🚀 BTC-PERP Absolute Trader - V1.0

Robô de Trading Algorítmico com Machine Learning

Um sistema completo de trading algorítmico para BTC-PERP (Bitcoin Perpetual Futures) que combina técnicas avançadas de Machine Learning, Deep Learning e análise quantitativa para gerar sinais de trading automatizados.

📋 Índice

🎯 Visão Geral

O **BTC-PERP Absolute Trader** é um sistema de trading algorítmico desenvolvido especificamente para operar contratos futuros perpétuos de Bitcoin. O sistema utiliza uma abordagem multi-modal que combina:

- **Análise Técnica Tradicional**: Indicadores como RSI, MACD, Bollinger Bands
- **Microestrutura de Mercado**: Análise do order book e flow de ordens
- **Dados On-Chain**: Métricas da blockchain Bitcoin
- **Machine Learning**: Modelos XGBoost e LSTM
- **Deep Learning**: Redes neurais para padrões complexos
- **Ensemble Learning**: Combinação inteligente de múltiplos modelos

🎯 Objetivos

1. **Maximizar Retornos**: Através de sinais precisos e timing otimizado
2. **Minimizar Riscos**: Sistema robusto de gestão de risco
3. **Automação Completa**: Operação 24/7 sem intervenção manual
4. **Transparência**: Métricas detalhadas e relatórios de performance
5. **Escalabilidade**: Arquitetura modular e extensível

✨ Características Principais

🔄 Coleta de Dados em Tempo Real

- **WebSocket Binance**: Dados de velas e order book em tempo real
- **APIs On-Chain**: Métricas da blockchain via múltiplas fontes
- **Dados de Sentimento**: Análise de notícias e redes sociais
- **Armazenamento Eficiente**: Formato Parquet com compressão

🧠 Engenharia de Features Avançada

- **45+ Features Técnicas**: Indicadores tradicionais e customizados
- **Features de Microestrutura**: Imbalance, pressure indicators
- **Features On-Chain**: Hash rate, dificuldade, atividade da rede
- **Features de Sentimento**: Scores de notícias e volume
- **Otimização Numba**: Performance acelerada para cálculos

🤖 Modelos de Machine Learning

- **XGBoost**: Modelo baseline robusto e interpretável
- **LSTM**: Redes neurais para padrões temporais complexos
- **Ensemble Avançado**: Combinação inteligente de modelos
- **Regime-Aware**: Adaptação automática a diferentes regimes de mercado

📊 Sistema de Backtesting

- **Simulação Realística**: Slippage, comissões e latência
- **Métricas Completas**: Sharpe, Sortino, Calmar, Maximum Drawdown
- **Análise de Trades**: Win rate, profit factor, distribuição de retornos
- **Visualizações**: Gráficos de equity curve e drawdown

⚡ Execução e Gestão de Risco

- **Motor de Execução Stub**: Simulação de ordens sem risco real
- **Position Sizing**: Kelly Criterion, volatility targeting
- **Stop Loss/Take Profit**: Níveis dinâmicos baseados em volatilidade
- **Risk Limits**: Controles automáticos de exposição

🌐 Interface Web e API

- **Dashboard Interativo**: Monitoramento em tempo real
- **API REST Completa**: Controle programático do sistema

- **Métricas em Tempo Real**: Performance e status do sistema
- **Controles Manuais**: Ordens manuais e ajustes de parâmetros

---

🚀 Inovações Implementadas

1. Detecção Automática de Regimes de Mercado

Sistema que classifica automaticamente o mercado em diferentes regimes (baixa/média/alta volatilidade, tendência/lateral) e ajusta os parâmetros dos modelos dinamicamente.

Benefícios: - Adaptação automática a mudanças de mercado - Melhor performance em diferentes condições - Redução de drawdowns em mercados laterais

2. Features de Sentimento Multi-Modal

Integração de dados de sentimento de múltiplas fontes (notícias, redes sociais, métricas on-chain) com pesos adaptativos baseados na relevância histórica.

Benefícios: - Antecipação de movimentos baseados em fundamentals - Melhor timing de entrada e saída - Redução de falsos sinais

3. Ensemble Consciente de Regimes

Sistema de ensemble que ajusta os pesos dos modelos baseado no regime atual do mercado, dando mais peso a modelos lineares em baixa volatilidade e modelos não-lineares em alta volatilidade.

Benefícios: - Melhor performance em diferentes condições de mercado - Maior robustez do sistema - Redução de overfitting

4. Microestrutura Avançada

Análise detalhada do order book incluindo imbalance, pressure indicators e flow de ordens para detectar movimentos de curto prazo.

Benefícios: - Melhor timing de execução - Redução de slippage - Detecção de movimentos institucionais

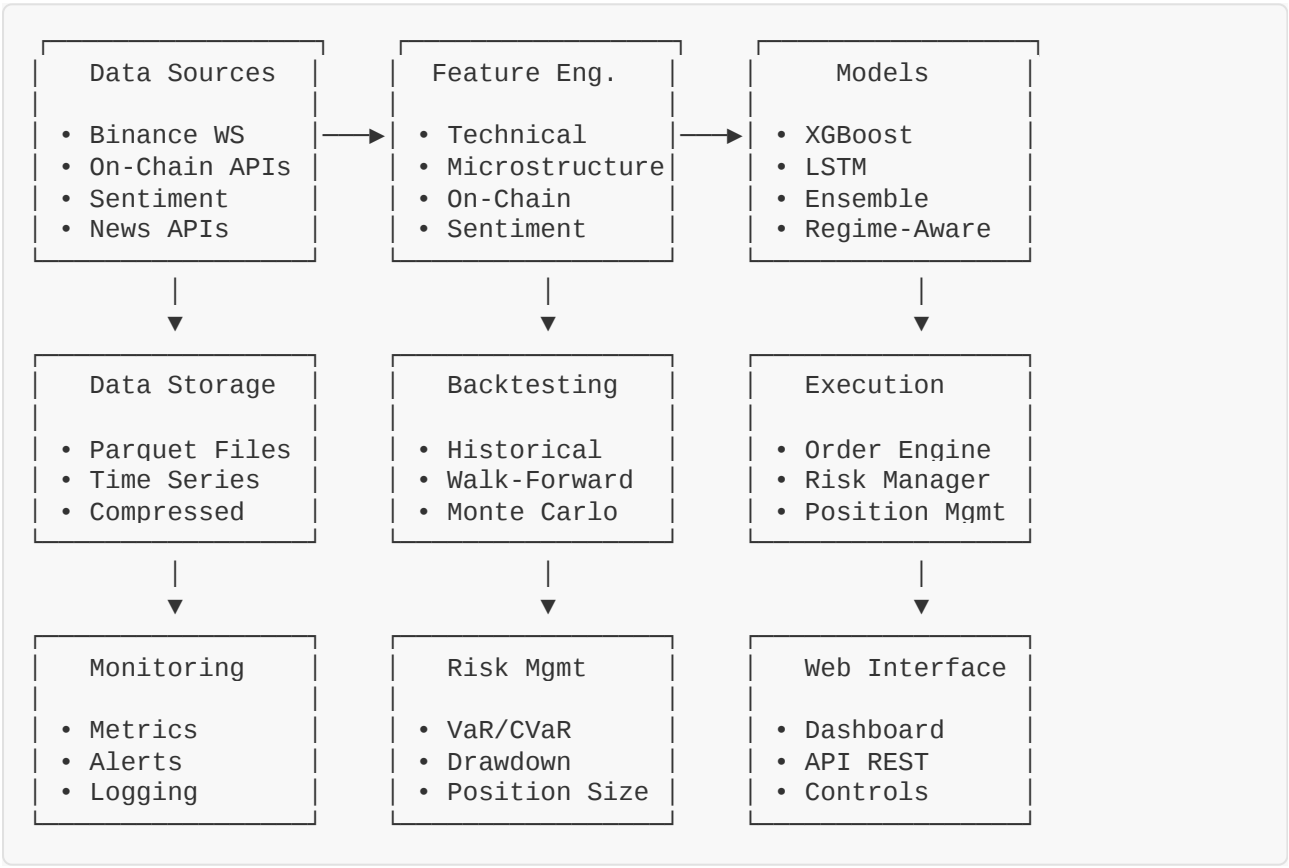
5. Position Sizing Adaptativo

Sistema de dimensionamento de posições que considera volatilidade atual, confiança do modelo e regime de mercado para otimizar o risco-retorno.

Benefícios: - Maximização do Sharpe ratio - Controle automático de risco - Adaptação a mudanças de volatilidade

---

🏗️ Arquitetura do Sistema



Componentes Principais

1. **Data Collectors** (`src/collectors/`): Coleta de dados em tempo real
2. **Feature Engineering** (`src/features/`): Processamento e criação de features
3. **Models** (`src/models/`): Modelos de ML/DL e ensemble

4. **Backtesting** (`src/backtest/`): Sistema de backtesting e análise
5. **Risk Management** (`src/risk/`): Gestão de risco e métricas
6. **Execution Engine** (`src/execution/`): Motor de execução de ordens
7. **Web API** (`trading_api/`): Interface web e API REST

🛠️ Instalação e Configuração

Pré-requisitos

- Python 3.11+
- Docker (opcional)
- 8GB+ RAM
- Conexão estável com internet

Instalação Local

```
# 1. Clonar o repositório
git clone <repository-url>
cd btc_perp_trader

# 2. Criar ambiente virtual
python -m venv venv
source venv/bin/activate  # Linux/Mac
# ou
venv\Scripts\activate  # Windows

# 3. Instalar dependências
pip install -r requirements.txt

# 4. Configurar variáveis de ambiente
cp config/.env.example config/.env
# Editar config/.env com suas configurações

# 5. Configurar parâmetros
# Editar config/config.yaml conforme necessário
```

Instalação com Docker

```
# 1. Construir imagem
docker-compose build

# 2. Iniciar serviços
docker-compose up -d

# 3. Verificar status
docker-compose ps
```

Configuração

Arquivo `config/config.yaml`

```yaml
# Configurações de trading
trading:
  initial_capital: 100000
  commission_rate: 0.001
  max_position_size: 0.1
  risk_free_rate: 0.02

# Configurações de dados
data:
  symbols: ["BTCUSDT"]
  timeframes: ["5m", "15m", "1h"]
  lookback_days: 30

# Configurações de modelos
models:
  xgboost:
    n_estimators: 1000
    max_depth: 6
    learning_rate: 0.1

  lstm:
    sequence_length: 60
    hidden_size: 128
    epochs: 100
```

Arquivo `config/.env`

```
# APIs (opcional para dados reais)
BINANCE_API_KEY=your_api_key
BINANCE_SECRET_KEY=your_secret_key

# Configurações de logging
LOG_LEVEL=INFO
LOG_FILE=logs/trading.log

# Configurações de banco de dados (se usar)
DATABASE_URL=sqlite:///data/trading.db
```

🎮 Uso do Sistema

Inicialização Rápida

```
# 1. Ativar ambiente
source venv/bin/activate

# 2. Iniciar API
cd trading_api
python src/main.py

# 3. Acessar interface web
# Abrir http://localhost:5000 no navegador
```

Uso Programático

```
from src.execution.execution_engine import ExecutionEngine, TradingBot
from src.models.ensemble_model import create_default_ensemble
from src.risk.risk_manager import RiskManager
from src.features.feature_engineering import FeatureEngineer

# 1. Criar componentes
engine = ExecutionEngine(initial_balance=100000)
risk_manager = RiskManager()
feature_engineer = FeatureEngineer()

# 2. Treinar modelo
ensemble = create_default_ensemble()
# ... código de treinamento ...

# 3. Criar bot
bot = TradingBot(engine, ensemble, risk_manager)

# 4. Iniciar trading
bot.start()

# 5. Processar dados em tempo real
# bot.process_market_data(symbol, data)
```

Backtesting

```python
from src.backtest.backtester import Backtester, MLTradingStrategy

# 1. Preparar dados
# data = load_historical_data()

# 2. Criar estratégia
strategy = MLTradingStrategy(model, threshold=0.001)

# 3. Executar backtesting
backtester = Backtester(initial_capital=100000)
results = backtester.run_backtest(data, strategy)

# 4. Analisar resultados
print(f"Total Return: {results['total_return_pct']:.2f}%")
print(f"Sharpe Ratio: {results['sharpe_ratio']:.2f}")
print(f"Max Drawdown: {results['max_drawdown_pct']:.2f}%")
```

🌐 API REST

Endpoints Principais

Status do Sistema

```
GET /api/trading/status
```

Retorna o status atual do sistema de trading.

Resposta:

```json
{
  "status": "running",
  "balance": 98500.50,
  "total_equity": 99200.75,
  "open_orders": 2,
  "total_trades": 45,
  "positions": 1,
  "timestamp": "2024-01-15T10:30:00Z"
}
```

Inicializar Sistema

```
POST /api/trading/initialize
Content-Type: application/json

{
  "initial_balance": 100000,
  "commission_rate": 0.001
}
```

Controle do Bot

```
POST /api/trading/start
POST /api/trading/stop
```

Colocar Ordem

```
POST /api/trading/place_order
Content-Type: application/json

{
  "symbol": "BTC-USDT",
  "side": "buy",
  "type": "market",
  "quantity": 0.1,
  "price": 50000.00
}
```

Obter Posições

```
GET /api/trading/positions
```

Obter Performance

```
GET /api/trading/performance
```

Documentação Completa

A documentação completa da API está disponível em `/api/docs` quando o servidor está rodando.

📊 Backtesting

Métricas Calculadas

Retorno e Risco

- **Total Return**: Retorno total do período
- **Annualized Return**: Retorno anualizado
- **Volatility**: Volatilidade anualizada
- **Sharpe Ratio**: Retorno ajustado ao risco
- **Sortino Ratio**: Retorno ajustado ao downside risk
- **Calmar Ratio**: Retorno anualizado / Maximum Drawdown

Drawdown

- **Maximum Drawdown**: Maior perda peak-to-trough
- **Average Drawdown**: Drawdown médio
- **Drawdown Duration**: Duração média dos drawdowns
- **Recovery Time**: Tempo médio de recuperação

Trades

- **Total Trades**: Número total de trades
- **Win Rate**: Percentual de trades vencedores
- **Profit Factor**: Lucro bruto / Perda bruta
- **Average Win**: Lucro médio por trade vencedor
- **Average Loss**: Perda média por trade perdedor
- **Largest Win/Loss**: Maior ganho/perda individual

Risk Metrics

- **Value at Risk (VaR)**: Perda máxima esperada (95% confiança)
- **Conditional VaR (CVaR)**: Perda esperada além do VaR
- **Beta**: Correlação com benchmark

- **Alpha**: Retorno em excesso ao benchmark

Exemplo de Relatório

```
 === BACKTEST RESULTS ===
Period: 2024-01-01 to 2024-12-31
Initial Capital: $100,000

PERFORMANCE METRICS
Total Return: 24.5%
Annualized Return: 22.8%
Volatility: 18.2%
Sharpe Ratio: 1.25
Sortino Ratio: 1.68
Calmar Ratio: 1.52

RISK METRICS
Maximum Drawdown: -8.5%
Average Drawdown: -2.1%
VaR (95%): -2.8%
CVaR (95%): -4.2%

TRADE ANALYSIS
Total Trades: 156
Win Rate: 58.3%
Profit Factor: 1.42
Average Win: $890
Average Loss: -$625
Largest Win: $3,200
Largest Loss: -$1,800
```

⚖️ Gestão de Risco

Controles Implementados

Position Sizing

- **Fixed Fraction**: Percentual fixo do capital
- **Kelly Criterion**: Otimização matemática baseada em win rate
- **Volatility Targeting**: Ajuste baseado na volatilidade atual
- **Risk Parity**: Equalização de risco entre posições

Stop Loss e Take Profit

- **Percentage-based**: Baseado em percentual do preço de entrada

- **ATR-based**: Baseado na volatilidade (Average True Range)
- **Dynamic**: Ajuste automático baseado em condições de mercado
- **Trailing**: Stop loss que acompanha movimentos favoráveis

Risk Limits

- **Maximum Position Size**: Limite por posição individual
- **Portfolio Risk**: Limite de exposição total
- **Daily Loss Limit**: Limite de perda diária
- **Maximum Drawdown**: Limite de drawdown total
- **Correlation Limits**: Limite de correlação entre posições

Monitoring

- **Real-time VaR**: Cálculo contínuo do Value at Risk
- **Stress Testing**: Simulação de cenários extremos
- **Risk Attribution**: Decomposição do risco por fonte
- **Alert System**: Notificações automáticas de violações

Configuração de Risco

```yaml
risk_management:
  position_sizing:
    method: "volatility_targeting"
    target_volatility: 0.15
    max_position_size: 0.10

  stop_loss:
    method: "atr_based"
    atr_multiplier: 2.0
    min_stop_loss: 0.02

  limits:
    max_daily_loss: 0.05
    max_drawdown: 0.15
    max_portfolio_risk: 0.20

  monitoring:
    var_confidence: 0.05
    stress_scenarios: ["2008_crisis", "covid_crash", "flash_crash"]
```

📁 Estrutura do Projeto

```
btc_perp_trader/
├── src/                          # Código fonte principal
│   ├── collectors/               # Coletores de dados
│   │   ├── __init__.py
│   │   ├── binance_ws.py         # WebSocket Binance
│   │   ├── orderbook_ws.py       # Order book collector
│   │   ├── onchain.py            # Dados on-chain
│   │   └── data_manager.py       # Gerenciador de dados
│   │
│   ├── features/                 # Engenharia de features
│   │   ├── __init__.py
│   │   └── feature_engineering.py
│   │
│   ├── models/                   # Modelos de ML/DL
│   │   ├── __init__.py
│   │   ├── base_model.py         # Classe base
│   │   ├── xgboost_model.py      # Modelo XGBoost
│   │   ├── lstm_model.py         # Modelo LSTM
│   │   └── ensemble_model.py     # Sistema de ensemble
│   │
│   ├── backtest/                # Sistema de backtesting
│   │   ├── __init__.py
│   │   └── backtester.py
│   │
│   ├── risk/                     # Gestão de risco
│   │   ├── __init__.py
│   │   └── risk_manager.py
│   │
│   └── execution/               # Motor de execução
│       ├── __init__.py
│       └── execution_engine.py
│
├── trading_api/                 # API Flask
│   ├── src/
│   │   ├── main.py              # Aplicação principal
│   │   ├── routes/
│   │   │   └── trading.py       # Rotas da API
│   │   └── static/
│   │       └── index.html       # Interface web
│   └── venv/                    # Ambiente virtual
│
├── config/                      # Configurações
│   ├── config.yaml             # Configurações principais
│   └── .env.example            # Exemplo de variáveis de ambiente
│
├── data/                       # Dados
│   ├── raw/                    # Dados brutos
│   ├── processed/              # Dados processados
│   └── models/                 # Modelos salvos
│
├── logs/                       # Logs do sistema
├── tests/                      # Testes
│   └── test_integration.py     # Testes de integração
│
├── docs/                       # Documentação
├── Dockerfile                  # Container Docker
├── docker-compose.yml          # Orquestração Docker
```

```
├── requirements.txt           # Dependências Python
├── pyproject.toml             # Configuração do projeto
└── README.md                  # Este arquivo
```

📦 Dependências

Core Dependencies

- **pandas**: Manipulação de dados
- **numpy**: Computação numérica
- **scikit-learn**: Machine learning
- **xgboost**: Gradient boosting
- **lightgbm**: Gradient boosting alternativo

Deep Learning

- **torch**: PyTorch para redes neurais
- **torchvision**: Visão computacional
- **torchaudio**: Processamento de áudio

Data Processing

- **polars**: DataFrames de alta performance
- **pyarrow**: Formato Parquet
- **duckdb**: Banco de dados analítico

Technical Analysis

- **pandas-ta**: Indicadores técnicos
- **ta**: Biblioteca de análise técnica
- **numba**: Aceleração de código

Networking

- **aiohttp**: Cliente HTTP assíncrono
- **cryptofeed**: Feeds de dados crypto

Web Framework

- **flask**: Framework web
- **flask-cors**: CORS para Flask
- **flask-sqlalchemy**: ORM para Flask

Visualization

- **matplotlib**: Gráficos
- **seaborn**: Visualização estatística

Utilities

- **pyyaml**: Configuração YAML
- **python-dateutil**: Manipulação de datas
- **requests**: Cliente HTTP

🤝 Contribuição

Como Contribuir

1. **Fork** o repositório
2. **Clone** seu fork localmente
3. **Crie** uma branch para sua feature (`git checkout -b feature/nova-feature`)
4. **Implemente** suas mudanças
5. **Teste** suas mudanças
6. **Commit** suas mudanças (`git commit -am 'Adiciona nova feature'`)

7. **Push** para a branch (`git push origin feature/nova-feature`)
8. **Abra** um Pull Request

Diretrizes

- Siga o estilo de código existente
- Adicione testes para novas funcionalidades
- Atualize a documentação quando necessário
- Use mensagens de commit descritivas
- Mantenha PRs focados e pequenos

Áreas de Contribuição

- **Novos Modelos**: Implementação de novos algoritmos de ML/DL
- **Features**: Novas features para engenharia de dados
- **Exchanges**: Suporte a novas exchanges
- **Otimizações**: Melhorias de performance
- **Testes**: Expansão da cobertura de testes
- **Documentação**: Melhorias na documentação
- **UI/UX**: Melhorias na interface web

📄 Licença

Este projeto está licenciado sob a Licença MIT - veja o arquivo LICENSE para detalhes.

Disclaimer

AVISO IMPORTANTE: Este software é fornecido apenas para fins educacionais e de pesquisa. O trading de criptomoedas envolve riscos significativos e pode resultar em perdas substanciais. Os desenvolvedores não se responsabilizam por quaisquer perdas financeiras decorrentes do uso deste software.

Use por sua própria conta e risco.

📞 Suporte

Documentação

- **README**: Este arquivo
- **API Docs**: Disponível em `/api/docs`
- **Code Docs**: Documentação inline no código

Comunidade

- **Issues**: Reporte bugs e solicite features
- **Discussions**: Discussões gerais sobre o projeto
- **Wiki**: Documentação adicional e tutoriais

Contato

- **Email**: [seu-email@exemplo.com]
- **LinkedIn**: [seu-perfil-linkedin]
- **Twitter**: [@seu-twitter]

🙏 Agradecimentos

Inspirações e Referências

- **"Advances in Financial Machine Learning"** por Marcos López de Prado
- **Comunidade Crypto Twitter** por insights e discussões
- **Papers Acadêmicos** sobre trading algorítmico e ML
- **Bibliotecas Open Source** que tornaram este projeto possível

Tecnologias Utilizadas

- **Python Ecosystem**: pandas, numpy, scikit-learn

- **PyTorch**: Framework de deep learning
- **XGBoost**: Gradient boosting de alta performance
- **Flask**: Framework web minimalista
- **Docker**: Containerização e deployment

Desenvolvido com ❤️ para a comunidade de trading algorítmico

Última atualização: Janeiro 2025