Kairos Alloy
PRD - MVP

Sistema de backtesting e execução em Rust com integração a agente DRL e sentimento em Python

Versão: v0.2 (rascunho)
Data: 26/01/2026
Autor: Marcus Vinicius Santos da Silva
Orientação: Maria Jose Pereira Dantas

Mudanças na v0.2: renomeação para Kairos Alloy; inclusão de mitigação para look-ahead bias em sentimento (sentiment_lag); recomendação de HTTP keep-alive e roadmap para gRPC/ZeroMQ; detalhamento de normalização no lado Python; divisão do milestone M2 em M2a/M2b.

1. Contexto e motivação

O projeto de IC investiga agentes de Deep Reinforcement Learning (DRL) para tomada de decisão no mercado de criptomoedas, incorporando indicadores derivados de um ensemble de LSTM/GRU e sinais de sentimento extraídos de redes sociais e tendências de busca. O plano de trabalho estabelece como resultado esperado a entrega de um software funcional e validado, capaz de operar em ambientes simulados e reais, com avaliação por métricas como Lucro líquido, Sharpe Ratio e Maximum Drawdown.

Este PRD define o MVP do sistema Kairos Alloy, implementado em Rust, responsável pela execução (simulada e real/paper), pelo motor de backtesting e pela integração com o agente em Python (treino/inferência).

2. Problema a resolver

No contexto do estudo, é necessário um sistema que permita: (i) executar backtests reprodutíveis com dados históricos; (ii) integrar sinais quantitativos e de sentimento; (iii) conectar um agente DRL treinado em Python a um motor de execução robusto; (iv) avaliar desempenho com métricas financeiras padronizadas e logs auditáveis.

3. Objetivos do produto

3.1 Objetivo do MVP

Entregar um aplicativo de linha de comando (CLI) em Rust que rode backtests e paper trading (simulado em tempo real), chamando um agente externo (serviço Python) para decidir ações, registrando trades e calculando métricas de desempenho.

3.2 Objetivos de pesquisa suportados

- Fornecer infraestrutura para testar e validar agentes DRL com variáveis de preço, indicadores técnicos, ensemble LSTM/GRU e sentimento.
- Permitir comparação contra estratégias baseline (ex.: buy-and-hold, média móvel) e contra benchmarks definidos no experimento.
- Gerar relatórios e artefatos para análise (curva de patrimônio, lista de operações, métricas).

4. Público-alvo e stakeholders

- Usuário primário: estudante (pesquisador) executando experimentos e validações.
- Stakeholder acadêmico: orientador(a) e grupo de pesquisa.
- Usuário técnico: futuros contribuidores do laboratório (reuso do motor).

5. Escopo do MVP

5.1 Em escopo (MVP)

- Importação de dados históricos de preço (OHLCV) a partir de arquivos (CSV como padrão do MVP).

- Importação de dados de sentimento (séries temporais agregadas por janela) a partir de arquivos (CSV/JSON).
- Sincronização temporal entre preço e sentimento (resampling por timeframe configurável).
- Motor de backtesting baseado em barras (bar-based), com execução de ordens a mercado e custos (fee, slippage configuráveis).
- Gestão de portfólio (caixa, posição, PnL realizado/não-realizado).
- Regras de risco do MVP (limite de exposição, limite de drawdown, posição máxima).
- Integração com agente via API local (HTTP/JSON) com esquema de mensagens versionado.
- Estratégias baseline embutidas em Rust para sanity check (buy-and-hold e média móvel).
- Cálculo de métricas: Lucro líquido, Sharpe Ratio, Maximum Drawdown; exportação de curva de patrimônio e lista de trades.
- Relatório final em JSON/CSV (e opcionalmente HTML simples) + logs estruturados.

5.2 Fora de escopo (para depois do MVP)

- Execução com dinheiro real em corretora/exchange; integração direta com MetaTrader 5 em tempo real.
- Treino de modelos (DRL, LSTM/GRU, NLP) dentro do Rust.
- Suporte completo a múltiplos ativos e rebalanceamento de portfólio multiativo.
- Derivativos (futuros, margem, alavancagem), ordens limit/stop complexas.
- Interface gráfica completa (web/desktop).

6. Requisitos funcionais

6.1 CLI e configuração

- O sistema deve expor um comando principal (kairos-alloy) com subcomandos: backtest, paper, validate, report.
- O sistema deve aceitar um arquivo de configuração (TOML) contendo paths de dados, timeframe, custos, limites de risco e parâmetros do agente.
- O sistema deve permitir seleção de estratégia: baseline (interna) ou agente externo (API).

6.2 Ingestão de dados

- Carregar OHLCV com timestamp UTC e campos: open, high, low, close, volume.
- Carregar série de sentimento com timestamp UTC e campos numéricos (ex.: score, volume_mencoes).
- Validar consistência: timestamps ordenados, duplicatas, gaps; gerar warnings e relatório de qualidade de dados.
- Aplicar regra de disponibilidade temporal para sentimento para evitar look-ahead bias: ao construir a observation no tempo t, usar apenas valores de sentimento com timestamp <= (t - sentiment_lag).

6.3 Pipeline de features (MVP)

- Gerar features básicas do preço: retorno log/percentual, volatilidade por janela, médias móveis simples (SMA), RSI (opcional).
- Acoplar features de sentimento já agregadas por janela (sem rodar NLP dentro do Rust).
- Gerar um vetor de observação (observation) padronizado para o agente externo.

- Normalização/escala: o Rust deve enviar features em formato cru ou em variações percentuais/retornos; o agente em Python aplica normalização usando os parâmetros salvos no treino (ex.: scaler).

6.4 Motor de backtesting e execução

- Simular execução a mercado na abertura da próxima barra (padrão), usando a action decidida no passo anterior; fees e slippage configuráveis.
- Suportar posição long-only no MVP (compra e venda do ativo), com fração do capital ou quantidade fixa.
- Registrar cada trade (timestamp, side, qty, price, fee, slippage, motivo/strategy_id).
- Calcular PnL e curva de patrimônio (equity curve) a cada barra.

6.5 Integração com agente (Python)

O MVP utilizará um serviço Python para inferência. O Rust envia observation e recebe action.

Esquema mínimo de mensagens (JSON)

- POST /v1/act: request contém run_id, timestamp, symbol, timeframe, observation[], portfolio_state{}; response contém action_type, size, confidence, model_version, latency_ms.
- Versionamento por campo api_version e feature_version.
- Timeout configurável; fallback: HOLD e redução de risco em caso de falha.
- O cliente HTTP do Rust deve suportar conexão persistente (keep-alive) no MVP. Evoluções pós-MVP podem incluir gRPC ou ZeroMQ para reduzir overhead em backtests de alta velocidade.

6.6 Métricas e relatórios

- Lucro líquido: diferença entre capital final e capital inicial (considerando fees).
- Sharpe Ratio: baseado em retornos do portfólio; janela e taxa livre de risco configuráveis (padrão 0).
- Maximum Drawdown: maior queda pico-vale da curva de patrimônio.
- Exportar: trades.csv, equity.csv, summary.json, config_snapshot.toml, logs.jsonl.

7. Requisitos não-funcionais

- Reprodutibilidade: dado o mesmo dataset + config + respostas do agente, o backtest deve ser determinístico.
- Performance: processar pelo menos 500k barras (aprox. 1 ano de 1-min) em menos de 30s em modo release, em máquina padrão de notebook.
- Confiabilidade: falhas no agente externo não podem derrubar o processo; deve haver retry limitado e fallback.
- Observabilidade: logs estruturados (JSON) e métricas internas (tempo por etapa, latência do agente).
- Portabilidade: suporte a Linux e Windows.

8. Arquitetura proposta (alto nível)

Arquitetura modular em Rust, com separação clara entre domínio, infraestrutura e integração.

- core: tipos de domínio (Bar, Order, Trade, Portfolio, Metrics).
- data: loaders/parsers (CSV) e validação.
- features: cálculo de indicadores e montagem do observation.

- engine: loop de backtest, execução simulada, geração de reward (opcional).
- risk: validação de ordens e limites.
- agents: estratégias internas + client HTTP para agente externo.
- report: exportação de artefatos (CSV/JSON/HTML).
- cli: interface de linha de comando e config.

9. Critérios de sucesso do MVP

- Rodar um backtest completo (ex.: BTCUSD) a partir de um dataset histórico exportado, gerando artefatos de resultado.
- Rodar o mesmo backtest com estratégia baseline e com agente externo, permitindo comparação direta.
- Gerar métricas (Lucro líquido, Sharpe, Max Drawdown) e logs para auditoria.
- Executar paper trading com feed simulado em tempo real (replay) por no mínimo 1 hora sem falhas.

10. Roadmap sugerido (até o MVP)

Cronograma alinhado às fases de implementação/validação do plano de trabalho (implementação e testes até 06/2026; validação em ambiente real 07-08/2026).

Milestone	Entrega	Conteúdo	Critério de aceite
M0	Setup	Repo Rust + CLI base + config TOML + logging	Compila e roda 'kairos-alloy --help'
M1	Data	Loader OHLCV + sentimento + validação	Carrega dataset e imprime resumo
M2a	Engine	Loop do backtest: iteração por barras + sincronização temporal + cálculo de indicadores/features	Itera sobre dataset e produz observation por barra (sem ordens)
M2b	Orders	Execução simulada + ordens (buy/sell/hold) + PnL + trades	Gera trades.csv e equity.csv com baseline
M3	Metrics	Lucro, Sharpe, Max Drawdown + summary.json	Métricas batem com caso de teste
M4	Agents	Baselines + client HTTP (keep-alive) para agente Python	Backtest roda com agente dummy e registra latência
M5	Paper	Replay em tempo real + timeouts/fallback + logs	1h de execução

	estruturados	contínua em replay

11. Riscos e mitigação

- Alinhamento temporal dos dados (preço vs sentimento) e risco de look-ahead bias: mitigar com regras de disponibilidade, atraso configurável (sentiment_lag) e relatórios de sincronização.
- Diferença entre ambiente de treino (Python) e execução (Rust): mitigar com esquema de observation versionado e testes de equivalência.
- Latência/falhas do agente externo: mitigar com timeout, retry limitado e fallback (HOLD).
- Escopo grande para IC: mitigar com MVP estrito (long-only, 1 ativo, bar-based).

12. Questões em aberto (para fechar na primeira sprint)

- Timeframe alvo do MVP (1-min, 5-min, 1h) para o backtest principal.
- Formato do dataset exportado (CSV vs Parquet) e campos disponíveis do MetaTrader 5.
- Conjunto mínimo de features para observation e como representar sentimento (score único vs múltiplos sinais).
- Política de slippage e fee (fixo, percentual, spread).
- Definição final do vetor observation: campos, ordem, tipos e responsabilidade de normalização (Rust vs Python).
- Decisão de protocolo para o agente em produção: HTTP keep-alive (MVP) e critérios para migrar para gRPC/ZeroMQ (pós-MVP).
- Definição de sentiment_lag (segundos/minutos) e regra exata de alinhamento sentimento->barra.

13. Referência interna

Baseado no Plano de Trabalho da IC 'Agentes de Deep Reinforcement Learning no Mercado de Criptomoedas' (seções: Objetivo Geral/Específicos, Métodos, Resultados Esperados e Cronograma).