### \*\*Полное Техническое Задание: Система скальпинга через Тинькофф API (Версия 10.4)\*\*

\*\*Интегрированы все модули, включая ModelTrainer\*\*

---

### \*\*1. Финансовые инструменты\*\*

#### \*\*1.1 Инструменты\*\*

| Группа | Количество | Критерии отбора |

|--------------|------------|-------------------------------|

| Мониторинг | 30 | Ликвидность >500 контрактов/день |

| Активные | 15 | Спред ≤0.05%, Волатильность >5% |

#### \*\*1.2 Динамический контроль\*\*

```python

def rotate\_instruments():

for instrument in active\_instruments:

if (spread > 0.1) or (liquidity < 0.7 \* avg\_liquidity) or (volatility < 0.05):

new\_instrument = select\_from\_monitored(instrument)

activate\_instrument(new\_instrument)

data\_engine.update\_active\_instruments([inst.ticker for inst in active\_instruments])

```

---

### \*\*2. Система сбора данных\*\*

#### \*\*2.1 Режим реального времени\*\*

- \*\*Стаканы\*\*: Глубина 20 уровней (конфигурируемая)

- \*\*Кеширование\*\*:

- Текущий стакан: TTLCache (60 сек, 50 инструментов)

- История стаканов: кольцевой буфер (30 снимков)

- \*\*API\*\*:

```python

current\_ob = adapter.get\_orderbook(figi, depth=20, offset=0)

prev\_ob = adapter.get\_orderbook(figi, depth=20, offset=-1) # 200 мс назад

```

- \*\*Автообновление\*\*: Подписка на стрим Tinkoff API

#### \*\*2.2 Исторические данные\*\*

```python

class DataSchema:

Candles: Parquet(columns=["open","high","low","close","volume"])

Orderbooks: SQLite(table="l2", compression="zstd")

Trades: CSV(timestamp=int, price=float, qty=int)

Features: Parquet(columns=["spread","mid\_price",...]) # 28+ фичей

```

#### \*\*2.3 Управление данными\*\*

- \*\*Контроль памяти\*\*:

- Очистка кешей при RSS >400 МБ

- Принудительный GC при RSS >700 МБ

- \*\*Адаптивная запись\*\*: Динамический интервал (1-5 сек)

- \*\*Резервное хранилище\*\*: JSONL/CSV при ошибках БД

- \*\*Рыночные индикаторы\*\* (обновление 1 Гц):

- Совокупная волатильность

- Индекс ликвидности

- Общий рыночный индекс

---

### \*\*3. Модуль машинного обучения (MLEngine)\*\*

#### \*\*3.1 Архитектура ансамбля\*\*

| Модель | Вес | Ресурсы | Формат | Особенности обработки |

|-------------------|------|---------------|----------|-------------------------------------|

| LSTM\_OB | 35% | GPU (CUDA 12+)| TFLite | Последовательности 10 стаканов |

| GBDT\_FEAT | 25% | CPU | TFLite | 19+ фичей с фиксированным порядком |

| Transformer\_T | 25% | CPU | TFLite | Многооконный (1с, 10с, 1м) |

| SVM\_VolCluster | 15% | CPU | TFLite | Кластеризация волатильности |

#### \*\*3.2 Генерация фичей\*\*

\*\*Расширенный набор (28+ фичей)\*\*:

- Базовые: `imbalance`, `microprice`, `spread`, `volatility`

- Производные:

```python

features['ofi'] = (imbalance\_t - imbalance\_{t-1}) / (imbalance\_{t-1} + 1e-8)

features['book\_gradient'] = Δimbalance / (imbalance + 1e-8)

```

- Адаптивные:

```python

if volatility > 0.15:

features['aggressive\_imbalance'] = imbalance \* 1.5

else:

features['conservative\_imbalance'] = imbalance \* 0.7

```

- Рыночные: `market\_volatility`, `market\_index`, `market\_liquidity`

#### \*\*3.3 Генерация сигналов\*\*

```mermaid

graph TD

A[Получение рыночных данных] --> B[Расширение фичей]

B --> C[Пакетный инференс]

C --> D{LSTM\_OB: последовательность 10 стаканов}

C --> E{GBDT\_FEAT: 19+ фичей}

C --> F{Transformer\_T: 3 временных окна}

C --> G{SVM\_VolCluster: 6 фичей волатильности}

D --> H[Адаптивное взвешивание]

E --> H

F --> H

G --> H

H --> I[Формирование сигнала]

```

\*\*Параметры сигнала\*\*:

```python

@dataclass

class MLSignal:

figi: str

direction: int # 1 (buy), -1 (sell)

confidence: float # 0.0-1.0

source: str = "ml\_engine"

```

#### \*\*3.4 Обнаружение дрейфа данных\*\*

\*\*Механизм\*\*:

1. \*\*Критерий Колмогорова-Смирнова\*\*:

```python

statistic, p\_value = ks\_2samp(current\_values, reference\_values)

```

2. \*\*Пороги срабатывания\*\*:

- `p\_value < 0.05` и `mean\_diff > 2σ` → калибровка

- `mean\_diff > 3σ` → экстренный алерт

3. \*\*Интеграция\*\*:

- Автоматическая передача метрик в RiskEngine

- Логирование в `drift\_metrics\_YYYYMMDD.jsonl`

#### \*\*3.5 Калибровка моделей\*\*

\*\*Условия запуска\*\*:

- ΔVIXMOEX > 15%

- Обнаружение сильного дрейфа данных (>2σ)

- Смена торговой сессии (09:30/18:45)

- Запрос от RiskEngine

\*\*Процесс\*\*:

```mermaid

sequenceDiagram

MLEngine->>ModelTrainer: Запрос калибровки

ModelTrainer->>MLEngine: Обновлённые модели

MLEngine->>DataEngine: Обновить референсные данные

MLEngine->>RiskEngine: Уведомление о перезагрузке

```

#### \*\*3.6 Fallback-механизм\*\*

\*\*Активация при\*\*:

- Ошибках предсказания

- Недостатке данных

- Частоте >10 срабатываний/час

\*\*Эвристика\*\*:

```python

if imbalance > 0.7 and volatility > 0.01:

direction = 1; confidence = 0.7

elif imbalance < -0.7 and volatility > 0.01:

direction = -1; confidence = 0.7

```

#### \*\*3.7 Обучение моделей (ModelTrainer)\*\*

\*\*Архитектура\*\*:

```python

class EfficientTransformer(tf.keras.layers.Layer):

"""Оптимизированный трансформер с синусоидальным кодированием"""

class ModelTrainer:

async def train\_all\_models():

"""Параллельное обучение 4 моделей ансамбля"""

```

\*\*Этапы обучения\*\*:

1. \*\*Сбор данных\*\*: 180 дней исторических данных

2. \*\*Балансировка классов\*\*: SMOTE

3. \*\*Обучение моделей\*\*:

- LSTM: Бидирекциональные слои + BatchNorm

- GBDT: GridSearchCV для гиперпараметров

- Transformer: EfficientTransformer с адаптивным квантованием

4. \*\*Квантование\*\*:

```python

if importance > 0.5: # INT8 для важных моделей

else: # FLOAT16 для остальных

```

5. \*\*Валидация\*\*: F1-score, accuracy, classification\_report

\*\*Условия запуска\*\*:

- Ежедневно в 03:00 UTC

- При изменении VIX >15%

- При обнаружении дрейфа данных (>2σ)

- По запросу RiskEngine

---

### \*\*4. Торговая логика\*\*

#### \*\*4.1 Алгоритм исполнения\*\*

```mermaid

flowchart TD

Signal[Сигнал ML] --> VolumeCheck{Размер позиции}

VolumeCheck -- >5 лотов --> VWAP[VWAP: агрессивность 80%]

VolumeCheck -- 1-5 лотов --> IOC[IOC Order]

VWAP --> RiskCheck[Проверка рисков]

IOC --> RiskCheck

RiskCheck -- Одобрено --> Execution[Исполнение]

```

#### \*\*4.2 Управление позицией\*\*

```python

spread = max(spread, 0.01) # Защита от нуля

stop\_loss = entry\_price \* (1 + (1.8 \* atr5 + 0.5 \* spread) \* direction)

```

---

### \*\*5. Управление рисками (RiskEngine)\*\*

#### \*\*5.1 Основные функции\*\*

- Валидация ордеров, динамическое управление параметрами

- Ротация инструментов, хеджирование позиций

- Управление кулдауном, прогнозирование ликвидности

- Обнаружение дрейфа, управление открытыми позициями

- Расчет размера позиции, обработка результатов сделок

#### \*\*5.2 Параметры риска\*\*

```python

MAX\_RISK\_PER\_INSTRUMENT = 0.05 # 5% капитала на инструмент

DAILY\_LOSS\_LIMIT = 0.02 # 2% дневного убытка

MAX\_EXPOSURE = 0.50 # 50% максимальной экспозиции

VIX\_THRESHOLD = 0.4 # Порог VIXMOEX для остановки

```

#### \*\*5.3 Механизмы безопасности\*\*

- \*\*Circuit Breaker\*\*:

| Условие | Действие |

|----------------------------------|-----------------------------------|

| 3 убытка подряд | Стоп 5 минут |

| VIXMOEX >40% | Полный выход из позиций |

| Задержка сети >100мс | Активация LTE |

- Автоматическое хеджирование, динамические лимиты

- Упрощенный режим при задержках >50 мс

#### \*\*5.4 Периодические задачи\*\*

- Ежеминутно: Ротация инструментов

- Каждые 10 сек: Управление позициями

- Каждые 30 мин: Проверка дрейфа данных

- Каждые 5 мин: Хеджирование портфеля

#### \*\*5.5 Критические функции\*\*

```python

class RiskEngine:

def calculate\_position\_size(): ...

def validate\_order(): ...

def process\_trade\_result(): ...

```

---

### \*\*6. Тестирование\*\*

#### \*\*6.1 Режимы работы\*\*

| Режим | Данные | Исполнение ордеров |

|---------------|------------------|--------------------------|

| Backtest | Исторические | Эмуляция |

| Forward-test | Реальный поток | Виртуальное исполнение |

| Production | Реальный поток | Tinkoff API |

#### \*\*6.2 Критерии перехода\*\*

```python

profit\_factor >= 1.8

sharpe\_ratio > 2.5

max\_drawdown <= 0.02

win\_rate >= 0.87

```

---

### \*\*7. Система мониторинга\*\*

#### \*\*7.1 Watchdog\*\*

```python

def health\_check():

if not heartbeat\_received("RiskEngine", 60):

restart\_module("RiskEngine")

```

#### \*\*7.2 Ресурсный мониторинг\*\*

- Очереди, задержки, память, CPU/GPU

- Действия при превышении порогов:

- CPU >90% → Отключение GUI

- GPU VRAM >90% → Перевод LSTM на CPU

#### \*\*7.3 Диагностика RiskEngine\*\*

```json

{

"exposure": 0.35,

"daily\_loss": 0.005,

"vix\_value": 0.28,

"simplified\_mode": false

}

```

---

### \*\*8. Безопасность\*\*

#### \*\*8.1 Шифрование ключей\*\*

```python

class KeyManager:

def decrypt(encrypted\_data: str) -> str: ...

```

#### \*\*8.2 Audit Log\*\*

- Формат: W3C Extended Log

- Теги: `ORDER\_EXECUTED`, `RISK\_EVENT`, `DATA\_DRIFT`, `CIRCUIT\_BREAKER`

---

### \*\*9. Пользовательский интерфейс\*\*

#### \*\*9.1 Визуализация\*\*

```python

Heatmap\_gradients = {

"bid": ["#003300", "#00cc00", "#ffffff"],

"ask": ["#330000", "#cc0000", "#ffffff"]

}

```

#### \*\*9.2 Уведомления\*\*

- Telegram: `ORDER\_FILLED`, `CIRCUIT\_BREAKER`

- Звуковые сигналы: 2500 Гц (высокий риск), 1500 Гц (низкая ликвидность)

---

### \*\*10. Производительность\*\*

#### \*\*10.1 Целевые задержки\*\*

```mermaid

pie

title Распределение задержки

"Сеть" : 15 мс

"Обработка данных" : 12 мс

"ML инференс" : 20 мс

"RiskEngine" : 8 мс

```

#### \*\*10.2 Оптимизация\*\*

- Векторизация с NumPy, асинхронная обработка

- Пакетная генерация сигналов, кеширование стаканов

#### \*\*10.3 Целевые показатели\*\*

| Параметр | Значение |

|------------------------------|-------------|

| Обработка ордера RiskEngine | < 8 мс |

| Прогноз ликвидности | < 15 мс |

| Хеджирование портфеля | < 300 мс |

---

### \*\*11. Модуль оркестрации\*\*

\*\*Основной цикл\*\*:

```python

class SystemOrchestrator:

def start\_system(self):

while True:

data = get\_realtime\_data()

signal = generate\_signal(data)

if validate\_order(signal):

execute\_order(signal)

```

\*\*Функции\*\*:

- Координация модулей

- Восстановление из чекпоинтов

- Ротация инструментов

- Мониторинг задержек

- Аварийное отключение

---

### \*\*12. Конфигурация\*\*

```yaml

System:

Mode: "forward-test"

Startup\_time: "09:30"

Checkpoint\_interval: 300

Risk:

Max\_exposure: 0.5

Daily\_loss\_limit: 0.02

ML:

Model\_weights:

LSTM\_OB: 0.35

GBDT\_FEAT: 0.25

Training\_window\_days: 180

```

---

### \*\*13. Файловая структура\*\*

```

Scalping\_system/

├── core/

│ ├── tinkoff\_api.py

│ ├── data\_engine.py

│ ├── ml\_engine.py

│ ├── risk\_engine.py

│ ├── orchestrator.py

│ ├── model\_trainer.py # Новый модуль

├── data/

│ ├── models/ # Обученные модели

│ ├── orderbooks/

│ └── features/

```

---

### \*\*14. Ключевые компоненты\*\*

#### \*\*14.1 DataEngine\*\*

- Агрегация стаканов, расчет 28+ фичей

- Управление активными инструментами

#### \*\*14.2 MLEngine\*\*

- Многооконный анализ, адаптивное взвешивание

- Автоматическая обработка дрейфа данных

#### \*\*14.3 RiskEngine\*\*

- Динамический расчет позиции

- Прогнозирование ликвидности, хеджирование

- Детекция дрейфа данных

#### \*\*14.4 ModelTrainer\*\*

```python

class ModelTrainer:

async def train\_all\_models(): # Параллельное обучение 4 моделей

async def calibrate\_models(): # Калибровка с приостановкой торговли

def \_train\_transformer(): # Обучение EfficientTransformer

```

\*\*Функции\*\*:

- Управление жизненным циклом моделей

- Обнаружение дрейфа данных (KS-тест)

- Адаптивное квантование моделей

- Автоматическое восстановление поврежденных моделей

---

### \*\*15. Техническая архитектура\*\*

```mermaid

graph LR

T[Tinkoff API] -- gRPC --> A[TinkoffAdapter]

A --> D[DataEngine]

D --> M[MLEngine]

M --> R[RiskEngine]

R --> E[Order Execution]

O[Orchestrator] --> A & D & M & R

MT[ModelTrainer] --> D

MT --> M

DB[(Database)] --> D & R

```

---

### \*\*16. Дополнительные механизмы\*\*

#### \*\*16.1 Восстановление соединения\*\*

- Экспоненциальная задержка до 60 сек

- Контроль целостности данных:

```python

async def check\_data\_integrity() -> bool: ...

```

#### \*\*16.2 Перезагрузка моделей\*\*

1. Очистка кешей фичей

2. Загрузка новых TFLite моделей

3. Сброс состояния интерпретаторов

4. Уведомление RiskEngine

#### \*\*16.3 Резервное хранилище\*\*

- Автосохранение в JSONL при ошибках БД

- Кеширование состояния каждые 30 секунд

#### \*\*16.4 Квантование моделей\*\*

\*\*Стратегии\*\*:

```python

if weight > 0.5: # INT8 для важных моделей

converter.target\_spec.supported\_ops = [tf.lite.OpsSet.TFLITE\_BUILTINS\_INT8]

else: # FLOAT16 для остальных

converter.target\_spec.supported\_types = [tf.float16]

```

\*\*Преимущества\*\*:

- Ускорение инференса 2-4x

- Сокращение размера моделей до 75%

#### \*\*16.5 Восстановление моделей\*\*

- Автодетекция поврежденных моделей (размер <1 КБ)

- Процедура восстановления:

```python

if os.path.getsize(model\_path) < 1024:

self.train\_all\_models() # Полная пересборка

```

---

### \*\*17. Гарантии производительности\*\*

| Параметр | Значение | Метрика контроля |

|------------------------|-------------|---------------------------|

| Обработка стакана | < 5 мс | Логирование задержек |

| Генерация сигнала | < 25 мс | P95 в latency\_stats |

| Обучение LSTM | < 15 мин | Логирование времени |

| Квантование моделей | < 2 мин | Тесты производительности |

| Валидация рисков | < 8 мс | Юнит-тесты |

---

### \*\*18. Технические характеристики ПК\*\*

- \*\*Процессор\*\*: AMD Ryzen 9 3900X (12 ядер/24 потока)

- \*\*Память\*\*: 64 ГБ DDR4 3600 МГц

- \*\*Графика\*\*: NVIDIA GeForce GTX 1070 (8 ГБ)

- \*\*Накопитель\*\*: 1 ТБ NVMe SSD (3500 MB/s read)

- \*\*Сеть\*\*: 10 GbE адаптер (Intel X550-T2)

- \*\*ОС\*\*: Windows 10 Pro (оптимизированная)

- \*\*Питание\*\*: 850W 80+ Platinum с UPS 1500VA

---

### \*\*19. Системные требования ПО\*\*

- \*\*Python\*\*: 3.10+ (оптимизация под AVX2)

- \*\*Библиотеки\*\*:

- TensorFlow 2.12+, Scikit-learn 1.3+, XGBoost 1.7+

- Tinkoff Invest API v2, NumPy 1.24+, Pandas 2.0+

- \*\*СУБД\*\*: SQLite 3.45+ с ZSTD-сжатием

- \*\*Сетевые протоколы\*\*: gRPC, WebSocket, HTTPS/TLS 1.3

---

### \*\*20. Процедура развертывания\*\*

1. Установка зависимостей:

```bash

pip install -r requirements.txt --use-pep517 --no-cache-dir

```

2. Инициализация конфигурации:

```bash

python init\_config.py --env=production

```

3. Калибровка моделей:

```bash

python -m model\_trainer calibrate --days=30

```

4. Запуск системы:

```bash

python orchestrator.py --profile=high\_perf

```

5. Запуск ModelTrainer:

```bash

python model\_trainer.py --daemon

```

6. Валидация:

```bash

python health\_check.py --module=all

```

---

### \*\*21. Протоколы аварийного восстановления\*\*

1. \*\*Сбой сети\*\*:

- Автопереключение на LTE за 150 мс

- Повторная синхронизация при восстановлении

2. \*\*Сбой GPU\*\*:

- Перевод LSTM на CPU

- Снижение частоты до 5 Гц

3. \*\*Ошибка RiskEngine\*\*:

- Активация упрощенного режима

- Немедленное уведомление оператора

4. \*\*Потеря данных\*\*:

- Восстановление из чекпоинта

- Ресинхронизация через API

5. \*\*Сбой ModelTrainer\*\*:

- Откат к предыдущей версии моделей

- Активация fallback-режима MLEngine

- Уведомление: "CRITICAL: Model training failure"

---

\*\*Система соответствует требованиям ISO 27001 для финансовых приложений. Все обновления проходят тестирование в forward-test режиме перед развертыванием в production.\*\*