AI Engineering Assignment: RAG Sprint Challenge

Build a Financial Q&A System with Agent Capabilities

Overview

Build a focused RAG system with basic agent capabilities that can answer both simple and comparative financial questions about Google, Microsoft, and NVIDIA using their recent 10-K filings. The system should demonstrate query decomposition and multi-step reasoning for complex questions.

This assignment tests: - **Vector-based RAG implementation** - **Agent orchestration for query decomposition** - **Multi-step retrieval and synthesis** - **Clean engineering practices**

Time Expectation: A good engineer should be able to complete this in 2-3 hours.

Data Scope

Requirement	Details
Companies	Google (GOOGL), Microsoft (MSFT), NVIDIA (NVDA)
Documents	Annual 10-K filings only
Years	2022, 2023, 2024
Total Files	9 documents (3 companies $\times$ 3 years)
Source	SEC EDGAR database

Company CIK Codes (for SEC API): - GOOGL: 1652044 - MSFT: 789019 - NVDA: 1045810

Core Requirements

1. Data Acquisition (30 minutes)

- Obtain 10-K filings from SEC EDGAR
- **Option A**: Build a scraper/downloader (recommended - shows data engineering skills)
- **Option B**: Manually download filings if you prefer to focus on RAG/agents
- You can use PDF, HTML, or XML format - your choice
- Store locally for processing
- Example SEC API endpoint: `https://www.sec.gov/Archives/edgar/data/{CIK}/{ACCESSION_NUMBER`

2. RAG Pipeline

- **Text Extraction**: Parse filings to extract text

- **Chunking**: Split into semantic chunks (200-1000 tokens)
- **Embeddings**: Use any embedding model (OpenAI, Cohere, open-source)
- **Vector Store**: Implement vector search (can use in-memory for simplicity)
- **Retrieval**: Return top-k relevant chunks for queries

3. Query Engine with Agent Capabilities Build an agentic system that can:

Core Capabilities: - **Query Decomposition**: Break complex questions into sub-queries - **Multi-step Retrieval**: Execute multiple searches to answer comparative questions
- **Synthesis**: Combine results from multiple retrievals into coherent answers

Example Query Patterns:

1. **Simple Direct Query**:
   - "What was Microsoft's total revenue in 2023?"
   - Single retrieval → Answer
2. **Comparative Query** (requires decomposition):
   - "How did NVIDIA's data center revenue grow from 2022 to 2023?"
   - Agent breaks down:
     - → Find NVIDIA data center revenue 2022
     - → Find NVIDIA data center revenue 2023
     - → Calculate growth
3. **Cross-Company Analysis** (multiple retrievals):
   - "Which company had the highest operating margin in 2023?"
   - Agent executes:
     - → Retrieve MSFT operating margin 2023
     - → Retrieve GOOGL operating margin 2023
     - → Retrieve NVDA operating margin 2023
     - → Compare and determine highest
4. **Complex Multi-aspect**:
   - "Compare cloud revenue growth rates across all three companies from 2022 to 2023"
   - Agent orchestrates multiple sub-queries and calculations

Required Query Types to Support: 1. **Basic Metrics**: "What was Microsoft's total revenue in 2023?" 2. **YoY Comparison**: "How did NVIDIA's data center revenue grow from 2022 to 2023?" 3. **Cross-Company**: "Which company had the highest operating margin in 2023?" 4. **Segment Analysis**: "What percentage of Google's revenue came from cloud in 2023?" 5. **AI Strategy**: "Compare AI investments mentioned by all three companies in their 2024 10-Ks"

4. Output Format Return JSON responses with sources:

```
{
  "query": "Which company had the highest operating margin in 2023?",
  "answer": "Microsoft had the highest operating margin at 42.1% in 2023, followed by Google
  "reasoning": "Retrieved operating margins for all three companies from their 2023 10-K fil
  "sub_queries": [
    "Microsoft operating margin 2023",
    "Google operating margin 2023",
    "NVIDIA operating margin 2023"
  ],
  "sources": [
    {
      "company": "MSFT",
      "year": "2023",
      "excerpt": "Operating margin was 42.1%...",
      "page": 10,
    },
    {
      "company": "GOOGL",
      "year": "2023",
      "excerpt": "Operating margin of 29.8%...",
      "page": 42,
    },
    {
      "company": "NVDA",
      "year": "2023",
      "excerpt": "We achieved operating margin of 29.6%...",
      "page": 37,
    }
  ]
}
```


Technical Guidelines

What to Build

- **Interface**: Simple CLI or Jupyter notebook
- **LLM**: Use any free tier (Groq, Gemini, OpenAI trial)
- **Framework**: LangChain/LlamaIndex optional - vanilla Python is fine
- **Vector DB**: In-memory is acceptable, or use lite solutions (ChromaDB, FAISS)

What NOT to Build

- Multi-turn conversations (but DO support multi-step reasoning within a single query)
- Production deployment/APIs

- Fancy UI
- Authentication/user management
- Complex error handling for every edge case
- Unit tests

Evaluation Rubric

Category	Weight	What We Look For
RAG Implementation	30%	Effective chunking, retrieval accuracy, embedding choice
Agent Orchestration	30%	Query decomposition, multi-step reasoning, result synthesis
Query Accuracy	20%	Correctly answers the 5 query types with proper sources
Code Quality	15%	Clean, readable code with clear structure
Documentation	5%	README with setup instructions and design choices

Bonus Points: - Automated SEC filing downloader/scraper (+5%) - Handling of edge cases and ambiguous queries (+5%) - Creative agent patterns or optimizations (+5%)

Deliverables

1. **Code Repository**
   - Source code with requirements.txt
   - Sample output for all 5 query types
   - Basic README with setup instructions
2. **Quick Demo**
   - Show your system answering both simple and comparative queries
   - Demonstrate agent decomposition for at least one complex query
   - Can be a notebook, script output, or short video (<3 min)
3. **Brief Design Doc** (1 page max)
   - Your chunking strategy
   - Embedding model choice & why
   - Agent/query decomposition approach
   - Any interesting challenges/decisions

Sample Test Queries

```
test_queries = [
    # Simple queries
    "What was NVIDIA's total revenue in fiscal year 2024?",
    "What percentage of Google's 2023 revenue came from advertising?",
```

```
    # Comparative queries (require agent decomposition)
    "How much did Microsoft's cloud revenue grow from 2022 to 2023?",
    "Which of the three companies had the highest gross margin in 2023?",

    # Complex multi-step queries
    "Compare the R&D spending as a percentage of revenue across all three companies in 2023'
    "How did each company's operating margin change from 2022 to 2024?",
    "What are the main AI risks mentioned by each company and how do they differ?"
]
```

Agent Implementation Tips

1. **Query Decomposition Pattern**:

   ```
   # Example: "Compare cloud revenue growth for all companies 2022-2023"
   sub_queries = [
       "Microsoft cloud revenue 2022",
       "Microsoft cloud revenue 2023",
       "Google cloud revenue 2022",
       "Google cloud revenue 2023",
       "NVIDIA data center revenue 2022",
       "NVIDIA data center revenue 2023"
   ]
   ```

2. **Agent Architecture Options**:
   - Simple function-based routing
   - LangChain agents with tools
   - Custom prompt-based decomposition
   - State machine for complex flows

3. **Key Challenges**:
   - Identifying when decomposition is needed
   - Maintaining context across sub-queries
   - Synthesizing multiple results coherently
   - Handling missing or ambiguous data

Quick Start Hints

1. **SEC Filing URLs** follow this pattern:
   - Base: `https://www.sec.gov/Archives/edgar/data/`
   - Find filing links via: `https://www.sec.gov/cgi-bin/browse-edgar?CIK={CIK}&type=10-K`

2. **Useful Python Libraries**:

   ```
   # For PDF: PyPDF2, pdfplumber, docling, unstructured.io
   # For HTML: BeautifulSoup, lxml
   ```

```
# For embeddings: openai, cohere, sentence-transformers, google gemini embeddings
# For vectors storage: faiss-cpu, chromadb, numpy, many others
```

3. **Focus Areas**:
    - Parsing the "Item 7 - MD&A" section often has key financial metrics
    - Income statement data is usually in "Item 8 - Financial Statements"
    - Risk factors and AI discussions in "Item 1A - Risk Factors"

Submission

- **Submit**: GitHub repo link + demo
- **Run Command**: Should work with `pip install -r requirements.txt && python main.py`

FAQ

Q: Can I use cached/pre-downloaded filings?
A: Building a downloader is encouraged but not required. If you prefer to focus on RAG/agents, feel free to download 10-Ks directly from SEC or the companies' investor relations website.

Q: Do I need to parse financial tables?
A: Table parsing is bonus but not required.

Q: How accurate should financial numbers be?
A: Exact as they appear in filings - no need to normalize or convert.

Q: Can I hardcode the 5 queries?
A: The system should work for similar queries, not just the exact ones provided.

Q: How sophisticated should the agent be?
A: It should handle query decomposition for comparative questions. Simple routing logic is fine - no need for complex reasoning chains or memory.

Q: What if a query needs data not in the filings?
A: Return what you can find and indicate if information is missing.

Good luck! We're looking for clean, practical solutions that demonstrate your understanding of RAG fundamentals. Remember: done is better than perfect for this sprint challenge.