Multi-Agent Architecture for Momentum Trading Strategy

vantuandang1990@gmail.com

Contents

1 Introduction

Algorithmic trading has revolutionized financial markets, allowing for sophisticated strategy implementation, rapid execution, and data-driven decision-making. Among various strategies, momentum trading stands as one of the most popular and effective approaches, capitalizing on the continuation of existing market trends.

This report presents a novel framework: a Multi-Agent Architecture for automating the development, testing, and evaluation of momentum trading strategies. The system leverages the AutoGen framework to create specialized AI agents that collaborate to implement, test, and optimize trading strategies based on moving average crossovers.

We will first explore the base architecture, which focuses on technical analysis using moving averages. Then, we will examine an enhanced version that incorporates machine learning-based forecasting to improve trading decisions. Both architectures are designed to provide comprehensive analysis with minimal human intervention, demonstrating the power of AI-assisted quantitative trading.

2 Context and Motivation

2.1 Traditional Strategy Development Challenges

Developing algorithmic trading strategies traditionally involves several manual, time-consuming steps:

1. **Strategy conceptualization**: Defining trading rules and signals
2. **Code implementation**: Writing and debugging strategy code
3. **Parameter testing**: Testing with various parameter combinations
4. **Performance evaluation**: Calculating returns, risk metrics, and other KPIs
5. **Optimization**: Fine-tuning parameters for optimal performance
6. **Visualization**: Creating charts and reports to analyze results

This process is often iterative, requiring multiple rounds of refinement. It can be error-prone, resource-intensive, and difficult to scale as the number of parameters and data increases.

2.2 Benefits of the Multi-Agent Approach

A Multi-Agent Architecture offers significant advantages:

1. **Automation**: Reduces manual coding and testing effort
2. **Separation of concerns**: Each agent specializes in a specific task
3. **Collaborative problem-solving**: Agents work together to implement and evaluate strategies
4. **Quality assurance**: Built-in evaluation of code quality and strategy performance
5. **Scalability**: Easily test multiple parameter combinations and strategies

6. **Comprehensive analysis**: Generate detailed reports and visualizations

By distributing tasks among specialized agents, the system can efficiently handle complex trading strategy development and evaluation, leading to better-informed trading decisions.

3 Base Multi-Agent Architecture

3.1 Architecture Overview

The base Multi-Agent Architecture for Momentum Trading Strategy comprises five distinct layers, each responsible for a specific aspect of the trading strategy development and evaluation process.

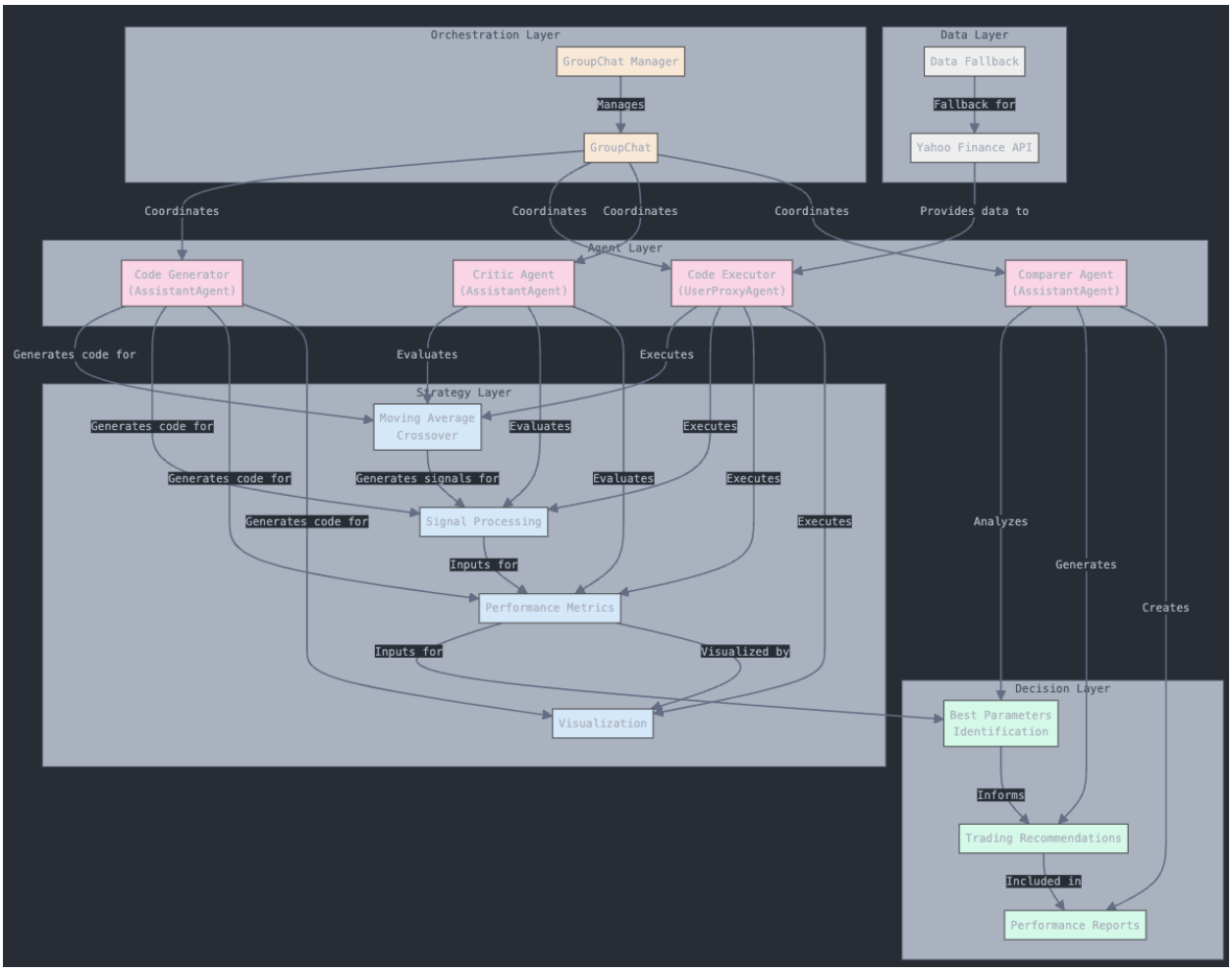

Figure 1: Multi-Agent Architecture for Momentum Trading Strategy

3.2 Core Layers and Components

3.2.1 Orchestration Layer

This layer coordinates the entire system and manages communication between agents:

- **GroupChat Manager**: Acts as the conductor of the system, orchestrating agent interactions.

- **GroupChat**: Provides the environment where agents exchange information and collaborate.

3.2.2 Agent Layer

This layer contains the specialized AI agents that perform different tasks:

- **Code Generator Agent** (AssistantAgent): Creates Python code for trading strategies based on specifications.
- **Code Executor Agent** (UserProxyAgent): Executes the generated code and interfaces with data sources.
- **Critic Agent** (AssistantAgent): Evaluates the quality of code and strategy implementation.
- **Comparer Agent** (AssistantAgent): Analyzes results across different parameter sets to identify optimal configurations.

3.2.3 Data Layer

This layer handles data acquisition and management:

- **Yahoo Finance API**: Primary source for historical stock price data.
- **Data Fallback**: Mechanism for generating synthetic data when real data is unavailable.

3.2.4 Strategy Layer

This layer implements the actual trading strategy logic:

- **Moving Average Crossover**: Implements the MA crossover algorithm for generating trading signals.
- **Signal Processing**: Processes raw signals into actionable trading decisions.
- **Performance Metrics**: Calculates key performance indicators for strategy evaluation.
- **Visualization**: Creates charts and visual representations of strategy performance.

3.2.5 Decision Layer

This layer focuses on analyzing results and making recommendations:

- **Best Parameters Identification**: Identifies optimal parameter combinations.
- **Trading Recommendations**: Generates specific trading advice based on the analysis.
- **Performance Reports**: Creates comprehensive reports documenting strategy performance.

3.3 Component Relationships

The relationships between components are crucial for understanding the system's operation:

1. **Orchestration to Agent relationships**:
   - GroupChat Manager coordinates the GroupChat.
   - GroupChat facilitates communication between all agents.
2. **Agent to Strategy relationships**:
   - Code Generator creates code for all Strategy layer components.
   - Code Executor executes the strategy implementations.
   - Critic Agent evaluates the quality of the strategy components.
   - Comparer Agent analyzes the Decision layer outputs.
3. **Data to Strategy relationships**:
   - Yahoo Finance provides data to the Code Executor.
   - Data Fallback serves as a contingency for data acquisition.
4. **Strategy to Decision relationships**:
   - Moving Average Crossover generates signals for Signal Processing.
   - Signal Processing provides inputs for Performance Metrics.
   - Performance Metrics feed into Best Parameters Identification.
   - Performance Metrics are visualized by the Visualization component.
5. **Decision layer internal relationships**:
   - Best Parameters Identification informs Trading Recommendations.
   - Trading Recommendations are included in Performance Reports.

These relationships create a cohesive system where data flows from acquisition to strategy implementation to evaluation and finally to decision-making.

4 End-to-End Process Flow

The end-to-end process flow illustrates how the system operates from initialization to final report generation.

4.1 Initialization Phase

The process begins when a user initiates an analysis with specific parameters:

1. **User initiates analysis**:
   - Specifies stock symbol (e.g., "NVDA")
   - Defines MA pairs to analyze (e.g., [(5,20), (10,50), (20,100), (50,200)])
   - Sets analysis time period.

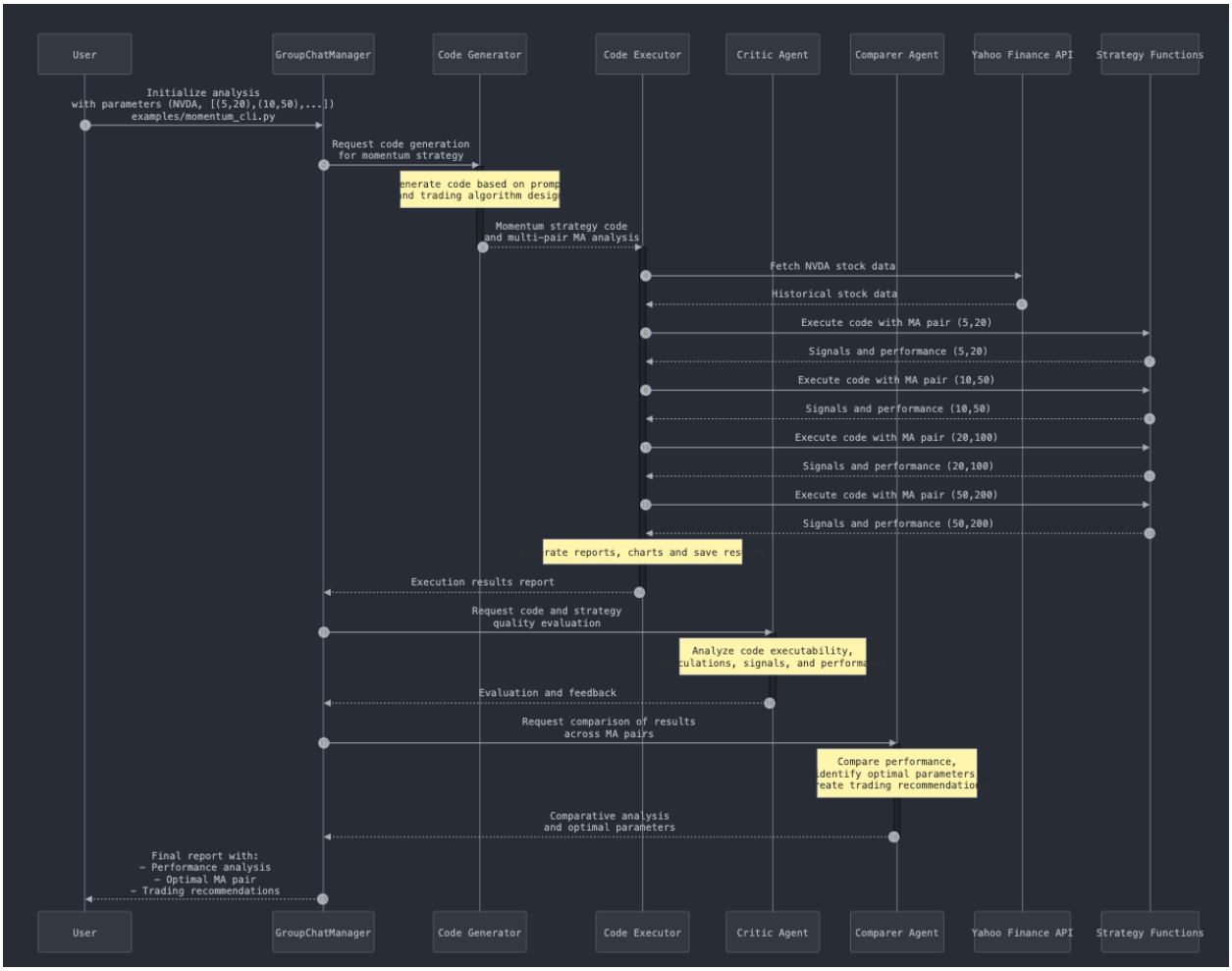

Figure 2: End-to-End Sequence Diagram of Multi-Agent System

2. **GroupChatManager orchestration**:
   - Creates and coordinates the group chat environment.
   - Distributes tasks to appropriate agents.
3. **Code Generator activation**:
   - Receives request to generate momentum strategy code.
   - Creates well-structured Python code based on prompt templates.
   - Delivers code to Code Executor.

4.2 Data Acquisition Phase

Once the code is generated, the system proceeds to acquire necessary data:

1. **Data retrieval**:
   - Code Executor requests historical data from Yahoo Finance.
   - System handles SSL issues automatically.

- Fallback mechanisms activate if needed.

2. **Data preparation**:
   - Historical price data is loaded into dataframes.
   - Basic data cleaning and preprocessing occurs.
   - Data is prepared for strategy implementation.

4.3 Strategy Implementation Phase

With data in hand, the system implements the trading strategy:

1. **Strategy execution for each MA pair**:
   - Code Executor runs the strategy with each parameter set.
   - Moving averages are calculated for each specified window.
   - Buy/sell signals are generated based on MA crossovers.
   - Performance metrics are calculated.

2. **Results storage and visualization**:
   - Results are saved for each parameter set.
   - Trading strategy charts are generated.
   - Buy/sell signal visualizations are created.
   - Performance reports are compiled.

4.4 Evaluation Phase

After implementation, the system evaluates the quality of both code and strategy:

1. **Critic Agent evaluation**:
   - Assesses code quality and executability.
   - Verifies calculation accuracy.
   - Validates signal generation logic.
   - Confirms performance metric calculations.

2. **Feedback mechanism**:
   - Critic provides detailed feedback on implementation.
   - Identifies potential improvements or issues.
   - Scores different aspects of the implementation.

4.5 Decision-Making Phase

Finally, the system analyzes results and makes recommendations:

1. **Comparer Agent analysis**:
    - Compares performance across different MA pairs.
    - Identifies trade-offs between parameters.
    - Determines which parameters performed best.
2. **Report generation**:
    - Comprehensive report with performance metrics.
    - Optimal parameter identification.
    - Trading recommendations based on findings.
    - Visualizations of strategy performance.
3. **User delivery**:
    - Final report with all analysis components.
    - Optimal parameters for the analyzed stock.
    - Actionable trading recommendations.

5 Enhanced Architecture with Forecasting

While the base architecture provides robust technical analysis capabilities, we can significantly enhance it by incorporating machine learning-based forecasting.

5.1 Forecasting Agent Integration

The enhanced architecture introduces a new specialized agent:

- **Forecasting Agent** (AssistantAgent): Specializes in creating and evaluating machine learning models for price prediction.
- Communicates with other agents via the GroupChat.
- Generates code for model training and prediction.
- Provides forecasting expertise to the system.

5.2 Machine Learning Component

New components are added to support machine learning capabilities:

1. **Historical Database**:
    - Stores historical price data in a structured format.
    - Enables efficient data retrieval for model training.
    - Provides consistent datasets for backtesting.

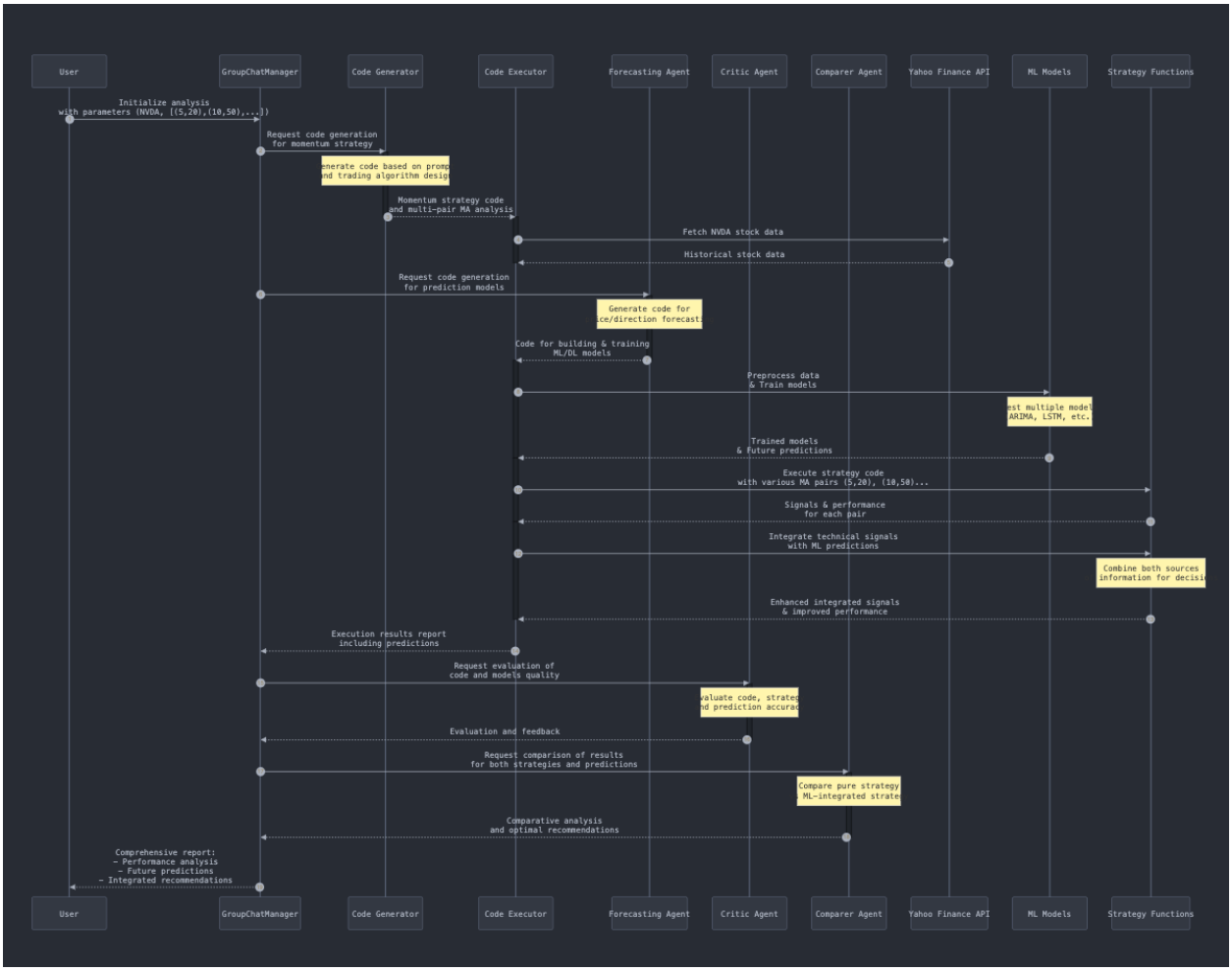

Figure 3: Enhanced Multi-Agent Architecture with Forecasting Agent

2. **ML Models**:
   - Implements various forecasting models:
     - Statistical models (ARIMA, GARCH)
     - Machine learning models (Random Forest, SVM)
     - Deep learning models (LSTM, GRU)
   - Handles model training, validation, and testing.
   - Manages model persistence and versioning.
3. **Forecast-based Predictions**:
   - Generates future price or direction predictions.
   - Calculates confidence intervals for predictions.
   - Provides prediction metrics and evaluation.

5.3 Extended Data Flow

The enhanced architecture introduces new data flows:

1. **Historical data storage**:
    - Yahoo Finance data is stored in the Historical Database.
    - Ensures data consistency across model training sessions.
2. **Model training flow**:
    - Forecasting Agent creates code for model training.
    - Code Executor trains models using historical data.
    - Models are evaluated and the best model is selected.
3. **Prediction integration flow**:
    - ML Models generate predictions.
    - Forecast-based Predictions process these predictions.
    - Predictions enhance Trading Recommendations.
    - Performance Reports include forecasting insights.

6 Enhanced End-to-End Process

The enhanced process incorporates model training and prediction integration into the workflow.

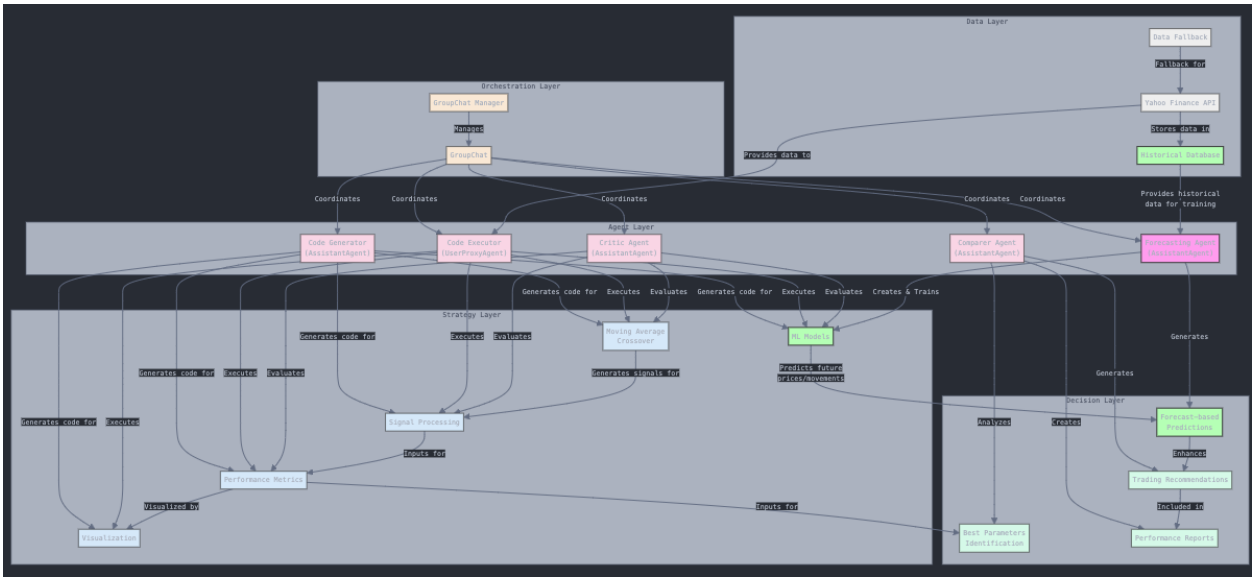

Figure 4: End-to-End Process with Forecasting Agent

6.1 Model Training Workflow

The enhanced process adds model training steps:

1. **Forecasting Agent activation**:
    - GroupChatManager requests Forecasting Agent to create prediction models.

- Forecasting Agent generates code for various model types.
- Code includes data preprocessing, feature engineering, and model training.

2. **Model training execution**:
   - Code Executor processes historical data.
   - Multiple model types are trained (ARIMA, LSTM, etc.).
   - Models are evaluated using appropriate metrics.
   - Best-performing model is selected.
3. **Future prediction generation**:
   - Selected model generates predictions for future periods.
   - Confidence intervals are calculated.
   - Predictions are formatted for integration.

6.2 Integrated Signal Generation

The process now combines technical and ML-based signals:

1. **Technical signal generation**:
   - Moving average crossover signals are generated.
   - Buy/sell points are identified based on MA crossovers.
2. **ML prediction integration**:
   - Forecasting predictions are aligned with technical timeframes.
   - ML confidence scores are calculated.
   - Technical signals are weighted based on ML predictions.
   - Integrated signals combine both information sources.
3. **Enhanced performance calculation**:
   - Performance metrics are calculated for:
     - Pure technical strategy
     - Pure ML strategy
     - Integrated strategy
   - Comparative analysis identifies optimal approach.

6.3 Enhanced Decision Making

Decision making now incorporates forecasting insights:

1. **Comprehensive evaluation**:
   - Critic Agent evaluates both technical and ML components.
   - Code quality assessment includes ML implementation.
   - Prediction accuracy metrics are included in evaluation.

2. **Enriched comparison**:
    - Comparer Agent analyzes technical vs. integrated approaches.
    - Trade-offs between different approaches are identified.
    - Risk-adjusted performance is highlighted.
3. **Advanced reporting**:
    - Reports include future predictions with confidence intervals.
    - Performance comparison across all approaches.
    - Optimal strategy identification (technical, ML, or integrated).
    - Risk-aware recommendations.

7 Implementation Guide

7.1 Environment Setup

To implement the Multi-Agent Architecture, follow these steps:

1. **Create Conda environment:**

```
# Clone repository
git clone https://github.com/danglive/momentum-trading-autogen.git
cd momentum-trading-autogen

# Create environment from file
conda env create -f environment.yaml

# Activate environment
conda activate momentum-trading
```

2. **Set up OpenAI API key:**

```
export OPENAI_API_KEY='your-api-key'
```

3. **Install additional packages for ML** (for enhanced architecture):

```
pip install tensorflow scikit-learn statsmodels prophet
```

7.2 Basic Configuration

To run an analysis with the base architecture:

1. **Single MA pair analysis:**

```
python examples/momentum_cli.py single --symbol NVDA --short 5 --long 20
```

2. **Multiple MA pairs analysis:**

```
python examples/momentum_cli.py pairs --symbol AAPL
```

3. **Multiple stocks analysis:**

```
python examples/momentum_cli.py stocks --short 10 --long 50
```

7.3 Advanced Configuration

For the enhanced architecture with forecasting:

1. **Create a Forecasting Agent:**

```
# Add to src/agents/setup_agents.py
forecasting_agent = AssistantAgent(
    name="Forecasting_agent",
    system_message=FORECASTING_AGENT_PROMPT,
    llm_config={"config_list": config_list},
    human_input_mode="NEVER"
)

# Add to agents dictionary
agents["forecasting_agent"] = forecasting_agent
```

2. **Run integrated analysis:**

```
# First run standard analysis
strategy_results = run_momentum_analysis(agents, symbol="NVDA", ma_pairs
    =[(5,20), (10,50)])

# Then run forecasting analysis
forecasting_results = run_forecasting_analysis(agents, symbol="NVDA",
    prediction_horizon=5)

# Integrate results
integrated_results = integrate_strategy_and_forecasting(strategy_results,
    forecasting_results)
```

8 Benefits and Limitations

8.1 Architectural Advantages

The Multi-Agent Architecture offers numerous benefits:

1. **Automation and efficiency**:
   - Reduces manual effort in strategy development.
   - Automates repetitive tasks like parameter testing.
   - Accelerates the development-evaluation cycle.
2. **Separation of concerns**:
   - Each agent focuses on its specialized task.
   - Modularity improves maintainability.
   - Easier to update individual components.
3. **Quality assurance**:
   - Built-in evaluation of code quality.
   - Systematic testing of strategies.
   - Comparison across multiple parameters.

4. **Enhanced decision making**:
    - Data-driven parameter selection.
    - Comprehensive performance evaluation.
    - Risk-aware trading recommendations.
5. **Forecasting integration** (in enhanced architecture):
    - Combines technical and predictive approaches.
    - Leverages ML for improved signal generation.
    - Provides forward-looking insights.

8.2 Challenges and Constraints

Despite its advantages, the architecture faces certain limitations:

1. **Dependency on external services**:
    - Requires OpenAI API for agent functionality.
    - Relies on Yahoo Finance for data access.
    - Potential cost implications for API usage.
2. **Technical complexity**:
    - Requires understanding of multiple technologies.
    - ML component adds additional complexity.
    - Debugging across distributed components can be challenging.
3. **Market limitations**:
    - Strategies may not perform consistently across all market conditions.
    - Past performance doesn't guarantee future results.
    - Market regime changes can affect strategy performance.
4. **Resource requirements**:
    - ML training can be computationally intensive.
    - Testing multiple parameters requires significant resources.
    - Real-time deployment requires robust infrastructure.

8.3 Performance Considerations

When implementing the architecture, consider these performance factors:

1. **Computational efficiency**:
    - Optimize data retrieval and storage.
    - Consider batch processing for multiple parameters.
    - Use efficient ML implementations.

2. **Scalability planning**:
    - Design for increasing data volumes.
    - Plan for additional strategy types.
    - Consider cloud deployment for resource flexibility.
3. **Latency management**:
    - Minimize API call latency.
    - Optimize model inference time.
    - Balance precision and speed in calculations.

9 Future Development Directions

9.1 Additional Strategies

The architecture can be extended to support more strategies:

1. **Technical strategies**:
    - MACD (Moving Average Convergence Divergence)
    - RSI (Relative Strength Index)
    - Bollinger Bands
    - Ichimoku Cloud
2. **Fundamental strategies**:
    - Earnings-based strategies
    - Value investing metrics
    - Growth factors
    - Sentiment analysis
3. **Hybrid approaches**:
    - Technical-fundamental combinations
    - Multi-factor models
    - Regime-switching strategies

9.2 Reinforcement Learning Integration

Reinforcement Learning (RL) offers promising extensions:

1. **RL agent addition**:
    - Create a dedicated RL Agent.
    - Implement Q-learning or policy gradient methods.
    - Train agents in market simulation environments.
2. **Dynamic strategy selection**:

- Use RL to switch between strategies.
- Adapt to changing market conditions.
- Optimize risk-return tradeoffs in real-time.

3. **Continuous learning**:
   - Implement online learning capabilities.
   - Update models as new data becomes available.
   - Adapt to regime changes autonomously.

9.3 Real-time Trading Capabilities

The architecture can evolve toward real-time trading:

1. **Real-time data integration**:
   - Connect to streaming market data APIs.
   - Process tick-level or minute-level data.
   - Implement efficient real-time feature calculation.
2. **Trading execution**:
   - Integrate with brokerage APIs.
   - Implement order management systems.
   - Add risk management controls.
3. **Monitoring and alerting**:
   - Create dashboards for strategy monitoring.
   - Implement alert systems for anomalies.
   - Develop performance tracking tools.

10 Conclusion

The Multi-Agent Architecture for Momentum Trading Strategy represents a significant advancement in automated trading strategy development and evaluation. By leveraging specialized AI agents, the system automates the entire process from code generation to performance evaluation, enabling more efficient and comprehensive strategy analysis.

The base architecture provides robust capabilities for technical analysis using moving average crossovers, while the enhanced version with forecasting integration adds predictive power through machine learning. This combination of technical analysis and predictive modeling offers a more holistic approach to trading strategy development.

Key takeaways include:

1. **Automation efficiency**: The multi-agent approach significantly reduces manual effort in strategy development and testing.
2. **Comprehensive analysis**: The architecture enables systematic testing across multiple parameters and strategies.

3. **Quality assurance**: Built-in evaluation mechanisms ensure code quality and strategy effectiveness.

4. **Enhanced decision-making**: The combination of technical analysis and machine learning provides more robust trading signals.

5. **Extensibility**: The modular design allows for easy addition of new strategies and capabilities.

While the architecture is not without challenges—including external dependencies and computational requirements—its benefits make it a valuable tool for quantitative traders and researchers. As financial markets continue to evolve, this architecture provides a flexible foundation that can adapt to changing conditions and incorporate new methodologies.

Future developments in reinforcement learning and real-time trading capabilities will further enhance the system's potential, making it an increasingly powerful platform for algorithmic trading strategy development and deployment.