Financial News Impact Analyzer: Multi-Agent System (Draconic AI Case Study)

---

1. System Architecture: Why Multiple Agents?

To effectively assess the financial impact of news articles, we adopted a **modular multi-agent architecture**. This system decomposes the problem into three independent analytical dimensions:

- **Sentiment Analysis** — Understands the emotional and tonal direction of the article.
- **Market Impact Prediction** — Evaluates the economic consequences the article may have on financial markets.
- **Risk Flagging** — Identifies forward-looking uncertainty, regulatory risks, and ethical concerns.

Each agent is a self-contained AI model powered by `pydantic-ai`, using a specialized `system_prompt` and structured `output_type`. This promotes separation of concerns and allows us to fine-tune and improve individual agents without impacting others.

Why Not a Single Agent?

- A single monolithic prompt would lack modularity, reduce explainability, and increase cognitive load on the LLM.
- Specialized agents are easier to prompt-engineer and evaluate.
- Multi-agent orchestration allows independent evolution and reuse of agents in other domains (e.g., general news, healthcare, or legal).

---

2. Design Decisions and Rationale

Agent Design

Each agent is implemented using `pydantic_ai.Agent()` with a:

- **System prompt** that simulates the role (e.g., "You are a financial sentiment analyzer")
- **Pydantic BaseModel** that defines the structured response

```
sentiment_agent = Agent(
    model=OpenAIModel(
        "gpt-4o",
        provider=OpenAIProvider(api_key=os.getenv("OPENAI_API_KEY"))
    ),
    output_type=SentimentOutput,
    system_prompt="You are a financial sentiment analyzer..."
)
```

Orchestration

The agents are orchestrated synchronously within a function `analyze_article()` in `main.py`. This function handles prompt delegation and output aggregation.

```
result = AnalysisOutput(
    sentiment=sentiment_agent.run_sync(article).output,
    market_impact=market_agent.run_sync(article).output,
    risks=risk_agent.run_sync(article).output
)
```

File Organization

- `agents/` directory: agent definitions
- `evaluation/`: test cases and test runner
- `.env`: API key management
- `main.py`: agent coordination

---

3. Prompt Engineering Iterations

Sentiment Agent

Prompt v1:

Classify article as positive, negative, or neutral.

Limitation: Output sometimes ambiguous or verbose.

Prompt v2:

Classify the sentiment as one of 'positive', 'neutral', or 'negative'. Provide a score between 0–1 and a short reasoning (max 2 lines).

Improvement: Now consistent, concise, and aligned with output type.

Prompt v3:

Emphasize investor sentiment and financial implications (added as system-level context).

Market Impact Agent

Prompt v1:

Estimate if the article has high, medium, or low market impact.

Prompt v2:

Include rationale + confidence score. Focus on company size, earnings, regulatory risks, or innovation.

Prompt v3:

Improved handling of ambiguous articles by instructing the model to make best-effort judgments when impact is unclear.

Risk Agent

Prompt v1:

Identify if article contains risk.

Prompt v2:

Explicitly instruct to return booleans for: forward-looking uncertainty, regulatory, and ethical risks.

Prompt v3:

Include "short explanation" for each True flag. Prevented hallucinated ethical risks.

---

4. Test Case Behaviors (Highlights)

✅FIN-001: Tesla Record Profits

- **Sentiment:** Positive
- **Impact:** High
- **Risk:** Forward-looking uncertainty (Musk's warning)

□□ FIN-002: CureGen FDA Approval

- Sentiment: Positive
- Impact: Medium (despite FDA approval — skepticism handled well)
- Ethical Risk: False (correctly ignored speculation)

✅FIN-003: Amazon AGI Announcement

- Sentiment: Neutral
- Impact: High
- Risk: Regulatory flagged, forward-looking flagged
- **Highlight:** Balanced reporting despite futuristic tone

□□ FIN-005: ByteDance Growth

- Sentiment: Positive
- Impact: Medium
- Regulatory Risk: Correctly flagged due to "regulatory clouds"

---

5. Evaluation Framework

Metrics Implemented (in `run_tests.py`):

- **Response Consistency:** All agents returned valid structured results
- **Sentiment vs Impact Agreement:** Positive sentiment often correlated with high/medium impact (expected trend)
- **Risk Coverage:** All 5 test cases triggered at least 1 boolean risk flag — validating model coverage

---

6. Final Thoughts

What Worked

- Agents were independently debuggable
- Prompt iterations improved reliability
- Using `pydantic-ai` allowed strict validation, reducing hallucinations

What Didn't Work Initially

- Relying on default model in `settings` didn’t always propagate — fixed by using `OpenAIModel()` per agent
- Early prompts were too broad and under-specified

- Ethical risk was often misclassified until prompt clarified it must be "explicitly mentioned"

✅Submission Ready

- Modular, testable architecture
- Compliant with evaluation framework
- All 5 test cases processed with structured output
- Easy to extend or plug into real-time financial analysis pipeline