AI Trading Bot Architecture

Reinforcement Learning System for Algorithmic Trading

AI Trading Bot Development Team

17 июля 2025 г.

Аннотация

Данный документ описывает архитектуру интеллектуального торгового бота, основанного на технологиях машинного обучения с подкреплением (Reinforcement Learning). Система предназначена для автоматизированной торговли на финансовых рынках с использованием анализа настроений, математических стратегий и альтернативных источников данных. Цель проекта - создание конкурентоспособной системы, способной соперничать с хедж-фондами по производительности и эффективности.

Содержание

1 Введение

Современные финансовые рынки характеризуются высокой волатильностью, быстрым изменением условий и огромными объемами данных. Традиционные подходы к торговле становятся менее эффективными, что создает потребность в интеллектуальных автоматизированных системах.

Предлагаемая архитектура представляет собой многоуровневую систему, способную:

- Обрабатывать различные источники данных в реальном времени
- Принимать торговые решения на основе RL-алгоритмов
- Управлять рисками и оптимизировать портфель
- Выполнять множественные сделки одновременно
- Непрерывно обучаться и адаптироваться к изменяющимся условиям

2 Архитектура системы

2.1 Обзор архитектуры

Система состоит из шести основных слоев:

1. **Data Collection & Processing Layer** - Сбор и предобработка данных
2. **AI/ML Core Layer** - Ядро машинного обучения
3. **Strategy & Risk Management Layer** - Управление стратегиями и рисками
4. **Execution Layer** - Исполнение сделок
5. **Testing & Validation Layer** - Тестирование и валидация
6. **Infrastructure Layer** - Инфраструктура

2.2 Слой сбора и обработки данных

2.2.1 Анализ новостей и настроений

Система осуществляет сбор новостей из различных источников и анализирует их тональность для принятия торговых решений.

Компоненты:

- News API Integration (NewsAPI, Alpha Vantage)
- Sentiment Analysis Engine (VADER, TextBlob, BERT)
- Real-time News Processing Pipeline
- Sentiment Score Calculation

Математическая модель анализа настроений:

$$S(t) = \sum_{i=1}^{n} w_i \cdot \text{sentiment}_i(t) \tag{1}$$

где $S(t)$ - совокупный индекс настроений в момент времени t, w_i - весовые коэффициенты источников, $\text{sentiment}_i(t)$ - оценка настроения i-го источника.

2.2.2 Рыночные данные

Потоковое получение рыночных данных в реальном времени через Capital.com API.

Технические индикаторы:

$$\text{SMA}(t) = \frac{1}{n} \sum_{i=0}^{n-1} P(t-i) \tag{2}$$

$$\text{EMA}(t) = \alpha \cdot P(t) + (1-\alpha) \cdot \text{EMA}(t-1) \tag{3}$$

$$\text{RSI}(t) = 100 - \frac{100}{1 + RS(t)} \tag{4}$$

где $P(t)$ - цена в момент времени t, α - коэффициент сглаживания, $RS(t)$ - отношение средних прибылей к убыткам.

2.2.3 Альтернативные источники данных

- Google Maps API для анализа экономической активности
- Social Media Sentiment (Twitter, Reddit)
- Макроэкономические индикаторы (FRED API)
- Satellite Data для commodity trading

2.3 Ядро машинного обучения

2.3.1 Reinforcement Learning Agent

Основной RL-агент использует Deep Q-Network (DQN) для принятия торговых решений:

Функция вознаграждения:

$$R(t) = \alpha \cdot \text{PnL}(t) + \beta \cdot \text{Sharpe}(t) - \gamma \cdot \text{Drawdown}(t) - \delta \cdot \text{Risk}(t) \tag{5}$$

где $\alpha, \beta, \gamma, \delta$ - весовые коэффициенты для балансировки различных метрик.

2.3.2 Генерация признаков

- Технические индикаторы (RSI, MACD, Bollinger Bands)
- Sentiment scores из новостных данных
- Макроэкономические показатели

Algorithm 1 DQN Trading Algorithm

```
Initialize replay buffer D
Initialize action-value function Q with random weights
Initialize target action-value function Q̂
for episode = 1 to M do
    Initialize state s_1
    for t = 1 to T do
        Select action a_t = ε-greedy(Q(s_t, ·))
        Execute action a_t and observe reward r_t and next state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in D
        Sample random minibatch from D
        Update Q using gradient descent
        if t mod C == 0 then
            Update target network: Q̂ ← Q
        end if
    end for
end for
```

- Volatility measures (VIX, GARCH)
- Cross-asset correlations

Feature Vector:

$$\mathbf{x}(t) = [P(t), V(t), S(t), I_1(t), I_2(t), \ldots, I_n(t)]^T \tag{6}$$

где $P(t)$ - цена, $V(t)$ - объем, $S(t)$ - sentiment score, $I_i(t)$ - технические индикаторы.

2.4 Управление стратегиями и рисками

2.4.1 Стратегический оркестратор

Система поддерживает множественные торговые стратегии:

- **Momentum Strategy:** Signal = $\text{sign}(\text{returns}_{t-k:t})$
- **Mean Reversion:** Signal = $-\text{sign}(P(t) - \text{SMA}(t))$
- **Multi-Exchange Arbitrage:** Signal = $\text{sign}(\text{spread}(t))$
- **Pairs Trading:** Signal = $\text{sign}(\text{zscore}(\text{spread}(t)))$

Детали арбитражной стратегии:
КРИТИЧЕСКИ ВАЖНО: Арбитражный модуль должен интегрироваться со **всеми доступными биржами** для обеспечения максимального покрытия рынка и выявления ценовых расхождений.

Требования к арбитражной системе:

- **Количество бирж:** Минимум 15-20 крупнейших бирж
- **Количество активов:** Минимум **300 торговых инструментов** для анализа

- **Частота обновления:** Real-time данные с латентностью < 100ms
- **Минимальный спред:** Только сделки с спредом $> 0.1\%$ (после комиссий)

Математическая модель арбитража:

$$\text{Arbitrage\_Opportunity}_{i,j} = \frac{P_i(t) - P_j(t)}{P_j(t)} - (\text{fee}_i + \text{fee}_j) - \text{slippage} \tag{7}$$

где $P_i(t)$ и $P_j(t)$ - цены актива на биржах i и j соответственно.

Приоритизация арбитражных возможностей:

$$\text{Priority\_Score} = \frac{\text{Expected\_Profit} \times \text{Liquidity\_Score}}{\text{Execution\_Time} \times \text{Risk\_Factor}} \tag{8}$$

Список целевых бирж для интеграции:

1. **Tier 1:** Binance, Coinbase Pro, Kraken, Bitfinex
2. **Tier 2:** OKX, Huobi, KuCoin, Gate.io, Bybit
3. **Tier 3:** Bitget, MEXC, Crypto.com, Gemini
4. **DEX:** Uniswap, SushiSwap, PancakeSwap (для DeFi арбитража)

Категории активов для мониторинга (300+ инструментов):

- **Major Cryptocurrencies (50):** BTC, ETH, BNB, ADA, SOL, MATIC, DOT, AVAX и др.
- **Altcoins (150):** Средней и малой капитализации
- **Forex Pairs (50):** EUR/USD, GBP/USD, USD/JPY, AUD/USD и др.
- **Commodities (30):** Gold, Silver, Oil, Natural Gas и др.
- **Indices (20):** S&P 500, NASDAQ, DAX, FTSE и др.

2.4.2 Управление рисками

Value at Risk (VaR):

$$\text{VaR}_\alpha = -\text{quantile}(\text{returns}, \alpha) \tag{9}$$

Position Sizing (Kelly Criterion):

$$f = \frac{bp - q}{b} \tag{10}$$

где b - odds, p - вероятность выигрыша, q - вероятность проигрыша.

Maximum Drawdown:

$$\text{MDD} = \max_{t \in [0,T]} \left[\max_{s \in [0,t]} X(s) - X(t) \right] \tag{11}$$

2.5 Слой исполнения сделок

2.5.1 Интеграция с Capital.com API

Для работы с русскими комментариями в коде, я вынес их отдельно:

Система арбитража между биржами:

```python
import asyncio
import aiohttp
from dataclasses import dataclass
from typing import Dict, List, Optional
import numpy as np

@dataclass
class ArbitrageOpportunity:
    buy_exchange: str
    sell_exchange: str
    asset: str
    buy_price: float
    sell_price: float
    profit_percentage: float
    volume_available: float
    execution_time_estimate: float

class MultiExchangeArbitrageEngine:
    def __init__(self):
        self.exchanges = {
            'binance': BinanceAPI(),
            'coinbase': CoinbaseAPI(),
            'kraken': KrakenAPI(),
            'bitfinex': BitfinexAPI(),
            'okx': OKXAPI(),
            'huobi': HuobiAPI(),
            'kucoin': KucoinAPI(),
            'gate_io': GateIoAPI(),
            'bybit': BybitAPI(),
            'capital_com': CapitalComAPI()
        }

        # CRITICAL: Minimum 300 assets for analysis
        self.target_assets = self.load_target_assets()
        self.min_profit_threshold = 0.001  # 0.1% minimum profit

    async def scan_arbitrage_opportunities(self):
        """Scanning arbitrage opportunities across all exchanges"""
        opportunities = []

        # Fetch prices from all exchanges simultaneously
        price_data = await self.fetch_all_prices()

        for asset in self.target_assets:
            if asset not in price_data:
                continue

            exchanges_with_asset = price_data[asset]

            # Find maximum spread between exchanges
            max_opportunity = self.find_max_spread(exchanges_with_asset,
    asset)
```

```
52
53          if max_opportunity and max_opportunity.profit_percentage >
    self.min_profit_threshold:
54              opportunities.append(max_opportunity)
55
56      # Sort by profitability
57      return sorted(opportunities, key=lambda x: x.profit_percentage,
    reverse=True)
58
59  def find_max_spread(self, exchanges_data: Dict, asset: str):
60      """Find maximum spread for asset"""
61      exchanges = list(exchanges_data.keys())
62      max_profit = 0
63      best_opportunity = None
64
65      for buy_exchange in exchanges:
66          for sell_exchange in exchanges:
67              if buy_exchange == sell_exchange:
68                  continue
69
70              buy_price = exchanges_data[buy_exchange]['ask']
71              sell_price = exchanges_data[sell_exchange]['bid']
72
73              # Account for fees and slippage
74              total_fees = self.get_trading_fees(buy_exchange) + self.
    get_trading_fees(sell_exchange)
75              estimated_slippage = self.estimate_slippage(asset,
    buy_exchange, sell_exchange)
76
77              net_profit = (sell_price - buy_price) / buy_price -
    total_fees - estimated_slippage
78
79              if net_profit > max_profit:
80                  max_profit = net_profit
81                  best_opportunity = ArbitrageOpportunity(
82                      buy_exchange=buy_exchange,
83                      sell_exchange=sell_exchange,
84                      asset=asset,
85                      buy_price=buy_price,
86                      sell_price=sell_price,
87                      profit_percentage=net_profit,
88                      volume_available=min(
89                          exchanges_data[buy_exchange]['volume'],
90                          exchanges_data[sell_exchange]['volume']
91                      ),
92                      execution_time_estimate=self.
    estimate_execution_time(buy_exchange, sell_exchange)
93                  )
94
95      return best_opportunity
96
97  async def execute_arbitrage(self, opportunity, trade_size):
98      """Execute arbitrage trade"""
99      try:
100         # Simultaneous buy and sell execution
101         buy_task = asyncio.create_task(
102             self.exchanges[opportunity.buy_exchange].buy(
103                 opportunity.asset, trade_size, opportunity.buy_price
```

```
                )
            )

            sell_task = asyncio.create_task(
                self.exchanges[opportunity.sell_exchange].sell(
                    opportunity.asset, trade_size, opportunity.
    sell_price
                )
            )

            # Wait for both trades to complete
            buy_result, sell_result = await asyncio.gather(buy_task,
    sell_task)

            return {
                'success': True,
                'buy_result': buy_result,
                'sell_result': sell_result,
                'actual_profit': self.calculate_actual_profit(buy_result
    , sell_result)
            }

        except Exception as e:
            return {'success': False, 'error': str(e)}

    def load_target_assets(self):
        """Load target assets (300+ instruments)"""
        return [
            # Major Cryptocurrencies (50)
            'BTC/USDT', 'ETH/USDT', 'BNB/USDT', 'ADA/USDT', 'SOL/USDT',
            'MATIC/USDT', 'DOT/USDT', 'AVAX/USDT', 'LINK/USDT', 'UNI/
    USDT',
            # ... (remaining 40 major crypto)

            # Altcoins (150)
            'ALGO/USDT', 'VET/USDT', 'THETA/USDT', 'FIL/USDT', 'EOS/USDT
    ',
            # ... (remaining 145 altcoins)

            # Forex Pairs (50) - via Capital.com
            'EUR/USD', 'GBP/USD', 'USD/JPY', 'AUD/USD', 'USD/CHF',
            'USD/CAD', 'NZD/USD', 'EUR/GBP', 'EUR/JPY', 'GBP/JPY',
            # ... (remaining 40 forex pairs)

            # Commodities (30)
            'GOLD', 'SILVER', 'OIL', 'NATGAS', 'COPPER', 'PLATINUM',
            # ... (remaining 24 commodities)

            # Indices (20)
            'SPX500', 'NAS100', 'GER30', 'UK100', 'FRA40', 'JPN225',
            # ... (remaining 14 indices)
        ]

class CapitalComAPI:
    def __init__(self, api_key, api_secret):
        self.api_key = api_key
        self.api_secret = api_secret
        self.base_url = "https://api.capital.com"
```

```
157
158     async def execute_arbitrage_trade(self, opportunity, size):
159         """Execute arbitrage trade via Capital.com"""
160         # If one of exchanges is Capital.com, use their API directly
161         if opportunity.buy_exchange == 'capital_com':
162             return await self.open_position(
163                 opportunity.asset, 'BUY', size,
164                 limit_price=opportunity.buy_price
165             )
166         elif opportunity.sell_exchange == 'capital_com':
167             return await self.open_position(
168                 opportunity.asset, 'SELL', size,
169                 limit_price=opportunity.sell_price
170             )
171
172         # Otherwise use Capital.com as hedge venue
173         return await self.hedge_position(opportunity, size)
```

Листинг 1: Multi-Exchange Arbitrage System

2.5.2 Система управления ордерами

- Market Orders для немедленного исполнения
- Limit Orders для контроля цены исполнения
- Stop Orders для управления рисками
- Trailing Stop для максимизации прибыли

2.6 Тестирование и валидация

2.6.1 Backtesting Engine

```
1  class BacktestEngine:
2      def __init__(self, strategy, data, initial_capital=100000):
3          self.strategy = strategy
4          self.data = data
5          self.initial_capital = initial_capital
6          self.portfolio = Portfolio(initial_capital)
7
8      def run_backtest(self, start_date, end_date):
9          """Run backtest for given period"""
10         results = []
11         for timestamp, market_data in self.data.iter_range(start_date,
       end_date):
12             signal = self.strategy.generate_signal(market_data)
13             trade = self.execute_trade(signal, market_data)
14             self.portfolio.update(trade)
15             results.append(self.portfolio.get_metrics())
16         return BacktestResults(results)
17
18     def calculate_metrics(self, results):
19         """Calculate performance metrics"""
20         returns = np.array([r['return'] for r in results])
21         sharpe_ratio = np.mean(returns) / np.std(returns) * np.sqrt(252)
22         max_drawdown = self.calculate_max_drawdown(results)
```

```
        win_rate = np.mean(returns > 0)

        return {
            'sharpe_ratio': sharpe_ratio,
            'max_drawdown': max_drawdown,
            'win_rate': win_rate,
            'total_return': (results[-1]['portfolio_value'] / self.
    initial_capital - 1) * 100
        }
```

Листинг 2: Backtesting Framework

2.6.2 Ключевые метрики производительности

Таблица 1: Целевые показатели производительности

Метрика	Целевое значение
Годовая доходность	15%
Коэффициент Шарпа	2.5
Максимальная просадка	-8%
Коэффициент Сортино	3.0
Бета	0.3
Процент прибыльных сделок	65%
Коэффициент Кальмара	1.9

2.7 Инфраструктурный слой

2.7.1 Облачная архитектура

- **Compute:** Kubernetes clusters на AWS/GCP
- **Database:** Supabase (PostgreSQL + Real-time + Auth + Storage)
- **Computer Vision:** Supervision для обработки визуальных данных
- **Local AI:** Ollama 3.2 для локальных LLM операций
- **Networking:** Load balancers, CDN для низкой латентности
- **Security:** WAF, DDoS protection, encryption at rest/in transit

2.7.2 Система мониторинга

```
class ArbitrageMonitor:
    def __init__(self):
        self.metrics = MetricsCollector()
        self.alerts = AlertManager()
        # CRITICAL: Monitor all 300+ assets and 15+ exchanges
        self.tracked_exchanges = 15  # Minimum exchange count
        self.tracked_assets = 300    # Minimum asset count

    def monitor_arbitrage_performance(self):
```

```
        """Monitor arbitrage system performance"""
        current_opportunities = self.scan_current_opportunities()

        # Alerts for critical metrics
        if len(current_opportunities) < 10:
            self.alerts.send_alert("LOW_ARBITRAGE_OPPORTUNITIES", len(current_opportunities))

        # Monitor latency across exchanges
        for exchange in self.exchanges:
            latency = self.measure_exchange_latency(exchange)
            if latency > 100:  # > 100ms critical for arbitrage
                self.alerts.send_alert("HIGH_EXCHANGE_LATENCY", exchange, latency)

        # Monitor asset availability
        available_assets = self.count_available_assets()
        if available_assets < self.tracked_assets:
            self.alerts.send_alert("ASSET_COVERAGE_LOW", available_assets)

    def collect_arbitrage_metrics(self):
        """Collect arbitrage system metrics"""
        return {
            'total_exchanges_connected': len(self.connected_exchanges()),
            'total_assets_monitored': self.count_monitored_assets(),
            'average_latency_ms': self.calculate_average_latency(),
            'opportunities_per_minute': self.count_opportunities_last_minute(),
            'successful_arbitrage_rate': self.calculate_success_rate(),
            'average_profit_per_trade': self.calculate_average_profit(),
            'total_volume_processed': self.get_total_volume(),
            'exchange_uptime_percentage': self.calculate_exchange_uptime()
        }

    def generate_arbitrage_report(self):
        """Generate arbitrage activity report"""
        return {
            'period': '24h',
            'total_opportunities_identified': self.count_opportunities_24h(),
            'opportunities_executed': self.count_executed_24h(),
            'total_profit_realized': self.calculate_total_profit_24h(),
            'best_performing_exchange_pair': self.get_best_exchange_pair(),
            'most_profitable_assets': self.get_top_profitable_assets(10),
            'average_execution_time': self.calculate_avg_execution_time(),
            'slippage_analysis': self.analyze_slippage_24h()
        }

# Configuration for arbitrage system
ARBITRAGE_CONFIG = {
    'MIN_EXCHANGES': 15,           # Minimum exchange count
    'MIN_ASSETS': 300,            # Minimum asset count
```

```
    'MAX_LATENCY_MS': 100,          # Maximum allowable latency
    'MIN_PROFIT_THRESHOLD': 0.001, # 0.1% minimum profit
    'MAX_SLIPPAGE': 0.0005,         # 0.05% maximum slippage
    'MIN_VOLUME_USD': 1000,         # Minimum volume for arbitrage
    'EXECUTION_TIMEOUT_SEC': 5,     # Trade execution timeout
    'PRICE_UPDATE_INTERVAL_MS': 50 # Price update interval
}
```

Листинг 3: Arbitrage Monitoring System

Инфраструктурные требования для арбитражной системы:

Таблица 2: Технические требования для Multi-Exchange Arbitrage

Компонент	**Требования**
Количество бирж	Минимум 15-20 (Tier 1: 4, Tier 2: 6, Tier 3: 5+)
Количество активов	**300+ инструментов**
Латентность	$<$ 100ms для выполнения арбитража
Пропускная способность	10,000+ price updates/second
Availability	99.9% uptime requirement
Memory	32GB+ RAM для real-time data
Storage	1TB+ SSD для historical data
Network	Dedicated lines к major exchanges
Geographical	Multi-region deployment (US, EU, Asia)

Критические метрики для арбитражной системы:

- **Opportunity Detection Rate:** $>$ 50 возможностей в час
- **Execution Success Rate:** $>$ 95% успешных сделок
- **Average Profit per Trade:** $>$ 0.15% после всех издержек
- **Exchange Coverage:** 100% uptime для Tier 1 бирж
- **Asset Coverage:** $>$ 90% от целевых 300 активов
- **Slippage Control:** $<$ 0.05% average slippage

3 Математические модели и алгоритмы

3.1 Оптимизация портфеля

3.1.1 Модель Марковица

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^T \Sigma \mathbf{w} \\ \text{subject to} \quad & \mathbf{w}^T \boldsymbol{\mu} = \mu_p \\ & \mathbf{w}^T \mathbf{1} = 1 \\ & \mathbf{w} \geq 0 \end{aligned} \tag{12}$$

где $\mathbf{w}$ - веса портфеля, Σ - ковариационная матрица, $\boldsymbol{\mu}$ - вектор ожидаемых доходностей.

3.1.2 Black-Litterman Model

$$\boldsymbol{\mu}_{BL} = \left[(\tau\Sigma)^{-1} + \mathbf{P}^T\Omega^{-1}\mathbf{P}\right]^{-1}\left[(\tau\Sigma)^{-1}\boldsymbol{\pi} + \mathbf{P}^T\Omega^{-1}\mathbf{Q}\right] \tag{13}$$

где $\boldsymbol{\pi}$ - равновесные доходности, $\mathbf{P}$ - матрица picking, $\mathbf{Q}$ - вектор прогнозов, Ω - матрица неопределенности.

3.2 Модели волатильности

3.2.1 GARCH(1,1)

$$\sigma_t^2 = \omega + \alpha\epsilon_{t-1}^2 + \beta\sigma_{t-1}^2 \tag{14}$$

3.2.2 Стохастическая волатильность (Heston)

$$dS_t = \mu S_t dt + \sqrt{v_t}S_t dW_t^S \tag{15}$$

$$dv_t = \kappa(\theta - v_t)dt + \sigma_v\sqrt{v_t}dW_t^v \tag{16}$$

4 Конкурентные преимущества

4.1 Скорость и автоматизация

- Латентность исполнения < 10 мс
- Торговля 24/7 без перерывов
- Отсутствие человеческих эмоций в принятии решений
- Параллельная обработка множественных стратегий

4.2 Искусственный интеллект

- Обработка $>$ 1M точек данных в секунду
- Адаптивное обучение на изменяющихся рынках
- Комплексный анализ паттернов
- Прогнозирование с использованием ансамблей моделей

4.3 Экономическая эффективность

- Операционные расходы на 90% ниже традиционных фондов
- Масштабируемость без пропорционального увеличения затрат
- Отсутствие комиссий управляющих
- Автоматизированная отчетность и комплаенс

5 Реализация и развертывание

5.1 Этапы разработки

1. **Фаза 1 (3 месяца):** Базовая архитектура, интеграция с Capital.com API
2. **Фаза 2 (4 месяца):** Разработка RL-агента, система управления рисками
3. **Фаза 3 (3 месяца):** Анализ настроений, альтернативные данные
4. **Фаза 4 (2 месяца):** Тестирование, оптимизация, развертывание

5.2 Технологический стек

Таблица 3: Технологический стек

Компонент	**Технология**
ML Framework	PyTorch, TensorFlow, Stable-Baselines3
Reinforcement Learning	Deep Q-Network (DQN), PPO, A3C
Database	Supabase (PostgreSQL + Real-time + Auth)
Computer Vision	Supervision, OpenCV, YOLO
Local LLM	Ollama 3.2 (Llama models)
Data Processing	Apache Spark, Pandas, NumPy
API	FastAPI, Flask, WebSocket
Container	Docker, Kubernetes
Cloud	AWS, GCP, Azure
Monitoring	Prometheus, Grafana, ELK Stack
CI/CD	GitLab CI, Jenkins, ArgoCD

6 Риски и ограничения

6.1 Технические риски

- Сбои в API Capital.com
- Латентность сетевых подключений
- Ошибки в алгоритмах RL
- Переобучение моделей

6.2 Рыночные риски

- Изменения рыночной структуры
- Регуляторные изменения
- Экстремальные рыночные события
- Ликвидность инструментов

6.3 Операционные риски

- Кибербезопасность
- Соответствие регуляторным требованиям
- Операционные сбои
- Управление данными

7 Заключение

Предложенная архитектура AI Trading Bot представляет собой комплексную систему, способную конкурировать с традиционными хедж-фондами за счет:

- Использования передовых технологий ИИ и Reinforcement Learning
- Интеграции с современными инструментами: Supabase, Supervision, Ollama 3.2
- Высокой скорости обработки данных и принятия решений
- Эффективного управления рисками и портфелем
- Непрерывного обучения и адаптации к рыночным условиям
- Значительного снижения операционных затрат
- Обработки визуальных данных для дополнительных торговых сигналов
- Локального анализа данных для повышения безопасности и скорости

Успешная реализация данной системы требует междисциплинарной команды экспертов в области машинного обучения, финансов, разработки ПО и инфраструктуры. Критически важным является соблюдение всех регуляторных требований и проведение тщательного тестирования перед запуском в продакшн.

8 Распределение обязанностей команды

Для эффективной реализации проекта команда из 3 человек должна быть организована следующим образом:

8.1 Разработчик 1: AI/ML Engineer

Основные обязанности:

- Разработка и обучение Reinforcement Learning моделей
- Интеграция с Ollama 3.2 для локальной обработки данных
- Создание системы анализа настроений
- Разработка алгоритмов feature engineering

- Оптимизация производительности ML pipeline
- Continuous learning и model retraining

Требуемые навыки:

- Глубокие знания ML/DL: PyTorch, TensorFlow
- Опыт с Reinforcement Learning (DQN, PPO, A3C)
- Знание NLP и sentiment analysis
- Опыт работы с LLM (Ollama, Llama models)
- Python, NumPy, Pandas, Scikit-learn
- Понимание финансовых рынков и торговых стратегий

8.2 Разработчик 2: Backend/Infrastructure Engineer

Основные обязанности:

- Интеграция с Capital.com API
- Настройка и оптимизация Supabase
- Разработка системы управления рисками
- Создание backtesting engine
- Настройка CI/CD pipeline
- Мониторинг и логирование системы
- Обеспечение безопасности и compliance

Требуемые навыки:

- Сильные навыки backend разработки (Python, FastAPI)
- Опыт с Supabase/PostgreSQL, real-time subscriptions
- Знание Docker, Kubernetes, облачных платформ
- REST API, WebSocket, асинхронное программирование
- DevOps практики, мониторинг (Prometheus, Grafana)
- Понимание финансовых API и торговых протоколов

8.3 Разработчик 3: Computer Vision/Data Engineer

Основные обязанности:

- Интеграция и настройка Supervision для computer vision
- Разработка системы обработки визуальных данных
- Создание data pipeline для альтернативных источников
- Анализ графиков и паттернов на изображениях
- Интеграция с Google Maps API и satellite data
- Оптимизация data ingestion и preprocessing

Требуемые навыки:

- Опыт с computer vision (OpenCV, YOLO, Supervision)
- Знание data engineering (Apache Spark, ETL)
- Работа с геоданными и APIs (Google Maps, satellite)
- Python, обработка изображений, паттерн recognition
- Понимание технического анализа в трейдинге
- Опыт с real-time data processing

A Приложение A: Примеры конфигураций

```
{
  "arbitrage_config": {
    "min_exchanges": 15,
    "min_assets": 300,
    "max_latency_ms": 100,
    "min_profit_threshold": 0.001,
    "max_slippage": 0.0005,
    "execution_timeout_sec": 5
  },
  "ml_config": {
    "model_type": "DQN",
    "learning_rate": 0.001,
    "batch_size": 32,
    "replay_buffer_size": 10000,
    "update_frequency": 100
  },
  "risk_config": {
    "max_position_size": 0.1,
    "max_drawdown": 0.08,
    "var_confidence": 0.95,
    "kelly_fraction": 0.25
  }
}
```

Листинг 4: Configuration Example

B Приложение B: API Спецификация

Таблица 4: REST API Endpoints

Endpoint	Method	Description
/api/v1/arbitrage/scan	GET	Scan arbitrage opportunities
/api/v1/arbitrage/execute	POST	Execute arbitrage trade
/api/v1/portfolio/status	GET	Get portfolio status
/api/v1/risk/metrics	GET	Get risk metrics
/api/v1/monitoring/health	GET	System health check
/api/v1/backtest/run	POST	Run backtest

C Приложение C: Развертывание

Команды для развертывания системы:

```
# Build Docker images
docker build -t ai-trading-bot:latest .

# Deploy to Kubernetes
kubectl apply -f k8s/

# Setup monitoring
helm install prometheus prometheus-community/kube-prometheus-stack

# Initialize database
python scripts/init_database.py

# Start trading bot
python main.py --config config/production.yaml
```

Листинг 5: Deployment Commands