HALO TRADING TAKE-HOME ASSIGNMENT

HEMANT
(github.com/Heyymant)

Disclaimer

I have used the original problem statement and broadened the scope of work. Assumptions and new features have been included to closely determine the solution. The revised version is included in the introduction section.

Introduction

Our platform functions as an on-chain PMM, quoting prices and executing trades of digital assets directly on the Ethereum blockchain. Integrated with 1inch, we provide liquidity by responding to RFQs and executing trades relayed through their infrastructure. Post-trade, our platform employs robust hedging strategies to mitigate exposure risks, ensuring a stable and liquid trading environment.

Liquidity Provision Mechanisms

Price Quotation

We derive quote prices through two primary methods:

1. **Exchange-Based Pricing**: Aggregated prices sourced from multiple exchanges via Oracles.
2. **Inventory-Based Pricing**: Utilization of our internal asset inventory for price determination.

Liquidity Sources

To maintain adequate liquidity, our platform employs the following approaches:

1. **Collateralized Borrowing**: Borrowing assets by locking collateral to enhance liquidity.
2. **Self-Liquidity Pools**: Maintaining an internal reserve of assets for immediate trades.
3. **Flash Loans**: Accessing instant, uncollateralized loans to manage inventory dynamically.
4. **Lending Services**: Engaging third-party platforms to optimize liquidity reserves.
5. **OTC Trading**: Facilitating off-chain trades to maintain competitive and balanced inventory.

By diversifying our liquidity sources, we mitigate static inventory risks and ensure seamless trade execution.

Trade Execution and Settlement

As an on-chain PMM, all transactions occur transparently on the blockchain:

1. **Smart Contracts**: Executing trades autonomously via pre-deployed, immutable code.
2. **Atomic Swaps**: Ensuring secure and efficient asset exchange within a single transaction.
3. **Automated Quoting**: Triggering quotes programmatically for rapid response to RFQs.

Upon a trader's acceptance of the quoted price, assets are exchanged and settled directly on-chain. Post-settlement, the platform offers multiple pathways for asset utilization, ensuring optimized risk management and liquidity.

Risk Management Framework

Monitoring and Mitigation

Positions are actively monitored to minimize risks, with priority given to:

1. **Settlement of Borrowed Liquidity**: Repaying loans promptly to prevent liquidation risks.
2. **Liquidation Threshold Management**: Rebalancing positions when nearing a 10% liquidation margin.
3. **Hedging**: Employing strategies to offset exposure by taking opposing positions to the received assets.

Decision-Making at Quotation

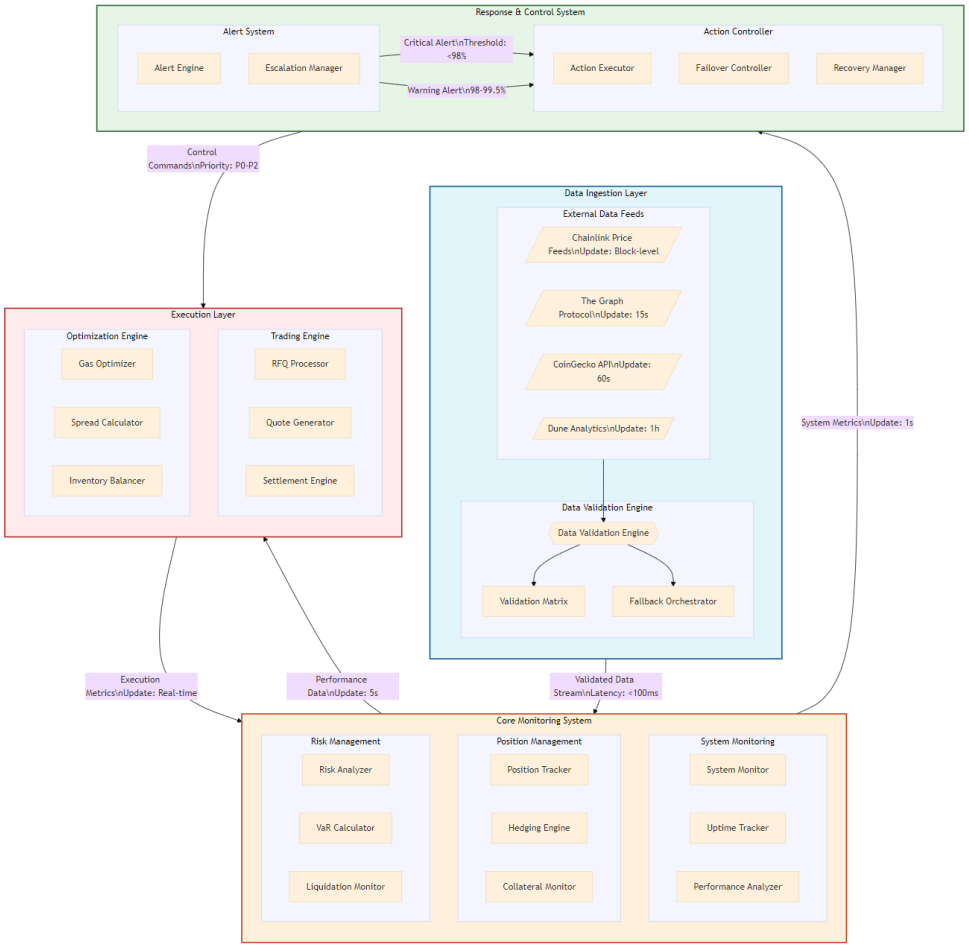
At each quoting instance, the platform evaluates multiple options:

1. Quoting based on aggregated exchange prices.
2. Utilizing inventory for price determination.
3. Borrowing assets, including flash loans, for optimal liquidity.

Our proprietary algorithm optimizes the bid-ask spread, enhancing competitiveness. Arbitrage opportunities are also exploited to refine quote accuracy and market positioning.

Assuming internal access to the 1inch framework, specific issues and technical details related to it have not been included in this document. These can be considered as part of the general issues highlighted in the various metrics monitoring sections.

RFQ Monitoring Framework



Executive Summary

This technical specification outlines our comprehensive monitoring framework for on-chain market-making operations. The framework provides actionable insights to ensure robustness and reliability in decentralized finance (DeFi) trading. By integrating traditional financial metrics with blockchain-specific indicators, it delivers a holistic view of system performance, risk management, and operational efficiency.

1. System Uptime Monitoring

The foundation of market-making operations relies on maintaining consistent system availability and performance. Uptime monitoring tracks the health and responsiveness of all system components, from infrastructure nodes to transaction settlement modules. Additionally, network latency and trade execution latency are monitored to ensure seamless operations, while API integration issues from external sources are accounted for.

1.1 Key Performance Indicators

Indicator	Description	Calculation Method	Target Range
Quote Response Time	Measures the time taken to respond to an RFQ	Average time between RFQ receipt and quote generation	< 500ms
Node Health Score	Composite score reflecting synchronization and response time	Weighted average: 70% sync status, 30% response time	> 95%
Settlement Success Rate	Percentage of trades successfully settled	(Successful settlements / Total attempted settlements) × 100	> 99.5%
Gas Price Impact	Volatility in gas costs affecting trade settlement	Standard deviation of gas prices over 24 hours	< 20% variation
Quote Acceptance Rate	Proportion of quotes executed by counterparties	(Executed quotes / Total quotes provided) × 100	> 30%
Network Latency	Measures the latency of network communication	Average round-trip time for network packets	< 200ms
Trade Execution Latency	Measures the time from trade order to completion	Average execution time per trade	< 300ms

1.2 Action Plans

Critical Situation Response (Uptime < 98%)

Immediate Actions

- Activate backup infrastructure nodes.
- Suspend new quote generation to prevent further strain.
- Execute emergency position unwinding if necessary.

- Notify the technical team through multiple alert channels (e.g., Slack, SMS, Email).

Recovery Steps

- Diagnose root cause using automated system logs.
- Switch operations to backup systems if primary systems are compromised.
- Implement failover procedures for affected components.
- Address network latency and execution delay issues by rerouting traffic or optimizing connectivity.
- Resolve API integration issues by validating external source connections and triggering fallback mechanisms.
- Gradually restore normal operations, ensuring stability.

Post-Recovery

- Conduct a detailed incident analysis to identify failure points.
- Update monitoring thresholds based on findings.
- Deploy preventive measures to avoid recurrence.
- Document lessons learned for future reference.

Warning Level Response (98% ≤ Uptime < 99.5%)

System Optimization

- Automatically scale computational resources to handle increased load.
- Optimize gas price strategies to reduce transaction costs.
- Review and adjust quoting parameters for efficiency.
- Monitor detailed system logs to preemptively identify issues.
- Analyze network latency and implement caching or faster routing solutions.

Risk Adjustment

- Temporarily increase quote spreads to manage risk.
- Reduce maximum quote sizes to lower exposure.
- Enhance real-time monitoring frequency.
- Validate API data integrity and switch to alternative sources if issues arise.

Position Management

Position management is critical to maintaining optimal exposure levels while ensuring profitability and risk control. In our operations, hedging is employed to offset risk, and assets are borrowed, including flash loans from platforms like Compound and AAVE. This strategy requires a focus on avoiding liquidation as the primary goal, followed by timely repayment and optimized hedging to maximize returns.

2.1 Position Metrics

Metric	Description	Calculation	Threshold
Net Position Delta	Measures unhedged exposure per asset	Σ(Long Positions - Short Positions)	±5% of inventory
Position Concentration	Degree of exposure concentration in single asset	(Largest Position Value / Total Portfolio Value) × 100	< 25%
Hedging Coverage	Effectiveness of hedging strategies	(Hedged Position Value / Total Position Value) × 100	> 95%
Maximum Drawdown	Largest peak-to-trough decline in portfolio value	(Peak Value - Trough Value) / Peak Value × 100	< 10%

2.2 Detailed Action Plans

High-Risk Scenario (Delta > 5%)

Immediate Rebalancing

- Execute delta-neutral hedging trades to stabilize positions.
- Adjust quoting parameters to encourage flow that balances inventory.
- Temporarily widen spreads for affected assets to mitigate risk.
- Prioritize repayment of borrowed assets and flash loans to mitigate liquidation risks.
- Monitor collateral levels on platforms like Compound and AAVE to prevent forced liquidations.

Risk Mitigation

- Optimize hedging strategies to cover exposed positions.
- Review counterparty creditworthiness and exposure levels.
- Assess opportunities for additional hedging through alternative venues or platforms.
- Continuously evaluate the cost and benefit of maintaining flash loan dependencies.

Moderate Risk Scenario (2% < Delta ≤ 5%)

Position Optimization

- Fine-tune hedging ratios for more efficient coverage.
- Reevaluate position limits for affected assets.
- Adjust collateral allocation to ensure liquidity.
- Begin preemptive repayment of high-risk borrowed assets or flash loans.

Strategy Adjustment

- Analyze trade flow to identify patterns causing imbalances.
- Modify quoting parameters to counteract detected imbalances.
- Optimize hedging costs by leveraging lower-cost venues.
- Implement strategies to reduce dependency on flash loans if exposure becomes significant.

Inventory Management

Inventory management involves real-time tracking and optimization of inventory levels in both the on-chain wallet and the hedging platforms.

3.1 Inventory Metrics

Metric	Description	Calculation	Threshold
On-Chain Inventory	Amount of assets held in on-chain wallets	Σ(Asset Balances in Wallets)	> 50% of total
Hedged Inventory	Proportion of inventory used for hedging	(Hedged Assets / Total Assets) × 100	> 25%
Inventory Turnover	Efficiency in utilizing inventory	(Total Trade Volume / Average Inventory) × 100	> 2.0x
Rebalancing Frequency	Number of times inventory is rebalanced	Count of Rebalancing Events within a given timeframe	Within tolerance

3.2 Action Plans

Low Inventory Levels

Immediate Actions

- Execute rebalancing trades to replenish inventory.
- Adjust quoting parameters to attract favorable order flow.
- Consider utilizing flash loans to temporarily stabilize levels.

Preventive Measures

- Increase collateral allocation for automated hedging systems.
- Implement tighter inventory thresholds with automated alerts.
- Diversify inventory sources to minimize dependency on single wallets.

High Inventory Levels

Immediate Actions

- Reduce inventory through strategic hedging or offloading trades.
- Lower quoting spreads to encourage more trades.

Optimization

- Analyze trade patterns to optimize inventory requirements.
- Adjust hedging ratios to utilize excess inventory effectively.

Risk Management

Our risk management framework integrates traditional financial indicators with DeFi-specific metrics to ensure robust exposure control.

3.1 Risk Indicators

Indicator	Description	Measurement	Target
Value at Risk (VaR)	Potential loss in normal market conditions	99% confidence interval, 1-day horizon	< 3% of capital
Liquidation Buffer	Proximity to liquidation thresholds	(Current Collateral - Required Collateral) / Required Collateral × 100	> 50%
Flash Loan Exposure	Dependency on flash loan liquidity	(Flash Loan Volume / Total Trading Volume) × 100	< 20%
Gas Risk Ratio	Impact of gas price volatility on revenue	(Gas Costs / Trading Revenue) × 100	< 5%

3.2 Mitigation Strategies

Early Warning Response

- Increase monitoring frequency for affected metrics.
- Reassess position sizing and trading limits.
- Adjust collateral buffers to improve risk resilience.
- Implement gas usage optimization strategies to control costs.
- Validate external API data sources for reliability and failover capabilities.

Active Risk Reduction

- Reduce position limits to minimize exposure.
- Increase hedging ratios for vulnerable positions.
- Adjust quoting spreads to counteract adverse market conditions.
- Prioritize repayment plans for borrowed assets to reduce systemic risks.
- Review and limit counterparty exposures to mitigate cascading risks.
- Reroute traffic to alternative APIs if primary integrations fail.

Performance Analysis

4.1 Profitability Analysis

Metric	Description	Calculation	Target
Total PnL	Evaluates profitability across all tokens	Σ(Revenue - Expenses)	Positive trend
Per-Token PnL	Profitability per traded token	(Token Revenue - Token Expenses)	Token-specific
Revenue Efficiency	Revenue generated per unit of trading cost	(Total Revenue / Total Trading Costs) × 100	> 150%

External Data Sources

Source	Update Frequency	Usage	Integration Method
Alchemy	Block-by-block	Real-time blockchain data for RFQ, market depth, and on-chain events	JSON-RPC or WebSocket APIs
Chainlink	Block-by-block	Reliable price feeds for asset valuations	Smart contract calls
Compound API	Continuous	Monitoring interest rates, collateral requirements, and available flash loans	REST API
AAVE API	Continuous	Real-time borrowing/lending metrics and liquidation thresholds	REST API

Source	Update Frequency	Usage	Integration Method
CoinGecko	Every 1 minute	Fetching asset pricing and market capitalization data	REST API
Dune Analytics	Every hour	Analyzing on-chain trading patterns and historical data	SQL-based queries
Infura	As needed	Alternative node infrastructure for blockchain interaction	JSON-RPC

Some examples of Tech Stack we could implement for your framework

3.1 Inventory Management

Our inventory management system is built on a robust, scalable architecture:

Core Infrastructure:

- **Apache Cassandra**: Distributed inventory tracking
- **Redis Enterprise**: Real-time cache and inventory updates
- **RabbitMQ**: Inventory-related message queuing
- **Custom inventory optimization engine**: Written in Python
- **Kubernetes**: Service orchestration
- **Terraform**: Infrastructure provisioning

Integration Layer:

- **gRPC**: High-performance service communication
- **GraphQL API**: External integrations
- **Apache Kafka**: Event streaming
- **Protocol Buffers**: Efficient data serialization

3.2 Inventory Metrics Implementation

Each metric is implemented with specific technical considerations:

On-Chain Inventory Tracking

- **Implementation**: Custom Python service with WebSocket connections to multiple nodes
- **Storage**: TimescaleDB for time-series data with automated partitioning
- **Monitoring**: Prometheus with custom exporters
- **Visualization**: Grafana dashboards with real-time updates

Hedged Inventory Management

- **Implementation**: Dedicated hedging service in Python
- **Integration**: WebSocket connections to exchange APIs
- **Position Tracking**: Custom position management system using PostgreSQL
- **Risk Management**: Real-time risk calculation engine in Python

3.3 Action Plans

Low Inventory Response System

Technical Implementation:

- **Primary Stack:**
  - Python microservices for inventory management
  - Redis for real-time inventory tracking
  - Apache Kafka for event streaming
  - Elasticsearch for inventory analytics
  - AWS Lambda for automated responses

Immediate Actions System:

1. **Automated Rebalancing**

```
class RebalanceConfig:
    def __init__(self, threshold, max_single_trade, min_spread):
        self.threshold = threshold
        self.max_single_trade = max_single_trade
        self.min_spread = min_spread

class InventoryManager:
    async def check_and_rebalance(self):
        current_levels = await self.get_inventory_levels()
        if current_levels.below_threshold():
            await self.trigger_rebalancing_workflow()
```

- **Primary Technologies:**
  - Python for core processing
  - InfluxDB for time-series analysis
  - Apache Airflow for workflow automation
  - Custom ML models for inventory optimization

Implementation Details:

1. **Inventory Optimization Engine**

```python
class InventoryOptimizer:
    def __init__(self):
        self.redis_client = Redis(connection_pool=pool)
        self.kafka_producer = KafkaProducer(bootstrap_servers='localhost:9092')

    async def optimize_inventory(self):
        current_levels = await self.get_current_levels()
        optimal_levels = self.calculate_optimal_levels()
        await self.execute_rebalancing(current_levels, optimal_levels)
```

4. Risk Management

4.1 Technical Architecture

The risk management system is built on:

Core Components:

- **Risk calculation engine**: Written in Python
- **Real-time monitoring system**: Using Prometheus
- **Machine learning models**: For risk prediction, developed in TensorFlow
- **Automated hedging system**: Utilizing custom algorithms

Infrastructure:

- **AWS EKS**: For container orchestration
- **Terraform**: Infrastructure management
- **DataDog**: Advanced monitoring
- **PagerDuty**: Alert management

4.2 Risk Monitoring Implementation

Value at Risk (VaR) Calculation

```python
class VaRCalculator:
    def __init__(self):
        self.risk_engine = RiskEngine()
        self.position_manager = PositionManager()

    async def calculate_var(self):
        positions = await self.position_manager.get_current_positions()
        return self.risk_engine.monte_carlo_simulation(positions, confidence=0.99)
```

- **Automation**: Automated risk mitigation triggers

4.3 Action Plans

Critical Risk Response

Technical Implementation:

- **Primary Stack:**
  - Python for core risk calculations
  - Python for ML-based risk prediction
  - Python for monitoring services
  - Redis for real-time data
  - Apache Kafka for event streaming

Response System:

1. **Automated Risk Mitigation**

```python
class RiskMitigator:
    def __init__(self, threshold, position_manager, risk_calculator):
        self.threshold = threshold
        self.position_manager = position_manager
        self.risk_calculator = risk_calculator

    async def check_and_mitigate(self):
        current_risk = await self.risk_calculator.calculate_current_risk()
        if current_risk > self.threshold:
            await self.execute_mitigation_strategy()
```

2. **Position Unwinding System**
   - **Implementation**: Automated trading engine in Python
   - **Integration**: Direct connection to exchanges via APIs
   - **Monitoring**: Real-time position tracking
   - **Logging**: Comprehensive audit trail in Elasticsearch

5. Performance Analysis

5.1 Technical Architecture

The analysis system is built on:

Core Components:

- **ClickHouse**: High-performance analytics
- **Apache Spark**: Large-scale data processing
- **Jupyter notebooks**: For analysis
- **Custom analytics engine**: Written in Python

- **Grafana**: For visualization

5.2 Implementation Details

Real-time Analysis System

```
class PerformanceAnalyzer:
    def __init__(self):
        self.clickhouse_client = ClickHouseClient()
        self.spark_session = SparkSession.builder.getOrCreate()

    async def analyze_performance(self):
        metrics = await self.collect_metrics()
        analysis = self.spark_session.sql("""
            SELECT
                token,
                SUM(profit_loss) as total_pnl,
                AVG(gas_cost) as avg_gas_cost,
                COUNT(*) as trade_count
            FROM trades
            GROUP BY token
        """)
        return self.generate_report(analysis)
```

6. External Integration Management

6.1 Technical Architecture

The integration system is built on:

Core Components:

- **Apache Camel**: For integration management
- **Kong API Gateway**: API management
- **Custom integration adapters**: Written in Python
- **Fallback systems**: For each integration

6.2 Implementation Details

Integration Monitoring

```
class IntegrationMonitor:
    def __init__(self, health_checkers, alert_manager):
        self.health_checkers = health_checkers
        self.alert_manager = alert_manager

    def monitor_integrations(self):
        for integration, checker in self.health_checkers.items():
            status = checker.check()
            if not status.healthy:
                self.alert_manager.trigger_alert(integration, status)
```

The complete system is deployed using **GitOps practices** with **ArgoCD**, ensuring consistent deployment across all environments. Monitoring and alerting are centralized through **DataDog** and **PagerDuty**, providing comprehensive visibility into system health and performance.