Technical Analysis RAG Agent

Simplified Architecture Overview

An agentic system that combines basic RAG with yfinance data to provide strategy improvement suggestions and stock screening.

Core Components

1. API Layer (FastAPI/Any)
python
Just 3 endpoints
POST /analyze-strategy   # Strategy improvement suggestions
POST /screen-stocks   # Find bullish stocks
GET /health   # Health check

2. Three Core Agents

Coordinator Agent
- Routes requests to appropriate handlers
- Combines responses from other agents
- Simple request/response format

TA Knowledge Agent
- Searches local vector database of TA PDFs
- Uses sentence-transformers (free embedding model)
- Simple similarity search, no complex RAG

Market Data Agent
- Fetches data via yfinance (free tier)
- Basic technical indicator calculations
- Limited to S&P 100 stocks to reduce API calls

Data Requirements

Document Collection
- *Size*: 10-15 technical analysis PDF books
- *Storage*: Local ChromaDB (free)
- *Processing*: One-time embedding using sentence-transformers
- *Examples*: "Technical Analysis of Stock Trends", "Japanese Candlestick Charting"

Market Data
- *Source*: yfinance (completely free)

- *Scope*: S&P 100 stocks only (reduces data volume)
- *History*: 1 year max to minimize storage
- *Caching*: Simple file-based caching

Workflows

Strategy Analysis Flow
1. *Input*: User describes strategy + win rate + average return
2. *Knowledge Search*: Find similar strategies in PDF collection
3. *Market Context*: Get recent performance of 2-3 relevant stocks
4. *Output*: 3-5 bullet points with improvement suggestions

Stock Screening Flow
1. *Input*: User specifies 2-3 technical indicators
2. *Data Fetch*: Get data for S&P 100 stocks only
3. *Calculate*: Basic indicators (RSI, MACD, SMA)
4. *Output*: Top 10 bullish stocks with confidence scores

Tech Stack

Core Technologies
- *API*: FastAPI/Any
- *LLM*: OpenAI GPT-3.5-turbo ($0.50/1M tokens)/others
- *Embeddings*: sentence-transformers/openai
- *Vector DB*: ChromaDB/Pinecone
- *Market Data*: yfinance (free)
- *Indicators*: pandas-ta (free)

Infrastructure
- *Development*: Local development only
- *Database*: SQLite for simple user data
- *Caching*: Simple pickle files (Not required to implement)
- **No Docker/Kubernetes needed

Cost Optimization Strategies

1. Use GPT-3.5-turbo (10x cheaper than GPT-4)

2. Reduce Data Volume
- Only S&P 100 stocks (not full market)
- 1-year historical data maximum
- Daily data only (no intraday)
- Basic indicators only (RSI, MACD, SMA, Volume)

3. Local Processing
- Run embeddings locally (sentence-transformers)/Openai embeddings
- Local vector database (ChromaDB)/Pinecone
- SQLite instead of PostgreSQL/Supabase

Example API Usage

Strategy Analysis Example
bash
```
curl -X POST "http://localhost:8000/analyze-strategy" \
  -H "Content-Type: application/json" \
  -d '{
    "strategy": "Buy when RSI < 30, sell when RSI > 70",
    "win_rate": 0.65,
    "avg_return": 0.08,
    "sample_stocks": ["AAPL", "MSFT"]
  }'
```

Simple Response:
json
```
{
  "improvements": [
    "Add volume confirmation to reduce false RSI signals",
    "Consider 14-period RSI instead of default for less noise",
    "Set stop-loss at 3% to improve risk/reward ratio"
  ],
  "similar_strategies": ["RSI Divergence", "RSI with Moving Average"],
  "recent_performance": {
    "AAPL": {"rsi": 45.2, "trend": "neutral"},
    "MSFT": {"rsi": 38.1, "trend": "oversold"}
  }
}
```

Stock Screening Example
bash
```
curl -X POST "http://localhost:8000/screen-stocks" \
  -H "Content-Type: application/json" \
  -d '{
    "criteria": {
      "rsi_max": 40,
      "macd_signal": "bullish_crossover",
      "volume_increase": true
```

```
  }
 }'
```

Simple Response:

```
json
{
  "bullish_stocks": [
    {"symbol": "AAPL", "score": 8.5, "rsi": 35.2, "reason": "RSI oversold + MACD crossover"},
    {"symbol": "GOOGL", "score": 7.8, "rsi": 38.9, "reason": "Strong volume + RSI recovery"},
    {"symbol": "TSLA", "score": 7.2, "rsi": 39.1, "reason": "MACD bullish + volume spike"}
  ],
  "total_screened": 100,
  "timestamp": "2024-01-15T10:30:00Z"
}
```

File Structure

```
technical-analysis-rag/
├── app/
│   ├── main.py             # FastAPI app
│   ├── agents/
│   │   ├── coordinator.py   # Main routing logic
│   │   ├── knowledge.py     # PDF search agent
│   │   └── market_data.py   # yfinance integration
│   ├── data/
│   │   ├── pdfs/           # TA book PDFs
│   │   ├── vectordb/       # ChromaDB storage
│   │   └── cache/          # Simple file cache
│   └── utils/
│       ├── indicators.py   # Technical calculations
│       └── embeddings.py   # Local embedding functions
├── requirements.txt        # Minimal dependencies
├── setup.py               # Simple setup script
└── README.md
```

Estimated Monthly Costs

Development Phase
- *OpenAI API*: $10-20/month (with careful usage)
- *Compute*: $0 (local development)
- *Storage*: $0 (local files)

- *Total*: $10-20/month

Key Cost Controls
- Limit to 1000 API calls per day
- Cache responses for 24 hours
- Use shortest possible prompts
- Only process top 100 stocks
- Local embedding model (no API costs)

Testing Strategy

Basic Tests
1. *test_strategy_analysis*: Verify strategy improvement suggestions
2. *test_stock_screening*: Check bullish stock detection
3. *test_knowledge_retrieval*: Ensure PDF search works
4. *test_market_data*: Confirm yfinance integration
5. *test_cost_limits*: Ensure API usage stays under budget

Success Criteria (Simplified)
- *Functionality*: System provides relevant strategy tips and finds bullish stocks
- *Cost*: Under $25/month during development
- *Performance*: Responses under 30 seconds
- *Accuracy*: Suggestions make sense based on TA principles
- *Learning*: Demonstrates agentic workflow concepts