Agentic Trading Challenge

Technical Interview — Backend / AI Engineering

February 18, 2026

Contents

Overview

In this challenge you will build an **agentic stock analysis workflow** in Go. Your program will:

1. Accept a stock ticker symbol (e.g. `AAPL`) as a command-line argument.
2. Use the **DeepSeek Chat** LLM to autonomously decide which tools to call.
3. Execute tool calls (get price data, fetch news, analyze sentiment, retrieve historical prices, financial metrics, insider trades, and analyst ratings) and feed results back to the LLM.
4. Repeat the loop until the LLM produces a final **Buy / Sell / Hold** recommendation.
5. Output a structured JSON `AnalysisReport`.

The high-level flow is:

$$\texttt{User Input} \rightarrow \texttt{LLM} \rightleftharpoons \texttt{Tools} \rightarrow \texttt{Final Report (JSON)}$$

You are provided with a project skeleton that includes mock tool implementations, data types, and a partially implemented LLM client. **Your job is to complete the agent loop and the tool-call parsing logic.**

Technical Constraints

- **Language**: Go 1.22+ (standard library only — no external dependencies)
- **LLM**: DeepSeek Chat (`deepseek-chat`) via OpenAI-compatible REST API
- **No AI frameworks**: Do not use LangChain, LlamaIndex, or similar. Manual function calling only.
- **Output**: Structured JSON to stdout; errors to stderr

Provided vs. Candidate-Implemented

Component	Status	File
Data types	Provided	`internal/models/types.go`
Mock tools (7 tools)	**Candidate**	`internal/tools/tools.go`
LLM client (HTTP call)	Provided	`internal/llm/client.go`
Tool-call parsing	**Candidate**	`internal/llm/client.go`
Tool dispatch (executeTool)	**Candidate**	`internal/agent/agent.go`
Agent loop	**Candidate**	`internal/agent/agent.go`
System prompt design	**Candidate**	`internal/agent/agent.go`
CLI entry point	Provided	`cmd/analyzer/main.go`

Available Tools

The following mock tools are provided in `internal/tools/tools.go`. They return deterministic data so you can focus on the agent logic.

GetStockPrice

```
func GetStockPrice(symbol string) (*models.StockPrice, error)
```

JSON schema for tool call:

```
{
  "name": "GetStockPrice",
  "arguments": { "symbol": "AAPL" }
}
```

Returns: symbol, price, change, change_pct, volume, high, low.

GetNews

```
func GetNews(symbol string) ([]models.NewsItem, error)
```

JSON schema for tool call:

```
{
  "name": "GetNews",
  "arguments": { "symbol": "AAPL" }
}
```

Returns: array of news items with title, source, summary, url, published.

AnalyzeSentiment

```
func AnalyzeSentiment(text string) (*models.SentimentResult, error)
```

JSON schema for tool call:

```
{
  "name": "AnalyzeSentiment",
  "arguments": { "text": "Apple Reports Record Q4 Revenue..." }
}
```

Returns: text, sentiment (positive/negative/neutral), confidence.

GetHistoricalPrices

```
func GetHistoricalPrices(symbol string) ([]models.HistoricalPrice,
    error)
```

JSON schema for tool call:

```
{
  "name": "GetHistoricalPrices",
  "arguments": { "symbol": "AAPL" }
}
```

Returns: array of 5 entries with date, open, high, low, close, volume.

GetFinancials

```go
func GetFinancials(symbol string) (*models.FinancialData, error)
```

JSON schema for tool call:

```
{
  "name": "GetFinancials",
  "arguments": { "symbol": "AAPL" }
}
```

Returns: symbol, revenue, net_income, eps, pe_ratio, market_cap, dividend_yield.

GetInsiderTrading

```go
func GetInsiderTrading(symbol string) ([]models.InsiderTrade, error)
```

JSON schema for tool call:

```
{
  "name": "GetInsiderTrading",
  "arguments": { "symbol": "AAPL" }
}
```

Returns: array of insider trades with name, title, trade_type, shares, price, date.

GetAnalystRatings

```go
func GetAnalystRatings(symbol string) ([]models.AnalystRating, error)
```

JSON schema for tool call:

```
{
  "name": "GetAnalystRatings",
  "arguments": { "symbol": "AAPL" }
}
```

Returns: array of analyst ratings with firm, rating, price_target, date.

Expected Workflow

The agent should follow this loop:

1. **Initialize**: Create a conversation with a system prompt that describes the agent's role and available tools.
2. **User message**: Add a user message asking the LLM to analyze a specific stock symbol.
3. **LLM call**: Send the conversation to the DeepSeek API.
4. **Parse response**: Check if the LLM's response contains a tool call (JSON with `name` and `arguments`).
5. **If tool call**: Execute tool calls (get price, news, sentiment, history, financials, insider trades, analyst ratings), append the result as a `tool` message, and go to step 3.

6. **If final answer**: Parse the response as an `AnalysisReport` and return it.
7. **Safety**: Stop after a maximum number of iterations to prevent infinite loops.

Important

The agent must be **autonomous**. The LLM decides which tools to call and in what order. Do **not** hardcode a fixed sequence of tool calls (e.g. "always call GetStockPrice first, then GetNews"). The LLM should drive the workflow.

Expected Output Format

The final output should be a JSON `AnalysisReport` printed to stdout:

```
{
  "symbol": "AAPL",
  "recommendation": "Buy",
  "confidence": 0.85,
  "reasoning": "Strong Q4 earnings, positive analyst sentiment, and
    upward price momentum suggest continued growth.",
  "price": {
    "symbol": "AAPL",
    "price": 178.72,
    "change": 2.35,
    "change_pct": 1.33,
    "volume": 52341000,
    "high": 179.50,
    "low": 175.80
  },
  "news": [ ... ],
  "sentiment": {
    "text": "...",
    "sentiment": "positive",
    "confidence": 0.85
  }
}
```

Example Scenarios

Symbol	Expected	Reasoning
AAPL	Buy	Positive earnings, analyst upgrades, price up 1.33%
GOOGL	Hold	Mixed signals: strong cloud growth but antitrust concerns
TSLA	Buy	Deliveries beat expectations, strong momentum (+3.41%)
NVDA	Sell	High P/E ratio (72.6), declining price trend, insider selling
META	Buy	Strong ad revenue growth, Reality Labs turnaround, analyst upgrades
AMZN	Hold	AWS growth slowing, flat price action, mixed analyst views

Implementation Guide

Manual Function Calling

Since DeepSeek's API is OpenAI-compatible but we are implementing function calling manually, here is the approach:

6.1.1 System Prompt Structure

Your system prompt should clearly define:

- The agent's role (stock analyst)
- Each available tool: name, parameters, JSON call format
- Instructions: respond with tool-call JSON when needing data
- Instructions: respond with final `AnalysisReport` JSON when done

6.1.2 Parsing Strategy

The LLM may return a tool call in several formats:

- Raw JSON: `{"name": "GetStockPrice", "arguments": {...}}`
- Inside markdown code fences: "‘`json ... "‘`
- With surrounding explanatory text

Your `ParseToolCall` function should handle all of these cases. A robust approach:

```
func (c *Client) ParseToolCall(content string) (*models.ToolCall, error
    ) {
    // 1. Strip markdown code fences if present
    // 2. Find JSON substring containing "name" and "arguments"
    // 3. Unmarshal into models.ToolCall
    // 4. Return nil if no tool call found (it's a final answer)
    return nil, nil
}
```

6.1.3 Tool Execution & Result Injection

After parsing a tool call:

1. Call the corresponding function from `internal/tools/`
2. Marshal the result to JSON
3. Append to conversation as a message with role `"tool"` and the tool's name
4. Send the updated conversation back to the LLM

Getting Started

1. **Clone the repository**:

```
git clone <repo-url>
cd stock-analyzer
```

2. **Configure your API key**:

```
export DEEPSEEK_API_KEY="your-key-here"
```

3. **Build**:

```
make build
```

4. **Run**:

```
./bin/analyzer AAPL
```

The skeleton compiles out of the box. Your task is to implement the 7 mock tool functions, the tool dispatch in `executeTool`, and the `ParseToolCall` method.

Evaluation Criteria

Automated Tests (Pass / Fail)

The repository includes a test suite. Run it with:

```
make test
```

Out of the box, **7 tests pass** and **33 tests fail**. Your goal is to make all 40 tests pass.

Package	File	Initial	Tests
`internal/tools`	`tools_test.go`	18 fail	Mock tools: 7 functions × 6 stocks
`internal/llm`	`client_test.go`	4 pass / 3 fail	ParseToolCall: raw JSON, code fences, surrounding
`internal/agent`	`agent_test.go`	1 pass / 12 fail	Tool dispatch, agent loop, conversation

Acceptance Checklist

Your submission passes if **all** of the following hold:

1. `make build` compiles without errors
2. `make test` — all 40 tests pass
3. `./bin/analyzer AAPL` produces valid `AnalysisReport` JSON to stdout (with a valid API key)
4. The agent makes ≥ 2 tool calls before producing a final answer (not hardcoded)
5. `recommendation` is one of `Buy`, `Sell`, or `Hold`
6. `confidence` is in the range $[0.0, 1.0]$
7. `reasoning` is a non-empty string

Scoring Rubric

Criterion	Weight	Description
Agent Loop	30%	Correct agentic loop: LLM drives tool selection, multi-turn conversation, termination condition
Function Calling	25%	Robust parsing of tool calls from LLM responses, correct dispatch and result injection
Code Quality	20%	Clean, idiomatic Go; proper error handling; clear naming
Output	15%	Correct JSON format; reasonable recommendation with supporting data
Edge Cases	10%	Handles unknown symbols, malformed LLM responses, API errors gracefully

Bonus (Optional)

If you finish early, consider implementing any of the following:

- **Streaming**: Use SSE streaming for the DeepSeek API and display partial responses in real time.
- **Retries with backoff**: Implement exponential backoff for transient API errors (429, 500, 503).
- **Multi-turn memory**: Allow the agent to remember previous analyses across runs (e.g. file-based cache).
- **Concurrent tool calls**: Use goroutines to execute independent tool calls in parallel (e.g. fetch price and news concurrently).
- **Additional tests**: Extend the test suite with your own edge-case tests.

Appendix A: Mock Tool Data

The tools return deterministic data. Below are the exact outputs for the six supported symbols.

GetStockPrice

Symbol	Price	Change	Change%	Volume	High / Low
AAPL	178.72	+2.35	+1.33%	52.3M	179.50 / 175.80
GOOGL	141.80	−0.95	−0.67%	23.1M	143.20 / 140.55
TSLA	248.50	+8.20	+3.41%	98.8M	251.00 / 239.80
NVDA	875.30	−12.40	−1.40%	45.2M	890.00 / 870.15
META	505.75	+15.30	+3.12%	32.8M	510.20 / 489.60
AMZN	178.25	+0.75	+0.42%	28.5M	179.80 / 176.90
Other	100.00	+0.50	+0.50%	10.0M	101.00 / 99.00

GetNews

AAPL (3 articles):

1. "Apple Reports Record Q4 Revenue" (Reuters) — strong iPhone and Services growth
2. "Apple Announces New AI Features" (Bloomberg) — on-device AI capabilities
3. "Analysts Raise Apple Price Target" (CNBC) — price targets raised after earnings

GOOGL (2 articles):

1. "Google Cloud Revenue Surges 28%" (Reuters) — beating analyst estimates
2. "DOJ Antitrust Case Update" (WSJ) — critical phase of antitrust case

TSLA (2 articles):

1. "Tesla Deliveries Beat Expectations" (Bloomberg) — 484k vehicles, beat 460k estimate
2. "Tesla Expands Charging Network" (Electrek) — doubling Supercharger network

NVDA (3 articles):

1. "NVIDIA AI Chip Demand Plateaus" (Reuters)
2. "NVIDIA CEO Sells $50M in Stock" (Bloomberg)
3. "NVIDIA Valuation Concerns Rise" (WSJ)

META (2 articles):

1. "Meta Reality Labs Shows Turnaround" (Bloomberg)
2. "Meta Ad Revenue Grows 25%" (Reuters)

AMZN (2 articles):

1. "AWS Growth Slows to 12%" (CNBC)
2. "Amazon Expands Same-Day Delivery" (Reuters)

Other symbols: 1 generic article ("`<SYMBOL>` Trades Steady").

AnalyzeSentiment

Sentiment is determined by keyword matching:

- **Positive** keywords: record, beat, surge, raise, growth, strong, exceeded, new high
- **Negative** keywords: decline, drop, fall, loss, miss, antitrust, lawsuit, warning
- If positive count > negative ⇒ `"positive"`, and vice versa. Tie ⇒ `"neutral"`.

GetHistoricalPrices

Returns 5 most recent trading days (oldest first). The last entry's close matches the current price. AAPL shows an upward trend. NVDA shows a downward trend.

GetFinancials

Symbol	Revenue (B)	Net Income (B)	EPS	P/E	Market Cap (B)	Div Yield
AAPL	383.28	97.00	6.42	27.8	2800.00	0.55%
GOOGL	307.39	73.80	5.80	24.4	1780.00	0.50%
TSLA	96.77	15.00	4.73	52.5	790.00	0.00%
NVDA	60.92	29.76	12.06	72.6	2150.00	0.03%
META	134.90	39.10	15.02	33.7	1300.00	0.40%
AMZN	574.78	30.42	2.90	61.5	1870.00	0.00%
Other	10.00	1.00	1.00	15.0	50.00	0.00%

GetInsiderTrading

- **AAPL**: 2 Buy trades (bullish signal)
- **GOOGL**: 1 Sell trade
- **TSLA**: 1 Buy trade
- **NVDA**: 2 Sell trades (bearish signal)
- **META**: 1 Buy trade
- **AMZN**: 1 Sell trade
- Unknown symbols: empty list

GetAnalystRatings

- **AAPL**: 3 ratings (2 Buy, 1 Hold) — majority bullish
- **GOOGL**: 2 ratings (1 Hold, 1 Buy) — mixed
- **TSLA**: 2 ratings (1 Buy, 1 Hold) — mixed bullish
- **NVDA**: 3 ratings (1 Sell, 1 Hold, 1 Buy) — mixed bearish
- **META**: 2 ratings (2 Buy) — bullish
- **AMZN**: 2 ratings (2 Hold) — neutral
- Unknown symbols: empty list

Appendix B: DeepSeek API Reference

Official Documentation

- Getting Started: https://api-docs.deepseek.com/
- Chat Completion API: https://api-docs.deepseek.com/api/create-chat-completion

Endpoint

```
POST https://api.deepseek.com/chat/completions
```

Note: The path `/v1/chat/completions` also works for OpenAI SDK compatibility, but `/v1` has no relationship to the model version.

Headers

```
Content-Type: application/json
Authorization: Bearer <DEEPSEEK_API_KEY>
```

Available Models

Model ID	Description
`deepseek-chat`	DeepSeek-V3 (use this for the challenge)
`deepseek-reasoner`	DeepSeek-V3 with chain-of-thought reasoning

Request Body

```
{
  "model": "deepseek-chat",
  "messages": [
    {"role": "system", "content": "You are a stock analyst..."},
    {"role": "user", "content": "Analyze AAPL"},
    {"role": "assistant", "content": "{\"name\": \"GetStockPrice\",
       ...}"},
    {"role": "tool", "name": "GetStockPrice", "content": "{...}"}
  ]
}
```

Response Body

```
{
  "id": "chatcmpl-abc123",
  "object": "chat.completion",
  "model": "deepseek-chat",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "..."
      },
```

```
      "finish_reason": "stop"
    }
  ],
  "usage": {
    "prompt_tokens": 256,
    "completion_tokens": 64,
    "total_tokens": 320
  }
}
```

curl Example

```
curl -X POST https://api.deepseek.com/chat/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $DEEPSEEK_API_KEY" \
  -d '{
    "model": "deepseek-chat",
    "messages": [
      {"role": "user", "content": "Hello, who are you?"}
    ]
  }'
```