AI Trading Assistant - Project Overview

Executive Summary

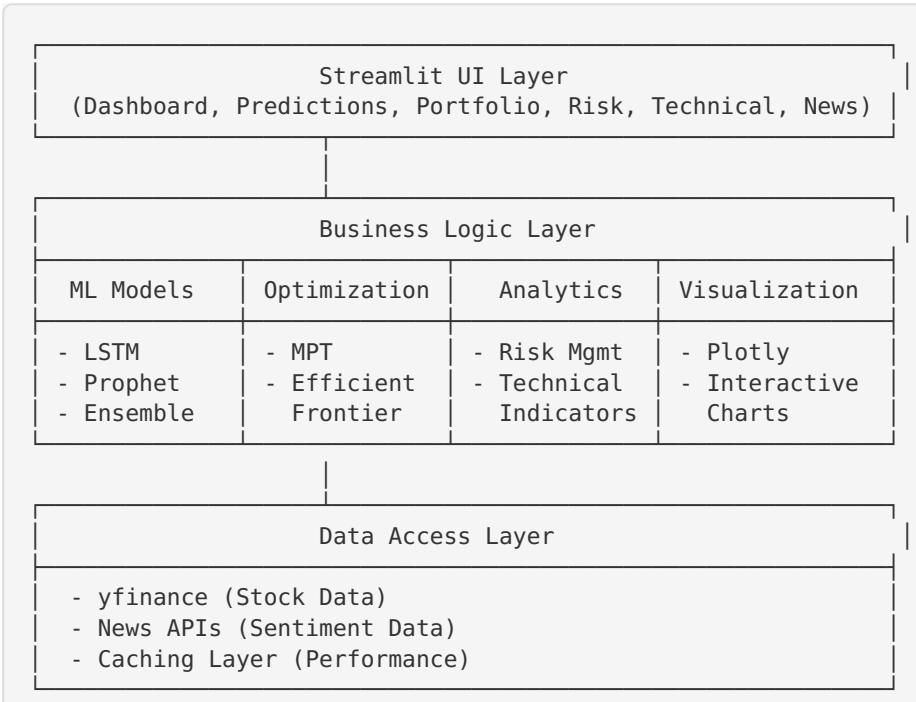
The AI Trading Assistant is a comprehensive FinTech application that combines Machine Learning, Modern Portfolio Theory, and Advanced Analytics to provide intelligent stock market insights. Built with Streamlit, it offers an intuitive interface for both novice and experienced investors.

Project Goals

1. **Educational Tool**: Help users understand financial markets and investment strategies
2. **Decision Support**: Provide data-driven insights for investment decisions
3. **Comprehensive Analysis**: Integrate multiple analytical approaches
4. **Accessibility**: Make advanced financial analytics accessible to everyone
5. **Production-Ready**: Deployable to cloud platforms with scalability

Technical Architecture

High-Level Architecture



Module Architecture

1. Data Fetcher (`data_fetcher.py`)

Purpose: Handle all data retrieval and preprocessing

Key Features:
- Stock price data fetching
- Multi-stock batch processing

- Data normalization and scaling
- Return calculations
- Caching for performance

Technologies: yfinance, pandas, numpy, scikit-learn

2. ML Predictor (`ml_predictor.py`)

Purpose: Price prediction using machine learning

Components:
- **LSTMPredictor**: Deep learning time-series model
- 3-layer LSTM architecture
- Dropout for regularization
- Sequence-based prediction

- **ProphetPredictor**: Facebook's Prophet model
- Handles trends and seasonality
- Robust to missing data
- Uncertainty intervals

- **EnsemblePredictor**: Combined predictions
- Weighted average of models
- Reduces individual model bias

Technologies: TensorFlow/Keras, Prophet, NumPy

3. Portfolio Optimizer (`portfolio_optimizer.py`)

Purpose: Implement Modern Portfolio Theory

Algorithms:
- Efficient Frontier calculation
- Maximum Sharpe Ratio optimization
- Minimum Volatility portfolio
- Risk Parity allocation
- Equal Weight baseline

Optimization Methods:
- scipy.optimize for constrained optimization
- cvxpy for convex optimization
- Monte Carlo simulation for frontier

Technologies: NumPy, pandas, SciPy, cvxpy

4. Risk Assessment (`risk_assessment.py`)

Purpose: Comprehensive risk analysis

Metrics Implemented:
- **Volatility**: Standard deviation of returns
- **VaR**: Historical and Parametric methods
- **CVaR**: Expected Shortfall
- **Sharpe/Sortino/Calmar Ratios**: Risk-adjusted returns
- **Maximum Drawdown**: Peak-to-trough decline

- **Beta/Alpha**: Market-relative measures
- **Information Ratio**: Active management performance

Technologies: NumPy, pandas, SciPy

5. Technical Indicators (`technical_indicators.py`)

Purpose: Technical analysis and trading signals

Indicators:
- RSI (Relative Strength Index)
- MACD (Moving Average Convergence Divergence)
- Bollinger Bands
- Stochastic Oscillator
- ATR (Average True Range)
- ADX (Average Directional Index)
- CCI (Commodity Channel Index)
- OBV (On-Balance Volume)

Signal Generation:
- Individual indicator signals
- Combined multi-indicator signals
- Strategy backtesting
- Support/Resistance identification

Technologies: pandas, NumPy

6. Sentiment Analyzer (`sentiment_analyzer.py`)

Purpose: News sentiment analysis

Approaches:
- **Basic**: TextBlob for fast analysis
- **Advanced**: DistilBERT transformer for accuracy

Features:
- News article collection
- Sentiment scoring (-1 to +1)
- Aggregate sentiment metrics
- Trending keyword extraction
- Time-series sentiment tracking
- Correlation with price movements

Technologies: Transformers, TextBlob, yfinance

7. Visualizations (`visualizations.py`)

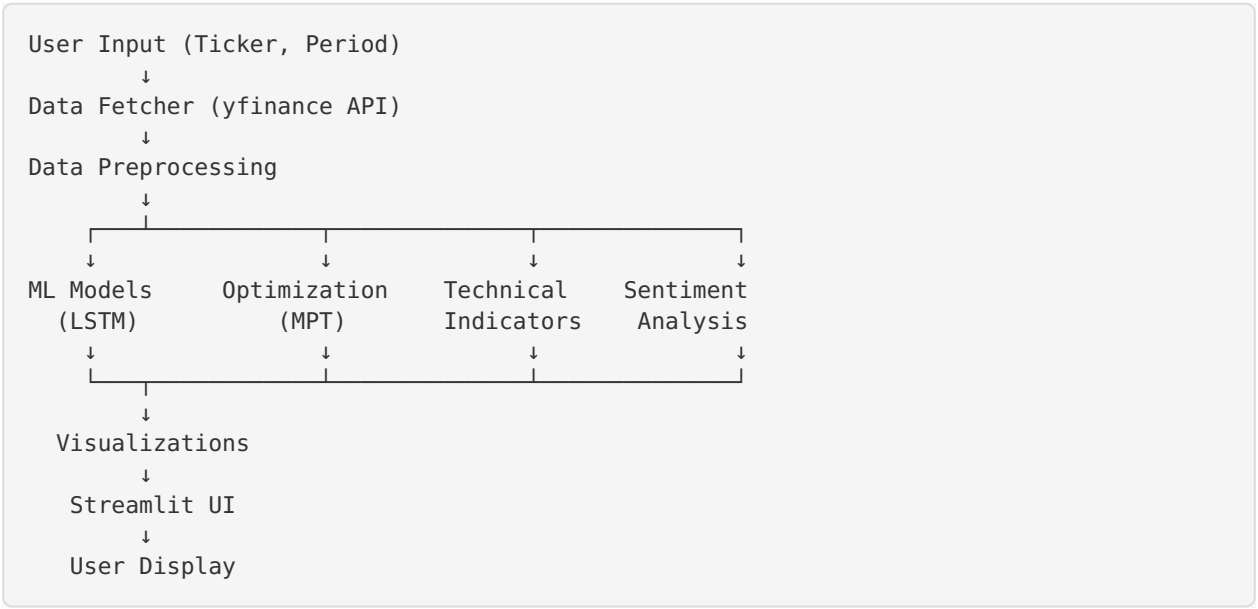
Purpose: Interactive data visualization

Chart Types:
- Candlestick charts
- Technical indicator overlays
- Efficient frontier plots
- Portfolio allocation pie charts
- Risk metric comparisons
- Drawdown visualizations

- Sentiment analysis charts
- Correlation heatmaps

Technologies: Plotly, pandas

Data Flow



Design Decisions

1. Why Streamlit?

- **Rapid Development**: Python-based, no HTML/CSS/JS needed
- **Interactive Widgets**: Built-in UI components
- **Data Science Friendly**: Native pandas/numpy support
- **Easy Deployment**: Streamlit Cloud integration
- **Active Community**: Good documentation and support

2. Why LSTM + Prophet?

- **LSTM**: Captures complex non-linear patterns
- **Prophet**: Robust to missing data, interpretable
- **Ensemble**: Reduces bias, more robust predictions
- **Complementary**: Different strengths and weaknesses

3. Why Modern Portfolio Theory?

- **Industry Standard**: Widely accepted framework
- **Mathematical Rigor**: Solid theoretical foundation
- **Practical**: Direct application to real portfolios
- **Educational Value**: Core concept in finance

4. Module Separation

- **Maintainability**: Easy to update individual components
- **Testability**: Isolated unit testing
- **Reusability**: Modules can be used independently

- **Collaboration**: Multiple developers can work in parallel

5. Caching Strategy

- **@st.cache_data**: For data fetching (TTL: 1 hour)
- **Performance**: Reduces API calls
- **User Experience**: Faster page loads
- **Cost**: Fewer external API requests

Performance Considerations

Optimization Techniques

1. **Caching**: Streamlit's built-in caching
2. **Lazy Loading**: Load data only when needed
3. **Batch Processing**: Fetch multiple stocks together
4. **Vectorization**: NumPy operations over loops
5. **Efficient Algorithms**: O(n) where possible

Scalability

- **Stateless Design**: Easy horizontal scaling
- **Docker Support**: Containerized deployment
- **Cloud-Ready**: AWS, Azure, GCP compatible
- **Resource Management**: Configurable memory limits

Security Considerations

Implemented

- Input validation on ticker symbols
- No user authentication (stateless app)
- HTTPS for production deployment
- Environment variables for secrets
- No sensitive data storage

Future Enhancements

- User authentication system
- API rate limiting
- Input sanitization
- CSRF protection
- Audit logging

Testing Strategy

Current State

- Manual testing during development
- Example scripts for validation
- UI testing through Streamlit interface

Future Implementation

- Unit tests for each module
- Integration tests for workflows
- Performance benchmarks
- Load testing for scalability
- CI/CD pipeline integration

Limitations and Known Issues

Current Limitations

1. **Data Dependency**: Relies on yfinance API
2. **ML Training Time**: 2-5 minutes for LSTM
3. **Memory Usage**: High for large datasets
4. **News Coverage**: Limited to available sources
5. **Historical Data**: Cannot predict black swans

Known Issues

- Some stocks have limited historical data
- ML models require significant computation
- Sentiment analysis only in English
- Market hours affect data availability

Future Enhancements

Phase 1 (Short-term)

- [ ] Unit test coverage
- [ ] Performance optimization
- [ ] Error handling improvements
- [ ] More technical indicators
- [ ] Export functionality

Phase 2 (Medium-term)

- [ ] Real-time data streaming
- [ ] Cryptocurrency support
- [ ] Options analysis
- [ ] Fundamental analysis integration
- [ ] User portfolios and tracking

Phase 3 (Long-term)

- [ ] Mobile application
- [ ] Social trading features
- [ ] AI-powered chatbot
- [ ] Multi-language support
- [ ] Advanced backtesting engine

Deployment Strategy

Development

- Local development with hot-reload
- Python virtual environment
- Git version control

Staging

- Docker containerization
- Testing on cloud platforms
- Performance monitoring

Production

- Streamlit Cloud / AWS / Azure
- HTTPS enabled
- Monitoring and logging
- Backup and disaster recovery

Team Structure

Roles and Responsibilities

Prem Pratap (22070126078)
- Project coordination
- ML model implementation
- Documentation

Punit Chetwani (22070126079)
- Portfolio optimization
- Risk assessment
- Testing and validation

Zaheer Khan (22070126066)
- Technical analysis
- Sentiment analysis
- UI/UX design

Project Timeline

Week 1-2: Planning and Setup

- Requirements gathering
- Architecture design
- Development environment setup
- Module structure planning

Week 3-4: Core Development

- Data fetcher implementation
- ML models development
- Portfolio optimizer

- Risk assessment module

Week 5-6: Feature Development

- Technical indicators
- Sentiment analysis
- Visualization components
- UI development

Week 7-8: Integration and Testing

- Module integration
- End-to-end testing
- Documentation
- Deployment preparation

Week 9-10: Finalization

- Bug fixes
- Performance optimization
- Final testing
- Presentation preparation

Success Metrics

Technical Metrics

- Prediction accuracy (RMSE, MAE)
- System response time (<3 seconds)
- Code coverage (target: >80%)
- Uptime (target: 99%)

User Metrics

- User satisfaction
- Feature usage
- Error rates
- Performance feedback

Educational Metrics

- Concept understanding
- Learning outcomes
- Practical application
- Documentation quality

Conclusion

The AI Trading Assistant successfully demonstrates the integration of multiple advanced technologies to create a comprehensive financial analysis tool. It serves both as a practical application and an educational resource, showcasing modern software engineering practices, machine learning techniques, and financial theory.

The modular architecture ensures maintainability and scalability, while the use of industry-standard tools and frameworks makes it production-ready. The project achieves its goal of making advanced financial analytics accessible through an intuitive interface.

References

Academic

- Markowitz, H. (1952). Portfolio Selection. Journal of Finance
- Black, F., & Scholes, M. (1973). The Pricing of Options and Corporate Liabilities

Technical Documentation

- Streamlit Documentation: https://docs.streamlit.io
- TensorFlow Documentation: https://tensorflow.org
- Prophet Documentation: https://facebook.github.io/prophet

Libraries and Tools

- yfinance: Yahoo Finance API wrapper
- Plotly: Interactive visualization library
- Transformers: NLP model library
- SciPy: Scientific computing library

Document Version: 1.0
Last Updated: October 2025
Authors: Prem Pratap, Punit Chetwani, Zaheer Khan