# ****Documentação do Sistema de Previsão de Preço de Ações com LSTM****

## Sumário

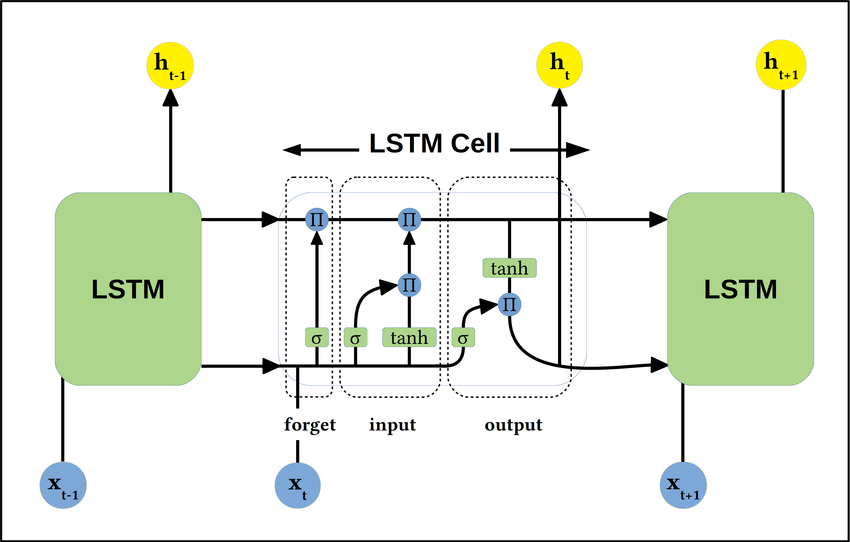
1. [Visão Geral do Projeto](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "1-visão-geral-do-projeto)
2. [Instalação e Configuração](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "2-instalação-e-configuração)
   * [2.1 Requisitos do Sistema](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "21-requisitos-do-sistema)
   * [2.2 Dependências](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "22-dependências)
   * [2.3 Instalação](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "23-instalação)
   * [2.4 Configuração Inicial](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "24-configuração-inicial)
3. [Estrutura do Projeto](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "3-estrutura-do-projeto)
4. [Funcionalidades Principais](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "4-funcionalidades-principais)
   * [4.1 Download de Dados de Ações](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "41-download-de-dados-de-ações)
   * [4.2 Processamento de Dados](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "42-processamento-de-dados)
   * [4.3 Modelo LSTM](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "43-modelo-lstm)
   * [4.4 Treinamento do Modelo](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "44-treinamento-do-modelo)
   * [4.5 Previsão de Preços](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "45-previsão-de-preços)
   * [4.6 Armazenamento em Banco de Dados](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "46-armazenamento-em-banco-de-dados)
   * [4.7 Visualizações](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "47-visualizações)
5. [Uso da Aplicação](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "5-uso-da-aplicação)
   * [5.1 Interface Web](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "51-interface-web)
   * [5.2 Fluxo de Uso](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "52-fluxo-de-uso)
6. [Monitoramento e Métricas](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "6-monitoramento-e-métricas)
   * [6.1 MLflow](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "61-mlflow)
   * [6.2 Prometheus](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "62-prometheus)
7. [Informações Adicionais](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "7-informações-adicionais)
   * [7.1 Considerações de Segurança](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "71-considerações-de-segurança)
   * [7.2 Tratamento de Erros](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "72-tratamento-de-erros)
8. [Limitações e Melhorias Futuras](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "8-limitações-e-melhorias-futuras)
9. [Apêndice: Referência de API](https://claude.ai/chat/afd3f2fe-35f8-4263-81e4-aeb67bd92dd3" \l "9-apêndice-referência-de-api)

## 1. Visão Geral do Projeto

## ****Objetivo****

O objetivo deste script é fornecer uma previsão do preço das ações de uma empresa usando o modelo LSTM (Long Short-Term Memory) para séries temporais. O usuário pode inserir um intervalo de datas para coletar os dados de ações, visualizar gráficos e obter previsões para o próximo dia e para os próximos 5 dias.

**LSTM** (Long Short-Term Memory) é um tipo de rede neural recorrente (RNN, do inglês Recurrent Neural Network) que foi projetada para permitir captura de dependências de longo prazo em dados sequenciais. É composta por **células de memória** que podem manter informações ao longo do tempo e são controladas por três portas principais: **Porta de entrada (input gate)**: Decide quais informações serão armazenadas na célula de memória. **Porta de esquecimento (forget gate)**: Determina quais informações da célula de memória serão descartadas e **Porta de saída (output gate)**: Controla o que será produzido como saída da célula.



## 

## 2 Instalação e Configuração

### 2.1 Requisitos do Sistema

* Python 3.6+ (recomendado Python 3.8 ou superior)
* Acesso à internet para download de dados de ações
* Mínimo de 4GB de RAM (8GB recomendado para datasets maiores)
* Processador multi-core (recomendado para treinamento de modelo)
* GPU compatível com CUDA (opcional, para treinamento acelerado)

### 2.2 Dependências

O sistema requer as seguintes bibliotecas Python:

| Biblioteca | Versão Mínima | Uso no Sistema |
| --- | --- | --- |
| Flask | 2.0.0+ | Framework de aplicação web |
| NumPy | 1.20.0+ | Computação numérica |
| Pandas | 1.3.0+ | Manipulação de dados |
| yfinance | 0.1.70+ | Download de dados de ações |
| Matplotlib | 3.4.0+ | Visualização de dados |
| scikit-learn | 1.0.0+ | Pré-processamento de dados |
| PyTorch | 1.9.0+ | Framework de deep learning |
| MLflow | 1.20.0+ | Rastreamento de experimentos (opcional) |
| Prometheus | 0.9.0+ | Monitoramento de métricas (opcional) |

### 2.3 Instalação

# Clonar o repositório

git clone [URL\_DO\_REPOSITÓRIO]

cd [NOME\_DO\_DIRETÓRIO]

# Criar ambiente virtual (recomendado)

python -m venv venv

source venv/bin/activate # No Windows: venv\Scripts\activate

# Instalar dependências

pip install -r requirements.txt

# Instalar dependências opcionais para GPU (se disponível)

pip install torch==1.12.0+cu113 -f https://download.pytorch.org/whl/torch\_stable.html

### 2.4 Configuração Inicial

Antes de executar o aplicativo, o sistema inicializa automaticamente o banco de dados SQLite. A inicialização do banco de dados é executada na primeira execução do aplicativo:

# Iniciar o servidor web

python app.py

O servidor estará disponível em http://localhost:5000.

Configurações adicionais podem ser ajustadas diretamente no código-fonte:

* Parâmetros do modelo (sequence\_length, hidden\_size, num\_epochs)
* Intervalo de tempo para download de dados
* Opções de visualização

## 3. Estrutura do Projeto

O projeto é organizado nos seguintes componentes principais:

stock-prediction-app/

├── app.py # Aplicativo principal (Flask + lógica de negócios)

├── model\_utils.py # Utilitários do modelo (importado pelo app.py)

├── requirements.txt # Dependências do projeto

├── static/ # Arquivos estáticos (CSS, JS, imagens)

├── templates/ # Templates HTML para interface web

│ ├── form.html # Formulário para entrada de dados

│ ├── result.html # Página de resultados da previsão

│ └── history.html # Página de histórico

└── stock\_prediction.db # Banco de dados SQLite (criado automaticamente)

### Componentes principais:

* **Aplicativo Flask**: Interface web para interação com o usuário
* **Modelo LSTM PyTorch**: Implementação do modelo de deep learning para previsão
* **Banco de dados SQLite**: Armazenamento de dados históricos e previsões
* **MLflow**: Rastreamento de experimentos e métricas
* **Prometheus**: Monitoramento de métricas em tempo real (opcional)

## 4. Funcionalidades Principais

### 4.1 Download de Dados de Ações

O sistema utiliza a biblioteca yfinance para baixar dados históricos de ações, implementando tratamento de erros e timeout para garantir robustez:

def download\_with\_timeout(symbol, start\_date, end\_date, timeout=180):

"""

Baixa dados históricos de ações usando yfinance.

Parâmetros:

- symbol: Símbolo da ação (ex: "QQQ", "AAPL")

- start\_date: Data inicial no formato YYYY-MM-DD

- end\_date: Data final no formato YYYY-MM-DD

- timeout: Tempo limite em segundos

Retorna:

- DataFrame do pandas com dados históricos de ações

"""

Os dados baixados incluem:

* Data
* Preço de abertura
* Preço máximo
* Preço mínimo
* Preço de fechamento
* Volume

Para a previsão, o sistema foca principalmente no preço de fechamento ajustado.

### 4.2 Processamento de Dados

As séries temporais são normalizadas usando MinMaxScaler e transformadas em sequências adequadas para o modelo LSTM:

def create\_sequences(data, seq\_length):

"""

Cria sequências temporais para o modelo LSTM.

Parâmetros:

- data: Dados normalizados

- seq\_length: Tamanho da sequência (número de dias anteriores)

Retorna:

- xs: Sequências de entrada

- ys: Valores alvo (próximo dia)

"""

O processamento de dados inclui:

1. Normalização dos dados para o intervalo [0, 1]
2. Criação de sequências de entrada/saída
3. Divisão de dados para treinamento

### 4.3 Modelo LSTM

O modelo LSTM é implementado usando PyTorch com a seguinte arquitetura:

class LSTMModel(nn.Module):

def \_\_init\_\_(self, input\_size=1, hidden\_layer\_size=150, output\_size=1, num\_layers=1):

"""

Implementação do modelo LSTM com PyTorch.

Parâmetros:

- input\_size: Tamanho da entrada (geralmente 1 para preço de fechamento)

- hidden\_layer\_size: Tamanho da camada oculta

- output\_size: Tamanho da saída (geralmente 1 para previsão do próximo dia)

- num\_layers: Número de camadas LSTM

"""

A arquitetura do modelo consiste em:

* Camada LSTM para processamento de sequências temporais
* Camada totalmente conectada (linear) para previsão
* Função de ativação para saída

### 4.4 Treinamento do Modelo

O treinamento do modelo é realizado com acompanhamento de métricas e suporte a GPU:

def train\_lstm(model, X, y, num\_epochs=55, batch\_size=32):

"""

Treina o modelo LSTM com PyTorch.

Parâmetros:

- model: Modelo LSTM PyTorch

- X, y: Dados de treinamento

- num\_epochs: Número de épocas de treinamento

- batch\_size: Tamanho do lote para treinamento

Retorna:

- model: Modelo treinado

- loss\_history: Histórico de perdas por época

- sigmoid\_value: Valor sigmóide baseado na perda média

"""

O processo de treinamento inclui:

1. Conversão de dados para tensores PyTorch
2. Criação de DataLoader para treinamento em lotes
3. Definição da função de perda (MSE) e otimizador (Adam)
4. Utilização de GPU se disponível
5. Loop de treinamento com rastreamento de métricas
6. Cálculo de métrica sigmóide a partir da perda média

### 4.5 Previsão de Preços

O modelo treinado é utilizado para prever preços futuros:

def predict\_prices(data, model, scaler, sequence\_length, days\_ahead=1):

"""

Gera previsões para dias futuros.

Parâmetros:

- data: Dados normalizados

- model: Modelo LSTM treinado

- scaler: Objeto para inverter a normalização

- sequence\_length: Tamanho da sequência

- days\_ahead: Número de dias a serem previstos

Retorna:

- prediction\_next\_day: Previsão para o próximo dia

- future\_predictions\_inv: Previsões para os próximos 'days\_ahead' dias

"""

O processo de previsão inclui:

1. Preparação da sequência atual com os últimos sequence\_length dias
2. Geração da previsão para o próximo dia
3. Para previsões de dias subsequentes, uso da previsão anterior como entrada
4. Inversão da normalização para obter valores reais

### 4.6 Armazenamento em Banco de Dados

O sistema utiliza SQLite para armazenar dados históricos, previsões e métricas do modelo:

def init\_db():

"""

Inicializa as tabelas do banco de dados SQLite.

Tabelas:

- raw\_stock\_data: Dados brutos das ações

- predictions: Previsões geradas

- model\_metrics: Métricas do modelo

"""

Estrutura do banco de dados:

1. **Tabela raw\_stock\_data**:
   * id (AUTOINCREMENT)
   * symbol (TEXT)
   * date (TEXT)
   * close\_price (REAL)
   * created\_at (TEXT)
2. **Tabela predictions**:
   * id (AUTOINCREMENT)
   * symbol (TEXT)
   * prediction\_date (TEXT)
   * target\_date (TEXT)
   * predicted\_price (REAL)
   * created\_at (TEXT)
3. **Tabela model\_metrics**:
   * id (AUTOINCREMENT)
   * symbol (TEXT)
   * run\_date (TEXT)
   * sequence\_length (INTEGER)
   * num\_epochs (INTEGER)
   * batch\_size (INTEGER)
   * loss\_history (TEXT - JSON)
   * sigmoid\_value (REAL)
   * created\_at (TEXT)

### 4.7 Visualizações

O sistema gera diversos gráficos para visualização de dados e resultados:

1. **Gráfico de linha**: Preço de fechamento ao longo do tempo

plt.plot(df\_close.index, df\_close['Close'], label='Preço de Fechamento')

1. **Histograma**: Distribuição do preço de fechamento

plt.hist(df\_close['Close'], bins=50, color='blue', edgecolor='black')

1. **Gráfico de previsão**: Previsões para os próximos 5 dias

plt.plot(range(1, 6), future\_predictions\_inv, marker='o', linestyle='-', label='Previsão')

1. **Curva de aprendizado**: Loss por época

plt.plot(range(1, num\_epochs+1), loss\_history, label='Loss (MSE)')

1. **Gráfico sigmóide**: Métrica baseada na perda média

plt.axhline(y=sigmoid\_value, color='r', linestyle='--', label=f'Valor Sigmoide: {sigmoid\_value:.4f}')

## 5. Uso da Aplicação

### 5.1 Interface Web

A aplicação possui duas páginas principais:

1. **Página inicial (**/**)**:
   * Formulário para definir parâmetros e executar previsões
   * Campos para símbolo da ação, data inicial e data final
   * Botão para iniciar o processo de previsão
2. **Página de resultados**:
   * Exibição da previsão para o próximo dia
   * Visualização das mudanças percentuais
   * Gráficos de histórico de preços e previsões
   * Métricas de desempenho do modelo
3. **Página de histórico (**/history**)**:
   * Visualização de previsões anteriores
   * Filtro por símbolo de ação

### 5.2 Fluxo de Uso

1. **Acesse a página inicial** (http://localhost:5000)
2. **Preencha o formulário com**:
   * Símbolo da ação (ex: "QQQ", "AAPL")
   * Data inicial (formato YYYY-MM-DD)
   * Data final (formato YYYY-MM-DD)
3. **Envie o formulário** para iniciar o processo:
   * Download dos dados históricos da ação
   * Processamento e normalização dos dados
   * Treinamento do modelo LSTM
   * Geração de previsões para os próximos dias
   * Criação de visualizações
   * Armazenamento de resultados no banco de dados
4. **Visualize os resultados**:
   * Previsão para o próximo dia de negociação
   * Variação percentual esperada
   * Gráficos de histórico e previsão
   * Métricas de desempenho do modelo
5. **Acesse o histórico de previsões** (http://localhost:5000/history):
   * Visualize previsões anteriores por símbolo
   * Compare previsões com resultados reais

## 6. Monitoramento e Métricas

### 6.1 MLflow

Se disponível, o sistema utiliza MLflow para rastrear experimentos:

# Exemplo de implementação com MLflow

mlflow.set\_experiment("LSTM Stock Prediction PyTorch")

with mlflow.start\_run():

mlflow.log\_param("sequence\_length", sequence\_length)

mlflow.log\_param("num\_epochs", num\_epochs)

mlflow.log\_param("batch\_size", batch\_size)

# Train the model with MLflow tracking

model, loss\_history, sigmoid\_value = train\_lstm(model, X, y, num\_epochs, batch\_size)

# Log the PyTorch model

mlflow.pytorch.log\_model(model, "lstm\_model")

MLflow registra:

* **Parâmetros do modelo**: sequence\_length, num\_epochs, batch\_size
* **Métricas de treinamento**: loss, rmse, mae
* **Artefatos**: Modelo PyTorch treinado

Os experimentos podem ser visualizados através da interface web do MLflow:

mlflow ui

### 6.2 Prometheus

Se disponível, o sistema utiliza Prometheus para monitoramento em tempo real:

# Exemplo de implementação com Prometheus

metrics = PrometheusMetrics(app)

metrics.info('appinfos', 'Application info', version='1.0.0')

MODEL\_LOSS = Summary('model\_loss\_count', 'Loss function during training')

MODEL\_ACCURACY = Summary('model\_accuracy\_count', 'Accuracy during training')

MODEL\_TRAINING\_TIME = Summary('model\_training\_duration\_seconds', 'Time spent training the model')

MODEL\_SIGMOID = Gauge('model\_sigmoid\_value', 'Sigmoid value computed from the average training loss', ['model'])

Prometheus monitora:

* **MODEL\_LOSS**: Função de perda durante o treinamento
* **MODEL\_ACCURACY**: Precisão durante o treinamento
* **MODEL\_TRAINING\_TIME**: Tempo gasto no treinamento
* **MODEL\_SIGMOID**: Valor sigmóide calculado a partir da perda média de treinamento

## 7. Informações Adicionais

### 7.1 Considerações de Segurança

O sistema desativa a verificação SSL para resolver problemas de certificado:

# Fix SSL certificate issues

ssl.\_create\_default\_https\_context = ssl.\_create\_unverified\_context

**Atenção**: Esta abordagem é útil para desenvolvimento e teste, mas deve ser utilizada com cuidado em ambientes de produção. Em ambientes de produção, é recomendável configurar certificados SSL válidos.

Outras considerações de segurança:

* O sistema não implementa autenticação de usuários
* Não há validação de entrada para proteger contra injeção SQL
* Dados financeiros são armazenados localmente sem criptografia

### 7.2 Tratamento de Erros

O sistema inclui tratamento de erros para:

1. **Falha no download de dados**:

try:

df = download\_with\_timeout(symbol, start\_date, end\_date)

if df is None or df.empty:

logging.error("Erro ao coletar os dados. Verifique as datas e tente novamente.")

return "Erro ao coletar os dados. Verifique as datas e tente novamente."

1. **Indisponibilidade de bibliotecas**:

try:

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader, TensorDataset

torch\_available = True

except ImportError:

torch\_available = False

print("PyTorch não disponível. O modelo LSTM não funcionará.")

1. **Erros de banco de dados**:

try:

with sqlite3.connect('stock\_prediction.db') as conn:

# Operações de banco de dados

except Exception as e:

logger.error(f"Erro ao salvar dados no banco de dados: {e}")

## 8. Limitações e Melhorias Futuras

### Limitações Atuais:

* Modelo LSTM simples pode não capturar todas as nuances do mercado financeiro
* Não considera fatores externos (notícias, eventos econômicos, etc.)
* Utiliza apenas o preço de fechamento como entrada
* Limitado a previsões de curto prazo (próximos dias)

### Melhorias Futuras:

1. **Aprimoramentos do Modelo**:
   * Implementação de modelos mais complexos (Transformers, GRU)
   * Adição de features como volume, indicadores técnicos, etc.
   * Implementação de ensemble de modelos para maior robustez
2. **Melhorias na Interface**:
   * Implementação de autenticação de usuários
   * Dashboard interativo com React/Vue.js
   * Notificações de previsões por email/SMS
3. **Funcionalidades Adicionais**:
   * Suporte a múltiplos modelos de machine learning
   * Integração com APIs adicionais de dados financeiros
   * Implementação de backtesting para validação de estratégias
   * Análise de sentimento de notícias e redes sociais
4. **Infraestrutura**:
   * Migração para banco de dados mais robusto (PostgreSQL)
   * Implementação de API REST para integração com outros sistemas
   * Containerização com Docker para facilitar implantação

## 9. Apêndice: Referência de API

### Endpoints da Aplicação Web

| Endpoint | Método | Descrição | Parâmetros |
| --- | --- | --- | --- |
| / | GET | Exibe o formulário de entrada | Nenhum |
| / | POST | Processa os dados e gera previsões | symbol, start\_date, end\_date |
| /history | GET | Exibe histórico de previsões | symbol (opcional) |

### Funções Principais

| Função | Descrição | Parâmetros |
| --- | --- | --- |
| download\_with\_timeout() | Baixa dados históricos | symbol, start\_date, end\_date, timeout |
| create\_sequences() | Cria sequências para LSTM | data, seq\_length |
| build\_lstm\_model() | Constrói o modelo LSTM | sequence\_length |
| train\_lstm() | Treina o modelo LSTM | model, X, y, num\_epochs, batch\_size |
| predict\_prices() | Gera previsões | data, model, scaler, sequence\_length, days\_ahead |
| save\_raw\_data() | Salva dados brutos | df, symbol |
| save\_predictions() | Salva previsões | symbol, prediction\_date, future\_predictions\_inv |
| save\_model\_metrics() | Salva métricas do modelo | symbol, sequence\_length, num\_epochs, batch\_size, loss\_history, sigmoid\_value |