. **Custos e@ gronegócio on line**, v. 10, n. 3, p. 216–255, 2014.

SARANGI, P. K. et al. Analysing the fluctuations in commodity prices and forecasting the future directions using machine learning techniques. **Procedia Computer Science**, Elsevier, v. 259, p. 1827–1836, 2025.

SARI, M. et al. Various optimized machine learning techniques to predict agricultural commodity prices. **Neural Computing and Applications**, Springer, v. 36, n. 19, p. 11439–11459, 2024.

SAYED, G. I. et al. An optimized and interpretable carbon price prediction: Explainable deep learning model. **Chaos, Solitons & Fractals**, Elsevier, v. 188, p. 115533, 2024.

SEDIYONO, E. et al. An integrated framework for multi-commodity agricultural price forecasting and anomaly detection using attention-boosted models. **Journal of Agriculture and Food Research**, Elsevier, v. 22, p. 102021, 2025.

SHA, X. Time series stock price forecasting based on genetic algorithm (ga)-long short-term memory network (lstm) optimization. **Advances in Economics, Management and Political Sciences**, v. 91, n. 1, p. 142–149, 2024.

SHAO, B. et al. Nickel price forecast based on the lstm neural network optimized by the improved pso algorithm. **Mathematical Problems in Engineering**, Wiley Online Library, v. 2019, n. 1, p. 1934796, 2019.

SUÁREZ-RODRÍGUEZ, C. H.; MANOTAS-DUQUE, D. F.; LARGO-AVILA, E. Financialized arabica coffee: Distributional asymmetries and the burden on smallholder farmers. **The Journal of Economic Asymmetries**, Elsevier, v. 33, p. e00458, 2026.

SUN, L. et al. Integration of genetic algorithms and trading simulation for enhanced stock price prediction and optimization of quantitative trading strategies. In: **Proceedings of the 2024 2nd International Conference on Electronics, Computers and Communication Technology**. New York, NY, USA: [s.n.], 2024. (CECCT '24), p. 166–171.

TAGHIPOUR, H.; REZAEE, A.; HAJATI, F. Gold price prediction using long short-term memory and multi-layer perceptron with gray wolf optimizer. **arXiv preprint arXiv:2512.22606**, 2025.

WANG, C.-N. et al. An integrated forecasting model for the coffee bean supply chain. **Applied Economics**, Taylor & Francis, v. 53, n. 28, p. 3321–3333, 2021.

WANG, J.; LI, X. A combined neural network model for commodity price forecasting with ssa. **Soft Computing**, Springer, v. 22, n. 16, p. 5323–5333, 2018.

WU, C. et al. Stock price prediction method based on optimized lstm and quantified news sentiment. In: **Proceedings of the 2025 International Conference on Artificial Intelligence and Digital Finance**. New York, NY, USA: Association for Computing Machinery, 2025. (AIDF '25), p. 25–34.

XU, X.; ZHANG, Y. Commodity price forecasting via neural networks for coffee, corn, cotton, oats, soybeans, soybean oil, sugar, and wheat. **Intelligent Systems in Accounting, Finance and Management**, Wiley Online Library, v. 29, n. 3, p. 169–181, 2022.

YANG, Q. et al. Can hybrid model improve the forecasting performance of stock price index amid covid-19? contextual evidence from the meemd-lstm-mlp approach. **The North American Journal of Economics and Finance**, Elsevier, v. 74, p. 102252, 2024.

YANG, R.; LIU, H.; LI, Y. A heterogeneous ensemble architecture coupling model selection sorting and residual error iterative correction for crude oil price forecasting. **Applied Soft Computing**, Elsevier, v. 148, p. 110865, 2023.

YASHAVANTH, B. et al. Forecasting prices of coffee seeds using vector autoregressive time series model. **Indian Journal of Agricultural Sciences**, v. 87, n. 6, p. 754–758, 2017.

Apêndices

APÊNDICE **A**

LSTM Clássica

```
# ============================================================
# CÓDIGO 1 - LSTM CLÁSSICA
# PREVISÃO DO PREÇO DO CAFÉ KC=F
# ============================================================

import os                                   # Permite configurar variaveis do ambiente, serve para interagir com o sistema op
                                            # permitindo manipular arquivos, diretórios (pastas), gerenciar caminhos e execut
import certifi                              # Conecta com segurança no Yahoo finance para coletar os dados.
os.environ['SSL_CERT_FILE'] = certifi.where()     # Diz o python onde estão os certificados confiaveis, evia erros de conexã
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'          # Reduz mensagens desnecessárias do TensorFlow, como avisos tecnicos.

import time                                 # Utilizada para medir o tempo para rodar o codigo
import numpy as np                          # Usuda para calculos, de media, raiz quadrada, etc..
import pandas as pd                         # Para manipular e organizar os dados em tabela
import matplotlib.pyplot as plt             # Usado para gerar os graficos dos resultados

import yfinance as yf                       # Baixar os dados financeiros

from tensorflow.keras.models import Sequential           # Cria uma rede neural em sequencia, e como montar em pilha de camd
from tensorflow.keras.layers import LSTM, Dense, Dropout # Imponta as camadas usadas
from tensorflow.keras.optimizers import Adam             # Ajusta os pesos da rede, reduzindo o erro durante o treinamento.
from tensorflow.keras import backend as K                # Permite limpar a memoria da sessão keras, para não acumular memor

from sklearn.preprocessing import MinMaxScaler           # Normaliza os preços na escala de 0 a 1, LSTM aprender melhor com
from sklearn.metrics import mean_squared_error, r2_score # importa as metricas utilizadas

# ============================================================
# CONFIGURAÇÕES GERAIS DO EXPERIMENTO
# ============================================================

TICKER = 'KC=F'                    # Define o ativo financeiro, no caso o Café futuro Arabica
                                   # Será utilizado 3 periodos graficos diferentes 4h (horas), 1d (dia), 1wk(semana)
# Escolha:
# '4h'  = últimos 500 dias
# '1d'  = 01/01/2000 até 31/12/2020
# '1wk' = 01/01/2000 até 31/12/2020

INTERVAL = '1wk'  # ALTERE AQUI     # Aqui e defido os intervalos de acordo com cada experimento.

if INTERVAL == '4h':
    PERIOD = '500d'
    START_DATE = None
    END_DATE = None
    NOME_BASE = '4h'

elif INTERVAL == '1d':
    PERIOD = None
    START_DATE = '2000-01-01'
    END_DATE = '2020-12-31'
    NOME_BASE = 'diario'

elif INTERVAL == '1wk':
    PERIOD = None
    START_DATE = '2000-01-01'
    END_DATE = '2020-12-31'
    NOME_BASE = 'semanal'

else:
    raise ValueError("Intervalo inválido. Use '4h', '1d' ou '1wk'.")

TIME_STEP = 30                      # Usando os 30 registros anteriores para prever o proximo.
TEST_SIZE_RATIO = 0.20              # Define 20% dos dados para o teste

# ============================================================
# HIPERPARÂMETROS FIXOS DA LSTM CLÁSSICA
# ============================================================

NUM_LAYERS = 2                      # Terá duas camadas
NUM_NEURONS = 50                    # Cada camada com 50 neurônios
LEARNING_RATE = 0.001               # Taxa de aprendizado da rede, controlo o tamanho dos ajustes nos pesos da rede.
DROPOUT_RATE = 0.2                  # Durante o treino 20% das conexões são desligadas aleatoriamente, reduz overfeatting
BATCH_SIZE = 32                     # Processa 32 amostrar por vez, divide a amostra em bolocos de 32.
EPOCHS = 50                         # Numero maximo de epocas, quantidade de vezes que a rede passsara por todo conjunto de

start_global = time.time()          # Começa a marca o tempo inicial do codigo.

# ============================================================
# 1. DOWNLOAD DOS DADOS
# ============================================================

if PERIOD is not None:              # Verifica se tem o periodo de 4h, baixa a coluna close, que é o preço de fechamento
```

```
if PERIOD is not None:                  # Verifica se tem o período de 4h, baixa a coluna Close, que é o preço de fechamento
    cafe = yf.download(
        TICKER,
        period=PERIOD,
        interval=INTERVAL
    )[['Close']]
else:                                   # Caso não seja o periodo de 4h, baixa os dados no intervalo definido
    cafe = yf.download(
        TICKER,
        start=START_DATE,
        end=END_DATE,
        interval=INTERVAL
    )[['Close']]

cafe = cafe.dropna()                    # Remove os valores faltantes
cafe.index = pd.to_datetime(cafe.index)  # Define a coluna data, como indice

# ============================================================
# 2. CRIAÇÃO DAS JANELAS TEMPORAIS
# ============================================================

def dataset_from_series(values, time_step=30):      # transforma a serie temporal em pares entrada "x" e saida "y"
    arr = np.asarray(values).reshape(-1)            # transforma os valores em um vetor

    X_raw, Y_raw = [], []                           # Cria duas listas vazias, que receberam os dados de entrada e saida esp

    for i in range(len(arr) - time_step):           # Percorre a serie e cria os valores de entrada e saida. Ex: se tem 1000
        X_raw.append(arr[i:i + time_step])          # Recebe os valores de 30 dias
        Y_raw.append(arr[i + time_step])            # Recebe o valor do dia seguinte.

    return np.array(X_raw), np.array(Y_raw)         #  Retorna a entrada e saida em valores em formato numpy

X_raw, Y_raw = dataset_from_series(                 # Aplica a função aos preços de fechamento do cafe.
    cafe['Close'].values,                           # Agora temos X_raw uma sequencia de 30 preços e Y_raw próximo preço após
    TIME_STEP
)

# ============================================================
# 3. DIVISÃO TREINO / TESTE TEMPORAL
# ============================================================

split_idx = int(len(X_raw) * (1 - TEST_SIZE_RATIO))   # Calcula o ponto de divisão de treino

X_train_raw = X_raw[:split_idx]                       # 80% para treino
X_test_raw = X_raw[split_idx:]                        # 20% para teste

Y_train_raw = Y_raw[:split_idx]                       # 80% para treino
Y_test_raw = Y_raw[split_idx:]                        # 20% para teste

# ============================================================
# 4. NORMALIZAÇÃO DE DADOS
# ============================================================

scaler = MinMaxScaler()                       # Cria o normalizador de dados

valores_treino_para_scaler = np.concatenate([ # Junta todos os valores de treino em uma coluna
    X_train_raw.flatten(),
    Y_train_raw.flatten()
]).reshape(-1, 1)

scaler.fit(valores_treino_para_scaler)        # Aprender os valores minimos e maximos dos dados para o treino, importante so

X_train_scaled = scaler.transform(            # Normaliza o "X" do treino e volta para o formato original
    X_train_raw.reshape(-1, 1)
).reshape(X_train_raw.shape)

X_test_scaled = scaler.transform(             # Normaliza o "X" do teste usando o scaler treinado no treino
    X_test_raw.reshape(-1, 1)
).reshape(X_test_raw.shape)

Y_train_scaled = scaler.transform(            # Normaliza o "Y" do treino
    Y_train_raw.reshape(-1, 1)
).reshape(-1)

Y_test_scaled = scaler.transform(             # Normaliza o "Y" do teste
    Y_test_raw.reshape(-1, 1)
).reshape(-1)

X_train = X_train_scaled.reshape(             # Prepara para o formato da LSTM -> número de amostras, quantidade de passos n
    X_train_scaled.shape[0],
    X_train_scaled.shape[1],
    1
)
```

```
X_test = X_test_scaled.reshape(              # Prepara para o formato da LSTM -> número de amostras, quantidade de passos n
    X_test_scaled.shape[0],
    X_test_scaled.shape[1],
    1
)

# ============================================================
# 5. MODELO LSTM CLÁSSICO FIXO
# ============================================================

def create_lstm_classica(input_shape):              # Cria uma função que monta o modelo
    modelo = Sequential()                           # Cria um modelo vazio

    modelo.add(                                     # Adiciona a primeira camada LSTM
        LSTM(
            NUM_NEURONS,                            # Tem 50 neurônios
            return_sequences=True,                  # porque terá outra LSTM depois
            input_shape=input_shape                 # Será de 30,1   -> 30 dias para prever o proximo dia.
        )
    )

    modelo.add(                                     # Adiciona a segunda camada LSTM
        LSTM(
            NUM_NEURONS,                            # Com 50 neurônios tambem
            return_sequences=False                  # Porque agora não precisa devolver uma sequencia completa.
        )
    )

    modelo.add(Dropout(DROPOUT_RATE))              # Desliga 20% das conexões dos neurônios
    modelo.add(Dense(25))                          # Adiciona uma camada densa com 25 neuronios
    modelo.add(Dense(1))                           # Camada final da rede, gera a previsao do proximo dia.

    modelo.compile(                                   # Compila o modelo
        optimizer=Adam(learning_rate=LEARNING_RATE),  # Usa otimizados adam, loss o mse e learning_rate = 0,001
        loss='mse'
    )

    return modelo                                     # Retorna o modelo pronto.

# ============================================================
# 6. MÉTRICAS
# ============================================================

def calcular_metricas(y_true_scaled, y_pred_scaled, scaler_used): # Função para calcular as metricas, ela recebe valor real
    y_true_inv = scaler_used.inverse_transform(                   # desfaz a normalização do valor real
        y_true_scaled.reshape(-1, 1)
    ).reshape(-1)

    y_pred_inv = scaler_used.inverse_transform(                   # Desfaz a normalização da previsão
        y_pred_scaled.reshape(-1, 1)
    ).reshape(-1)

    mse = mean_squared_error(y_true_inv, y_pred_inv)              # Erro quadratico médio, se ele e grande o MSE cresce bast
    rmse = np.sqrt(mse)                                           # Ele fica na escala original do preço, mais facil de visu

    mape = np.mean(                                               # Erro percentual médio
        np.abs(
            (y_true_inv - y_pred_inv) /
            np.maximum(y_true_inv, 1e-6)
        )
    ) * 100

    r2 = r2_score(y_true_inv, y_pred_inv)                         # Quanto mais proximo de 1 melhor.

    return mape, mse, rmse, r2, y_true_inv, y_pred_inv            # Retorna as metricas e os valores desnormalizados.

# ============================================================
# 7. TREINAMENTO
# ============================================================

K.clear_session()                              # Limpa modelos anteriores da memoria

modelo = create_lstm_classica(                 # Cria o modelo, (30,1)
    (X_train.shape[1], X_train.shape[2])       # O modelo recebe 30 passos temporais, 1  variável.
)

# O EarlyStopping foi removido para padronizar a comparação com LSTM + AG e LSTM + AGT.
# Assim, a LSTM clássica treina diretamente com todo o conjunto de treino (80%)
# pelo número fixo de épocas definido em EPOCHS.
inicio_treino = time.time()                    # inicia o treinamento

history = modelo.fit(                          # Treina o modelo
```

```
    X_train,                                # entrada de treino completa
    Y_train_scaled,                         # saídas esperadas normalizadas
    epochs=EPOCHS,                          # número fixo de épocas
    batch_size=BATCH_SIZE,                  # blocos de 32 amostras
    verbose=1                               # mostra o progresso do treino
)

tempo_treino = time.time() - inicio_treino    # Calcula quanto tempo o treinamento durou.

# ============================================================
# 8. PREVISÕES
# ============================================================

inicio_teste = time.time()                      # Inicio da etapa de previsão

y_pred_train_scaled = modelo.predict(           # Faz previsão do conjunto de treino
    X_train,
    verbose=0                                   # Não gera nenhuma mensagem
).reshape(-1)

y_pred_test_scaled = modelo.predict(            # Previsao do conjunto de teste
    X_test,
    verbose=0
).reshape(-1)

tempo_teste = time.time() - inicio_teste        # Calcula o tempo das previsões

# ============================================================
# 9. CÁLCULO DAS MÉTRICAS
# ============================================================

mape_train, mse_train, rmse_train, r2_train, y_train_inv, y_pred_train_inv = calcular_metricas(
    Y_train_scaled,
    y_pred_train_scaled,
    scaler
)

mape_test, mse_test, rmse_test, r2_test, y_test_inv, y_pred_test_inv = calcular_metricas(
    Y_test_scaled,
    y_pred_test_scaled,
    scaler
)

tempo_total = time.time() - start_global      # Calcula o tempo total do script

periodo_utilizado = (                         # Cria uma variavel com o periodo usado, se for 4h será 500, se for diario ou
    PERIOD
    if PERIOD is not None
    else f'{START_DATE} a {END_DATE}'
)

# ============================================================
# 10. RESULTADOS
# ============================================================

resultados = pd.DataFrame({                       # Criando uma tabela com os resultados
    'Modelo': ['LSTM_Classica_Fixa'],             # Nome do modelo
    'Ticker': [TICKER],                           # Ativo utilizado
    'Intervalo': [INTERVAL],                      # Intervalo dos dados
    'Periodo': [periodo_utilizado],               # Período dos dados
    'Time_Step': [TIME_STEP],                     # janela temporal

    'MAPE_Treino': [mape_train],                  # Metricas do treino
    'MSE_Treino': [mse_train],
    'RMSE_Treino': [rmse_train],
    'R2_Treino': [r2_train],

    'MAPE_Teste': [mape_test],                    # Metricas do teste
    'MSE_Teste': [mse_test],
    'RMSE_Teste': [rmse_test],
    'R2_Teste': [r2_test],

    'Tempo_Treino': [tempo_treino],               # Tempos de execuções
    'Tempo_Teste': [tempo_teste],
    'Tempo_Total': [tempo_total],

    'Batch_Size': [BATCH_SIZE],                   # Hiperparametros utilizados na rede LSTM
    'Epochs': [EPOCHS],
    'Camadas': [NUM_LAYERS],
    'Neuronios': [NUM_NEURONS],
    'Learning_Rate': [LEARNING_RATE],
    'Dropout': [DROPOUT_RATE]
})
```

```
})


pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', None)

print("\nRESULTADOS - LSTM CLÁSSICA FIXA")          # Mostra a tabela na tela
display(resultados)

resultados.to_csv(                                    # Salva os resultados em CSV.
    f"resultado_lstm_classica_fixa_{NOME_BASE}.csv",
    index=False
)

# ============================================================
# 11. GRÁFICO TREINAMENTO
# ============================================================

plt.figure(figsize=(12, 5))                    # Cria um figura

plt.plot(                                      # Plota os valores reais do treino
    y_train_inv,
    label='Real'
)

plt.plot(                                      # Plota os valores previstos no treino
    y_pred_train_inv,
    label='Previsto'
)

plt.title(                                             # Define o título do grafico
    f"LSTM Clássica Fixa - Treinamento - {NOME_BASE}"
)

plt.xlabel("Tempo")                    # Define o nome dos eixos
plt.ylabel("Preço do Café")
plt.legend()                           # Adiciona legenda, grade e ajusta layout
plt.grid()
plt.tight_layout()

plt.savefig(                                            # Salva o gráfico
    f"lstm_classica_fixa_treinamento_{NOME_BASE}.png"
)

plt.show()
plt.close()                                             # Fecha a figura, para liberar memoria

# ============================================================
# 12. GRÁFICO TESTE
# ============================================================

plt.figure(figsize=(12, 5))          # Cria uma figura

plt.plot(                            # Mostra se o modelo conseguiu prever bem os dados que não foram usados no treino.
    y_test_inv,
    label='Real'
)

plt.plot(
    y_pred_test_inv,
    label='Previsto'
)

plt.title(
    f"LSTM Clássica Fixa - Teste - {NOME_BASE}"
)

plt.xlabel("Tempo")
plt.ylabel("Preço do Café")
plt.legend()
plt.grid()
plt.tight_layout()

plt.savefig(                                      # Salva o grafico de teste.
    f"lstm_classica_fixa_teste_{NOME_BASE}.png"
)

plt.show()
plt.close()

# ============================================================
# 13. GRÁFICO DE CONVERGÊNCIA
```

```
# ============================================================

plt.figure(figsize=(10, 5))             # Cria o gráfico

plt.plot(                               # Ploto o erro do treino em cada epoca
    history.history['loss'],
    label='Loss - Treino'
)


plt.title(
    f"Convergência - LSTM Clássica Fixa - {NOME_BASE}"
)

plt.xlabel("Épocas")
plt.ylabel("Erro MSE")
plt.legend()
plt.grid()
plt.tight_layout()

plt.savefig(
    f"convergencia_lstm_classica_fixa_{NOME_BASE}.png"
)

plt.show()
plt.close()

print(f"\nTempo total de execução: {tempo_total:.2f} segundos")   # Mostra o tempo total.
```

Mostrar saída oculta

APÊNDICE **B**

LSTM AG

```
# ============================================================
# CÓDIGO 2 - LSTM + AG
# ALGORITMO GENÉTICO PARA OTIMIZAÇÃO DE HIPERPARÂMETROS DA LSTM
# ============================================================

import os
import certifi
os.environ['SSL_CERT_FILE'] = certifi.where()
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import time
import random                       # Importa funções aleatorias, no AG gera a população inicial, seleciona pais e etc...
from collections import Counter
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import yfinance as yf
import tensorflow as tf

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import backend as K

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import TimeSeriesSplit

from google.colab import files

# ============================================================
# CONFIGURAÇÕES GERAIS DO EXPERIMENTO
# ============================================================

TICKER = 'KC=F'

# Escolha:
# '4h'  = últimos 500 dias
# '1d'  = 01/01/2000 até 31/12/2020
# '1wk' = 01/01/2000 até 31/12/2020

INTERVAL = '4h'  # ALTERE AQUI

if INTERVAL == '4h':
    PERIOD = '500d'
    START_DATE = None
    END_DATE = None
    NOME_BASE = '4h'

elif INTERVAL == '1d':
    PERIOD = None
    START_DATE = '2000-01-01'
    END_DATE = '2020-12-31'
    NOME_BASE = 'diario'

elif INTERVAL == '1wk':
    PERIOD = None
    START_DATE = '2000-01-01'
    END_DATE = '2020-12-31'
    NOME_BASE = 'semanal'

else:
    raise ValueError("Intervalo inválido. Use '4h', '1d' ou '1wk'.")

TIME_STEP = 30
TEST_SIZE_RATIO = 0.20

# ============================================================
# CONFIGURAÇÕES DO ALGORITMO GENÉTICO
# ============================================================

GENERATIONS = 5
POPULATION_SIZE = 30
MUTATION_RATE = 0.1
N_SPLITS = 2

start_global = time.time()

# ============================================================
# 1. DOWNLOAD DOS DADOS
```

```
# 1. DOWNLOAD DOS DADOS
# ============================================================

if PERIOD is not None:
    cafe = yf.download(
        TICKER,
        period=PERIOD,
        interval=INTERVAL
    )[['Close']]
else:
    cafe = yf.download(
        TICKER,
        start=START_DATE,
        end=END_DATE,
        interval=INTERVAL
    )[['Close']]

cafe = cafe.dropna()
cafe.index = pd.to_datetime(cafe.index)

# ============================================================
# 2. CRIAÇÃO DAS JANELAS TEMPORAIS
# ============================================================

def dataset_from_series(values, time_step=30):
    arr = np.asarray(values).reshape(-1)
    X_raw, Y_raw = [], []

    for i in range(len(arr) - time_step):
        X_raw.append(arr[i:i + time_step])
        Y_raw.append(arr[i + time_step])

    return np.array(X_raw), np.array(Y_raw)

X_raw, Y_raw = dataset_from_series(
    cafe['Close'].values,
    TIME_STEP
)

# ============================================================
# 3. DIVISÃO TREINO / TESTE TEMPORAL
# ============================================================

split_idx = int(len(X_raw) * (1 - TEST_SIZE_RATIO))

X_train_raw = X_raw[:split_idx]
X_test_raw = X_raw[split_idx:]

Y_train_raw = Y_raw[:split_idx]
Y_test_raw = Y_raw[split_idx:]

# ============================================================
# 4. NORMALIZAÇÃO SEM VAZAMENTO DE DADOS
# ============================================================

scaler = MinMaxScaler()

valores_treino_para_scaler = np.concatenate([
    X_train_raw.flatten(),
    Y_train_raw.flatten()
]).reshape(-1, 1)

scaler.fit(valores_treino_para_scaler)

X_train_scaled = scaler.transform(
    X_train_raw.reshape(-1, 1)
).reshape(X_train_raw.shape)

X_test_scaled = scaler.transform(
    X_test_raw.reshape(-1, 1)
).reshape(X_test_raw.shape)

Y_train_scaled = scaler.transform(
    Y_train_raw.reshape(-1, 1)
).reshape(-1)

Y_test_scaled = scaler.transform(
    Y_test_raw.reshape(-1, 1)
).reshape(-1)

X_train = X_train_scaled.reshape(
    X_train_scaled.shape[0],
    X_train_scaled.shape[1],
```

```
    1
)

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

# ============================================================
# 5. CRIAÇÃO DO MODELO LSTM
# ============================================================
# Cria uma LSTM variável, com os parâmetros escolhidos pelo AG
def create_model(input_shape, num_layers, num_neurons, learning_rate):
    num_layers = int(num_layers)             # Número de camadas
    num_neurons = int(num_neurons)           # Número de neurônios
    learning_rate = float(learning_rate)     # Taxa de aprendizado

    model = Sequential()                     # Cria um modelo de sequencial vazio

    model.add(                                 # Adiciona a primeira camada da LSTM
        LSTM(                                  # Recebe o numero de neurônios do AG
            num_neurons,
            return_sequences=(num_layers > 1),  # Se houver mais de uma camada, retorna a sequencia completa.
            input_shape=input_shape             # Se houver apenas uma camada, retorna so a saida final.
        )
    )

    for _ in range(max(0, num_layers - 2)): # Cria camadas intermediária
        model.add(                   # Adiciona camada LSTM intermediária
            LSTM(
                num_neurons,
                return_sequences=True # Retorna outra sequencia porque ainda haverá outra camada depois
            )
        )
        model.add(Dropout(0.2))       # Adiciona dropout de 20% após a camada intermediária

    if num_layers > 1:                # verifica se há mais de uma camada
        model.add(                    # Adiciona a última camada LSTM
            LSTM(num_neurons)
        )

    model.add(Dense(1))               # Adiciona a camada de saída
                                      # Ela gera um único número: o próximo preço previsto
    model.compile(                    # Compila o modelo, usa otimizador Adam, a taxa de aprendizado definido pelo individuo
        optimizer=Adam(learning_rate=learning_rate),
        loss='mse'
    )

    return model        # Retorna o modelo pronto

# ============================================================
# 6. FUNÇÃO FITNESS
# Menor MSE médio = melhor indivíduo
# ============================================================
# Essa função avalia a qualidade de um individuo
def fitness_function(individual, X_local, Y_local):

    if X_local.shape[0] <= N_SPLITS: # Se houver poucas amostras retorna um erro muito grande
        return 1e10

    try:                             # Começa um bloco de tentativa, se algo der errado o codigo nao trava.
        num_layers, num_neurons, lr, batch_size, epochs = individual # Separa os genes do individuo.
                                     # Garante que cada valor esteja no tipo correto
        num_layers = int(num_layers)
        num_neurons = int(num_neurons)
        lr = float(lr)
        batch_size = int(batch_size)
        epochs = int(epochs)

    except Exception:      # Se algo der errado na leitura dos genes, retorna um valor alto.
        return 1e10
    # Evita o overfitting temporal
    tscv = TimeSeriesSplit(n_splits=N_SPLITS)   # Cria uma validação temporal
    mse_scores = []        # Cria uma lista para guardar os MSEs de cada validação

    try:                   # Novo bloco de tentativa
        for train_idx, val_idx in tscv.split(X_local): # Percorre as divisões temporais
                                                       # uma parte anterior será treino e outra posterior será validação.
            X_tr = X_local[train_idx]     # Separa as entradas de treino e validação
            X_val = X_local[val_idx]

            Y_tr = Y_local[train_idx]     # Separa os alvos de treino e validação.
```

```
            Y_val = Y_local[val_idx]

            K.clear_session()             # Limpa a memória antes de criar nova LSTM.
                                          # importante porque essa função será chamda muitas vezes.
            model = create_model(         # Cria uma LSTM com os genes do indivíduo
                (X_tr.shape[1], X_tr.shape[2]), # o formato será (30,1)
                num_layers,
                num_neurons,
                lr
            )

            model.fit(                # Treina a LSTM
                X_tr,
                Y_tr,
                epochs=epochs,
                batch_size=batch_size,
                verbose=0             # Significa que não mostra o progresso na tela
            )

            preds = model.predict(    # Faz previsões no conjunto de validação
                X_val,
                verbose=0
            ).reshape(-1)

            mse_scores.append(        # Calcula o MSE da validação e guarda na lista
                mean_squared_error(
                    Y_val.reshape(-1),
                    preds
                )
            )

        return float(np.mean(mse_scores))  # Retorna o MSE médio das divisões

    except Exception:   # Se qualquer erro acontecer no treinamento, retorna erro alto.
        return 1e10

# ============================================================
# 7. MÉTRICAS
# ============================================================
# Calcula as métricas depois que o modelo final é treinado
def calcular_metricas(y_true_scaled, y_pred_scaled, scaler_used):
    # Desnormaliza os valores reais.
    y_true_inv = scaler_used.inverse_transform(
        y_true_scaled.reshape(-1, 1)
    ).reshape(-1)
    # Desnormaliza os valores previstos
    y_pred_inv = scaler_used.inverse_transform(
        y_pred_scaled.reshape(-1, 1)
    ).reshape(-1)
    # Calcula o MSE e o RMSE
    mse = mean_squared_error(y_true_inv, y_pred_inv)
    rmse = np.sqrt(mse)
    # Calcula o Mape, evitando divisão por zero
    mape = np.mean(
        np.abs(
            (y_true_inv - y_pred_inv) /
            np.maximum(y_true_inv, 1e-6)
        )
    ) * 100
    # Calcula o r2
    r2 = r2_score(y_true_inv, y_pred_inv)
    # Retorna as métricas e vallores desnormalizados.
    return mape, mse, rmse, r2, y_true_inv, y_pred_inv

# ============================================================
# 8. OPERADORES GENÉTICOS
# ============================================================
# Cria a população inicial
def initialize_population(size):
    # Retorna uma lista de individuos
    return [
        [
            random.randint(1, 4),                 # número de camadas
            random.randint(10, 100),               # neurônios
            round(random.uniform(0.0001, 0.01), 4),  # learning rate
            random.randint(16, 128),               # batch size
            random.randint(10, 100)                # epochs
        ]
        for _ in range(size)     # Repete isso até completar a quantidade de individuos definidos
    ]

def tournament_selection(pop, scores, k=3):  # Função de seleção por torneio, escolhe bons individuos para serem os pais
```

```
    k = min(k, len(pop))       # Garante que o torneio não terá mais canditados que a população

    idxs = random.sample(      # Sorteia 3 individuos aleatórios
        range(len(pop)),
        k
    )

    best_idx = min(            # Escolhe entre esses 3 o indivíduo com menor MSE
        idxs,
        key=lambda i: scores[i]
    )

    return pop[best_idx][:]    # retorna uma cópia desse individuo, ele será usado como pai.

def crossover(p1, p2):         # Cra a função de cruzamento
    # Cria copia dos pais para não alterar os originais
    p1 = p1[:]
    p2 = p2[:]
    # Escolhe aleatoriamente um ponto de corte, como tem 5 genes, o ponto pode ser entre 1 e 4.
    point = random.randint(
        1,
        len(p1) - 1
    )

    c1 = p1[:point] + p2[point:] # Recebe a primeira parte do pai 1 e a segunda parte do pai 2.
    c2 = p2[:point] + p1[point:] # Recebe a primeira parte do pai 2 e a segunda parte do pai 1.

    return c1, c2                # Retorna os dois filhos

def mutate(individual):          # Cria a função de mutação

    ind = individual[:]          # Faz uma cópia do individuo

    if random.random() < MUTATION_RATE:  # Sorteia um número entre 0 e 1.
                                         # Se menor que 0,1 ocorre a mutação
        gene_idx = random.randint(0, len(ind) - 1)  # Escolhe qual gene vai mutar

        if gene_idx == 0:           # Se o gene escolhido for 0, muda o número de camadas.
            ind[0] = random.randint(1, 4)

        elif gene_idx == 1:         # Ser for 1, muda o número de neurônios
            ind[1] = random.randint(10, 100)

        elif gene_idx == 2:         # Se for 2, muda a taxa de aprendizado
            ind[2] = round(
                random.uniform(0.0001, 0.01),
                4
            )

        elif gene_idx == 3:         # Se for 3, muda o batch size
            ind[3] = random.randint(16, 128)

        elif gene_idx == 4:         # Se for 4, muda o número de épocas
            ind[4] = random.randint(10, 100)

    return ind      # Retorna o indivíduo, mutado ou não


# ============================================================
# INSTRUMENTAÇÃO DA BUSCA EVOLUTIVA - FASE 2
# Registra cromossomos, fitness, elite e diversidade por geração.
# Esses arquivos permitem analisar a dinâmica da busca na dissertação.
# ============================================================
GENE_NAMES = ["Camadas", "Neuronios", "Learning_Rate", "Batch_Size", "Epochs"]

def _entropia_gene(valores):
    valores = list(valores)
    if len(valores) == 0:
        return 0.0
    contagem = Counter(valores)
    total = float(sum(contagem.values()))
    probs = np.array([v / total for v in contagem.values()], dtype=float)
    return float(-np.sum(probs * np.log2(probs + 1e-12)))

def calcular_diversidade_populacao(populacao):
    arr = np.array(populacao, dtype=object)
    diversidade = {
        "Tamanho_Populacao": len(populacao),
        "Cromossomos_Unicos": len({tuple(ind) for ind in populacao}),
        "Proporcao_Cromossomos_Unicos": len({tuple(ind) for ind in populacao}) / max(len(populacao), 1)
    }
    for j, nome in enumerate(GENE_NAMES):
```

```
        valores = arr[:, j].tolist() if len(populacao) > 0 else []
        diversidade[f"{nome}_Unicos"] = len(set(valores))
        diversidade[f"{nome}_Entropia"] = _entropia_gene(valores)
        if nome in ["Camadas", "Neuronios", "Batch_Size", "Epochs"]:
            valores_num = np.array(valores, dtype=float)
            diversidade[f"{nome}_Media"] = float(np.mean(valores_num))
            diversidade[f"{nome}_Desvio"] = float(np.std(valores_num))
        else:
            valores_num = np.array(valores, dtype=float)
            diversidade[f"{nome}_Media"] = float(np.mean(valores_num))
            diversidade[f"{nome}_Desvio"] = float(np.std(valores_num))
    return diversidade

def registrar_populacao_busca(historico_populacao, geracao, etapa, populacao, scores=None,
                              elite_indices=None, algoritmo="AG", taxa_transgenica=None):
    elite_indices = set([] if elite_indices is None else [int(i) for i in elite_indices])
    for idx, ind in enumerate(populacao):
        registro = {
            "Algoritmo": algoritmo,
            "Geracao": geracao,
            "Etapa": etapa,
            "Individuo": idx,
            "Camadas": int(ind[0]),
            "Neuronios": int(ind[1]),
            "Learning_Rate": float(ind[2]),
            "Batch_Size": int(ind[3]),
            "Epochs": int(ind[4]),
            "Fitness_MSE": float(scores[idx]) if scores is not None else np.nan,
            "Elite": idx in elite_indices,
            "Taxa_Transgenica": taxa_transgenica
        }
        historico_populacao.append(registro)

# ============================================================
# 9. ALGORITMO GENÉTICO PRINCIPAL
# ============================================================
# Cria a função principal do AG, recebe os dados de treino.
def genetic_algorithm(X, Y):
    # Cria a população inicial com 30 indivíduos
    population = initialize_population(
        POPULATION_SIZE
    )

    # Cria lista para guardar o tempo de cada geração
    generation_times = []
    best_mse_por_geracao = []  # Cria lista para guardar o melhor MSE de cada geração
    historico_geracoes = []    # Guarda estatísticas agregadas de diversidade por geração
    historico_populacao = []   # Guarda todos os cromossomos avaliados

    # Guarda a melhor configuração encontrada em TODAS as gerações
    # Isso evita perder um indivíduo muito bom que apareceu antes da última geração.
    global_best_individual = None
    global_best_mse = float("inf")
    global_best_generation = None
    global_best_origin = None

    # Define quantos indivíduos serão preservados por elitismo.
    top_n = max(
        1,
        POPULATION_SIZE // 2
    )

    for gen in range(GENERATIONS):   # Começa o laço das gerações

        inicio_geracao = time.time()   # Marca o inicio da geração

        # Avaliação da população: cada indivíduo recebe um MSE médio.
        scores = [
            fitness_function(ind, X, Y)
            for ind in population
        ]

        # Pega o menor MSE da geração.
        best_mse = float(min(scores))
        best_mse_por_geracao.append(best_mse)   # Guarda esse melhor MSE

        # Identifica o melhor indivíduo desta geração.
        best_idx_gen = int(
            np.argmin(scores)
        )

        # Atualiza o melhor global se o melhor desta geração for superior
        # ao melhor já encontrado anteriormente.
```

```
        if best_mse < global_best_mse:
            global_best_mse = best_mse
            global_best_individual = population[best_idx_gen][:]
            global_best_generation = gen + 1
            global_best_origin = "avaliacao_da_geracao"

        # Mostra na tela a geração atual, o melhor MSE da geração
        # e o melhor MSE global até o momento.
        print(
            f"Geração {gen + 1} "
            f"- Melhor MSE da geração: {best_mse:.6f} "
            f"| Melhor MSE global: {global_best_mse:.6f}"
        )

        # Elitismo: seleciona os 50% melhores indivíduos
        best_idx = np.argsort(scores)[:top_n]

        # Instrumentação: registra população avaliada, fitness, elites e diversidade.
        diversidade = calcular_diversidade_populacao(population)
        diversidade.update({
            "Algoritmo": "AG",
            "Geracao": gen + 1,
            "Ordem_Grafico": gen + 1,
            "Rotulo_Geracao": f"G{gen + 1}",
            "Etapa": "avaliacao_inicio_geracao",
            "Melhor_MSE_Geracao": best_mse,
            "Media_MSE_Geracao": float(np.mean(scores)),
            "Desvio_MSE_Geracao": float(np.std(scores)),
            "Melhor_MSE_Global_Ate_Geracao": global_best_mse
        })
        historico_geracoes.append(diversidade)
        registrar_populacao_busca(
            historico_populacao,
            geracao=gen + 1,
            etapa="avaliacao_inicio_geracao",
            populacao=population,
            scores=scores,
            elite_indices=best_idx,
            algoritmo="AG"
        )

        # Copia os melhores indivíduos. Estes serão preservados.
        best_parents = [
            population[i][:]
            for i in best_idx
        ]

        # Geração dos filhos por seleção, crossover e mutação
        offspring = []

        # Gera filhos até completar a população.
        while len(offspring) < (
            POPULATION_SIZE - top_n
        ):
            # Seleciona o primeiro pai por torneio
            p1 = tournament_selection(
                population,
                scores
            )

            # Seleciona o segundo pai por torneio
            p2 = tournament_selection(
                population,
                scores
            )

            # Faz o crossover entre os pais e gera dois filhos.
            o1, o2 = crossover(p1, p2)

            # Aplica mutação no filho 1 e adiciona à lista de filhos.
            offspring.append(
                mutate(o1)
            )

            # Verifica se ainda precisa de mais filhos.
            if len(offspring) < (
                POPULATION_SIZE - top_n
            ):
                offspring.append(
                    mutate(o2)
                )

        # Nova população da próxima geração, contendo os melhores da geração anterior
```

```
        # mais os filhos gerados.
        population = best_parents + offspring

        fim_geracao = time.time()     # Marca o fim da geração

        generation_times.append(           # Guarda o tempo gasto naquela geração.
            fim_geracao - inicio_geracao
        )

    # Avaliação final da população, depois das gerações.
    final_scores = [
        fitness_function(ind, X, Y)
        for ind in population
    ]

    # Pega o melhor indivíduo da população final.
    best_index_final = int(
        np.argmin(final_scores)
    )

    best_final_mse = float(
        final_scores[best_index_final]
    )

    # Compara o melhor da população final com o melhor global encontrado nas gerações.
    # Se a população final tiver um indivíduo melhor, ele passa a ser o melhor global.
    if best_final_mse < global_best_mse:
        global_best_mse = best_final_mse
        global_best_individual = population[best_index_final][:]
        global_best_generation = "Final"
        global_best_origin = "avaliacao_final_da_populacao"

    diversidade_final = calcular_diversidade_populacao(population)
    diversidade_final.update({
        "Algoritmo": "AG",
        "Geracao": "Final",
        "Ordem_Grafico": GENERATIONS + 1,
        "Rotulo_Geracao": "Final",
        "Etapa": "avaliacao_final_populacao",
        "Melhor_MSE_Geracao": best_final_mse,
        "Media_MSE_Geracao": float(np.mean(final_scores)),
        "Desvio_MSE_Geracao": float(np.std(final_scores)),
        "Melhor_MSE_Global_Ate_Geracao": global_best_mse
    })
    historico_geracoes.append(diversidade_final)
    registrar_populacao_busca(
        historico_populacao,
        geracao="Final",
        etapa="avaliacao_final_populacao",
        populacao=population,
        scores=final_scores,
        elite_indices=[best_index_final],
        algoritmo="AG"
    )

    return (                          # Retorna:
        global_best_individual,       # Melhor configuração global encontrada
        best_mse_por_geracao,         # Melhor MSE por geração
        generation_times,             # Tempo de cada geração
        sum(generation_times),        # Tempo total do AG
        global_best_mse,              # Melhor MSE global
        global_best_generation,       # Geração onde o melhor global foi encontrado
        global_best_origin,           # Origem do melhor global
        historico_geracoes,           # Estatísticas de diversidade por geração
        historico_populacao           # Cromossomos, fitness e elites por geração
    )

# ============================================================
# 10. EXECUÇÃO DO AG
# ============================================================
# Marca inicio da execução do AG
inicio_ag = time.time()
# Executa o AG usando os dados de treino. Retorna a melhor configuração.
(
    best_config,
    mse_geracoes,
    tempos_geracoes,
    tempo_ag,
    melhor_mse_global,
    geracao_melhor_individuo,
    origem_melhor_individuo,
    historico_geracoes_ag,
```

```
    historico_populacao_ag
) = genetic_algorithm(
    X_train,
    Y_train_scaled
)
# Calcula o tempo total real do AG.
tempo_ag = time.time() - inicio_ag

# Exporta os logs da busca evolutiva para análise posterior da dissertação.
df_historico_geracoes_ag = pd.DataFrame(historico_geracoes_ag)
df_historico_populacao_ag = pd.DataFrame(historico_populacao_ag)
df_historico_geracoes_ag.to_csv(f"historico_geracoes_ag_{NOME_BASE}.csv", index=False)
df_historico_populacao_ag.to_csv(f"historico_populacao_ag_{NOME_BASE}.csv", index=False)
print("Logs da busca AG salvos:")
print(f"- historico_geracoes_ag_{NOME_BASE}.csv")
print(f"- historico_populacao_ag_{NOME_BASE}.csv")

# Separa os genes da melhor configuração
num_layers, num_neurons, lr, batch_size, epochs = best_config
# Converte para os tipos corretos
num_layers = int(num_layers)
num_neurons = int(num_neurons)
lr = float(lr)
batch_size = int(batch_size)
epochs = int(epochs)

print("\nMELHOR CONFIGURAÇÃO GLOBAL ENCONTRADA - AG")   # Mostra a melhor configuração global encontrada.
print(f"Melhor MSE global: {melhor_mse_global:.6f}")
print(f"Geração do melhor indivíduo: {geracao_melhor_individuo}")
print(f"Origem do melhor indivíduo: {origem_melhor_individuo}")
print(f"Camadas: {num_layers}")
print(f"Neurônios: {num_neurons}")
print(f"Learning Rate: {lr}")
print(f"Batch Size: {batch_size}")
print(f"Epochs: {epochs}")

# ============================================================
# 11. TREINAMENTO FINAL COM A MELHOR CONFIGURAÇÃO
# ============================================================
# Limpa a sessão antes de criar o modelo final.
K.clear_session()
# Cria a LSTM final com a melhor configuração encontrada pelo AG.
final_model = create_model(
    (X_train.shape[1], X_train.shape[2]),
    num_layers,
    num_neurons,
    lr
)
# Mostra o início do treino final.
inicio_treino = time.time()
# Treina o modelo final usando todo o conjunto de treino.
final_model.fit(
    X_train,
    Y_train_scaled,
    epochs=epochs,
    batch_size=batch_size,
    verbose=1
)
# Calcula o tempo do treino final
tempo_treino = time.time() - inicio_treino

# ============================================================
# 12. PREVISÕES
# ============================================================
# Marca o inicio do tempo da etapa de previsão.
inicio_teste = time.time()
# Faz previsão no treino
y_pred_train_scaled = final_model.predict(
    X_train,
    verbose=0
).reshape(-1)
# Faz previsão no teste
y_pred_test_scaled = final_model.predict(
    X_test,
    verbose=0
).reshape(-1)
# Calcula o tempo das previsões
tempo_teste = time.time() - inicio_teste

# ============================================================
# 13. MÉTRICAS
# ============================================================
# Calcula as metricas no treino
```

```
mape_train, mse_train, rmse_train, r2_train, y_train_inv, y_pred_train_inv = calcular_metricas(
    Y_train_scaled,
    y_pred_train_scaled,
    scaler
)
# Calcula as metricas no teste
mape_test, mse_test, rmse_test, r2_test, y_test_inv, y_pred_test_inv = calcular_metricas(
    Y_test_scaled,
    y_pred_test_scaled,
    scaler
)
# Calcula o tempo total do script
tempo_total = time.time() - start_global
# Cria uma variavel com o periodo usado
periodo_utilizado = (
    PERIOD
    if PERIOD is not None
    else f'{START_DATE} a {END_DATE}'
)

# ============================================================
# 14. RESULTADOS
# ============================================================
# Cria a tabela final
resultados = pd.DataFrame({
    'Modelo': ['LSTM_AG'],
    'Ticker': [TICKER],
    'Intervalo': [INTERVAL],
    'Periodo': [periodo_utilizado],
    'Time_Step': [TIME_STEP],

    'MAPE_Treino': [mape_train],
    'MSE_Treino': [mse_train],
    'RMSE_Treino': [rmse_train],
    'R2_Treino': [r2_train],

    'MAPE_Teste': [mape_test],
    'MSE_Teste': [mse_test],
    'RMSE_Teste': [rmse_test],
    'R2_Teste': [r2_test],

    'Tempo_AG': [tempo_ag],
    'Tempo_Treino': [tempo_treino],
    'Tempo_Teste': [tempo_teste],
    'Tempo_Total': [tempo_total],
     # Melhores hiperparametros encontrados
    'Batch_Size': [batch_size],
    'Epochs': [epochs],
    'Camadas': [num_layers],
    'Neuronios': [num_neurons],
    'Learning_Rate': [lr],
     # Configurações do AG
    'Generations': [GENERATIONS],
    'Population_Size': [POPULATION_SIZE],
    'Mutation_Rate': [MUTATION_RATE],
    'Melhor_MSE_AG': [melhor_mse_global],
    'Geracao_Melhor_Individuo': [geracao_melhor_individuo],
    'Origem_Melhor_Individuo': [origem_melhor_individuo]
})
# Mostra todos os resultados
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', None)

print("\nRESULTADOS - LSTM + AG")
display(resultados)
# Salva a tabela em CSV
resultados.to_csv(
    f"resultado_lstm_ag_{NOME_BASE}.csv",
    index=False
)

# ============================================================
# 15. GRÁFICO DE CONVERGÊNCIA
# ============================================================

plt.figure(figsize=(10, 5))    # Cria uma figura
# Plota o melhor MSE no início de cada geração e inclui a avaliação final da população.
# A avaliação final corresponde aos indivíduos gerados após a última reprodução,
# evitando confusão entre "melhor da geração 5" e "melhor da população final".
df_convergencia_ag = df_historico_geracoes_ag[
    df_historico_geracoes_ag["Etapa"].isin(["avaliacao_inicio_geracao", "avaliacao_final_populacao"])
].copy()
```

```
df_convergencia_ag = df_convergencia_ag.sort_values("Ordem_Grafico")

plt.plot(
    df_convergencia_ag["Rotulo_Geracao"],
    df_convergencia_ag["Melhor_MSE_Geracao"],
    marker='o'
)
# Titulo do grafico
plt.title(
    f"Convergência - LSTM + AG - {NOME_BASE}"
)
# Define os eixos e adiciona grade e ajusta layout.
plt.xlabel("Etapa da busca")
plt.ylabel("Melhor MSE")
plt.grid()
plt.tight_layout()
# Salva o grafico
plt.savefig(
    f"convergencia_lstm_ag_{NOME_BASE}.png"
)
# Mostra e depois fecha o grafico
plt.show()
plt.close()

# ============================================================
# 16. GRÁFICO TREINAMENTO
# ============================================================
# Cria o grafico
plt.figure(figsize=(12, 5))
# Plota o grafico de valores reais do treino
plt.plot(
    y_train_inv,
    label='Real'
)
# Plota valores previstos no treino
plt.plot(
    y_pred_train_inv,
    label='Previsto'
)
# Titulo do grafico
plt.title(
    f"LSTM + AG - Treinamento - {NOME_BASE}"
)
# Eixos, legenda, grade e ajuste
plt.xlabel("Tempo")
plt.ylabel("Preço do Café")
plt.legend()
plt.grid()
plt.tight_layout()
# Salva o grafico
plt.savefig(
    f"lstm_ag_treinamento_{NOME_BASE}.png"
)
# Mostra e fecha o grafico
plt.show()
plt.close()

# ============================================================
# 17. GRÁFICO TESTE
# ============================================================

plt.figure(figsize=(12, 5))
# Plota valores reais do teste
plt.plot(
    y_test_inv,
    label='Real'
)
# Plota previsões do teste, aqui mostra o desempenho em dados que o modelo não usou no treino.
plt.plot(
    y_pred_test_inv,
    label='Previsto'
)

plt.title(
    f"LSTM + AG - Teste - {NOME_BASE}"
)

plt.xlabel("Tempo")
plt.ylabel("Preço do Café")
plt.legend()
plt.grid()
plt.tight_layout()
```

```
plt.savefig(
    f"lstm_ag_teste_{NOME_BASE}.png"
)
# Mostra e fecha o grafico
plt.show()
plt.close()

# ============================================================
# 18. TEMPOS POR GERAÇÃO
# ============================================================

print("\nTEMPOS POR GERAÇÃO")
for i, t in enumerate(tempos_geracoes):    # Percorre os tempos salvos
    print(f"Geração {i + 1}: {t:.2f} segundos")    # Mostra o tempo de cada geração

print(f"\nTempo total AG: {tempo_ag:.2f} segundos")  # Mostra o tempo total gasto pelo AG
print(f"Tempo total do script: {tempo_total:.2f} segundos") # Mostra o tempo do código inteiro.




files.download(f"historico_geracoes_ag_{NOME_BASE}.csv")
files.download(f"historico_populacao_ag_{NOME_BASE}.csv")
```

Mostrar saída oculta

APÊNDICE C

LSTM AGT

```
# ============================================================
# CÓDIGO 4 - LSTM + AGT ORIGINAL
# AGT ADAPTADO COM SUBSTITUIÇÃO DOS PIORES INDIVÍDUOS
# MÓDULOS M1, M2, M3 E M4 GERANDO INDIVÍDUOS TRANSGÊNICOS
# TAXA TRANSGÊNICA CÍCLICA POR GERAÇÃO: 10%, 30%, 50%, 70% E 90%
# ============================================================

# ============================================================
import os                                   # Mexe nas configurações do ambiente
import certifi                              # Ajuda a encontrar certificados de segurança para baixar dados da internet
os.environ['SSL_CERT_FILE'] = certifi.where() # Ajuda baixar os dados do yahoo finance sem erro de segurança
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'      # Reduz mensagens tecnicas do tensorflow

import time                                 # Usada para medir o tempo
import random                               # Usada para sorteios
import numpy as np                          # Usada para calculos numericos, arrays, médias e ordenação.
import pandas as pd                         # Cria tabelas
import matplotlib.pyplot as plt             # Cria graficos
from collections import Counter             # Conta quantas vezes cada gene aparece entre os melhores individuos

import yfinance as yf                       # Baixa os dados financeiros do yahoo
import tensorflow as tf                     # Usado para criar a rede LSTM

from tensorflow.keras.models import Sequential            # Importa o modelo sequencial, monta a rede camada por camada.
from tensorflow.keras.layers import LSTM, Dense, Dropout  # Importa as camadas da rede
from tensorflow.keras.optimizers import Adam              # Ajusta os pesos da rede durante o treinamento.
from tensorflow.keras import backend as K                 # Usado para limpar a memoria antes de criar novas redes.

from sklearn.preprocessing import MinMaxScaler            # Normaliza os dados na escala de 0 a 1.
from sklearn.metrics import mean_squared_error, r2_score  # Importanto as métricas
from sklearn.model_selection import TimeSeriesSplit       # Validação temporal, o modelo treine com dados anteriores e valid

from google.colab import files
# ============================================================
# CONFIGURAÇÕES GERAIS DO EXPERIMENTO

# ============================================================
TICKER = 'KC=F'                      # Simbolo do ativo, Cafe futuro

# Escolha:
# '4h'  = últimos 500 dias
# '1d'  = 01/01/2000 até 31/12/2020
# '1wk' = 01/01/2000 até 31/12/2020

INTERVAL = '4h'  # ALTERE AQUI

if INTERVAL == '4h':
    PERIOD = '500d'
    START_DATE = None
    END_DATE = None
    NOME_BASE = '4h'

elif INTERVAL == '1d':
    PERIOD = None
    START_DATE = '2000-01-01'
    END_DATE = '2020-12-31'
    NOME_BASE = 'diario'

elif INTERVAL == '1wk':
    PERIOD = None
    START_DATE = '2000-01-01'
    END_DATE = '2020-12-31'
    NOME_BASE = 'semanal'

else:
    raise ValueError("Intervalo inválido. Use '4h', '1d' ou '1wk'.")

TIME_STEP = 30              # Define a janela temporal, ultiliza os 30 ultimos preços para prever o proximo
TEST_SIZE_RATIO = 0.20      # Define 20% dos dados para teste

# ============================================================
# CONFIGURAÇÕES DO ALGORITMO GENÉTICO TRANSGÊNICO

# ============================================================
GENERATIONS = 5        # Define 5 gerações para o AG
POPULATION_SIZE = 30   # Define a população inicial de 30 individuos
MUTATION_RATE = 0.1    # Taxa de mutação de 10%
N_SPLITS = 2           # Define duas divisoes temporais na validação, cada individuo será avaliado em 2 blocos temporais

# Taxas transgênicas que serão aplicadas ciclicamente por geração
# Foi adicionada a taxa de 90% para evitar repetição da taxa de 10% na 5ª geração
```

```
# Foi adicionada a taxa de 90% para evitar repetição da taxa de 10% na 5ª geração.
TRANSGENIC_RATES = [0.10, 0.30, 0.50, 0.70, 0.90]

start_global = time.time()    # Marca o inicio da execução do codigo

# ============================================================
# 1. DOWNLOAD DOS DADOS

# ============================================================
if PERIOD is not None:       # Verifica se o codigo deve baixar dados por periodo relativo, acontece no 4h.
    cafe = yf.download(      # Baixa os dados da coluna close, que é o preço de fechamento.
        TICKER,
        period=PERIOD,
        interval=INTERVAL
    )[['Close']]
else:                        # Se não for 4h, baixa utilizando data inicial e final.
    cafe = yf.download(
        TICKER,
        start=START_DATE,
        end=END_DATE,
        interval=INTERVAL
    )[['Close']]

cafe = cafe.dropna()         # Remove linhas vazias
cafe.index = pd.to_datetime(cafe.index)   # Garante que o indice seja tratado como data/hora

# ============================================================
# 2. CRIAÇÃO DAS JANELAS TEMPORAIS

# ============================================================
# Cria uma função para transformar a série de preços em entradas e saídas
def dataset_from_series(values, time_step=30):
    arr = np.asarray(values).reshape(-1)      # Transforma os preços em um vetor simples.
    X_raw, Y_raw = [], []                     # Cria duas listas vazias, guarda as entradas e saidas previstas.

    for i in range(len(arr) - time_step):     # Percorre a série criando janelas
        X_raw.append(arr[i:i + time_step])    # Pega 30 valores consecutivos
        Y_raw.append(arr[i + time_step])      # Pega o próximo valor apos esses 30.

    return np.array(X_raw), np.array(Y_raw)   # Retorna as entradas e saídas como arrays Numpy.

X_raw, Y_raw = dataset_from_series(           # Aplica a função aos preços de fechamento.
    cafe['Close'].values,
    TIME_STEP
)

# ============================================================
# 3. DIVISÃO TREINO / TESTE TEMPORAL

# ============================================================
split_idx = int(len(X_raw) * (1 - TEST_SIZE_RATIO))  # Calcula o ponto de corte, sendo 80% treino e 20% teste.

X_train_raw = X_raw[:split_idx]     # Divide as entradas em treino e teste.
X_test_raw = X_raw[split_idx:]
                                    # Divide os valores reais em treino e teste.
Y_train_raw = Y_raw[:split_idx]
Y_test_raw = Y_raw[split_idx:]

# ============================================================
# 4. NORMALIZAÇÃO SEM VAZAMENTO DE DADOS

# ============================================================
scaler = MinMaxScaler()                       # Cria o normalizador.

valores_treino_para_scaler = np.concatenate([  # Junta todos os valores do treino, usa apenas no treino.
    X_train_raw.flatten(),
    Y_train_raw.flatten()
]).reshape(-1, 1)

scaler.fit(valores_treino_para_scaler)         # Aprende o minimo e o maximo do treino.

X_train_scaled = scaler.transform(             # Normaliza as entradas de treino usando o scaler do treino.
    X_train_raw.reshape(-1, 1)
).reshape(X_train_raw.shape)

X_test_scaled = scaler.transform(              # Normaliza as entradas de teste usando o scaler do treino.
    X_test_raw.reshape(-1, 1)
).reshape(X_test_raw.shape)

Y_train_scaled = scaler.transform(             # Normaliza  os valores reais do treino.
    Y_train_raw.reshape(-1, 1)
).reshape(-1)
```

```
Y_test_scaled = scaler.transform(             # Normaliza os valores reais do teste.
    Y_test_raw.reshape(-1, 1)
).reshape(-1)

X_train = X_train_scaled.reshape(             # Coloca o X de treino no formato da LSTM (amostras, passos temporais, variáv
    X_train_scaled.shape[0],
    X_train_scaled.shape[1],
    1
)

X_test = X_test_scaled.reshape(               # faz o mesmo para o teste.
    X_test_scaled.shape[0],
    X_test_scaled.shape[1],
    1
)

# ============================================================
# 5. CRIAÇÃO DO MODELO LSTM

# ============================================================
# Cria uma LSTM com base nos genes do indivíduo.
def create_model(input_shape, num_layers, num_neurons, learning_rate):
    num_layers = int(num_layers)           # Converte os valores para os tipos corretos.
    num_neurons = int(num_neurons)
    learning_rate = float(learning_rate)

    model = Sequential()                   # Cria um modelo vazio.

    model.add(                             # Cria a primeira camada LSTM
        LSTM(
            num_neurons,
            return_sequences=(num_layers > 1), # Se houver mais de uma LSTM, usa return_sequences=True
            input_shape=input_shape
        )
    )

    for _ in range(max(0, num_layers - 2)):    # Cria camadas LSTM intermediárias.
        model.add(                             # Adiciona uma LSTM intermediária.
            LSTM(
                num_neurons,
                return_sequences=True
            )
        )
        model.add(Dropout(0.2))                # Adiciona dropout de 20%

    if num_layers > 1:                         # Se houver mais de uma LSTM, adiciona a última LSTM.
        model.add(
            LSTM(num_neurons)
        )

    model.add(Dense(1))                        # Adiciona a camada de saída, gera um único valor previsto.

    model.compile(                             # Compila o modelo usando Adam e MSE.
        optimizer=Adam(learning_rate=learning_rate),
        loss='mse'
    )

    return model                               # Retorna o modelo pronto.

# ============================================================
# 6. FUNÇÃO FITNESS
# Menor MSE médio = melhor indivíduo

# ============================================================
def fitness_function(individual, X_local, Y_local):

    if X_local.shape[0] <= N_SPLITS:    # Se houver poucos dados, retorna erro muito alto.
        return 1e10

    try:
        num_layers, num_neurons, lr, batch_size, epochs = individual  # Separa os genes do individuo
                                       # Converte os genes para tipos corretos
        num_layers = int(num_layers)
        num_neurons = int(num_neurons)
        lr = float(lr)
        batch_size = int(batch_size)
        epochs = int(epochs)

    except Exception:                  # Se houver erro, retorna erro alto.
        return 1e10

    tscv = TimeSeriesSplit(n_splits=N_SPLITS)  # Cria validação temporal
    mse_scores = []                            # Cria lista para guardar os MSEs.
```

```
    try:
        for train_idx, val_idx in tscv.split(X_local):  # Para cada divisão temporal, treina no passado e valida no futuro.

            X_tr = X_local[train_idx]        # Separa treino e validação
            X_val = X_local[val_idx]

            Y_tr = Y_local[train_idx]
            Y_val = Y_local[val_idx]

            K.clear_session()                # Limpa a memória antes de criar nova LSTM

            model = create_model(            # Cria uma LSTM com os genes daqueles indivíduo.
                (X_tr.shape[1], X_tr.shape[2]),
                num_layers,
                num_neurons,
                lr
            )

            model.fit(                       # treina a LSTM
                X_tr,
                Y_tr,
                epochs=epochs,
                batch_size=batch_size,
                verbose=0
            )

            preds = model.predict(           # Faz previsões na validação
                X_val,
                verbose=0
            ).reshape(-1)

            mse_scores.append(               # Calcula e guarda o MSE
                mean_squared_error(
                    Y_val.reshape(-1),
                    preds
                )
            )

        return float(np.mean(mse_scores))    # Retorna o MSE médio

    except Exception:
        return 1e10

# ============================================================
# 7. MÉTRICAS

# ============================================================
# Calcula as metricas depois que o modelo final for treinado
def calcular_metricas(y_true_scaled, y_pred_scaled, scaler_used):

    y_true_inv = scaler_used.inverse_transform(    # Desnormalia o valor real
        y_true_scaled.reshape(-1, 1)
    ).reshape(-1)

    y_pred_inv = scaler_used.inverse_transform(    # Desnormaliza a previsao
        y_pred_scaled.reshape(-1, 1)
    ).reshape(-1)

    mse = mean_squared_error(y_true_inv, y_pred_inv)
    rmse = np.sqrt(mse)

    mape = np.mean(
        np.abs(
            (y_true_inv - y_pred_inv) /
            np.maximum(y_true_inv, 1e-6)
        )
    ) * 100

    r2 = r2_score(y_true_inv, y_pred_inv)

    return mape, mse, rmse, r2, y_true_inv, y_pred_inv # Retorna as métricas e os valores desnormalizados.

# ============================================================
# 8. OPERADORES GENÉTICOS BÁSICOS

# ============================================================
def initialize_population(size):

    return [
        [
            random.randint(1, 4),                  # número de camadas LSTM
            random.randint(10, 100)                # número de neurônios
```

```
            random.randint(16, 128),  # Número de neurônios
            round(random.uniform(0.0001, 0.01), 4),  # learning rate
            random.randint(16, 128),                # batch size
            random.randint(10, 100)                 # epochs
        ]
        for _ in range(size)
    ]

def tournament_selection(pop, scores, k=3):   # Seleciona um pai

    k = min(k, len(pop))

    idxs = random.sample(                # Sorteia 3 individuos
        range(len(pop)),
        k
    )

    best_idx = min(                      # Escolhe o indivíduo com menor MSE entre os 3.
        idxs,
        key=lambda i: scores[i]
    )

    return pop[best_idx][:]              # Retorna uma cópia do vencedor.

def crossover(p1, p2):                   # Recebe dois pais

    p1 = p1[:]                           # Cria copia dos pais
    p2 = p2[:]

    point = random.randint(              # Escolhe um ponto de corte
        1,
        len(p1) - 1
    )

    c1 = p1[:point] + p2[point:]         # Cria dois filhos misturando genes dos pais, apartir do ponto de corte.
    c2 = p2[:point] + p1[point:]

    return c1, c2                        # Retorna os filhos

def mutate(individual):                  # Cria a função de mutação

    ind = individual[:]                  # Faz uma copia do individuo

    if random.random() < MUTATION_RATE:  # 10% de chance que ocorra mutação.

        gene_idx = random.randint(0, len(ind) - 1)  # Escolhe aleatoriamente qual gene será alterado.

        if gene_idx == 0:
            ind[0] = random.randint(1, 4)

        elif gene_idx == 1:
            ind[1] = random.randint(10, 100)

        elif gene_idx == 2:
            ind[2] = round(
                random.uniform(0.0001, 0.01),
                4
            )

        elif gene_idx == 3:
            ind[3] = random.randint(16, 128)

        elif gene_idx == 4:
            ind[4] = random.randint(10, 100)

    return ind


# ============================================================
# INSTRUMENTAÇÃO DA BUSCA EVOLUTIVA - FASE 2
# Registra cromossomos, fitness, elite e diversidade por geração.
# Esses arquivos permitem analisar a dinâmica da busca na dissertação.
# ============================================================
GENE_NAMES = ["Camadas", "Neuronios", "Learning_Rate", "Batch_Size", "Epochs"]

def _entropia_gene(valores):
    valores = list(valores)
    if len(valores) == 0:
        return 0.0
    contagem = Counter(valores)
    total = float(sum(contagem.values()))
    probs = np.array([v / total for v in contagem.values()], dtype=float)
    return float(-np.sum(probs * np.log2(probs + 1e-12)))
```

```
def calcular_diversidade_populacao(populacao):
    arr = np.array(populacao, dtype=object)
    diversidade = {
        "Tamanho_Populacao": len(populacao),
        "Cromossomos_Unicos": len({tuple(ind) for ind in populacao}),
        "Proporcao_Cromossomos_Unicos": len({tuple(ind) for ind in populacao}) / max(len(populacao), 1)
    }
    for j, nome in enumerate(GENE_NAMES):
        valores = arr[:, j].tolist() if len(populacao) > 0 else []
        diversidade[f"{nome}_Unicos"] = len(set(valores))
        diversidade[f"{nome}_Entropia"] = _entropia_gene(valores)
        if nome in ["Camadas", "Neuronios", "Batch_Size", "Epochs"]:
            valores_num = np.array(valores, dtype=float)
            diversidade[f"{nome}_Media"] = float(np.mean(valores_num))
            diversidade[f"{nome}_Desvio"] = float(np.std(valores_num))
        else:
            valores_num = np.array(valores, dtype=float)
            diversidade[f"{nome}_Media"] = float(np.mean(valores_num))
            diversidade[f"{nome}_Desvio"] = float(np.std(valores_num))
    return diversidade

def registrar_populacao_busca(historico_populacao, geracao, etapa, populacao, scores=None,
                              elite_indices=None, algoritmo="AG", taxa_transgenica=None):
    elite_indices = set([] if elite_indices is None else [int(i) for i in elite_indices])
    for idx, ind in enumerate(populacao):
        registro = {
            "Algoritmo": algoritmo,
            "Geracao": geracao,
            "Etapa": etapa,
            "Individuo": idx,
            "Camadas": int(ind[0]),
            "Neuronios": int(ind[1]),
            "Learning_Rate": float(ind[2]),
            "Batch_Size": int(ind[3]),
            "Epochs": int(ind[4]),
            "Fitness_MSE": float(scores[idx]) if scores is not None else np.nan,
            "Elite": idx in elite_indices,
            "Taxa_Transgenica": taxa_transgenica
        }
        historico_populacao.append(registro)

# ============================================================
# 9. HISTÓRICO TRANSGÊNICO
# Conta os genes dos melhores indivíduos selecionados por elitismo

# ============================================================
# Cria uma classe, para guardar a frequencia dos genes.
class HistoricoTransgenicoOriginal:

    def __init__(self, n_genes):   # Inica o historico
        self.n_genes = n_genes     # Guarda a quantidade de genes
        self.ocorrencias = [       # Cria um contador para cada posição genetica
            Counter()
            for _ in range(n_genes)
        ]

    def atualizar(self, melhores_individuos):  # Atualiza o historico usando os melhores individuos da geração.
        """
        Atualiza a frequência genética com base nos melhores
        indivíduos da geração atual.
        """

        self.ocorrencias = [              # Zera os contadores a cada geração, isso significa que o historico é da geração a
            Counter()
            for _ in range(self.n_genes)
        ]

        for ind in melhores_individuos:     # Percorre os melhores individuos

            for i, gene in enumerate(ind):  # Percorre cada gene do individuo e gera as posições de cada um.

                self.ocorrencias[i][gene] += 1  # Conta os genes

    def selecionar_genes_original(self, k):   # Seleciona "k" genes, os mais frequentes tem mais chances.
        """
        Seleciona k genes usando frequência direta.
        Genes mais frequentes entre os melhores indivíduos
        têm maior probabilidade de serem escolhidos.
        """

        genes_escolhidos = {}         # Cria um dicionario vazio, guarda a posição e o gene escolhido.
```

```
        positions_with_counts = [     # Cria uma lista com posições que tem contagem
            i
            for i, c in enumerate(self.ocorrencias)
            if len(c) > 0
        ]

        if not positions_with_counts:  # Se não tiver, retorna vazio
            return {}

        k_real = min(k, len(positions_with_counts))

        posicoes_escolhidas = random.sample(
            positions_with_counts,
            k_real
        )

        for idx in posicoes_escolhidas:

            valores = list(
                self.ocorrencias[idx].keys()
            )

            frequencias = np.array(
                list(self.ocorrencias[idx].values()),
                dtype=float
            )

            valor = random.choices(
                valores,
                weights=frequencias,
                k=1
            )[0]

            genes_escolhidos[idx] = valor

        return genes_escolhidos

# ============================================================
# 10. OPERADOR TRANSGÊNICO
# M1, M2, M3 e M4 aplicados em paralelo
# N1 = N2 = N3 = N4

# ============================================================
# Calcula quantos individuos cada módulo vai modificar
def calcular_N_modulos(population_size, taxa_transgenica):
    """
    Calcula N1, N2, N3 e N4.

    No artigo:
    N1 = N2 = N3 = N4.

    Cada N representa a quantidade de indivíduos
    modificados por cada módulo transgênico.
    """
                                # max evita que taxas pequenas, o resultado vire zero.
    n_por_modulo = max(         # Primeiro calcula quantos individuos, no total seriam afetados pela taxa.
        1,
        int((population_size * taxa_transgenica) / 4))  # Exemplo: população = 50, taxa = 0,30 então 50 x 0,30 = 15 indivídu
                                                        # Depois divide por 4, então chega a 3,75 e arredonda para 3 por ser
    return {                  # Retorna em forma de dicionário. Cada módulo recebe a mesma quantidade de indivíduos.
        "M1": n_por_modulo,
        "M2": n_por_modulo,
        "M3": n_por_modulo,
        "M4": n_por_modulo
    }


def aplicar_transgenico_original_modular(
    populacao,
    historico,
    taxa_transgenica,
    scores_populacao=None
):
    """
    Aplica o operador transgênico adaptado com quatro módulos.

    M1 altera 1 gene.
    M2 altera 2 genes.
    M3 altera 3 genes.
    M4 altera 4 genes.

    Os módulos trabalham em paralelo no sentido de que todos partem
    da mesma população-base da geração. Porém, para reduzir o custo
```

```
computacional com LSTM, cada módulo gera apenas alguns indivíduos
transgênicos, e não uma população completa.

Fluxo desta versão:
- A população-base é copiada.
- Cada módulo escolhe genes frequentes no histórico.
- Cada módulo cria N indivíduos transgênicos.
- Os indivíduos transgênicos substituem os piores indivíduos
  da população provisória, segundo o MSE.
- O tamanho final da população permanece constante.
"""

populacao_base = [                # Cria uma cópia da população.
    ind[:]
    for ind in populacao
]

N_modulos = calcular_N_modulos(   # Calcula N1, N2, N3 e N4.
    len(populacao_base),
    taxa_transgenica
)

configuracao_modulos = {          # Define quantos genes cada módulo altera.
    "M1": 1,
    "M2": 2,
    "M3": 3,
    "M4": 4
}

individuos_transgenicos = []      # Guarda os indivíduos modificados por todos os módulos.
detalhes_transgenicos = []        # Guarda gene a gene o que foi alterado em cada indivíduo transgênico.
detalhes_modulos = {}
id_transgenico = 0

for nome_modulo, quantidade_genes in configuracao_modulos.items():  # Percorre M1, M2, M3 e M4.

    N_modulo = N_modulos[nome_modulo]              # Quantos indivíduos esse módulo irá gerar/modificar.

    genes_frequentes = historico.selecionar_genes_original(  # Seleciona os genes por frequência/probabilidade.
        quantidade_genes
    )

    for _ in range(N_modulo):                      # Gera N_modulo indivíduos transgênicos.

        base = random.choice(populacao_base)       # Escolhe um indivíduo da população-base.
        base_original = base[:]                    # Guarda o cromossomo antes da transgenia.
        transg = base[:]                           # Faz uma cópia desse indivíduo.
        genes_substituidos = []
        valores_antigos = {}
        valores_novos = {}

        for i, valor in genes_frequentes.items():  # Substitui os genes selecionados pelos valores frequentes.

            if 0 <= i < len(transg):
                valores_antigos[GENE_NAMES[i]] = transg[i]
                valores_novos[GENE_NAMES[i]] = valor
                if transg[i] != valor:
                    genes_substituidos.append(GENE_NAMES[i])
                transg[i] = valor

        individuos_transgenicos.append(transg)     # Guarda o indivíduo transgênico.
        detalhes_transgenicos.append({
            "ID_Transgenico": id_transgenico,
            "Modulo": nome_modulo,
            "Quantidade_Genes_Modulo": quantidade_genes,
            "Genes_Selecionados": str([GENE_NAMES[i] for i in genes_frequentes.keys()]),
            "Genes_Efetivamente_Alterados": str(genes_substituidos),
            "Valores_Antigos": str(valores_antigos),
            "Valores_Novos": str(valores_novos),
            "Cromossomo_Base": str(base_original),
            "Cromossomo_Transgenico": str(transg),
            "Indice_Substituido": None,
            "Score_Individuo_Substituido": None
        })
        id_transgenico += 1

    detalhes_modulos[f"{nome_modulo}_Genes_Alterados"] = quantidade_genes
    detalhes_modulos[f"{nome_modulo}_Individuos_Gerados"] = N_modulo

nova_populacao = populacao_base[:]                 # A população continua com o mesmo tamanho.

indices_substituidos = []
```

```
    if len(individuos_transgenicos) > 0:

        qtd_substituir = min(                          # Evita inserir mais indivíduos do que o tamanho da população.
            len(individuos_transgenicos),
            len(nova_populacao)
        )

        if scores_populacao is not None and len(scores_populacao) == len(nova_populacao):
            # Ordena os indivíduos pelo MSE.
            # Como menor MSE é melhor, os piores são aqueles com maior MSE.
            indices_piores = np.argsort(scores_populacao)[::-1][:qtd_substituir]

            for ordem, (idx, transg) in enumerate(zip(indices_piores, individuos_transgenicos[:qtd_substituir])):
                nova_populacao[int(idx)] = transg
                indices_substituidos.append(int(idx))
                if ordem < len(detalhes_transgenicos):
                    detalhes_transgenicos[ordem]["Indice_Substituido"] = int(idx)
                    detalhes_transgenicos[ordem]["Score_Individuo_Substituido"] = float(scores_populacao[int(idx)])

        else:
            # Segurança: se os scores não forem informados, mantém o comportamento antigo.
            nova_populacao[-qtd_substituir:] = individuos_transgenicos[:qtd_substituir]
            indices_substituidos = list(
                range(len(nova_populacao) - qtd_substituir, len(nova_populacao))
            )
            for ordem, idx in enumerate(indices_substituidos):
                if ordem < len(detalhes_transgenicos):
                    detalhes_transgenicos[ordem]["Indice_Substituido"] = int(idx)

    total_transgenicos = len(individuos_transgenicos)
    total_aplicados = min(
        total_transgenicos,
        len(nova_populacao)
    )

    info_modulos = {                                                # Guarda informações da aplicação.
        "N1": N_modulos["M1"],
        "N2": N_modulos["M2"],
        "N3": N_modulos["M3"],
        "N4": N_modulos["M4"],

        "Total_Transgenicos_Gerados": total_transgenicos,
        "Total_Transgenicos_Aplicados": total_aplicados,

        "Taxa_Transgenica_Teorica": taxa_transgenica,
        "Taxa_Transgenica_Real_Aproximada": total_aplicados / len(nova_populacao),

        "Estrategia_Modulos": "Paralelo_adaptado_com_substituicao_dos_piores",
        "Estrategia_Substituicao": "Substitui_piores_individuos_por_MSE",
        "Indices_Substituidos": str(indices_substituidos)
    }

    info_modulos.update(detalhes_modulos)

    return nova_populacao, info_modulos, detalhes_transgenicos
# Retorna a população modificada, as informações dos módulos e detalhes gene a gene.

# ============================================================
# 11. ALGORITMO GENÉTICO TRANSGÊNICO ORIGINAL
# Agora recebe a taxa transgênica como parâmetro

# ============================================================
def genetic_algorithm_transgenico_original(X, Y):   # Executa o AGT usando uma taxa transgênica por geração, em ciclo.

    population = initialize_population(                     # Cria população inicial com 30 individuos
        POPULATION_SIZE
    )

    historico = HistoricoTransgenicoOriginal(               # Cria o historico transgênico
        n_genes=len(population[0])
    )

    generation_times = []                                 # Guardar os resultados
    best_mse_por_geracao = []
    historico_modulos = []
    historico_geracoes = []
    historico_populacao = []
    historico_transgenicos = []

    # Guarda o melhor indivíduo encontrado em TODO o processo evolutivo.
    # Isso evita perder uma configuração muito boa encontrada em uma geração intermediária.
    global_best_individual = None
```

```
    global_best_mse = float("inf")
    global_best_generation = None
    global_best_origin = None

    top_n = max(                                        # Define que metade da população será preservada por elitismo.
        1,
        POPULATION_SIZE // 2
    )

    for gen in range(GENERATIONS):                      # Repete por 5 gerações

        inicio_geracao = time.time()

        # Escolhe a taxa transgênica da geração de forma cíclica.
        # Exemplo com 5 gerações:
        # Geração 1 -> 10%, Geração 2 -> 30%, Geração 3 -> 50%,
        # Geração 4 -> 70%, Geração 5 -> 90%.
        taxa_transgenica = TRANSGENIC_RATES[gen % len(TRANSGENIC_RATES)]

        # Avaliação da população
        scores = [
            fitness_function(ind, X, Y)
            for ind in population
        ]

        best_mse = float(min(scores))               # Pega o melhor MSE da geração
        best_mse_por_geracao.append(best_mse)

        # Verifica se o melhor indivíduo desta geração é também
        # o melhor indivíduo encontrado em todas as gerações até agora.
        best_idx_gen = int(np.argmin(scores))

        if best_mse < global_best_mse:
            global_best_mse = best_mse
            global_best_individual = population[best_idx_gen][:]
            global_best_generation = gen + 1
            global_best_origin = "Geracao"

        print(
            f"Taxa {taxa_transgenica:.0%} | "
            f"Geração {gen + 1} "
            f"- Melhor MSE: {best_mse:.6f} | "
            f"Melhor MSE global: {global_best_mse:.6f}"
        )

        # Elitismo: seleciona os 50% melhores indivíduos
        best_idx = np.argsort(scores)[:top_n]

        # Instrumentação: registra população avaliada, fitness, elites e diversidade.
        diversidade = calcular_diversidade_populacao(population)
        diversidade.update({
            "Algoritmo": "AGT",
            "Geracao": gen + 1,
            "Ordem_Grafico": gen + 1,
            "Rotulo_Geracao": f"G{gen + 1}",
            "Etapa": "avaliacao_inicio_geracao",
            "Taxa_Transgenica": taxa_transgenica,
            "Melhor_MSE_Geracao": best_mse,
            "Media_MSE_Geracao": float(np.mean(scores)),
            "Desvio_MSE_Geracao": float(np.std(scores)),
            "Melhor_MSE_Global_Ate_Geracao": global_best_mse
        })
        historico_geracoes.append(diversidade)
        registrar_populacao_busca(
            historico_populacao,
            geracao=gen + 1,
            etapa="avaliacao_inicio_geracao",
            populacao=population,
            scores=scores,
            elite_indices=best_idx,
            algoritmo="AGT",
            taxa_transgenica=taxa_transgenica
        )

        best_parents = [                        # Copia os melhores individuos
            population[i][:]
            for i in best_idx
        ]

        # Atualiza o histórico usando os melhores indivíduos
        historico.atualizar(best_parents)

        # Geração dos filhos por seleção, crossover e mutação
```

```
        offspring = []

        while len(offspring) < (                # Gera filhos até completar a população
            POPULATION_SIZE - top_n
        ):

            p1 = tournament_selection(          # Seleciona o Pai 1
                population,
                scores
            )

            p2 = tournament_selection(         # Seleciona o Pai 2
                population,
                scores
            )

            o1, o2 = crossover(p1, p2)        # Cruza os pais

            offspring.append(                  # Aplica a mutação no filho 1
                mutate(o1)
            )

            if len(offspring) < (             # Controla o tamanho da populçao.
                POPULATION_SIZE - top_n
            ):

                offspring.append(             # Aplica mutação no filho 2 se necessario, caso a população ainda não seja 30.
                    mutate(o2)
                )

        # Nova população provisória antes do transgênico:
        # 50% elite + 50% filhos gerados por crossover e mutação.
        population = best_parents + offspring

        # Avalia a população provisória para identificar, de fato,
        # quais são os piores indivíduos antes da entrada dos transgênicos.
        # Menor MSE = melhor indivíduo; maior MSE = pior indivíduo.
        scores_populacao_provisoria = [
            fitness_function(ind, X, Y)
            for ind in population
        ]

        # Atualiza também o melhor indivíduo global caso algum filho
        # da população provisória seja melhor do que todos os anteriores.
        best_idx_provisorio = int(np.argmin(scores_populacao_provisoria))
        best_mse_provisorio = float(scores_populacao_provisoria[best_idx_provisorio])

        if best_mse_provisorio < global_best_mse:
            global_best_mse = best_mse_provisorio
            global_best_individual = population[best_idx_provisorio][:]
            global_best_generation = gen + 1
            global_best_origin = "Populacao_Provisoria_Antes_Transgenico"

        # Instrumentação: registra população provisória antes da transgenia.
        registrar_populacao_busca(
            historico_populacao,
            geracao=gen + 1,
            etapa="populacao_provisoria_antes_transgenico",
            populacao=population,
            scores=scores_populacao_provisoria,
            elite_indices=None,
            algoritmo="AGT",
            taxa_transgenica=taxa_transgenica
        )

        # Operador transgênico:
        # M1, M2, M3 e M4 partem da mesma população-base.
        # Para reduzir o custo computacional, eles geram apenas indivíduos
        # transgênicos. Nesta versão, esses transgênicos substituem os
        # piores indivíduos da população provisória, segundo o MSE.
        population, info_modulos, detalhes_transgenicos = aplicar_transgenico_original_modular(
            population,
            historico,
            taxa_transgenica,
            scores_populacao=scores_populacao_provisoria
        )

        for detalhe in detalhes_transgenicos:
            detalhe["Geracao"] = gen + 1
            detalhe["Taxa_Transgenica"] = taxa_transgenica
        historico_transgenicos.extend(detalhes_transgenicos)
```

```
        registrar_populacao_busca(
            historico_populacao,
            geracao=gen + 1,
            etapa="apos_transgenico",
            populacao=population,
            scores=None,
            elite_indices=None,
            algoritmo="AGT",
            taxa_transgenica=taxa_transgenica
        )

        info_modulos["Geracao"] = gen + 1                  # Registra a geração
        info_modulos["Melhor_MSE_Geracao"] = best_mse      # Melhor MSE da população avaliada no início da geração
        info_modulos["Melhor_MSE_Populacao_Provisoria"] = best_mse_provisorio
        info_modulos["Melhor_MSE_Global_Apos_Transgenico"] = global_best_mse

        historico_modulos.append(info_modulos)             # Guarda o historico dos módulos.

        fim_geracao = time.time()                          # Registra o tempo de finalização da geração

        generation_times.append(                           # Guarda o tempo da geração
            fim_geracao - inicio_geracao
        )

    # Avaliação final da população
    final_scores = [
        fitness_function(ind, X, Y)
        for ind in population
    ]

    best_index = int(                          # Encontra o melhor individuo final
        np.argmin(final_scores)
    )

    best_final_mse = float(                    # Guarda o melhor MSE da população final.
        final_scores[best_index]
    )

    # Compara também a população final com o melhor global já encontrado.
    # Assim, se a última população tiver uma configuração ainda melhor,
    # ela também será considerada.
    if best_final_mse < global_best_mse:
        global_best_mse = best_final_mse
        global_best_individual = population[best_index][:]
        global_best_generation = "Final"
        global_best_origin = "Avaliacao_Final"

    diversidade_final = calcular_diversidade_populacao(population)
    diversidade_final.update({
        "Algoritmo": "AGT",
        "Geracao": "Final",
        "Ordem_Grafico": GENERATIONS + 1,
        "Rotulo_Geracao": "Final",
        "Etapa": "avaliacao_final_populacao",
        "Taxa_Transgenica": None,
        "Melhor_MSE_Geracao": best_final_mse,
        "Media_MSE_Geracao": float(np.mean(final_scores)),
        "Desvio_MSE_Geracao": float(np.std(final_scores)),
        "Melhor_MSE_Global_Ate_Geracao": global_best_mse
    })
    historico_geracoes.append(diversidade_final)
    registrar_populacao_busca(
        historico_populacao,
        geracao="Final",
        etapa="avaliacao_final_populacao",
        populacao=population,
        scores=final_scores,
        elite_indices=[best_index],
        algoritmo="AGT"
    )

    return (
        global_best_individual,
        best_mse_por_geracao,
        generation_times,
        sum(generation_times),
        historico_modulos,
        global_best_mse,
        global_best_generation,
        global_best_origin,
        historico_geracoes,
        historico_populacao,
        historico_transgenicos
```

```
    )

# ============================================================
# 12. EXECUÇÃO DO AGT ORIGINAL COM TAXA CÍCLICA POR GERAÇÃO

# ============================================================
print("\n================================================")
print("EXECUTANDO AGT ORIGINAL COM TAXA TRANSGÊNICA CÍCLICA POR GERAÇÃO")
print("Sequência das taxas:", TRANSGENIC_RATES)
print("================================================")

inicio_agt_original = time.time()

(
    best_config,
    mse_geracoes,
    tempos_geracoes,
    tempo_agt_original,
    historico_modulos,
    melhor_mse_global,
    geracao_melhor_individuo,
    origem_melhor_individuo,
    historico_geracoes_agt,
    historico_populacao_agt,
    historico_transgenicos_agt
) = genetic_algorithm_transgenico_original(
    X_train,
    Y_train_scaled
)

tempo_agt_original = time.time() - inicio_agt_original

# Histórico de módulos e taxas aplicadas em cada geração
df_historico_modulos = pd.DataFrame(
    historico_modulos
)

df_historico_modulos.to_csv(
    f"historico_modulos_agt_original_ciclico_parcial_{NOME_BASE}.csv",
    index=False
)

# Exporta logs completos da busca transgênica para análise posterior da dissertação.
df_historico_geracoes_agt = pd.DataFrame(historico_geracoes_agt)
df_historico_populacao_agt = pd.DataFrame(historico_populacao_agt)
df_historico_transgenicos_agt = pd.DataFrame(historico_transgenicos_agt)
df_historico_geracoes_agt.to_csv(f"historico_geracoes_agt_{NOME_BASE}.csv", index=False)
df_historico_populacao_agt.to_csv(f"historico_populacao_agt_{NOME_BASE}.csv", index=False)
df_historico_transgenicos_agt.to_csv(f"historico_transgenicos_agt_{NOME_BASE}.csv", index=False)
print("Logs da busca AGT salvos:")
print(f"- historico_geracoes_agt_{NOME_BASE}.csv")
print(f"- historico_populacao_agt_{NOME_BASE}.csv")
print(f"- historico_transgenicos_agt_{NOME_BASE}.csv")

# Define a melhor configuração global para o treinamento final
num_layers, num_neurons, lr, batch_size, epochs = best_config # Separa os genes

num_layers = int(num_layers)
num_neurons = int(num_neurons)
lr = float(lr)
batch_size = int(batch_size)
epochs = int(epochs)

print("\nMELHOR CONFIGURAÇÃO ENCONTRADA - AGT ORIGINAL CÍCLICO")
print(f"Melhor MSE global encontrado: {melhor_mse_global:.6f}")
print(f"Melhor indivíduo encontrado na geração: {geracao_melhor_individuo}")
print(f"Origem do melhor indivíduo: {origem_melhor_individuo}")
print(f"Camadas: {num_layers}")
print(f"Neurônios: {num_neurons}")
print(f"Learning Rate: {lr}")
print(f"Batch Size: {batch_size}")
print(f"Epochs: {epochs}")
print("Taxas aplicadas ciclicamente por geração:", TRANSGENIC_RATES)
print("Estratégia dos módulos: módulos paralelos com geração de indivíduos transgênicos e substituição parcial")

# ============================================================
# 13. TREINAMENTO FINAL COM A MELHOR CONFIGURAÇÃO

# ============================================================
K.clear_session()

final_model = create_model(                    # Cria a LSTM final com os melhores hiperparâmetros
    (X_train.shape[1], X_train.shape[2]),
```

```
    num_layers,
    num_neurons,
    lr
)

inicio_treino = time.time()

final_model.fit(
    X_train,
    Y_train_scaled,
    epochs=epochs,
    batch_size=batch_size,
    verbose=1
)

tempo_treino = time.time() - inicio_treino

# ============================================================
# 14. PREVISÕES

# ============================================================
inicio_teste = time.time()

y_pred_train_scaled = final_model.predict(       # Previsões no treino
    X_train,
    verbose=0
).reshape(-1)

y_pred_test_scaled = final_model.predict(       # Previsões do teste
    X_test,
    verbose=0
).reshape(-1)

tempo_teste = time.time() - inicio_teste

# ============================================================
# 15. MÉTRICAS

# ============================================================
mape_train, mse_train, rmse_train, r2_train, y_train_inv, y_pred_train_inv = calcular_metricas(
    Y_train_scaled,
    y_pred_train_scaled,
    scaler
)

mape_test, mse_test, rmse_test, r2_test, y_test_inv, y_pred_test_inv = calcular_metricas(
    Y_test_scaled,
    y_pred_test_scaled,
    scaler
)

tempo_total = time.time() - start_global

periodo_utilizado = (
    PERIOD
    if PERIOD is not None
    else f'{START_DATE} a {END_DATE}'
)

# ============================================================
# 16. RESULTADOS

# ============================================================
resultados = pd.DataFrame({
    'Modelo': ['LSTM_AGT_Original'],
    'Ticker': [TICKER],
    'Intervalo': [INTERVAL],
    'Periodo': [periodo_utilizado],
    'Time_Step': [TIME_STEP],

    'MAPE_Treino': [mape_train],
    'MSE_Treino': [mse_train],
    'RMSE_Treino': [rmse_train],
    'R2_Treino': [r2_train],

    'MAPE_Teste': [mape_test],
    'MSE_Teste': [mse_test],
    'RMSE_Teste': [rmse_test],
    'R2_Teste': [r2_test],

    'Tempo_AGT_Original': [tempo_agt_original],
    'Tempo_Treino': [tempo_treino],
    'Tempo_Teste': [tempo_teste],
```

```
    'Tempo_Teste': [tempo_teste],
    'Tempo_Total': [tempo_total],

    'Batch_Size': [batch_size],
    'Epochs': [epochs],
    'Camadas': [num_layers],
    'Neuronios': [num_neurons],
    'Learning_Rate': [lr],

    'Generations': [GENERATIONS],
    'Population_Size': [POPULATION_SIZE],
    'Mutation_Rate': [MUTATION_RATE],

    'Taxas_Testadas': [str(TRANSGENIC_RATES)],
    'Estrategia_Taxa_Transgenica': ['Ciclica_por_geracao_com_substituicao_parcial'],
    'Estrategia_Modulos_Transgenicos': ['Modulos_paralelos_com_substituicao_parcial'],
    'Melhor_MSE_AGT_Original': [melhor_mse_global],
    'Geracao_Melhor_Individuo': [geracao_melhor_individuo],
    'Origem_Melhor_Individuo': [origem_melhor_individuo]
})

pd.set_option("display.max_columns", None)
pd.set_option("display.width", None)
pd.set_option("display.max_colwidth", None)

print("\nRESULTADOS - LSTM + AGT ORIGINAL")
display(resultados)
display(resultados.T)

resultados.to_csv(
    f"resultado_lstm_agt_original_ciclico_parcial_{NOME_BASE}.csv",
    index=False
)

# ============================================================
# 17. GRÁFICO DE CONVERGÊNCIA DA MELHOR TAXA

# ============================================================
plt.figure(figsize=(10, 5))

# Plota o melhor MSE no início de cada geração e inclui a avaliação final da população.
# A avaliação final corresponde aos indivíduos gerados após a última reprodução/transgenia,
# evitando confusão entre "melhor da geração 5" e "melhor da população final".
df_convergencia_agt = df_historico_geracoes_agt[
    df_historico_geracoes_agt["Etapa"].isin(["avaliacao_inicio_geracao", "avaliacao_final_populacao"])
].copy()
df_convergencia_agt = df_convergencia_agt.sort_values("Ordem_Grafico")

plt.plot(
    df_convergencia_agt["Rotulo_Geracao"],
    df_convergencia_agt["Melhor_MSE_Geracao"],
    marker='o'
)

plt.title(
    f"Convergência - LSTM + AGT Original Cíclico Parcial Adaptado - {NOME_BASE}"
)

plt.xlabel("Etapa da busca")
plt.ylabel("Melhor MSE")
plt.grid()
plt.tight_layout()

plt.savefig(
    f"convergencia_lstm_agt_original_ciclico_parcial_{NOME_BASE}.png"
)

plt.show()
plt.close()

# ============================================================
# 18. GRÁFICO DAS TAXAS TRANSGÊNICAS APLICADAS POR GERAÇÃO

# ============================================================
plt.figure(figsize=(10, 5))

plt.plot(
    df_historico_modulos['Geracao'],
    df_historico_modulos['Taxa_Transgenica_Teorica'],
    marker='o'
)

plt.title(
```

```
    f"Taxa Transgênica Aplicada por Geração - AGT Original Cíclico Parcial Adaptado - {NOME_BASE}"
)

plt.xlabel("Geração")
plt.ylabel("Taxa Transgênica")
plt.grid()
plt.tight_layout()

plt.savefig(
    f"taxas_por_geracao_agt_original_ciclico_parcial_{NOME_BASE}.png"
)

plt.show()
plt.close()

# ============================================================
# 19. GRÁFICO TREINAMENTO

# ============================================================
plt.figure(figsize=(12, 5))

plt.plot(
    y_train_inv,
    label='Real'
)

plt.plot(
    y_pred_train_inv,
    label='Previsto'
)

plt.title(
    f"LSTM + AGT Original Parcial Adaptado - Treinamento - {NOME_BASE}"
)

plt.xlabel("Tempo")
plt.ylabel("Preço do Café")
plt.legend()
plt.grid()
plt.tight_layout()

plt.savefig(
    f"lstm_agt_original_ciclico_parcial_treinamento_{NOME_BASE}.png"
)

plt.show()
plt.close()

# ============================================================
# 20. GRÁFICO TESTE

# ============================================================
plt.figure(figsize=(12, 5))

plt.plot(
    y_test_inv,
    label='Real'
)

plt.plot(
    y_pred_test_inv,
    label='Previsto'
)

plt.title(
    f"LSTM + AGT Original Parcial Adaptado - Teste - {NOME_BASE}"
)

plt.xlabel("Tempo")
plt.ylabel("Preço do Café")
plt.legend()
plt.grid()
plt.tight_layout()

plt.savefig(
    f"lstm_agt_original_ciclico_parcial_teste_{NOME_BASE}.png"
)

plt.show()
plt.close()

# ============================================================
# 21. TEMPOS POR GERAÇÃO DA MELHOR TAXA
```

```
# ============================================================
print("\nTEMPOS POR GERAÇÃO - MELHOR TAXA")
for i, t in enumerate(tempos_geracoes):
    print(f"Geração {i + 1}: {t:.2f} segundos")

print(f"\nEstratégia de taxa transgênica: cíclica por geração {TRANSGENIC_RATES}")
print(f"Tempo total AGT Original Cíclico Parcial Adaptado: {tempo_agt_original:.2f} segundos")
print(f"Tempo total do script: {tempo_total:.2f} segundos")

files.download(f"historico_geracoes_agt_{NOME_BASE}.csv")
files.download(f"historico_populacao_agt_{NOME_BASE}.csv")
files.download(f"historico_transgenicos_agt_{NOME_BASE}.csv")
files.download(f"historico_modulos_agt_original_ciclico_parcial_{NOME_BASE}.csv")
```

Mostrar saída oculta

APÊNDICE **D**

Treinamento Final

```
# ============================================================
# TREINAMENTO FINAL - LSTM PARA PREVISÃO DO CAFÉ ARÁBICA
# Fluxo simplificado:
# Melhores parâmetros encontrados -> Treinamento final único -> Salvar modelo/scaler -> Backtesting
# ============================================================

# === IMPORTS ===
import os
import certifi
os.environ['SSL_CERT_FILE'] = certifi.where()
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import time
import json
import joblib
import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error, r2_score

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import backend as K

from google.colab import files

# ============================================================
# 1. CONFIGURAÇÕES DO EXPERIMENTO
# ============================================================

TICKER = "KC=F"

# Escolha o intervalo:
# "4h"  = dados intraday, máximo permitido pelo Yahoo Finance
# "1d"  = dados diários
# "1wk" = dados semanais
INTERVALO = "1wk"  # ALTERE AQUI: "4h", "1d" ou "1wk"

# Nome do modelo de origem:
# "LSTM_Classica"
# "LSTM_AG"
# "LSTM_AGT"
NOME_EXPERIMENTO = "LSTM_AGT"

# ============================================================
# 2. PERÍODO DO TREINAMENTO FINAL
# ============================================================
# Para diário e semanal:
# Treinamento final: 2000-01-01 até 2023-12-31
# Backtesting separado: 2024-01-02 até 2025-12-31
#
# Para 4h:
# Treinamento final: últimos 730 dias disponíveis
# Backtesting separado: período definido no código de backtesting
# ============================================================

if INTERVALO in ["1d", "1wk"]:
    START_DATE = "2000-01-01"
    END_DATE = "2023-12-31"
    PERIOD = None
    NOME_BASE = "diario" if INTERVALO == "1d" else "semanal"

elif INTERVALO == "4h":
    START_DATE = None
    END_DATE = None
    PERIOD = "730d"
    NOME_BASE = "4h"

else:
    raise ValueError("INTERVALO inválido. Use '4h', '1d' ou '1wk'.")

# ============================================================
# 3. PARÂMETROS ENCONTRADOS NOS CÓDIGOS DE BUSCA
# ============================================================
# Altere esses valores conforme o melhor resultado encontrado
# em cada modelo: LSTM clássica, LSTM_AG ou LSTM_AGT_Original.
# ============================================================
```

```
# ============================================================

num_layers = 1
num_neurons = 43
lr = 0.0088
batch_size = 41
epochs = 68

TIME_STEP = 30
DROPOUT_RATE = 0.2

# ============================================================
# 4. NOMES DOS ARQUIVOS DE SAÍDA
# ============================================================

NOME_MODELO = f"modelo_{NOME_EXPERIMENTO}_{NOME_BASE}.keras"
NOME_MODELO_H5 = f"modelo_{NOME_EXPERIMENTO}_{NOME_BASE}.h5"
NOME_SCALER = f"scaler_{NOME_EXPERIMENTO}_{NOME_BASE}.pkl"
NOME_METADADOS = f"metadados_{NOME_EXPERIMENTO}_{NOME_BASE}.json"
NOME_HISTORICO = f"historico_treinamento_{NOME_EXPERIMENTO}_{NOME_BASE}.csv"
NOME_RESULTADOS = f"metricas_treinamento_final_{NOME_EXPERIMENTO}_{NOME_BASE}.csv"
NOME_GRAFICO = f"convergencia_treinamento_final_{NOME_EXPERIMENTO}_{NOME_BASE}.png"
NOME_GRAFICO_TREINO = f"treinamento_final_real_previsto_{NOME_EXPERIMENTO}_{NOME_BASE}.png"

start_global = time.time()

# ============================================================
# 5. BAIXAR OS DADOS DO PERÍODO DE TREINAMENTO FINAL
# ============================================================

print("================================================")
print("TREINAMENTO FINAL ÚNICO DA LSTM")
print("================================================")
print(f"Ticker: {TICKER}")
print(f"Intervalo: {INTERVALO}")
print(f"Experimento: {NOME_EXPERIMENTO}")

if PERIOD is not None:
    print(f"Período usado no treinamento final: {PERIOD}")
    dados = yf.download(
        TICKER,
        period=PERIOD,
        interval=INTERVALO
    )
else:
    print(f"Período usado no treinamento final: {START_DATE} até {END_DATE}")
    dados = yf.download(
        TICKER,
        start=START_DATE,
        end=END_DATE,
        interval=INTERVALO
    )

fechamento = dados[["Close"]].dropna()
fechamento.index = pd.to_datetime(fechamento.index)

print(f"Total de registros coletados: {len(fechamento)}")

if len(fechamento) <= TIME_STEP:
    raise ValueError("Quantidade de dados insuficiente para criar as janelas temporais.")

# ============================================================
# 6. FUNÇÕES AUXILIARES
# ============================================================

def criar_dados(seq, passos=30):
    """Cria janelas temporais para previsão one-step ahead."""
    X, y = [], []

    for i in range(passos, len(seq)):
        X.append(seq[i - passos:i, 0])
        y.append(seq[i, 0])

    return np.array(X), np.array(y)


def construir_modelo_lstm(input_shape, num_layers, num_neurons, learning_rate, dropout_rate):
    """Constrói uma LSTM com a melhor configuração encontrada na busca."""
    K.clear_session()

    modelo = Sequential()

    if int(num_layers) == 1:
```

```
        modelo.add(
            LSTM(
                int(num_neurons),
                input_shape=input_shape
            )
        )

    else:
        modelo.add(
            LSTM(
                int(num_neurons),
                return_sequences=True,
                input_shape=input_shape
            )
        )

        for _ in range(max(0, int(num_layers) - 2)):
            modelo.add(
                LSTM(
                    int(num_neurons),
                    return_sequences=True
                )
            )

        modelo.add(
            LSTM(
                int(num_neurons)
            )
        )

    modelo.add(Dropout(float(dropout_rate)))
    modelo.add(Dense(1))

    opt = Adam(learning_rate=float(learning_rate))

    modelo.compile(
        optimizer=opt,
        loss="mse"
    )

    return modelo


def calcular_metricas(y_true_scaled, y_pred_scaled, scaler_usado):
    """Calcula MAPE, MSE, RMSE e R² após desfazer a normalização."""
    y_true_inv = scaler_usado.inverse_transform(
        y_true_scaled.reshape(-1, 1)
    ).reshape(-1)

    y_pred_inv = scaler_usado.inverse_transform(
        y_pred_scaled.reshape(-1, 1)
    ).reshape(-1)

    mse = mean_squared_error(y_true_inv, y_pred_inv)
    rmse = np.sqrt(mse)
    mape = mean_absolute_percentage_error(y_true_inv, y_pred_inv) * 100
    r2 = r2_score(y_true_inv, y_pred_inv)

    return mape, mse, rmse, r2, y_true_inv, y_pred_inv

# ============================================================
# 7. NORMALIZAÇÃO E CRIAÇÃO DAS JANELAS DO TREINAMENTO FINAL
# ============================================================
# Nesta versão não há divisão 80/20 dentro do treinamento final.
# O modelo é treinado uma única vez com TODO o período definido acima.
# O backtesting deve ser feito em outro código, usando dados fora da amostra.
# ============================================================

scaler_final = MinMaxScaler(feature_range=(0, 1))
fechamento_scaled_final = scaler_final.fit_transform(fechamento)

X_final, y_final = criar_dados(fechamento_scaled_final, TIME_STEP)
X_final = X_final.reshape(X_final.shape[0], X_final.shape[1], 1)

print("\n================================================")
print("DADOS DO TREINAMENTO FINAL")
print("================================================")
print(f"Registros usados no treinamento final: {len(fechamento)}")
print(f"Amostras supervisionadas do treinamento final: {len(X_final)}")
print(f"Time step: {TIME_STEP}")

# ============================================================
# 8. TREINAMENTO FINAL ÚNICO
```

```
# ============================================================

modelo_final = construir_modelo_lstm(
    input_shape=(X_final.shape[1], X_final.shape[2]),
    num_layers=int(num_layers),
    num_neurons=int(num_neurons),
    learning_rate=float(lr),
    dropout_rate=float(DROPOUT_RATE)
)

inicio_treino_final = time.time()

hist_final = modelo_final.fit(
    X_final,
    y_final,
    epochs=int(epochs),
    batch_size=int(batch_size),
    verbose=1
)

tempo_treino_final = time.time() - inicio_treino_final

# ============================================================
# 9. MÉTRICAS INTERNAS DO TREINAMENTO FINAL
# ============================================================
# Essas métricas são calculadas apenas sobre o próprio período de treinamento final.
# A avaliação fora da amostra será feita no backtesting.
# ============================================================

inicio_predicao = time.time()
pred_final_scaled = modelo_final.predict(X_final, verbose=0).reshape(-1)
tempo_predicao = time.time() - inicio_predicao

mape_final, mse_final, rmse_final, r2_final, y_final_inv, y_pred_final_inv = calcular_metricas(
    y_final,
    pred_final_scaled,
    scaler_final
)

print("\n==============================")
print("MÉTRICAS INTERNAS DO TREINAMENTO FINAL")
print("==============================")
print(f"MAPE : {mape_final:.4f}%")
print(f"MSE  : {mse_final:.6f}")
print(f"RMSE : {rmse_final:.6f}")
print(f"R²   : {r2_final:.6f}")

# ============================================================
# 10. SALVAR RESULTADOS, MODELO, SCALER, HISTÓRICO E METADADOS
# ============================================================

tempo_total = time.time() - start_global
periodo_utilizado = PERIOD if PERIOD is not None else f"{START_DATE} a {END_DATE}"

resultados = pd.DataFrame({
    "Modelo": [NOME_EXPERIMENTO],
    "Ticker": [TICKER],
    "Intervalo": [INTERVALO],
    "Periodo_Treinamento_Final": [periodo_utilizado],
    "Time_Step": [TIME_STEP],

    "MAPE_Treinamento_Final": [mape_final],
    "MSE_Treinamento_Final": [mse_final],
    "RMSE_Treinamento_Final": [rmse_final],
    "R2_Treinamento_Final": [r2_final],

    "Tempo_Treino_Final": [tempo_treino_final],
    "Tempo_Predicao_Treinamento_Final": [tempo_predicao],
    "Tempo_Total": [tempo_total],

    "Camadas": [int(num_layers)],
    "Neuronios": [int(num_neurons)],
    "Learning_Rate": [float(lr)],
    "Batch_Size": [int(batch_size)],
    "Epochs": [int(epochs)],
    "Dropout": [float(DROPOUT_RATE)]
})

pd.set_option("display.max_columns", None)
pd.set_option("display.width", None)
print("\nRESULTADOS DO TREINAMENTO FINAL")
display(resultados)
resultados.to_csv(NOME_RESULTADOS, index=False)
```

```
resultados.to_csv(NOME_RESULTADOS, index=False)

modelo_final.save(NOME_MODELO)
modelo_final.save(NOME_MODELO_H5)
joblib.dump(scaler_final, NOME_SCALER)

historico_df = pd.DataFrame({
    "loss_treinamento_final": pd.Series(hist_final.history["loss"])
})

historico_df.to_csv(NOME_HISTORICO, index=False)

metadados = {
    "ticker": TICKER,
    "intervalo": INTERVALO,
    "nome_experimento": NOME_EXPERIMENTO,
    "periodo_treinamento_final": periodo_utilizado,
    "period": PERIOD,
    "start_date": START_DATE,
    "end_date": END_DATE,
    "time_step": TIME_STEP,
    "num_layers": int(num_layers),
    "num_neurons": int(num_neurons),
    "learning_rate": float(lr),
    "batch_size": int(batch_size),
    "epochs_usadas": int(epochs),
    "dropout_rate": float(DROPOUT_RATE),
    "total_registros_fechamento": int(len(fechamento)),
    "total_amostras_treinamento_final": int(len(X_final)),
    "tempo_treino_final_segundos": float(tempo_treino_final),
    "tempo_predicao_treinamento_final_segundos": float(tempo_predicao),
    "tempo_total_segundos": float(tempo_total),
    "metricas_treinamento_final": {
        "MAPE": float(mape_final),
        "MSE": float(mse_final),
        "RMSE": float(rmse_final),
        "R2": float(r2_final)
    }
}

with open(NOME_METADADOS, "w", encoding="utf-8") as f:
    json.dump(metadados, f, indent=4, ensure_ascii=False)

print("\nArquivos salvos:")
print(NOME_MODELO)
print(NOME_MODELO_H5)
print(NOME_SCALER)
print(NOME_HISTORICO)
print(NOME_METADADOS)
print(NOME_RESULTADOS)

# ============================================================
# 11. GRÁFICOS
# ============================================================

plt.figure(figsize=(10, 5))
plt.plot(hist_final.history["loss"], label="Loss - Treinamento final")
plt.title(f"Convergência do Treinamento Final - {NOME_EXPERIMENTO} - {NOME_BASE}")
plt.xlabel("Épocas")
plt.ylabel("Erro MSE")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig(NOME_GRAFICO)
plt.show()
plt.close()

plt.figure(figsize=(12, 5))
plt.plot(y_final_inv, label="Real")
plt.plot(y_pred_final_inv, label="Previsto")
plt.title(f"Treinamento Final - Real x Previsto - {NOME_EXPERIMENTO} - {NOME_BASE}")
plt.xlabel("Tempo")
plt.ylabel("Preço do Café")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig(NOME_GRAFICO_TREINO)
plt.show()
plt.close()

# ============================================================
# 12. DOWNLOAD DOS ARQUIVOS NO GOOGLE COLAB
# ============================================================
```

```
files.download(NOME_MODELO)
files.download(NOME_MODELO_H5)
files.download(NOME_SCALER)
files.download(NOME_HISTORICO)
files.download(NOME_METADADOS)
files.download(NOME_RESULTADOS)
files.download(NOME_GRAFICO)
files.download(NOME_GRAFICO_TREINO)

print("\n✅ Treinamento final concluído.")
print("✅ O modelo foi treinado uma única vez com todo o período definido no início do código.")
print("✅ Modelo e scaler salvos para uso no backtesting fora da amostra.")
print("✅ No backtesting, use este modelo e este scaler sem reajustar o scaler.")
```

Mostrar saída oculta

APÊNDICE **E**

Backtesting

```
# ========================================
# BACKTESTING + SIMULAÇÃO DE CAPITAL - KC=F
# Com Stop-Loss / Take-Profit persistente
# ========================================
import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from tensorflow.keras.losses import MeanSquaredError
import joblib

# ========================================
# 🔹 CONFIGURAÇÕES
# ========================================
# Ajuste aqui o modelo/scaler que deseja testar.
# Exemplos:
#   4h      -> intervalo = "4h"  | backtesting = últimos 90 dias disponíveis
#   diário  -> intervalo = "1d"  | backtesting = 2024-01-02 até 2025-12-31
#   semanal -> intervalo = "1wk" | backtesting = 2024-01-02 até 2025-12-31

caminho_modelo = "/content/modelo_LSTM_AGT_semanal.h5"
caminho_scaler = "/content/scaler_LSTM_AGT_semanal.pkl"

# Escolha a frequência do backtesting: "4h", "1d" ou "1wk"
intervalo = "1wk"

# Organização temporal do backtesting
if intervalo == "4h":
    # O Yahoo Finance limita dados intradiários; por isso o backtesting 4h usa os últimos 90 dias disponíveis.
    backtest_start = None
    backtest_end = None
    backtest_period = "90d"
elif intervalo in ["1d", "1wk", "1w"]:
    # Backtesting diário e semanal fora da amostra
    backtest_start = "2024-01-02"
    backtest_end = "2025-12-31"
    backtest_period = None
    if intervalo == "1w":
        intervalo = "1wk"
else:
    raise ValueError("intervalo deve ser '4h', '1d' ou '1wk'.")

capital_inicial = 100000.0
stop_loss = 0.005      # 0.5%
take_profit = 0.02     # 2%

# filtro mínimo de diferença relativa entre predicted e open para entrar (0.5%)
min_diff_rel = 0.005

# ========================================
# 🔹 BAIXAR DADOS (robusto a MultiIndex / tuplas nas colunas)
# ========================================
print("🔹 Baixando dados...")
if backtest_period is not None:
    dados_back = yf.download("KC=F", period=backtest_period, interval=intervalo)
    periodo_configurado = f"últimos {backtest_period} disponíveis"
else:
    dados_back = yf.download("KC=F", start=backtest_start, end=backtest_end, interval=intervalo)
    periodo_configurado = f"{backtest_start} → {backtest_end}"

# flatten possible MultiIndex columns (yfinance sometimes returns tuples)
def _flatten_col(c):
    if isinstance(c, tuple):
        for prefer in ("Open","High","Low","Close","Adj Close"):
            if prefer in c:
                return prefer
        try:
            return str(next(x for x in reversed(c) if isinstance(x, str)))
        except StopIteration:
            return "_".join(map(str, c))
    return str(c)

dados_back.columns = [_flatten_col(c) for c in dados_back.columns]

# keep only OHLC (require them)
expected = ['Open','High','Low','Close']
missing = [c for c in expected if c not in dados_back.columns]
if missing:
    raise KeyError(f"Faltam colunas OHLC nos dados baixados: {missing}")
```

```
dados_back = dados_back[expected].dropna()
if len(dados_back) == 0:
    raise ValueError("Nenhum dado foi baixado. Verifique o intervalo e o período configurados.")
periodo_real_inicio = dados_back.index.min()
periodo_real_fim = dados_back.index.max()
print(f"✅ Dados baixados: {len(dados_back)} registros ({intervalo})")
print(f"🗓️ Período configurado: {periodo_configurado}")
print(f"🗓️ Período real baixado: {periodo_real_inicio} → {periodo_real_fim}")

# ========================================
# 🔹 CARREGAR MODELO E SCALER
# ========================================
print("🔹 Carregando modelo e scaler...")
model = load_model(caminho_modelo, custom_objects={'mse': MeanSquaredError()})
scaler = joblib.load(caminho_scaler)
print("✅ Modelo e scaler carregados com sucesso!")

# ========================================
# 🔹 PREPARAR SÉRIE 'Close' E JANELAS
# ========================================
# scaler foi treinado apenas com Close (shape (N,1))
scaled_close = scaler.transform(dados_back[['Close']])

janela = 30
X_backtest = np.array([scaled_close[i-janela:i] for i in range(janela, len(scaled_close))])
# garantir shape (samples, timesteps, features)
if X_backtest.ndim == 2:
    X_backtest = X_backtest.reshape((X_backtest.shape[0], X_backtest.shape[1], 1))

print(f"✅ Formato de entrada para previsão: {X_backtest.shape} (samples, timesteps, features)")

# ========================================
# 🔹 PREVISÕES
# ========================================
print("🔹 Fazendo previsões...")
y_pred_scaled = model.predict(X_backtest, verbose=0)
# garante 1D e desfaz escala
y_pred = scaler.inverse_transform(np.asarray(y_pred_scaled).reshape(-1, 1)).ravel()

# alinhar com datas originais (a primeira previsão corresponde ao índice 'janela')
dados_back_aligned = dados_back.iloc[janela:].copy()
dados_back_aligned = dados_back_aligned.rename(columns={'Close': 'Close_Real'})
dados_back_aligned['Predicted'] = y_pred

# 🔥 SHIFT PARA REPRESENTAR T+1
dados_back_aligned['Predicted'] = dados_back_aligned['Predicted'].shift(1)

# Remover primeira linha que ficará NaN
dados_back_aligned = dados_back_aligned.dropna().copy()

# ========================================
# 🔹 GARANTIR TIPOS NUMÉRICOS (robusto)
# ========================================
def ensure_1d_numeric(series_like, index=None):
    arr = np.asarray(series_like)
    if arr.ndim > 1:
        # tentar obter primeira coluna
        try:
            arr = arr.reshape(arr.shape[0], -1)[:, 0]
        except Exception:
            arr = arr.ravel()
    arr = np.asarray(arr).ravel()
    s = pd.Series(arr, index=index)
    s = pd.to_numeric(s, errors='coerce')
    return s

for col in ['Open','High','Low','Close_Real','Predicted']:
    if col not in dados_back_aligned.columns:
        dados_back_aligned[col] = np.nan
    dados_back_aligned[col] = ensure_1d_numeric(dados_back_aligned[col], index=dados_back_aligned.index).astype(float)

# remover linhas que ficaram inválidas
dados_back_aligned = dados_back_aligned.dropna(subset=['Open','High','Low','Close_Real','Predicted']).copy()
print(f"✅ Registros válidos após limpeza: {len(dados_back_aligned)}")

# ========================================
# 🔹 NOVA LÓGICA PROFISSIONAL (RETORNO ESPERADO)
# ========================================
def tipo_operacao(open_price, predicted, min_diff_rel=0.005):
    try:
        open_price = float(open_price)
        predicted = float(predicted)
```

```
    except:
        return 'neutro'

    if open_price == 0:
        return 'neutro'

    diff_rel = (predicted - open_price) / open_price

    if diff_rel >= min_diff_rel:
        return 'compra'
    elif diff_rel <= -min_diff_rel:
        return 'venda'
    else:
        return 'neutro'

dados_back_aligned['Tipo_Operacao'] = [
    tipo_operacao(opn, pred, min_diff_rel)
    for opn, pred in zip(
        dados_back_aligned['Open'],
        dados_back_aligned['Predicted']
    )
]

# ========================================
#  🔹 SIMULAÇÃO: 1 POSIÇÃO POR VEZ, SL/TP INCLUSIVOS (>= / <=)
#   - ADIÇÃO: se houver posição aberta e o sinal do dia for OPOSTO,
#     fechamos a posição ao OPEN do dia e imediatamente abrimos a posição inversa.
# ========================================
def simular_capital(dados):
    capital = capital_inicial
    historico = []
    trades = []

    posicao_aberta = False
    tipo_posicao = None
    preco_entrada = None
    stop = None
    alvo = None
    capital_no_entrada = None
    data_entrada = None

    num_buy_opened = 0
    num_sell_opened = 0

    for i in range(len(dados)):
        row = dados.iloc[i]
        date = row.name
        open_p = float(row['Open'])
        high_p = float(row['High'])
        low_p = float(row['Low'])
        tipo_sinal = row['Tipo_Operacao']

        # 1) verificar fechamento por SL/TP
        if posicao_aberta:
            fechado_por_sl_tp = False
            if tipo_posicao == 'compra':
                if low_p <= stop:  # stop atingido (inclusivo)
                    preco_saida = stop
                    retorno_frac = (preco_saida - preco_entrada) / preco_entrada
                    profit_money = capital_no_entrada * retorno_frac
                    trades.append({
                        'Data_Entrada': data_entrada,
                        'Data_Saida': date,
                        'Tipo': tipo_posicao,
                        'Preco_Entrada': preco_entrada,
                        'Preco_Saida': preco_saida,
                        'Retorno_frac': retorno_frac,
                        'Duracao_Dias': (date - data_entrada).days if hasattr(date, 'day') else 0,
                        'Capital_Entrada': capital_no_entrada,
                        'Profit_Money': profit_money
                    })
                    capital *= (1 + retorno_frac)
                    posicao_aberta = False
                    fechado_por_sl_tp = True
                elif high_p >= alvo:  # take atingido (inclusivo)
                    preco_saida = alvo
                    retorno_frac = (preco_saida - preco_entrada) / preco_entrada
                    profit_money = capital_no_entrada * retorno_frac
                    trades.append({
                        'Data_Entrada': data_entrada,
                        'Data_Saida': date,
                        'Tipo': tipo_posicao,
                        'Preco_Entrada': preco_entrada,
```

```
                    'Preco_Saida': preco_saida,
                    'Retorno_frac': retorno_frac,
                    'Duracao_Dias': (date - data_entrada).days if hasattr(date, 'day') else 0,
                    'Capital_Entrada': capital_no_entrada,
                    'Profit_Money': profit_money
                })
                capital *= (1 + retorno_frac)
                posicao_aberta = False
                fechado_por_sl_tp = True

            elif tipo_posicao == 'venda':
                if high_p >= stop:  # stop atingido (inclusivo)
                    preco_saida = stop
                    retorno_frac = (preco_entrada - preco_saida) / preco_entrada
                    profit_money = capital_no_entrada * retorno_frac
                    trades.append({
                        'Data_Entrada': data_entrada,
                        'Data_Saida': date,
                        'Tipo': tipo_posicao,
                        'Preco_Entrada': preco_entrada,
                        'Preco_Saida': preco_saida,
                        'Retorno_frac': retorno_frac,
                        'Duracao_Dias': (date - data_entrada).days if hasattr(date, 'day') else 0,
                        'Capital_Entrada': capital_no_entrada,
                        'Profit_Money': profit_money
                    })
                    capital *= (1 + retorno_frac)
                    posicao_aberta = False
                    fechado_por_sl_tp = True
                elif low_p <= alvo:  # take atingido (inclusivo)
                    preco_saida = alvo
                    retorno_frac = (preco_entrada - preco_saida) / preco_entrada
                    profit_money = capital_no_entrada * retorno_frac
                    trades.append({
                        'Data_Entrada': data_entrada,
                        'Data_Saida': date,
                        'Tipo': tipo_posicao,
                        'Preco_Entrada': preco_entrada,
                        'Preco_Saida': preco_saida,
                        'Retorno_frac': retorno_frac,
                        'Duracao_Dias': (date - data_entrada).days if hasattr(date, 'day') else 0,
                        'Capital_Entrada': capital_no_entrada,
                        'Profit_Money': profit_money
                    })
                    capital *= (1 + retorno_frac)
                    posicao_aberta = False
                    fechado_por_sl_tp = True
            else:
                fechado_por_sl_tp = False

            # se fechou por SL/TP já atualizado capital e trade, então não faz reversão baseada no mesmo sinal
            if fechado_por_sl_tp:
                # posicao_aberta agora False; continue para possível nova abertura mais abaixo
                pass

            else:
                # 2) se não fechou por SL/TP e há sinal oposto, fazer REVERSÃO:
                #    fechar a posição pelo OPEN do dia e abrir a posição inversa no mesmo OPEN.
                #    (isso evita acumular várias posições em aberto)
                if tipo_sinal in ['compra','venda'] and tipo_sinal != tipo_posicao:
                    # fechar posição atual ao OPEN do dia
                    preco_saida = open_p
                    if tipo_posicao == 'compra':
                        retorno_frac = (preco_saida - preco_entrada) / preco_entrada
                    else:  # venda
                        retorno_frac = (preco_entrada - preco_saida) / preco_entrada
                    profit_money = capital_no_entrada * retorno_frac
                    trades.append({
                        'Data_Entrada': data_entrada,
                        'Data_Saida': date,
                        'Tipo': tipo_posicao,
                        'Preco_Entrada': preco_entrada,
                        'Preco_Saida': preco_saida,
                        'Retorno_frac': retorno_frac,
                        'Duracao_Dias': (date - data_entrada).days if hasattr(date, 'day') else 0,
                        'Capital_Entrada': capital_no_entrada,
                        'Profit_Money': profit_money
                    })
                    capital *= (1 + retorno_frac)
                    # fechar
                    posicao_aberta = False

                    # depois de fechado, ABRIR a posição inversa no MESMO OPEN (reversão)
```

```
                    posicao_aberta = True
                    tipo_posicao = tipo_sinal
                    preco_entrada = open_p
                    data_entrada = date
                    capital_no_entrada = capital
                    if tipo_posicao == 'compra':
                        stop = preco_entrada * (1 - stop_loss)
                        alvo = preco_entrada * (1 + take_profit)
                        num_buy_opened += 1
                    else:
                        stop = preco_entrada * (1 + stop_loss)
                        alvo = preco_entrada * (1 - take_profit)
                        num_sell_opened += 1
                    # terminou reversão

        # 3) Se NÃO havia posição aberta, apenas abrir quando sinal pedir
        if (not posicao_aberta) and (tipo_sinal in ['compra','venda']):
            posicao_aberta = True
            tipo_posicao = tipo_sinal
            preco_entrada = open_p
            data_entrada = date
            capital_no_entrada = capital
            if tipo_posicao == 'compra':
                stop = preco_entrada * (1 - stop_loss)
                alvo = preco_entrada * (1 + take_profit)
                num_buy_opened += 1
            else:
                stop = preco_entrada * (1 + stop_loss)
                alvo = preco_entrada * (1 - take_profit)
                num_sell_opened += 1

        historico.append(capital)

    # se posição ainda aberta no final, fechamos ao último close (saída forçada)
    posicoes_abertas_ao_final = 0
    if posicao_aberta:
        last_date = dados.index[-1]
        last_close = float(dados.iloc[-1]['Close_Real'])
        if tipo_posicao == 'compra':
            retorno_frac = (last_close - preco_entrada) / preco_entrada
        else:
            retorno_frac = (preco_entrada - last_close) / preco_entrada
        profit_money = capital_no_entrada * retorno_frac
        trades.append({
            'Data_Entrada': data_entrada,
            'Data_Saida': last_date,
            'Tipo': tipo_posicao,
            'Preco_Entrada': preco_entrada,
            'Preco_Saida': last_close,
            'Retorno_frac': retorno_frac,
            'Duracao_Dias': (last_date - data_entrada).days if hasattr(last_date, 'day') else 0,
            'Capital_Entrada': capital_no_entrada,
            'Profit_Money': profit_money
        })
        capital *= (1 + retorno_frac)
        historico.append(capital)
        posicoes_abertas_ao_final = 0  # fechamos forçadamente

    trades_df = pd.DataFrame(trades)
    if not trades_df.empty:
        trades_df['Retorno_pct'] = trades_df['Retorno_frac'] * 100

    return historico, trades_df, num_buy_opened, num_sell_opened, posicoes_abertas_ao_final

# ========================================
# 🔹 RODAR BACKTEST
# ========================================
historico, trades_df, num_buy_opened, num_sell_opened, posicoes_abertas_ao_final = simular_capital(dados_back_aligned)
capital_final = float(historico[-1]) if len(historico) > 0 else capital_inicial

# ========================================
# 🔹 CALCULAR MÉTRICAS SOLICITADAS
# ========================================
# operações fechadas
operacoes = trades_df.shape[0] if not trades_df.empty else 0

if operacoes > 0:
    vencedoras = trades_df[trades_df['Retorno_frac'] > 0].copy()
    perdedoras = trades_df[trades_df['Retorno_frac'] <= 0].copy()

    n_vencedoras = len(vencedoras)
    n_perdedoras = len(perdedoras)
```

```
    perc_vencedoras = n_vencedoras / operacoes * 100

    media_lucro_pct = vencedoras['Retorno_frac'].mean() * 100 if n_vencedoras > 0 else 0
    media_prejuizo_pct = perdedoras['Retorno_frac'].mean() * 100 if n_perdedoras > 0 else 0

    # razão média lucro/prejuízo (magnitude)
    razao_media = (abs(vencedoras['Retorno_frac'].mean()) / abs(perdedoras['Retorno_frac'].mean())) if (n_vencedoras>0 and n

    # lucro/prejuízo em R$
    lucro_liquido = vencedoras['Profit_Money'].sum() if 'Profit_Money' in vencedoras.columns else (vencedoras['Retorno_frac'
    prejuizo_bruto = - (perdedoras['Profit_Money'].sum() if 'Profit_Money' in perdedoras.columns else (perdedoras['Retorno_f

    # fator de lucro
    fator_de_lucro = (lucro_liquido / prejuizo_bruto) if prejuizo_bruto != 0 else np.nan

    # durações médias
    media_tempo_win = vencedoras['Duracao_Dias'].mean() if n_vencedoras>0 else 0
    media_tempo_loss = perdedoras['Duracao_Dias'].mean() if n_perdedoras>0 else 0
else:
    n_vencedoras = n_perdedoras = 0
    perc_vencedoras = media_lucro_pct = media_prejuizo_pct = razao_media = np.nan
    lucro_liquido = prejuizo_bruto = fator_de_lucro = np.nan
    media_tempo_win = media_tempo_loss = np.nan

# drawdown como % do saldo (equity)
equity = np.array(historico) if len(historico)>0 else np.array([capital_inicial])
peak = np.maximum.accumulate(equity)
drawdowns = (equity - peak) / peak
max_drawdown_pct = float(drawdowns.min() * 100) if len(drawdowns)>0 else 0

# saldo líquido e lucro total
saldo_liquido_total = capital_final
lucro_liquido_total = capital_final - capital_inicial
patrimonio_necessario = trades_df['Capital_Entrada'].max() if (not trades_df.empty and 'Capital_Entrada' in trades_df.column

# ========================================
# 🔹 EXIBIR RESULTADOS
# ========================================
print("\n=== RESULTADOS DO BACKTESTING ===")
print(f"🗓️ Período configurado: {periodo_configurado}")
print(f"🗓️ Período real utilizado: {periodo_real_inicio} → {periodo_real_fim}")
print(f"💰 Capital inicial: R$ {capital_inicial:,.2f}")
print(f"💰 Capital final: R$ {capital_final:,.2f}")
print(f"🔁 Operações (fechadas): {operacoes}")
print(f"✅ Operações vencedoras: {n_vencedoras} | ❌ Perdedoras: {n_perdedoras} | Win%: {perc_vencedoras:.2f}%")
print(f"📈 Média Lucro (% das vencedoras): {media_lucro_pct:.4f}%")
print(f"📉 Média Prejuízo (% das perdedoras): {media_prejuizo_pct:.4f}%")
print(f"⚖️ Razão média (média lucro / média prejuízo): {razao_media if not np.isnan(razao_media) else 'N/A'}")
print(f"📉 Max Drawdown: {max_drawdown_pct:.2f}%")
print(f"🏦 Saldo líquido total: R$ {saldo_liquido_total:,.2f}")
print(f"💹 Lucro líquido total: R$ {lucro_liquido_total:,.2f}")
print(f"💸 Lucro líquido (somatório vencedoras em R$): R$ {lucro_liquido:.2f}")
print(f"📉 Prejuízo bruto (somatório perdas em R$): R$ {prejuizo_bruto:.2f}")
print(f"📊 Fator de lucro: {fator_de_lucro if not np.isnan(fator_de_lucro) else 'N/A'}")
print(f"⏱️ Tempo médio Win: {media_tempo_win:.2f} dias | Loss: {media_tempo_loss:.2f} dias")
print(f"🏦 Patrimônio necessário para maior operação: R$ {patrimonio_necessario:,.2f}")
print(f"📌 Total de compras abertas (aberturas): {num_buy_opened}")
print(f"📌 Total de vendas abertas (aberturas): {num_sell_opened}")
print(f"📌 Posições abertas ao final (fechadas forçadamente): {posicoes_abertas_ao_final}")

if not trades_df.empty:
    print("\n🔎 Amostra de trades (primeiras 10):")
    display_cols = ['Data_Entrada','Data_Saida','Tipo','Preco_Entrada','Preco_Saida','Retorno_pct','Duracao_Dias','Capital_E
    display_cols = [c for c in display_cols if c in trades_df.columns]
    print(trades_df[display_cols].to_string(index=False))

# ========================================
# 🔹 GRÁFICOS
# ========================================
plt.figure(figsize=(14,6))
plt.plot(dados_back_aligned.index, dados_back_aligned['Close_Real'], label='Fechamento Real', color='blue')
plt.plot(dados_back_aligned.index, dados_back_aligned['Predicted'], label='Fechamento Previsto', color='red', linestyle='--'

plt.scatter(dados_back_aligned.index[dados_back_aligned['Tipo_Operacao']=='compra'],
            dados_back_aligned['Close_Real'][dados_back_aligned['Tipo_Operacao']=='compra'],
            marker='^', color='green', label='Sinal Compra', s=50)

plt.scatter(dados_back_aligned.index[dados_back_aligned['Tipo_Operacao']=='venda'],
            dados_back_aligned['Close_Real'][dados_back_aligned['Tipo_Operacao']=='venda'],
            marker='v', color='orange', label='Sinal Venda', s=50)

plt.title('Backtesting - Previsão x Real')
plt.legend()
```

```
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()

plt.figure(figsize=(12,5))
plt.plot(range(len(historico)), historico, color='green', linewidth=2)
plt.title('Evolução do Capital')
plt.xlabel('Dias')
plt.ylabel('Capital (R$)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

Mostrar saída oculta

```
# ========================================
# 🔹 SALVAR, COMPACTAR E BAIXAR RESULTADOS DO BACKTESTING
# ========================================
import os
import shutil
from pathlib import Path

# Cria nome do experimento usando o intervalo e o nome do modelo
modelo_nome = Path(caminho_modelo).stem if 'caminho_modelo' in globals() else "modelo_lstm"
intervalo_nome = str(intervalo).replace("/", "_").replace(" ", "")
nome_experimento = f"backtesting_{modelo_nome}_{intervalo_nome}"

pasta_saida = f"/content/resultados_{nome_experimento}"
os.makedirs(pasta_saida, exist_ok=True)

# 1) Salvar todos os trades executados
trades_path = os.path.join(pasta_saida, f"{nome_experimento}_trades.csv")
trades_df.to_csv(trades_path, index=False, encoding="utf-8-sig")

# 2) Salvar previsões, preços reais e sinais
previsoes_path = os.path.join(pasta_saida, f"{nome_experimento}_previsoes_vs_real.csv")
dados_back_aligned.to_csv(previsoes_path, encoding="utf-8-sig")

# 3) Salvar histórico de capital
historico_capital_df = pd.DataFrame({
    "Passo": range(len(historico)),
    "Capital": historico
})
historico_capital_path = os.path.join(pasta_saida, f"{nome_experimento}_historico_capital.csv")
historico_capital_df.to_csv(historico_capital_path, index=False, encoding="utf-8-sig")

# 4) Salvar métricas finais do backtesting
metricas = {
    "Modelo": modelo_nome,
    "Intervalo": intervalo,
    "Periodo_Config": periodo_configurado if 'periodo_configurado' in globals() else f"{backtest_start} -> {backtest_end}",
    "Periodo_Real_Inicio": periodo_real_inicio if 'periodo_real_inicio' in globals() else "",
    "Periodo_Real_Fim": periodo_real_fim if 'periodo_real_fim' in globals() else "",
    "Capital_Inicial": capital_inicial,
    "Capital_Final": capital_final,
    "Operacoes_Fechadas": operacoes,
    "Operacoes_Vencedoras": n_vencedoras,
    "Operacoes_Perdedoras": n_perdedoras,
    "Win_Rate_pct": perc_vencedoras,
    "Media_Lucro_pct": media_lucro_pct,
    "Media_Prejuizo_pct": media_prejuizo_pct,
    "Razao_Media_Lucro_Prejuizo": razao_media,
    "Max_Drawdown_pct": max_drawdown_pct,
    "Saldo_Liquido_Total": saldo_liquido_total,
    "Lucro_Liquido_Total": lucro_liquido_total,
    "Lucro_Bruto": lucro_liquido,
    "Prejuizo_Bruto": prejuizo_bruto,
    "Fator_Lucro": fator_de_lucro,
    "Tempo_Medio_Win": media_tempo_win,
    "Tempo_Medio_Loss": media_tempo_loss,
    "Patrimonio_Necessario": patrimonio_necessario,
    "Total_Compras_Abertas": num_buy_opened,
    "Total_Vendas_Abertas": num_sell_opened,
    "Posicoes_Abertas_Ao_Final": posicoes_abertas_ao_final
}

metricas_df = pd.DataFrame([metricas])
metricas_path = os.path.join(pasta_saida, f"{nome_experimento}_metricas.csv")
metricas_df.to_csv(metricas_path, index=False, encoding="utf-8-sig")

# 5) Salvar um arquivo Excel consolidado com as principais abas
```

```
# Função auxiliar para salvar no Excel sem erro de timezone.
# O Excel não aceita datas com fuso horário, então removemos o timezone
# tanto do índice quanto das colunas datetime.
def remover_timezone_excel(df):
    df_excel = df.copy()

    # Remove timezone do índice, se houver
    if isinstance(df_excel.index, pd.DatetimeIndex):
        if df_excel.index.tz is not None:
            df_excel.index = df_excel.index.tz_localize(None)

    # Remove timezone de colunas datetime com timezone
    for col in df_excel.columns:
        if pd.api.types.is_datetime64tz_dtype(df_excel[col]):
            df_excel[col] = df_excel[col].dt.tz_localize(None)
        elif pd.api.types.is_object_dtype(df_excel[col]):
            # Converte objetos Timestamp com timezone para string limpa, se existirem
            df_excel[col] = df_excel[col].apply(
                lambda x: x.tz_localize(None) if hasattr(x, 'tzinfo') and x.tzinfo is not None and hasattr(x, 'tz_localize') 
            )

    return df_excel

excel_path = os.path.join(pasta_saida, f"{nome_experimento}_resultados.xlsx")
with pd.ExcelWriter(excel_path, engine="openpyxl") as writer:
    remover_timezone_excel(metricas_df).to_excel(writer, sheet_name="metricas", index=False)
    remover_timezone_excel(trades_df).to_excel(writer, sheet_name="trades", index=False)
    remover_timezone_excel(dados_back_aligned).to_excel(writer, sheet_name="previsoes_vs_real")
    remover_timezone_excel(historico_capital_df).to_excel(writer, sheet_name="historico_capital", index=False)

# 6) Salvar gráfico Previsão x Real
grafico_previsao_path = os.path.join(pasta_saida, f"{nome_experimento}_previsao_vs_real.png")

plt.figure(figsize=(14,6))
plt.plot(dados_back_aligned.index, dados_back_aligned['Close_Real'], label='Fechamento Real', color='blue')
plt.plot(dados_back_aligned.index, dados_back_aligned['Predicted'], label='Fechamento Previsto', color='red', linestyle='--'

plt.scatter(dados_back_aligned.index[dados_back_aligned['Tipo_Operacao']=='compra'],
            dados_back_aligned['Close_Real'][dados_back_aligned['Tipo_Operacao']=='compra'],
            marker='^', color='green', label='Sinal Compra', s=50)

plt.scatter(dados_back_aligned.index[dados_back_aligned['Tipo_Operacao']=='venda'],
            dados_back_aligned['Close_Real'][dados_back_aligned['Tipo_Operacao']=='venda'],
            marker='v', color='orange', label='Sinal Venda', s=50)

plt.title('Backtesting - Previsão x Real')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.savefig(grafico_previsao_path, dpi=300, bbox_inches='tight')
plt.show()

# 7) Salvar gráfico Evolução do Capital
grafico_capital_path = os.path.join(pasta_saida, f"{nome_experimento}_evolucao_capital.png")

plt.figure(figsize=(12,5))
plt.plot(range(len(historico)), historico, color='green', linewidth=2)
plt.title('Evolução do Capital')
plt.xlabel('Passo')
plt.ylabel('Capital (R$)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.savefig(grafico_capital_path, dpi=300, bbox_inches='tight')
plt.show()

# 8) Compactar todos os arquivos em ZIP para facilitar o download
zip_path = f"{pasta_saida}.zip"
if os.path.exists(zip_path):
    os.remove(zip_path)

shutil.make_archive(pasta_saida, 'zip', pasta_saida)

print("\n✅ Arquivos do backtesting salvos com sucesso!")
print(f"📁 Pasta de saída: {pasta_saida}")
print(f"📦 Arquivo ZIP: {zip_path}")
print(f"📊 Excel consolidado: {excel_path}")
print(f"📄 Trades: {trades_path}")
print(f"📄 Previsões x Real: {previsoes_path}")
print(f"📄 Histórico de capital: {historico_capital_path}")
print(f"📄 Métricas: {metricas_path}")
print(f"🖼️ Gráfico Previsão x Real: {grafico_previsao_path}")
print(f"🖼️ Gráfico Evolução do Capital: {grafico_capital_path}")
```

```
# 9) Download automático no Google Colab
try:
    from google.colab import files
    print("\n⬇️ Iniciando download automático do ZIP...")
    files.download(zip_path)
except Exception:
    print("\n⚠️ Download automático disponível apenas no Google Colab.")
    print("Baixe manualmente o arquivo ZIP no caminho abaixo:")
    print(zip_path)
```

Mostrar saída oculta

Comece a programar ou gere código com IA.