Multi-Agent Financial Analysis System - Design Document

Version: 1.0 | **Date**: 5 October 2025 | **Type**: LLD

1. System Overview

A multi-agent AI system that analyzes US public companies by combining SEC 10-K filings with real-time market data. Uses specialized agents coordinated by an orchestrator, advanced RAG for document retrieval, and dual-memory architecture for personalization.

Core Stack: phidata (orchestration) | OpenAI GPT-4o | ChromaDB (vectors) | yfinance (market data) | SQLite (persistence)

2. Agent Roles

Orchestrator (orchestrator.py)

Purpose: Multi-agent coordinator
Key Function: create_plan() decomposes queries into DAG of agent tasks, execute_plan() runs agents sequentially based on dependencies
Example Plan:

```
{"steps": [
  {"agent": "sec_researcher", "dependencies": []},
  {"agent": "market_data", "dependencies": []},
  {"agent": "analyst", "dependencies": ["sec_researcher", "market_data"]},
  {"agent": "auditor", "dependencies": ["analyst"]}
]}
```

SEC Researcher (sec_researcher.py)

Purpose: 10-K filing analyst using RAG
Key Functions: analyze_risks(ticker, years) → risk factors by year | get_financial_trends() → revenue/margin/cash flow
RAG Strategy: Query decomposition → Hybrid search (dense + sparse) → Re-ranking → Top 5 results

Market Data Agent (market_data.py)

Purpose: Real-time metrics via Yahoo Finance
Key Function: get_comprehensive_data(ticker) → {price, market_cap, P/E, beta, 52-week range, timestamp}
Validation: Verifies market_cap ≈ price × shares (±5% tolerance)

Financial Analyst (analyst.py)

Purpose: Report synthesizer
Key Function: synthesize(sec_data, market_data, ticker) → Markdown report with sections: Executive Summary, Market Metrics, Business Overview, Risk Factors, Financial Trends, Investment View
Citation: Tracks all sources via CitationTracker

Auditor (auditor.py)

Purpose: Quality assurance
Key Function: verify(report, citations, market_data) → {citation_check, numeric_check, claim_check, overall_confidence}
Validation Rules: All factual claims need citations | Market cap calculations within 5% | Confidence = (passed checks) / 3

3. RAG Pipeline Architecture

File: rag_pipeline.py | **Class**: AdvancedRAGPipeline

4-Stage Retrieval Process

Stage 1: Query Decomposition

- Uses GPT-3.5-turbo to break complex queries into 2-3 sub-queries
- Fallback: Original query if LLM fails or returns invalid JSON

Stage 2: Hybrid Search (Top 20)

- **Dense**: Sentence-BERT embeddings (768-dim) + cosine similarity
- **Sparse**: BM25 keyword matching (exact terms/numbers)
- **Fusion**: RRF (Reciprocal Rank Fusion) score = 1/(60+rank) merges both

Stage 3: Re-ranking (Top 5)

- Cross-encoder model: cross-encoder/ms-marco-MiniLM-L-6-v2
- Scores each passage for relevance
- More accurate than bi-encoders but slower (only used on top-20)

Stage 4: Cross-Document Analysis

- compare_across_years(query, [2020-2024]) → Results grouped by year
- Enables trend analysis across multiple 10-K filings

4. Memory Schema

Global Memory (Persistent) - memory.py

Storage: SQLite (global_memory.db) + ChromaDB

Tables:

```
user_preferences (
   user_id, risk_taxonomy JSON, writing_style,
   preferred_kpis JSON, version, created_at, ttl_expires
)

analysis_history (
   ticker, analysis_date, summary, key_metrics JSON, embedding BLOB
)
```

Use Cases: Personalized reports | Similar company search | Analysis history tracking
TTL: 365 days for preferences.