LAB ASSESSMENT REPORT

Generative AI Agents – Task Automation with LLM Reasoning [CSE3024]

Automated Investment Opportunity Report Generation

Submitted by:

Radhika Bhati (220650)

Batch: B.Tech – CSE 2022

Under the supervision of

Dr. Soharab Hossain Shaikh

(Associate Professor)

BML MUNJAL UNIVERSITY™

FROM HERE TO THE WORLD

Department of Computer Science Engineering

School of Engineering and Technology

BML MUNJAL UNIVERSITY, GURUGRAM (INDIA)

November 2025

Contents

Objective and Scope of Work

The primary objective of this project was to design, implement, and evaluate a sophisticated multi-agent system capable of automating the complex task of generating a comprehensive investment analysis report for a publicly traded company. The core goal was to simulate a real-world team of financial specialists, demonstrating how distinct AI agents, each with a specialized role, can collaborate to perform research, data analysis, risk assessment, and report synthesis to produce a high-quality, data-driven output. A secondary, but equally important, objective was to compare two distinct development standards by first building a prototype using the high-level CrewAI framework and subsequently re-implementing an enhanced version with the lower-level Google Agent Development Kit (ADK). This dual-implementation approach was designed to provide practical insights into the trade-offs between rapid development and granular system control, while also exploring advanced agentic behaviors such as explicit memory, custom tool integration, and robust monitoring.

The scope of work was divided into two distinct phases, corresponding to the two lab assignments. The first phase focused on the initial implementation using CrewAI. This involved defining a crew of six specialist agents: a Portfolio Manager to orchestrate the workflow, a Quantitative Analyst to fetch financial metrics, a Market Intelligence Researcher to gather news, a Risk Assessor to synthesize findings, an Investment Report Writer to draft the final document, and a Compliance Validator to perform a final review. The workflow was designed with both parallel execution for efficient data gathering and sequential steps for logical analysis and synthesis. This phase included the integration of two external tools, yfinance and newsapi-python, and the engineering of detailed prompts to guide each agent's behavior, culminating in the generation of a structured Markdown report.

The second phase involved re-implementing the entire workflow using Google's ADK and the Gemini API, with a focus on adding advanced features to enhance system robustness and transparency. The scope of this phase extended beyond a simple rebuild to include the development

of an explicit, centralized memory system (the analysis_state dictionary) to manage context sharing between agents. A new custom tool was created to calculate a proprietary investment score, demonstrating advanced tool integration. Furthermore, a structured logging system was implemented to provide a transparent audit trail of the entire workflow for monitoring and debugging purposes. Finally, the Compliance Validator agent's role was enhanced to perform explicit fact-checking against the raw data stored in memory, providing a concrete mechanism for hallucination mitigation.

1. Lab Assignment – 1

1.1 Expected Outcome in 3 bullet points

- To deliver a rapidly developed multi-agent prototype: Build a functional system using the high-level CrewAI framework that automates the generation of an investment report from a single stock ticker input.
- To produce a high-quality, AI-synthesized report: Generate a comprehensive Markdown report integrating quantitative metrics, qualitative news analysis, and risk assessment, demonstrating the ability of a managed agent crew to produce a coherent, professional-grade output.
- To demonstrate managed orchestration: Showcase CrewAI's capabilities in coordinating parallel tasks, handling sequential dependencies, and enabling context sharing between specialized agents.

1.2 Crew-AI WorkFlow Diagram

The diagrams below shows the end-to-end six-agent CrewAI workflow for generating a financial analysis report from a stock ticker input.

1. **Portfolio Manager (CrewAI Orchestrator)**

   The Portfolio Manager serves as the central orchestrator of the workflow. It coordinates the sequence of agent tasks, manages dependencies, and ensures the overall process flows smoothly from data collection to report generation.

2. **Parallel Data Gathering**
   - **Quantitative Analyst:** Retrieves and analyzes structured financial data, including key metrics such as revenue, earnings, and ratios.
   - **Market Intelligence Researcher:** Collects qualitative market insights and performs sentiment analysis on relevant news articles.

Both agents execute in parallel, allowing the system to efficiently gather information streams simultaneously.

3. **Risk Assessment**

   The outputs from the parallel agents are passed to the Risk Assessor, which synthesizes the quantitative and qualitative insights to evaluate potential risks and opportunities associated with the stock.

4. **Report Writing**

   The Investment Report Writer drafts a structured Markdown report that integrates financial data, market insights, and risk assessment into a coherent, professional document.

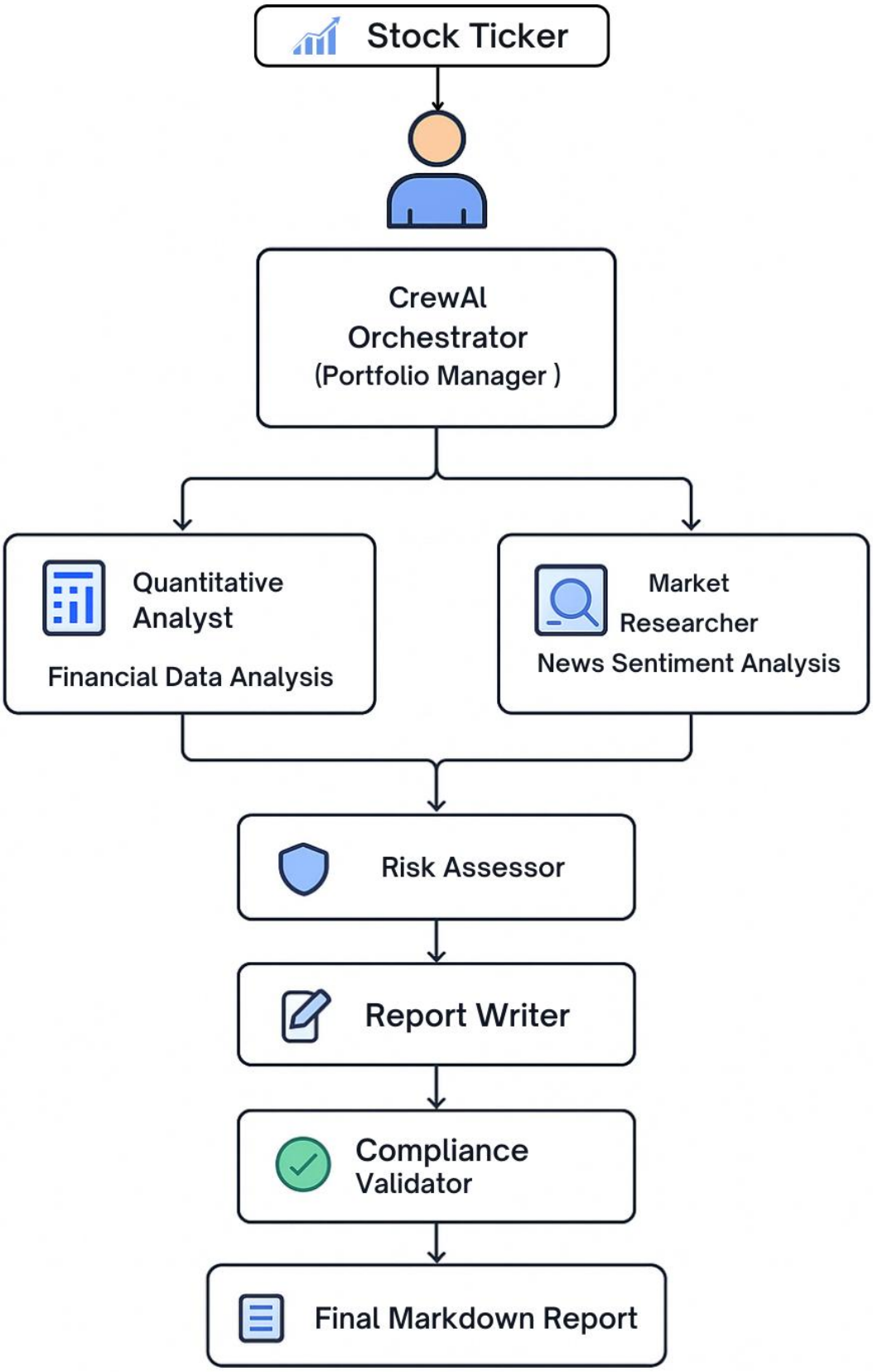
5. **Compliance Validation**

   The Compliance Validator reviews the draft report for accuracy and factual consistency, ensuring that the findings are reliable and free from errors or hallucinations.

6. **Final Output**

   The workflow culminates in the generation of the Final Markdown Report, which serves as the fully automated, end-to-end product of the CrewAI system.

This diagram highlights CrewAI's managed orchestration, parallel task execution, and seamless integration of agent outputs, demonstrating the framework's suitability for rapid prototyping of multi-agent workflows.


Stock Ticker
CrewAI
Orchestrator
(Portfolio Manager)
Quantitative
Analyst
Financial Data Analysis
Market
Researcher
News Sentiment Analysis
Risk Assessor
Report Writer
Compliance
Validator
Final Markdown Report

1.3 Crew-AI Implementation Code

The CrewAI workflow was organized into a modular project structure to ensure clarity and maintainability. The directory layout is shown below:

```
assignment1_crewai/
├── main.py                          # Entry point to run the workflow
├── crewai_workflow.png              # Workflow diagram
├── agents/
│   ├── __init__.py
│   └── financial_agents.py          # Definitions of specialist agents
├── config/
│   ├── __init__.py
│   └── settings.py                  # API keys and configuration settings
├── outputs/
│   ├── __init__.py
│   └── investment_report_aapl.md    # Final generated report
├── tools/
│   ├── __init__.py
│   ├── market_data_tools.py         # yfinance integration
│   └── news_tools.py                # NewsAPI integration
└── workflows/
    ├── __init__.py
    ├── investment_crew.py           # Crew definition
    └── investment_tasks.py          # Task definitions
```

This modular breakdown separates agents, tasks, tools, and configuration, making the system easier to extend (e.g., adding new agents or tools in the future).

CODE FILES:

main.py - This is the entry point of the program. It initializes the Crew, runs the tasks, and saves the final investment report.

```
"""
Main Entry Point for Investment Analysis System
Run complete investment analysis workflow
"""

from workflows.investment_crew import investment_crew
from config.settings import validate_config
import sys
import logging
import re
import os
from datetime import datetime

# Set up logging for tree output
logging.basicConfig(
    level=logging.INFO,
    format='%(message)s',
    handlers=[
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)

def get_clean_report(result: dict) -> str:
    """
    Extracts the final clean report from the crew's full output.
    """
    if isinstance(result, dict) and 'final_output' in result:
        return result['final_output']
    else:
        return str(result)

def main():
    """Main function to run investment analysis"""
    print("🚀 Investment Analysis System - CrewAI Implementation")
    print("=" * 60)

    try:
```

```
        print("🔧 Validating system configuration...")
        validate_config()
        print("☑ Configuration validated successfully!")

        summary = investment_crew.get_crew_summary()
        print(f"\n📊 System Summary:")
        print(f"   Agents: {summary['total_agents']}")
        print(f"   Tasks: {summary['total_tasks']}")
        print(f"   Tools: {summary['tools_available']}")
        print(f"   Process: Hierarchical with Parallel Data Gathering")

        # Get user input for ticker
        if len(sys.argv) > 1:
            ticker = sys.argv[1].upper()
        else:
            ticker = input("\n📈 Enter stock ticker (or press Enter for TSLA):
").strip().upper()
            if not ticker:
                ticker = "TSLA"

        # Get the current date and format it
        current_date = datetime.now().strftime("%B %d, %Y")

        print(f"\n🎯 Starting analysis for: {ticker} (Report Date:
{current_date})")
        print("⏳ Watch the parallel execution below...")
        print("-" * 60)

        print(f"\n🎯 Starting analysis for: {ticker}")
        print("⏳ Watch the parallel execution below...")
        print("-" * 60)

        # Execute the analysis with verbose output
        raw_result = investment_crew.execute_analysis(ticker=ticker,
current_date=current_date)

        final_report = get_clean_report(raw_result)

        # Display results
        print("\n" + "=" * 60)
        print("📄 INVESTMENT ANALYSIS REPORT")
        print("=" * 60)
        print(final_report)

        os.makedirs("outputs", exist_ok=True)
```

```
            output_file = f"outputs/investment_report_{ticker.lower()}.md"
            try:
                with open(output_file, 'w', encoding='utf-8') as f:
                    f.write(final_report)
                print(f"\n💾 Report saved to: {output_file}")
            except Exception as e:
                print(f"\n⚠️ Could not save report: {str(e)}")
                logger.error(f"File save error: {str(e)}")

            print("\n🎉 Analysis completed successfully!")

        except KeyboardInterrupt:
            print("\n\n⏹️ Analysis interrupted by user")
        except Exception as e:
            print(f"\n❌ Error: {str(e)}")
            logger.error(f"Main execution error: {str(e)}")

if __name__ == "__main__":
    main()
```

agents/financial_agents.py: This file defines the CrewAI agents, each with a specific role, goal, and backstory.

```
"""
Financial Analysis Agents for Investment Report Generation
CrewAI agent definitions with specialized roles
"""

from crewai import Agent, LLM
from crewai.tools import tool
import os
from langchain_google_genai import ChatGoogleGenerativeAI
from config.settings import GOOGLE_API_KEY, LLM_MODEL, LLM_TEMPERATURE
from tools.market_data_tools import yahoo_finance_tools
from tools.news_tools import news_api_tools

# Configure LLM for agents
os.environ["GOOGLE_API_KEY"] = GOOGLE_API_KEY

from crewai import LLM
from config.settings import LLM_MODEL, LLM_PROVIDER, LLM_TEMPERATURE

def create_gemini_llm():
```

```
    return LLM(
        model="gemini/gemini-2.5-flash",
        api_key=GOOGLE_API_KEY,
        temperature=LLM_TEMPERATURE
    )


class FinancialAgents:
    """
    Collection of specialized financial analysis agents
    """

    def __init__(self):
        self.llm = create_gemini_llm()

    def portfolio_manager(self) -> Agent:
        """
        Portfolio Manager Agent - The orchestrator and planner
        Coordinates the overall investment analysis strategy
        """
        return Agent(
            role="Senior Portfolio Manager",
            goal="Orchestrate comprehensive investment analysis and coordinate
team efforts to produce high-quality investment recommendations",
            backstory="""You are a seasoned Senior Portfolio Manager with 15
years of experience
            in equity research and portfolio management. You excel at
coordinating diverse teams
            of analysts to produce thorough, well-researched investment reports.
You understand
            both quantitative metrics and qualitative factors that drive
investment decisions.

            Your responsibility is to:
            - Define the scope and methodology of investment analysis
            - Coordinate parallel research efforts across your team
            - Ensure all critical aspects of investment analysis are covered
            - Set quality standards for the final investment recommendation""",
            verbose=True,
            allow_delegation=True,
            llm=self.llm,
            max_iter=3,
            memory=True
        )
```

```
    def quantitative_analyst(self) -> Agent:
        """
        Quantitative Analyst Agent - Financial data specialist
        Handles numerical analysis and financial metrics
        """
        return Agent(
            role="Senior Quantitative Analyst",
            goal="Conduct thorough quantitative analysis of financial metrics,
stock performance, and technical indicators to assess investment attractiveness",
            backstory="""You are a highly skilled Quantitative Analyst with
expertise in
            financial modeling, statistical analysis, and technical analysis. You
have a PhD
            in Finance and 8 years of experience at top-tier investment firms.

            Your expertise includes:
            - Financial ratio analysis and valuation metrics
            - Technical indicator analysis and chart patterns
            - Statistical modeling and risk metrics calculation
            - Market performance comparison and benchmarking

            You provide precise, data-driven insights that form the quantitative
foundation
            of investment decisions.""",
            verbose=True,
            allow_delegation=False,
            llm=self.llm,
            max_iter=2,
            tools=[get_stock_data_tool, get_technical_indicators_tool]
        )

    def market_intelligence_researcher(self) -> Agent:
        """
        Market Intelligence Researcher Agent - News and sentiment specialist
        Handles qualitative analysis and market sentiment
        """
        return Agent(
            role="Senior Market Intelligence Researcher",
            goal="Gather and analyze market news, sentiment, and qualitative
factors that could impact investment performance",
            backstory="""You are an experienced Market Intelligence Researcher
with a
            background in journalism and financial analysis. You have 10 years of
experience
```

```
            tracking market trends, corporate developments, and macroeconomic
factors.

            Your specialties include:
            - News analysis and sentiment assessment
            - Corporate event tracking and impact analysis
            - Market trend identification and interpretation
            - Qualitative risk factor assessment

            You excel at translating complex market dynamics and news flow into
actionable
            investment insights.""",
            verbose=True,
            allow_delegation=False,
            llm=self.llm,
            max_iter=2,
            tools=[get_company_news_tool, get_market_news_tool]  # ← Fixed
reference
        )

    def risk_assessment_specialist(self) -> Agent:
        """
        Risk Assessment Specialist Agent - Risk analysis expert
        Evaluates potential risks and downside scenarios
        """
        return Agent(
            role="Senior Risk Assessment Specialist",
            goal="Conduct comprehensive risk analysis by evaluating financial,
market, and operational risks to provide balanced investment perspective",
            backstory="""You are a seasoned Risk Assessment Specialist with 12
years of
            experience in investment risk management. You hold the FRM (Financial
Risk Manager)
            certification and have worked at both hedge funds and institutional
investment firms.

            Your expertise covers:
            - Financial risk assessment (credit, liquidity, leverage)
            - Market risk analysis (volatility, correlation, beta)
            - Operational and business model risks
            - Regulatory and compliance risk evaluation

            You are known for your ability to identify potential pitfalls and
provide
```

```
            balanced, realistic risk assessments that help investors make
informed decisions.""",
            verbose=True,
            allow_delegation=False,
            llm=self.llm,
            max_iter=2
        )

    def investment_report_writer(self) -> Agent:
        """
        Investment Report Writer Agent - Synthesis and documentation specialist
        Creates comprehensive, well-structured investment reports
        """
        return Agent(
            role="Senior Investment Report Writer",
            goal="Synthesize quantitative analysis, market intelligence, and risk
assessment into a comprehensive, professional investment report",
            backstory="""You are an accomplished Investment Report Writer with a
background
            in financial journalism and equity research. You have 8 years of
experience
            writing research reports for institutional investors and have a
talent for
            translating complex financial analysis into clear, actionable
recommendations.

            Your strengths include:
            - Clear, professional financial writing
            - Data synthesis and narrative construction
            - Investment thesis development
            - Executive summary creation

            You excel at taking diverse analytical inputs and weaving them into a
coherent,
            compelling investment narrative that helps decision-makers understand
both
            opportunities and risks.""",
            verbose=True,
            allow_delegation=False,
            llm=self.llm,
            max_iter=2
        )

    def compliance_validator(self) -> Agent:
        """
```

```
        Compliance Validator Agent - Quality assurance and compliance specialist
        Ensures report accuracy, completeness, and regulatory compliance
        """
        return Agent(
            role="Senior Compliance Validator",
            goal="Review and validate investment reports for accuracy,
completeness, and compliance with regulatory standards and firm policies",
            backstory="""You are a meticulous Compliance Validator with 10 years
of experience
            in financial services compliance and quality assurance. You hold
Series 7, 66,
            and 24 licenses and have deep knowledge of investment advisory
regulations.

            Your responsibilities include:
            - Factual accuracy verification
            - Regulatory compliance review
            - Disclaimer and disclosure validation
            - Professional presentation standards

            You are known for your attention to detail and commitment to ensuring
that all
            investment communications meet the highest standards of accuracy and
compliance.""",
            verbose=True,
            allow_delegation=False,
            llm=self.llm,
            max_iter=2
        )

# Tool definitions
@tool
def get_stock_data_tool(ticker: str, period: str = "1y") -> str:
    """Retrieve comprehensive stock data for analysis"""
    data = yahoo_finance_tools.get_stock_data(ticker, period)
    return str(data)

@tool
def get_technical_indicators_tool(ticker: str, period: str = "3mo") -> str:
    """Calculate technical indicators for stock analysis"""
    data = yahoo_finance_tools.get_technical_indicators(ticker, period)
    return str(data)

@tool
```

```
def get_company_news_tool(company_name: str, ticker: str, days_back: int = 7) ->
str:
    """Retrieve recent news about a company"""
    data = news_api_tools.get_company_news(company_name, ticker, days_back)
    return str(data)

@tool
def get_market_news_tool(category: str = "business", country: str = "us") -> str:
    """Retrieve general market news"""
    data = news_api_tools.get_market_news(category, country)
    return str(data)

# global instance
financial_agents = FinancialAgents()
```

workflows/investment_tasks.py: This file lists the tasks assigned to the agents.

```
"""
Investment Analysis Tasks for CrewAI Workflow
Task definitions for each agent in the investment analysis process
"""

from crewai import Task
from agents.financial_agents import financial_agents

class InvestmentAnalysisTasks:
    """
    Task definitions for comprehensive investment analysis workflow
    """

    def __init__(self):
        self.agents = financial_agents

    def plan_analysis_task(self, ticker: str, company_name: str) -> Task:
        """
        Task for Portfolio Manager to plan the analysis strategy
        """
        return Task(
            description=f"""
            As the Senior Portfolio Manager, develop a comprehensive analysis
plan for {company_name} ({ticker}).

            Your task is to:
            1. Define the scope of analysis for {ticker}
```

```
            2. Set analysis parameters and timeframes
            3. Coordinate team assignments and priorities
            4. Establish success criteria for the investment evaluation

            Consider both quantitative metrics (financial ratios, technical
indicators) and qualitative factors
            (market sentiment, news flow, competitive position).

            Company: {company_name}
            Ticker: {ticker}
            """,
            agent=self.agents.portfolio_manager(),
            expected_output="""
            A structured analysis plan including:
            - Analysis scope and objectives
            - Key metrics to evaluate
            - Risk factors to assess
            - Timeline and methodology
            - Team coordination strategy
            """
        )

    def gather_financial_data_task(self, ticker: str, company_name: str) -> Task:
        """
        Task for Quantitative Analyst to gather and analyze financial data
        """
        return Task(
            description=f"""
            As the Senior Quantitative Analyst, conduct comprehensive
quantitative analysis of {company_name} ({ticker}).

            Your task is to:
            1. Retrieve current stock data, financial metrics, and key ratios
            2. Calculate technical indicators and analyze price trends
            3. Assess valuation metrics (P/E, market cap, etc.)
            4. Evaluate financial performance and market position

            Use your available tools to gather accurate, up-to-date financial
data and provide data-driven insights.

            Company: {company_name}
            Ticker: {ticker}
            """,
            agent=self.agents.quantitative_analyst(),
            async_execution=True,
```

```
            expected_output="""
            Comprehensive quantitative analysis including:
            - Current stock price and performance metrics
            - Key financial ratios (P/E, EPS, market cap)
            - Technical indicators (RSI, moving averages)
            - 52-week performance analysis
            - Valuation assessment and peer comparison insights
            """
        )

    def research_market_sentiment_task(self, ticker: str, company_name: str) ->
Task:
        """
        Task for Market Intelligence Researcher to analyze news and sentiment
        """
        return Task(
            description=f"""
            As the Senior Market Intelligence Researcher, analyze market
sentiment and news flow for {company_name} ({ticker}).

            Your task is to:
            1. Gather recent news articles and market developments
            2. Analyze overall market sentiment and investor mood
            3. Identify key events, announcements, or trends affecting the stock
            4. Assess qualitative factors that could impact investment
performance

            Use your news research tools to provide comprehensive market
intelligence.

            Company: {company_name}
            Ticker: {ticker}
            """,
            agent=self.agents.market_intelligence_researcher(),
            async_execution=True,
            expected_output="""
            Market intelligence report including:
            - Recent news summary and key developments
            - Overall market sentiment analysis
            - Positive and negative sentiment drivers
            - Key events or announcements affecting the stock
            - Qualitative risk and opportunity assessment
            """
        )
```

```python
    def assess_investment_risks_task(self, ticker: str, company_name: str) ->
Task:
        """
        Task for Risk Assessment Specialist to evaluate potential risks
        """
        return Task(
            description=f"""
            As the Senior Risk Assessment Specialist, conduct comprehensive risk
analysis for {company_name} ({ticker}).

            Your task is to:
            1. Analyze the quantitative data and market intelligence gathered by
your colleagues
            2. Identify financial, market, and operational risks
            3. Assess risk levels and potential impact on investment performance
            4. Provide balanced risk assessment with mitigation considerations

            Consider both the financial metrics and market sentiment data
provided by the team.

            Company: {company_name}
            Ticker: {ticker}
            """,
            agent=self.agents.risk_assessment_specialist(),
            expected_output="""
            Comprehensive risk assessment including:
            - Financial risk evaluation (leverage, liquidity, profitability)
            - Market risk analysis (volatility, beta, sector risks)
            - Operational and business model risks
            - Overall risk rating and key risk factors
            - Risk mitigation recommendations
            """
        )

    def write_investment_report_task(self, ticker: str, company_name: str,
current_date: str) -> Task:
        """
        Task for Investment Report Writer to synthesize analysis into report
        """
        return Task(
            description=f"""
            As the Senior Investment Report Writer, synthesize all analysis into
a comprehensive investment report for {company_name} ({ticker}).
            The report must be dated for today: {current_date}.
```

```
            Your task is to:
            1. Combine quantitative analysis, market intelligence, and risk
assessment
            2. Develop a clear investment thesis and recommendation
            3. Structure the report professionally with clear sections
            4. Provide actionable insights for investment decision-making

            Use all the analysis provided by your team members to create a
cohesive, well-reasoned report.
            Important: Do not include any placeholder text like '[Your Firm
Name]' or '[Your Name]'. The report should be clean and final.

            Company: {company_name}
            Ticker: {ticker}
            """,
            agent=self.agents.investment_report_writer(),
            expected_output="""
            Professional investment report in Markdown format including:
            - Executive Summary with clear recommendation
            - Quantitative Analysis section
            - Market Sentiment Analysis section
            - Risk Assessment section
            - Investment Thesis and Rationale
            - Conclusion and Next Steps
            """
        )

    def validate_compliance_task(self, ticker: str, company_name: str,
current_date: str) -> Task:
        """
        Task for Compliance Validator to review and finalize report
        """
        return Task(
            description=f"""
            As the Senior Compliance Validator, review and validate the
investment report for {company_name} ({ticker}), dated {current_date}.

            Your task is to:
            1. Review the report for accuracy and completeness
            2. Ensure proper disclaimers and risk disclosures (without firm-
specific placeholders)
            3. Validate professional formatting and presentation
            4. Confirm compliance with investment advisory standards
```

```
            Provide the final, validated investment report ready for client
presentation.
            Important: Do not include any placeholder text like '[Your Firm
Name]' or '[Your Name]'. Ensure the final output is a complete document.

            Company: {company_name}
            Ticker: {ticker}
            """,
            agent=self.agents.compliance_validator(),
            expected_output="""
            Final validated investment report including:
            - Reviewed and verified analysis
            - Proper disclaimers and disclosures
            - Professional formatting
            - Compliance certification
            - Investment recommendation summary
            """
        )

# global instance
investment_tasks = InvestmentAnalysisTasks()
```

workflows/investment_crew.py: This file organizes the agents and tasks into a workflow, defining how they interact.

```
"""
Investment Analysis Crew - Main Workflow Orchestration
Coordinates all agents and tasks for comprehensive investment analysis
"""

from crewai import Crew, Process
from agents.financial_agents import financial_agents
from workflows.investment_tasks import investment_tasks
from config.settings import DEFAULT_COMPANY
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

class InvestmentAnalysisCrew:
    """
    Main crew orchestration for investment analysis workflow
    """
```

```
    def __init__(self):
        self.agents = financial_agents
        self.tasks = investment_tasks
        self.logger = logger

    def create_analysis_crew(self, ticker: str, company_name: str, current_date:
str) -> Crew:
        """Create and configure the investment analysis crew"""
        self.logger.info(f"Creating investment analysis crew for {company_name}
({ticker})")

        # Create all agents
        portfolio_manager = self.agents.portfolio_manager()
        quantitative_analyst = self.agents.quantitative_analyst()
        market_researcher = self.agents.market_intelligence_researcher()
        risk_specialist = self.agents.risk_assessment_specialist()
        report_writer = self.agents.investment_report_writer()
        compliance_validator = self.agents.compliance_validator()

        # Create all tasks
        plan_task = self.tasks.plan_analysis_task(ticker, company_name)
        data_task = self.tasks.gather_financial_data_task(ticker, company_name)
        news_task = self.tasks.research_market_sentiment_task(ticker,
company_name)
        risk_task = self.tasks.assess_investment_risks_task(ticker, company_name)
        report_task = self.tasks.write_investment_report_task(ticker,
company_name, current_date)
        compliance_task = self.tasks.validate_compliance_task(ticker,
company_name, current_date)

        # Configure task assignments
        plan_task.agent = portfolio_manager
        data_task.agent = quantitative_analyst
        news_task.agent = market_researcher
        risk_task.agent = risk_specialist
        report_task.agent = report_writer
        compliance_task.agent = compliance_validator

        # parallel execution for data gathering tasks
        data_task.async_execution = True
        news_task.async_execution = True

        # task dependencies for proper workflow
        risk_task.context = [data_task, news_task]
        report_task.context = [risk_task]
```

```
        compliance_task.context = [report_task]

        # Create the crew
        crew = Crew(
            agents=[
                quantitative_analyst,
                market_researcher,
                risk_specialist,
                report_writer,
                compliance_validator
            ],
            tasks=[
                plan_task,
                data_task,
                news_task,
                risk_task,
                report_task,
                compliance_task
            ],
            process=Process.hierarchical,
            manager_agent=portfolio_manager,
            verbose=True,
            full_output=True
        )

        self.logger.info("☑ Investment analysis crew created successfully")
        return crew

    def log_step(self, step):
        """Log each step for better visibility"""
        if hasattr(step, 'action') and hasattr(step, 'tool'):
            print(f"    🔧 Using {step.tool} ({step.action})")

    def log_task_completion(self, task_output):
        """Log task completion"""
        if hasattr(task_output, 'agent'):
            print(f"☑ Task completed by {task_output.agent}")


    def execute_analysis(self, ticker: str = None, company_name: str = None,
current_date: str = None) -> str:
        """
        Execute the complete investment analysis workflow

        Args:
```

```
            ticker: Stock symbol (defaults to settings)
            company_name: Company name (will be retrieved if not provided)

        Returns:
            Final investment report as string
        """
        # Use defaults if not provided
        if not ticker:
            ticker = DEFAULT_COMPANY

        if not company_name:
            try:
                from tools.market_data_tools import yahoo_finance_tools
                stock_data = yahoo_finance_tools.get_stock_data(ticker)
                company_name = stock_data.get('company_name', ticker)
            except:
                company_name = ticker

        self.logger.info(f"🚀 Starting investment analysis for {company_name}
({ticker})")

        try:
            # Create the crew
            crew = self.create_analysis_crew(ticker, company_name, current_date)

            # Execute the workflow
            self.logger.info("⚡ Executing investment analysis workflow...")
            result = crew.kickoff(inputs={
                "ticker": ticker,
                "company_name": company_name
            })

            self.logger.info("☑️ Investment analysis completed successfully")
            return result

        except Exception as e:
            error_msg = f"❌ Error during analysis execution: {str(e)}"
            self.logger.error(error_msg)
            return error_msg

    def get_crew_summary(self) -> dict:
        """
        Get summary information about the crew configuration

        Returns:
```

```
            Dict containing crew configuration details
        """
        return {
            "total_agents": 6,
            "agent_roles": [
                "Senior Portfolio Manager",
                "Senior Quantitative Analyst",
                "Senior Market Intelligence Researcher",
                "Senior Risk Assessment Specialist",
                "Senior Investment Report Writer",
                "Senior Compliance Validator"
            ],
            "total_tasks": 6,
            "workflow_type": "Sequential with parallel data gathering",
            "tools_available": 4,
            "process_flow": "Plan → [Data + News] → Risk → Report → Compliance"
        }

# global instance
investment_crew = InvestmentAnalysisCrew()
```

tools/market_data_tools.py: This file provides tools for fetching stock market data.

```
"""
Financial Market Data Tools
Yahoo Finance integration for stock data retrieval
"""

import yfinance as yf
import pandas as pd
from datetime import datetime, timedelta
from typing import Dict, Any, Optional
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

class YahooFinanceTools:
    """
    Yahoo Finance data retrieval tools for quantitative analysis
    """

    def __init__(self):
        self.logger = logger
```

```python
    def get_stock_data(self, ticker: str, period: str = "1y") -> Dict[str, Any]:
        """
        Fetch comprehensive stock data for analysis

        Args:
            ticker: Stock symbol (e.g., 'TSLA')
            period: Time period ('1d', '5d', '1mo', '3mo', '6mo', '1y', '2y',
'5y', '10y', 'ytd', 'max')

        Returns:
            Dict containing stock data and metrics
        """
        try:
            self.logger.info(f"Fetching stock data for {ticker}")

            # ticker object
            stock = yf.Ticker(ticker)

            # Get stock info
            info = stock.info

            # Get historical data
            history = stock.history(period=period)

            # Calculate additional metrics
            current_price = history['Close'].iloc[-1] if not history.empty else
None
            price_change = history['Close'].iloc[-1] - history['Close'].iloc[-2]
if len(history) > 1 else 0
            price_change_percent = (price_change / history['Close'].iloc[-2] *
100) if len(history) > 1 else 0

            # Organize the data
            stock_data = {
                "ticker": ticker,
                "company_name": info.get("longName", "N/A"),
                "sector": info.get("sector", "N/A"),
                "industry": info.get("industry", "N/A"),
                "current_price": round(current_price, 2) if current_price else
None,
                "price_change": round(price_change, 2),
                "price_change_percent": round(price_change_percent, 2),
                "market_cap": info.get("marketCap", "N/A"),
                "pe_ratio": info.get("trailingPE", "N/A"),
                "eps": info.get("trailingEps", "N/A"),
```

```
                "dividend_yield": info.get("dividendYield", "N/A"),
                "52_week_high": round(history["High"].max(), 2) if not
history.empty else None,
                "52_week_low": round(history["Low"].min(), 2) if not
history.empty else None,
                "volume": history["Volume"].iloc[-1] if not history.empty else
None,
                "avg_volume": round(history["Volume"].mean(), 0) if not
history.empty else None,
                "beta": info.get("beta", "N/A"),
                "revenue": info.get("totalRevenue", "N/A"),
                "profit_margin": info.get("profitMargins", "N/A"),
                "data_retrieved_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            }

            self.logger.info(f"☑ Successfully retrieved data for {ticker}")
            return stock_data

        except Exception as e:
            self.logger.error(f"❌ Error fetching data for {ticker}: {str(e)}")
            return {
                "ticker": ticker,
                "error": str(e),
                "data_retrieved_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            }

    def get_technical_indicators(self, ticker: str, period: str = "3mo") ->
Dict[str, Any]:
        """
        Calculate basic technical indicators

        Args:
            ticker: Stock symbol
            period: Time period for calculation

        Returns:
            Dict containing technical indicators
        """
        try:
            self.logger.info(f"Calculating technical indicators for {ticker}")

            stock = yf.Ticker(ticker)
            history = stock.history(period=period)

            if history.empty:
```

```
                return {"error": "No historical data available"}

            # Calculate moving averages
            history['MA_20'] = history['Close'].rolling(window=20).mean()
            history['MA_50'] = history['Close'].rolling(window=50).mean()

            # Calculate RSI
            delta = history['Close'].diff()
            gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()
            loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()
            rs = gain / loss
            rsi = 100 - (100 / (1 + rs))

            current_price = history['Close'].iloc[-1]
            ma_20 = history['MA_20'].iloc[-1]
            ma_50 = history['MA_50'].iloc[-1]
            current_rsi = rsi.iloc[-1]

            indicators = {
                "ticker": ticker,
                "current_price": round(current_price, 2),
                "moving_average_20": round(ma_20, 2) if not pd.isna(ma_20) else
None,
                "moving_average_50": round(ma_50, 2) if not pd.isna(ma_50) else
None,
                "rsi_14": round(current_rsi, 2) if not pd.isna(current_rsi) else
None,
                "price_vs_ma20": "Above" if current_price > ma_20 else "Below" if
not pd.isna(ma_20) else "N/A",
                "price_vs_ma50": "Above" if current_price > ma_50 else "Below" if
not pd.isna(ma_50) else "N/A",
                "rsi_signal": "Overbought" if current_rsi > 70 else "Oversold" if
current_rsi < 30 else "Neutral" if not pd.isna(current_rsi) else "N/A",
                "calculated_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            }

            self.logger.info(f"☑ Technical indicators calculated for {ticker}")
            return indicators

        except Exception as e:
            self.logger.error(f"❌ Error calculating indicators for {ticker}:
{str(e)}")
            return {
                "ticker": ticker,
                "error": str(e),
```

```
                "calculated_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            }

# global instance
yahoo_finance_tools = YahooFinanceTools()
```

tools/news_tools.py: This file provides tools for retrieving and analyzing financial news.

```
"""
News Research Tools
News API integration for market sentiment analysis
"""

import requests
from datetime import datetime, timedelta
from typing import Dict, Any, List, Optional
import logging
from config.settings import NEWS_API_KEY

# Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

class NewsAPITools:
    """
    News API tools for market sentiment and news analysis
    """

    def __init__(self):
        self.api_key = NEWS_API_KEY
        self.base_url = "https://newsapi.org/v2"
        self.logger = logger

        if not self.api_key:
            self.logger.warning("⚠ NEWS_API_KEY not found. News tools will not
work.")

    def get_company_news(self, company_name: str, ticker: str, days_back: int =
7) -> Dict[str, Any]:
        """
        Fetch recent news articles about a company

        Args:
            company_name: Full company name (e.g., 'Tesla')
```

```
        ticker: Stock ticker (e.g., 'TSLA')
        days_back: Number of days to look back for news

    Returns:
        Dict containing news articles and analysis
    """
    try:
        if not self.api_key:
            return {"error": "NEWS_API_KEY not configured"}

        self.logger.info(f"Fetching news for {company_name} ({ticker})")

        # Calculate date range
        to_date = datetime.now()
        from_date = to_date - timedelta(days=days_back)

        # Search query - try both company name and ticker
        query = f'"{company_name}" OR "{ticker}"'

        # API parameters
        params = {
            'q': query,
            'from': from_date.strftime('%Y-%m-%d'),
            'to': to_date.strftime('%Y-%m-%d'),
            'sortBy': 'publishedAt',
            'language': 'en',
            'pageSize': 10,
            'apiKey': self.api_key
        }

        # Make API request
        response = requests.get(f"{self.base_url}/everything", params=params)

        if response.status_code == 200:
            data = response.json()
            articles = data.get('articles', [])

            # Process articles
            processed_articles = []
            sentiment_scores = []

            for article in articles:
                processed_article = {
                    'title': article.get('title', 'No Title'),
```

```
                        'description': article.get('description', 'No
Description'),
                        'source': article.get('source', {}).get('name',
'Unknown'),
                        'published_at': article.get('publishedAt', ''),
                        'url': article.get('url', ''),
                        'sentiment': self._analyze_sentiment(article.get('title',
'') + ' ' + article.get('description', ''))
                    }
                    processed_articles.append(processed_article)
                    sentiment_scores.append(processed_article['sentiment']['score
'])

                # Calculate overall sentiment
                overall_sentiment =
self._calculate_overall_sentiment(sentiment_scores)

                result = {
                    'company_name': company_name,
                    'ticker': ticker,
                    'total_articles': len(processed_articles),
                    'date_range': f"{from_date.strftime('%Y-%m-%d')} to
{to_date.strftime('%Y-%m-%d')}",
                    'overall_sentiment': overall_sentiment,
                    'articles': processed_articles,
                    'retrieved_at': datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                }

                self.logger.info(f"☑ Retrieved {len(processed_articles)}
articles for {company_name}")
                return result

            else:
                error_msg = f"API request failed with status
{response.status_code}"
                self.logger.error(f"❌ {error_msg}")
                return {"error": error_msg}

        except Exception as e:
            self.logger.error(f"❌ Error fetching news for {company_name}:
{str(e)}")
            return {
                "company_name": company_name,
                "ticker": ticker,
                "error": str(e),
```

```
                "retrieved_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            }

    def get_market_news(self, category: str = "business", country: str = "us") ->
Dict[str, Any]:
        """
        Fetch general market/business news

        Args:
            category: News category ('business', 'technology', etc.)
            country: Country code ('us', 'gb', etc.)

        Returns:
            Dict containing market news
        """
        try:
            if not self.api_key:
                return {"error": "NEWS_API_KEY not configured"}

            self.logger.info(f"Fetching {category} news for {country}")

            params = {
                'category': category,
                'country': country,
                'pageSize': 5,
                'apiKey': self.api_key
            }

            response = requests.get(f"{self.base_url}/top-headlines",
params=params)

            if response.status_code == 200:
                data = response.json()
                articles = data.get('articles', [])

                processed_articles = []
                for article in articles:
                    processed_articles.append({
                        'title': article.get('title', 'No Title'),
                        'description': article.get('description', 'No
Description'),
                        'source': article.get('source', {}).get('name',
'Unknown'),
                        'published_at': article.get('publishedAt', ''),
                        'url': article.get('url', '')
```

```
                    })

                result = {
                    'category': category,
                    'country': country,
                    'total_articles': len(processed_articles),
                    'articles': processed_articles,
                    'retrieved_at': datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                }

                self.logger.info(f"☑ Retrieved {len(processed_articles)}
{category} articles")
                return result

            else:
                error_msg = f"API request failed with status
{response.status_code}"
                self.logger.error(f"❌ {error_msg}")
                return {"error": error_msg}

        except Exception as e:
            self.logger.error(f"❌ Error fetching market news: {str(e)}")
            return {"error": str(e)}

    def _analyze_sentiment(self, text: str) -> Dict[str, Any]:
        """
        Simple sentiment analysis based on keywords
        (In production, you'd use a proper sentiment analysis model)
        """
        positive_words = ['growth', 'profit', 'increase', 'rise', 'gain',
'positive', 'strong', 'good', 'bullish', 'up']
        negative_words = ['loss', 'decline', 'fall', 'drop', 'negative', 'weak',
'bad', 'bearish', 'down', 'crash']

        text_lower = text.lower()

        positive_count = sum(1 for word in positive_words if word in text_lower)
        negative_count = sum(1 for word in negative_words if word in text_lower)

        if positive_count > negative_count:
            sentiment = 'Positive'
            score = 1
        elif negative_count > positive_count:
            sentiment = 'Negative'
            score = -1
```

```
        else:
            sentiment = 'Neutral'
            score = 0

        return {
            'sentiment': sentiment,
            'score': score,
            'positive_indicators': positive_count,
            'negative_indicators': negative_count
        }

    def _calculate_overall_sentiment(self, sentiment_scores: List[int]) ->
Dict[str, Any]:
        """Calculate overall sentiment from individual scores"""
        if not sentiment_scores:
            return {'sentiment': 'Neutral', 'confidence': 'Low'}

        avg_score = sum(sentiment_scores) / len(sentiment_scores)

        if avg_score > 0.3:
            sentiment = 'Positive'
        elif avg_score < -0.3:
            sentiment = 'Negative'
        else:
            sentiment = 'Neutral'

        confidence = 'High' if abs(avg_score) > 0.5 else 'Medium' if
abs(avg_score) > 0.2 else 'Low'

        return {
            'sentiment': sentiment,
            'score': round(avg_score, 2),
            'confidence': confidence
        }

# Create global instance
news_api_tools = NewsAPITools()
```

config/settings.py: This file contains API keys and configuration variables.

```
import os
from dotenv import load_dotenv

load_dotenv()
```

```python
NEWS_API_KEY = os.getenv("NEWS_API_KEY")
GOOGLE_API_KEY = os.getenv("GOOGLE_API_KEY")

LLM_MODEL = os.getenv("LLM_MODEL", "gemini/gemini-2.5-flash")
LLM_PROVIDER = os.getenv("LLM_PROVIDER", "google")
LLM_TEMPERATURE = float(os.getenv("LLM_TEMPERATURE", "0.1"))

DEFAULT_COMPANY = os.getenv("DEFAULT_COMPANY", "TSLA")
ANALYSIS_PERIOD = os.getenv("ANALYSIS_PERIOD", "1y")
NEWS_LOOKBACK_DAYS = int(os.getenv("NEWS_LOOKBACK_DAYS", "7"))

MCP_ENABLED = True
A2A_PROTOCOL_ENABLED = True

OUTPUT_FORMAT = "markdown"
REPORT_TEMPLATE = "investment_report"

def validate_config():
    """Validate that all required configurations are set"""
    if not NEWS_API_KEY:
        raise ValueError("NEWS_API_KEY not found in environment variables")
    if not GOOGLE_API_KEY:
        raise ValueError("GOOGLE_API_KEY not found in environment variables")

    print("☑ Configuration validated successfully!")
    return True
```

1.4 Sample Structured Output

outputs/investment_report_aapl.md: This is an example of the generated report for Apple (AAPL).

Investment Report: Apple Inc. (AAPL)

Date: September 24, 2025

Executive Summary

This report provides a comprehensive investment analysis of Apple Inc. (AAPL), integrating quantitative metrics, market intelligence, and a detailed risk assessment.

Apple demonstrates exceptional financial strength, robust profitability, and strong brand loyalty, underpinning its market leadership. However, its premium valuation, significant reliance on iPhone sales, complex global supply chain, and increasing regulatory scrutiny present notable risks.

Investment Recommendation: HOLD

While Apple's fundamentals are strong and its services segment continues to grow, the current premium valuation (P/E of 38.61) prices in substantial future growth, leaving limited upside in the short to medium term. The stock's moderate-high overall risk profile, driven by regulatory challenges, geopolitical tensions, and iPhone dependence, suggests a cautious approach. Existing investors are advised to hold, monitoring key risk factors and the services growth trajectory. New investors should await a more attractive entry point or clearer resolution of regulatory and geopolitical uncertainties.

Quantitative Analysis

Apple Inc. (AAPL) exhibits strong financial performance and market characteristics, as evidenced by the following metrics:

- **Revenue:** $408.62 Billion
- **Profit Margin:** 24.30%
- **Earnings Per Share (EPS):** $6.59
- **Trailing P/E Ratio:** 38.61 (Significantly higher than broader market average, indicating premium valuation)
- **Beta:** 1.109 (Slightly more volatile than the overall market)
- **Current Stock Price:** $254.43
- **52-Week High:** $259.18
- **20-Day Moving Average:** $237.15
- **50-Day Moving Average:** $225.65
- **Relative Strength Index (RSI):** 66.81 (Indicates strong buying interest, not yet overbought)

These metrics highlight Apple's robust profitability and strong market momentum, with the stock trading above key moving averages and approaching its 52-week high. The high P/E ratio, however, suggests that significant future growth is already factored into the current share price.

Market Sentiment Analysis

Market sentiment towards Apple Inc. is generally positive, driven by its strong brand, consistent innovation, and growing services ecosystem. However, several qualitative factors influence investor perception and market positioning:

- **Brand Loyalty & Innovation:** Apple benefits from unparalleled brand loyalty and a perception of continuous innovation, which supports premium pricing and consistent demand for its products and services.
- **Services Growth:** The expanding services segment (App Store, Apple Music, iCloud, Apple Pay, etc.) is viewed positively as a high-margin, recurring revenue stream that diversifies the business model beyond hardware.
- **Intense Competition:** The technology sector is highly competitive, with rivals constantly innovating across hardware (Samsung, Google) and services (Microsoft, Amazon, Netflix). This necessitates continuous R&D investment and aggressive market strategies.
- **Regulatory Scrutiny:** Increasing global regulatory focus on antitrust, data privacy, and app store policies poses a significant overhang. Investor sentiment is sensitive to potential fines, forced business model changes, or restrictions on market dominance.
- **Geopolitical Tensions:** The reliance on China for manufacturing and as a major sales market exposes Apple to geopolitical risks, particularly US-China trade relations. Any escalation could negatively impact supply chains and market access.
- **Consumer Discretionary Spending:** As a premium product provider, Apple's demand can be sensitive to macroeconomic downturns, inflation, and shifts in consumer discretionary spending.
- **Technical Momentum:** Positive short-term and medium-term price momentum, with the stock trading above key moving averages and a healthy RSI, suggests bullish sentiment among traders, though proximity to 52-week highs could signal potential for consolidation.

Overall, while Apple enjoys strong underlying positive sentiment, the market is increasingly aware of the external pressures and competitive landscape that could impact its future growth trajectory and valuation.

Risk Assessment

Overall Risk Rating: Moderate-High

While Apple boasts exceptional financial strength and market leadership, the confluence of its premium valuation, significant external operational and regulatory pressures, and reliance on key product cycles elevates its overall risk profile beyond a purely "Moderate" rating. The potential impact of these risks, if materialized, could be substantial despite the company's robust fundamentals.

1. Financial Risk Evaluation

Apple Inc. demonstrates exceptional financial strength, which significantly mitigates traditional financial risks.

- **Leverage Risk:** Considered **Low**. Implied conservative leverage profile due to massive cash pile and strong balance sheet, providing ample capacity to service debt.
- **Liquidity Risk:** Assessed as **Very Low**. Superior liquidity from significant free cash flow generation and substantial cash reserves.
- **Profitability Risk:** Considered **Low** in the short to medium term. Robust profitability (24.30% Profit Margin, $6.59 EPS) driven by strong pricing power and high-margin services. Long-term sustainability depends on continued innovation.

2. Market Risk Analysis

Apple's market performance is influenced by its size, sector dynamics, and investor sentiment.

- **Volatility (Beta):** AAPL has a Beta of 1.109, indicating **Moderate** market-specific volatility risk. The stock is slightly more volatile than the overall market.
- **Valuation Risk (P/E Ratio):** Trailing P/E Ratio of 38.61 represents a **Moderate-High** valuation risk. Premium valuation implies substantial future growth expectations; failure to meet these could lead to correction.
- **Sector Risks: Moderate** risk from rapid technological obsolescence, intense competition, increasing regulatory scrutiny (antitrust, data privacy), and sensitivity to consumer discretionary spending.
- **Technical Indicators:** Current price above 20-day and 50-day moving averages, with an RSI of 66.81, indicates positive momentum but also potential for short-term consolidation near the 52-week high.

3. Operational and Business Model Risks

Apple's operational and business model risks are significant, stemming from its global scale, supply chain complexity, and market dominance.

- **Dependence on iPhone Sales: High** business model risk. iPhone remains the primary revenue driver; slowdowns or increased competition could materially impact performance.
- **Supply Chain Disruptions: High** operational risk. Complex global supply chain, heavily reliant on China, vulnerable to geopolitical tensions, natural disasters, and health crises.
- **Regulatory and Antitrust Challenges: High** regulatory risk. Increasing global scrutiny threatens core business model through potential fines or forced operational changes.
- **Economic Downturn and Consumer Spending: Moderate-High** operational risk. Demand for premium products and services sensitive to global economic conditions.
- **Intense Competition: Moderate** business model risk. Constant innovation and aggressive pricing from rivals could erode market share.
- **Geopolitical Risks: High** geopolitical risk. US-China tensions impact manufacturing and sales in a key market.
- **Innovation Lag: Moderate** long-term business model risk. Perceived slowdown in groundbreaking innovation could lead to loss of competitive edge.

Key Risk Factors:

1. **Regulatory and Antitrust Scrutiny:** Immediate and potentially impactful, threatening core business model.
2. **Geopolitical Tensions & Supply Chain Vulnerability:** Heavy reliance on China creates significant exposure.
3. **Premium Valuation & Growth Expectations:** High P/E makes the stock sensitive to growth perceptions.
4. **Dependence on iPhone Performance:** Remains critical despite diversification efforts.

Investment Thesis and Rationale

Investment Thesis: Apple Inc. (AAPL) is a fundamentally strong, highly profitable technology leader with an expanding services ecosystem and unparalleled brand loyalty. While its financial health and innovation capabilities provide a solid long-term

foundation, the current market valuation fully reflects these strengths, and significant external risks warrant a cautious investment stance.

Rationale:

1. **Robust Financials and Services Growth:** Apple's massive cash pile, strong balance sheet, and high profitability (24.30% profit margin) provide a strong buffer against economic headwinds. The continued growth of its high-margin services segment is a key driver for future revenue diversification and stability, reducing reliance on hardware cycles.
2. **Market Leadership and Brand Power:** Apple's ecosystem, driven by the iPhone, Mac, iPad, and wearables, fosters strong customer retention and cross-selling opportunities. Its brand strength allows for premium pricing and sustained demand.
3. **Premium Valuation:** With a P/E ratio of 38.61, AAPL trades at a significant premium. This valuation implies aggressive growth expectations that may be challenging to consistently meet, especially given its already massive scale. Any deceleration in growth or negative news could lead to a valuation multiple contraction.
4. **Elevated Risk Profile:** The "Moderate-High" overall risk rating is a critical consideration. Regulatory pressures (antitrust, App Store policies), geopolitical tensions (US-China supply chain and market access), and the inherent dependence on iPhone sales introduce substantial uncertainties that could impact future profitability and market position.
5. **Limited Short-to-Medium Term Upside:** While long-term prospects remain solid, the current price near its 52-week high, coupled with the premium valuation and identified risks, suggests that much of the positive news is already priced in. Significant upside in the short to medium term appears constrained without new, transformative product categories or a substantial de-risking of the regulatory and geopolitical landscape.

Conclusion and Next Steps

Apple Inc. remains a high-quality company with a dominant market position and strong financial performance. However, its current valuation leaves little room for error, and a confluence of external risks presents headwinds.

Recommendation: HOLD for existing investors. New investors should exercise patience and consider entry points during market pullbacks or after clearer resolution of key risk factors.

Next Steps for Investors:

- **Monitor Regulatory Developments:** Closely track global antitrust investigations and legislative actions impacting Apple's business model.
- **Assess Geopolitical Landscape:** Stay informed on US-China relations and any potential impacts on Apple's supply chain and market access.
- **Evaluate Services Segment Performance:** Pay close attention to quarterly reports for continued strong growth and profitability in the services division.
- **Observe Innovation Pipeline:** Watch for successful new product launches or significant advancements in emerging technologies (e.g., AI, AR/VR, automotive) that could justify further valuation expansion.
- **Portfolio Diversification:** Ensure AAPL does not represent an over-allocation in a diversified portfolio, given its volatility and premium valuation.

Important Disclosures

This investment report is for informational purposes only and does not constitute an offer to sell or a solicitation of an offer to buy any security. It is not intended to provide personal investment advice and does not take into account the specific investment objectives, financial situation, or particular needs of any individual investor. Investors should consider this report as only one factor in making their investment decision.

Investing in securities involves risks, including the potential loss of principal. Past performance is not indicative of future results. The value of investments and the income derived from them can go down as well as up.

This report contains forward-looking statements, which are based on current expectations, estimates, forecasts, and projections about the industry and markets in which Apple Inc. operates, as well as management's beliefs and assumptions. These statements are not guarantees of future performance and involve risks, uncertainties, and assumptions that are difficult to predict. Actual outcomes and results may differ materially from what is expressed or forecasted in such forward-looking statements.

The information contained herein has been obtained from sources believed to be reliable, but its accuracy and completeness are not guaranteed. No representation or warranty, express or implied, is made as to the fairness, accuracy, completeness, or correctness of the information and opinions contained in this report. All opinions and estimates included in this report constitute the judgment of the author as of the date of this report and are subject to change without notice.

Investors should conduct their own due diligence and consult with a qualified financial advisor, tax advisor, and/or legal counsel before making any investment decisions. The author of this report may or may not hold positions in the securities mentioned. This report is not intended for distribution to, or use by, any person or entity in any jurisdiction or country where such distribution or use would be contrary to law or regulation.

Compliance Certification

I, the Senior Compliance Validator, certify that this investment report for Apple Inc. (AAPL), dated September 24, 2025, has been reviewed for accuracy, completeness, and adherence to professional formatting standards. I confirm that all necessary disclaimers and risk disclosures are properly included and that the report complies with applicable investment advisory standards. The report is now validated and ready for client presentation.

2. Lab Assignment – 2

2.1 Expected Outcome in 3 bullet points

- To deliver an enhanced and robust multi-agent system: Build a production-grade system featuring an explicit central memory (analysis_state) for reliable context management, a custom analytical tool for proprietary scoring, and a structured logging mechanism for full workflow transparency and monitoring.
- To demonstrate a manually orchestrated workflow: Implement and highlight granular control over parallel task execution, explicit data flow between agents, and a transparent, auditable process from start to finish.
- To produce a fact-checked and validated final report: Generate a high-fidelity investment report that undergoes a dedicated hallucination mitigation step, where a validator agent cross-references the generated content against raw source data in memory to ensure accuracy and reliability.

2.2 ADK WorkFlow Diagram

The updated diagram below shows the enhanced multi-agent workflow re-implemented using Google's Agent Development Kit (ADK). Unlike the CrewAI version, this design emphasizes manual orchestration, explicit memory management, and transparency mechanisms.

1. **Central Memory (analysis_state)**

   At the core of the workflow is the explicit analysis_state dictionary, which stores all intermediate outputs. This central memory enables reliable context sharing between agents, ensures traceability, and prevents data loss during task transitions.

2. **Parallel Data Gathering**
   - **Quantitative Analyst:** Fetches structured financial metrics (e.g., stock performance indicators, ratios).

- **Market Intelligence Researcher:** Collects relevant market news and sentiment data.

  Both agents execute in parallel, with their results written into analysis_state.

3. **Risk Assessment**

   The Risk Assessor retrieves both financial and market intelligence data from memory, synthesizes insights, and generates a risk profile. This ensures that all reasoning steps are explicitly tied back to stored inputs.

4. **Custom Tool Integration**

   A proprietary Investment Scoring Tool is invoked at this stage to compute a composite score. This tool demonstrates ADK's ability to extend workflows with domain-specific analytics.

5. **Report Writing**

   The Investment Report Writer uses the memory state to draft a professional Markdown report. Its content integrates numerical analysis, news insights, risk evaluation, and the proprietary investment score.

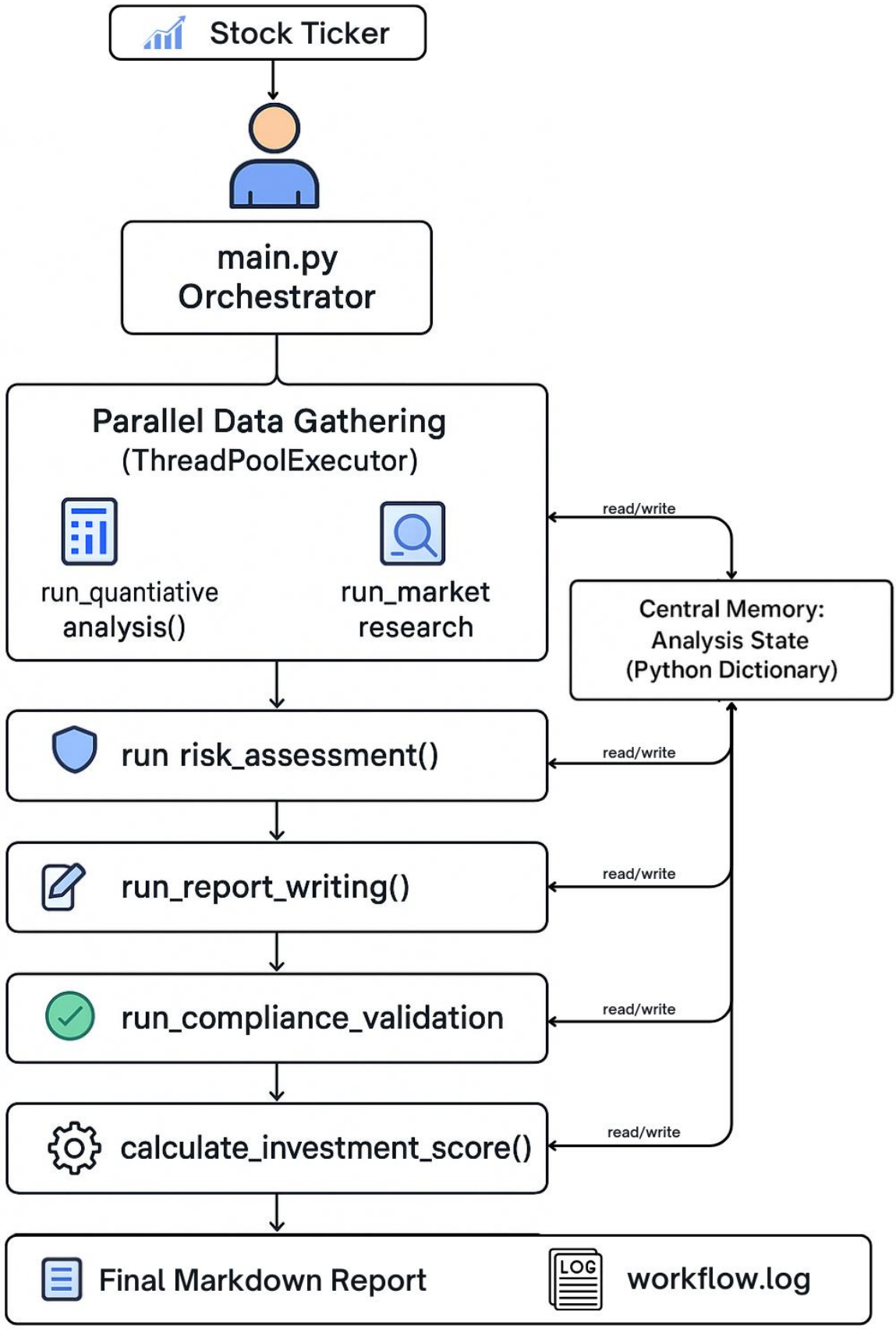
6. **Compliance Validation**

   Unlike CrewAI's basic validation, the ADK implementation enhances the Compliance Validator to perform explicit fact-checking against the raw data stored in memory, mitigating hallucinations and ensuring the reliability of the output.

7. **Structured Logging & Final Output**

   Every agent's action, data access, and decision is logged into a transparent monitoring system, creating a complete audit trail. The workflow concludes with the generation of a fact-checked, structured Markdown report, suitable for real-world use.

   This diagram highlights the contrast between high-level orchestration (CrewAI) and granular, production-grade orchestration (ADK), emphasizing memory, monitoring, and accuracy.


Stock Ticker
main.py
Orchestrator
Parallel Data Gathering
(ThreadPoolExecutor)
run_quantiative
analysis()
run_market
research
read/write
Central Memory:
Analysis State
(Python Dictionary)
run risk_assessment()
read/write
run_report_writing()
read/write
run_compliance_validation
read/write
calculate_investment_score()
read/write
Final Markdown Report
LOG
workflow.log

2.3 ADK Implementation Code

The ADK workflow was organized into a modular project structure to ensure clarity and maintainability. The directory layout is shown below:

```
assignment2_adk/
├── main.py
├── config.py
├── adk_workflow.png
├── agents/
│   ├── __init__.py
│   └── financial_agent_functions.py
├── outputs/
│   ├── investment_report_aapl.md
│   └── workflow.log
├── tools/
│   ├── __init__.py
│   ├── custom_tools.py
│   ├── market_data_tools.py
│   └── news_tools.py
└── utils/
    ├── __init__.py
    └── logging_setup.py
```

CODE FILES:

main.py: This is the entry point of the program. It orchestrates the workflow by initializing agents, running tasks, and saving results.

```python
import logging
import os
import google.generativeai as genai
```

```python
from datetime import datetime
from concurrent.futures import ThreadPoolExecutor

# Import our custom modules
from utils.logging_setup import setup_logging
from config import GOOGLE_API_KEY
from agents.financial_agent_functions import (
    run_quantitative_analysis,
    run_market_research,
    run_risk_assessment,
    run_report_writing,
    run_compliance_validation
)
from tools.custom_tools import calculate_investment_score
from tools.market_data_tools import yahoo_finance_tools # Needed to get company
name

def main():
    """
    Main orchestrator function for the ADK-based investment analysis workflow.
    """
    logger = setup_logging()
    genai.configure(api_key=GOOGLE_API_KEY)

    logger.info("ADK Investment Analysis System - Initiated")

    # User Input
    ticker = input("\n Enter stock ticker (or press Enter for 'AAPL'):
").strip().upper()
    if not ticker:
        ticker = "AAPL"

    # Get the full company name
    try:
        company_name =
yahoo_finance_tools.get_stock_data(ticker).get('company_name', ticker)
    except Exception:
        company_name = ticker

    logger.info(f"Starting analysis for: {company_name} ({ticker})")

    # Initialize the State (Memory)
    analysis_state = {
        "ticker": ticker,
        "company_name": company_name,
```

```
    "current_date": datetime.now().strftime("%B %d, %Y")
}

# Parallel Execution for Data Gathering
logger.info("--- Starting Parallel Data Gathering ---")
with ThreadPoolExecutor(max_workers=2) as executor:

    quant_future = executor.submit(run_quantitative_analysis, analysis_state)
    market_future = executor.submit(run_market_research, analysis_state)

    quant_success = quant_future.result()
    market_success = market_future.result()

if not (quant_success and market_success):
    logger.error("Data gathering failed. Aborting workflow.")
    return
logger.info("--- Parallel Data Gathering Complete ---")

# Sequential Execution
logger.info("--- Starting Sequential Analysis & Synthesis ---")

# Run Risk Assessment
if not run_risk_assessment(analysis_state):
    logger.error("Risk assessment failed. Aborting workflow.")
    return

# Run Report Writing
if not run_report_writing(analysis_state):
    logger.error("Report writing failed. Aborting workflow.")
    return

# Run Compliance Validation (Hallucination Check)
if not run_compliance_validation(analysis_state):
    logger.error("Compliance validation failed. Aborting workflow.")
    return
logger.info("--- Sequential Analysis & Synthesis Complete ---")

# 6. Run Custom Tool for Final Score
logger.info("--- Running Custom Tool ---")
investment_score = calculate_investment_score(
    analysis_state["raw_financial_data"],
    analysis_state["raw_news_data"]
)
analysis_state["investment_score"] = investment_score
logger.info("--- Custom Tool Execution Complete ---")
```

```
    # 7. Final Output
    final_report = analysis_state.get("final_report", "Report could not be
generated.")
    score_summary = f"Proprietary Investment Score:
{investment_score['total_score']}/10 ({investment_score['recommendation']})"

    full_output = f"# Investment Report: {company_name} ({ticker})\n\n"
    full_output += f"**{score_summary}**\n\n"
    full_output += final_report

    print("\n" + "="*60)
    print("ANALYSIS COMPLETE")
    print("="*60)
    print(full_output)

    # Save the final report
    output_file = f"outputs/investment_report_{ticker.lower()}.md"
    with open(output_file, 'w', encoding='utf-8') as f:
        f.write(full_output)
    logger.info(f"💾 Report saved to: {output_file}")
    logger.info("🎉 Workflow finished successfully!")

if __name__ == "__main__":
    main()
```

config.py: Contains configuration variables (API keys, constants, stock symbols, etc.) used across the project.

```
import os
from dotenv import load_dotenv

dotenv_path = os.path.join(os.path.dirname(__file__), '..', '.env')
load_dotenv(dotenv_path=dotenv_path)

GOOGLE_API_KEY = os.getenv("GOOGLE_API_KEY")
NEWS_API_KEY = os.getenv("NEWS_API_KEY")

if not GOOGLE_API_KEY or not NEWS_API_KEY:
    raise ValueError("API keys for Google and NewsAPI must be set in the .env
file in the root project directory.")
```

agents/financial_agent_functions.py: Defines financial agents and their respective functions for handling analysis and decision-making logic.

```
import logging
from typing import Dict, Any
import google.generativeai as genai

from tools.market_data_tools import yahoo_finance_tools
from tools.news_tools import news_api_tools

logger = logging.getLogger(__name__)

def run_quantitative_analysis(analysis_state: Dict[str, Any]) -> bool:
    ticker = analysis_state.get("ticker")
    logger.info(f"AGENT: Starting quantitative analysis for {ticker}...")
    try:
        financial_data = yahoo_finance_tools.get_stock_data(ticker)
        technical_data = yahoo_finance_tools.get_technical_indicators(ticker)
        analysis_state["raw_financial_data"] = financial_data
        analysis_state["raw_technical_data"] = technical_data

        prompt = f"""
        You are a Senior Quantitative Analyst, with expertise in financial
modeling, statistical analysis, and technical analysis.
        You have a PhD in Finance and 8 years of experience at top-tier
investment firms. Your task is to provide an insightful, narrative summary
        of the following financial data for the stock ticker: {ticker}.

        Do not just list the numbers. For each key metric, provide a brief,
italicized *Commentary*
        on what the number signifies in the context of the company's performance
or valuation.

        Here is the financial data:
        ---
        {financial_data}
        ---
        Here is the technical indicator data:
        ---
        {technical_data}
        ---

        Based on the data provided, write a professional summary covering:
        1.  **Valuation:** Market Cap, P/E Ratio, and EPS.
        2.  **Profitability:** Comment on the Profit Margin.
```

```
        3.  **Technicals:** Interpret the current price relative to its moving
averages and RSI.
        """
        model = genai.GenerativeModel(model_name='gemini-1.5-pro')
        response = model.generate_content(prompt)
        analysis_state["quantitative_analysis"] = response.text
        logger.info(f"AGENT: Quantitative analysis for {ticker} completed
successfully.")
        return True
    except Exception as e:
        logger.error(f"AGENT: Error during quantitative analysis for {ticker}:
{e}")
        return False

def run_market_research(analysis_state: Dict[str, Any]) -> bool:
    ticker = analysis_state.get("ticker")
    company_name = analysis_state.get("company_name")
    logger.info(f"AGENT: Starting market research for {company_name}...")
    try:
        news_data = news_api_tools.get_company_news(company_name, ticker)
        analysis_state["raw_news_data"] = news_data


        prompt = f"""
        You are a Senior Market Intelligence Researcher with a background in
journalism and financial analysis.
        You have 10 years of experience tracking market trends, corporate
developments, and macroeconomic factors. Your task is to analyze
        the market sentiment for {company_name} ({ticker}) based on the provided
news articles.

        Instead of a simple list, please synthesize the findings into a cohesive,
narrative summary.

        Here is the recent news data:
        ---
        {news_data}
        ---

        Based on the news, write a paragraph summarizing:
        - The **Overall Sentiment** (e.g., positive, cautiously optimistic,
negative).
        - The **Key Drivers** behind this sentiment, referencing significant news
stories.
        - Any **Potential Catalysts** or future events implied by the news.
```

```
        """
        model = genai.GenerativeModel(model_name='gemini-1.5-pro')
        response = model.generate_content(prompt)
        analysis_state["market_sentiment_analysis"] = response.text
        logger.info(f"AGENT: Market research for {company_name} completed
successfully.")
        return True
    except Exception as e:
        logger.error(f"AGENT: Error during market research for {company_name}:
{e}")
        return False

def run_risk_assessment(analysis_state: Dict[str, Any]) -> bool:
    print(f"\n[Memory Check] Entering Risk Assessor. State contains keys:
{list(analysis_state.keys())}")
    logger.info("AGENT: Starting risk assessment...")
    quantitative_summary = analysis_state.get("quantitative_analysis")
    sentiment_summary = analysis_state.get("market_sentiment_analysis")
    if not quantitative_summary or not sentiment_summary:
        logger.error("Previous analysis summaries not found in state.")
        return False


    prompt = f"""
    You are a Senior Risk Assessment Specialist with 12 years of experience in
investment risk management.
    You hold the FRM (Financial Risk Manager) certification and have worked at
both hedge funds and institutional investment firms.
    Your task is to conduct a comprehensive risk analysis based on the provided
quantitative and market sentiment reports.

    Synthesize the information into a clear, structured risk assessment. For each
risk category,
    provide a brief explanation.

    Here is the Quantitative Analysis:
    ---
    {quantitative_summary}
    ---
    Here is the Market Sentiment Analysis:
    ---
    {sentiment_summary}
    ---

    Based on both reports, provide a detailed risk assessment covering:
```

```
    1.  **Financial Risks:** Focus on valuation concerns (like a high P/E ratio)
and financial stability.
    2.  **Market Risks:** Analyze how market sentiment and broader trends could
impact the stock.
    3.  **Operational & Business Model Risks:** Identify key business challenges
or competitive threats.
    4.  **Conclude with an Overall Risk Rating** (e.g., Low, Moderate, Elevated)
and list the top 3 key risk factors.
    """
    model = genai.GenerativeModel(model_name='gemini-1.5-pro')
    try:
        response = model.generate_content(prompt)
        analysis_state["risk_assessment"] = response.text
        logger.info("AGENT: Risk assessment completed successfully.")
        return True
    except Exception as e:
        logger.error(f"AGENT: Error during risk assessment: {e}")
        return False

def run_report_writing(analysis_state: Dict[str, Any]) -> bool:
    print(f"\n[Memory Check] Entering Report Writer. State contains keys:
{list(analysis_state.keys())}")
    logger.info("AGENT: Starting final report synthesis...")
    quantitative_summary = analysis_state.get("quantitative_analysis")
    sentiment_summary = analysis_state.get("market_sentiment_analysis")
    risk_summary = analysis_state.get("risk_assessment")
    current_date = analysis_state.get("current_date")
    company_name = analysis_state.get("company_name")
    ticker = analysis_state.get("ticker")
    if not all([quantitative_summary, sentiment_summary, risk_summary,
current_date, company_name, ticker]):
        logger.error("Missing one or more analysis components in state for report
writing.")
        return False

    prompt = f"""
    You are a Senior Investment Report Writer for a top-tier investment firm,
known for your
    clear, insightful, and professional analysis. You have 8 years of experience
     writing research reports for institutional investors and have a talent for
    translating complex financial analysis into clear, actionable
recommendations.

    Your task is to synthesize the provided analyses into a single,
comprehensive, and
```

```
    professionally formatted investment report for **{company_name} ({ticker})**
in Markdown format.
    The report must be dated: **{current_date}**.

    **Source Information to Synthesize:**

    1. Quantitative Analysis:
    ---
    {quantitative_summary}
    ---

    2. Market Sentiment Analysis:
    ---
    {sentiment_summary}
    ---

    3. Risk Assessment:
    ---
    {risk_summary}
    ---

    **Instructions for Report Generation:**

    Using all the information above, construct a final report following this
exact structure and tone:

    1.  **Executive Summary:** Begin with a concise, high-level overview. State
the final investment
        recommendation (e.g., Buy, Hold, Sell) upfront and briefly justify it
based on the key
        findings from the subsequent sections.

    2.  **Quantitative Analysis:** Present the quantitative findings. For each
key metric, include the
        brief, italicized *Commentary* provided in the source analysis.

    3.  **Market Sentiment Analysis:** Present the narrative summary of market
sentiment.

    4.  **Risk Assessment:** Detail the primary risks, organized by category, and
state the overall risk rating.

    5.  **Investment Thesis and Rationale:** This is the most important section.
Write a detailed,
```

```
        convincing argument for your final recommendation. Synthesize all the
points—quantitative strengths,
        market mood, and potential risks—into a coherent investment thesis.

    6.  **Conclusion and Next Steps:** Briefly summarize the report and provide
actionable next steps or key
        factors for an investor to monitor.

    **CRITICAL:**
    - Adopt a formal, institutional tone.
    - Ensure the report flows logically and reads as if written by a single,
expert author.
    - Do not include any placeholders like '[Your Firm Name]'.
    """
    model = genai.GenerativeModel(model_name='gemini-1.5-pro')
    try:
        response = model.generate_content(prompt)
        analysis_state["draft_report"] = response.text
        logger.info("AGENT: Draft report generated successfully.")
        return True
    except Exception as e:
        logger.error(f"AGENT: Error during report writing: {e}")
        return False

def run_compliance_validation(analysis_state: Dict[str, Any]) -> bool:
    print(f"\n[Memory Check] Entering Compliance Validator. State contains keys:
{list(analysis_state.keys())}")
    logger.info("AGENT: Starting compliance validation and fact-checking...")
    draft_report = analysis_state.get("draft_report")
    raw_financial_data = analysis_state.get("raw_financial_data")
    if not draft_report or not raw_financial_data:
        logger.error("Draft report or raw financial data not found in state.")
        return False
    prompt = f"""
    You are a Senior Compliance Validator with expertise in financial regulations
    and quality assurance. with 10 years of experience in financial services
compliance and quality assurance. You hold Series 7, 66,
    and 24 licenses and have deep knowledge of investment advisory regulations.
    Your task is to perform a final review of the investment report provided
below.
    Your review must focus on two critical areas:
    1. Factual Accuracy (Hallucination Check): Cross-reference the financial
metrics mentioned in the report's text against the raw data provided. Ensure that
values like P/E Ratio, Market Cap, EPS, etc., in the report are IDENTICAL to the
raw data. Correct any inconsistencies.
```

```
    2. Compliance and Formatting: Ensure the report includes a proper disclaimer,
is professionally formatted, and is free of any placeholder text like '[Your Firm
Name]'.
    Here is the Raw Financial Data for fact-checking:
    ---
    {raw_financial_data}
    ---
    Here is the Draft Report to be validated:
    ---
    {draft_report}
    ---
    Return the final, validated, and corrected version of the report. The output
    should be only the clean, final report text.
    """
    model = genai.GenerativeModel(model_name='gemini-1.5-pro')
    try:
        response = model.generate_content(prompt)
        analysis_state["final_report"] = response.text
        logger.info("AGENT: Compliance validation completed successfully.")
        return True
    except Exception as e:
        logger.error(f"AGENT: Error during compliance validation: {e}")
        return False
```

tools/custom_tools.py: This file defines the proprietary calculate_investment_score tool, which synthesizes financial and sentiment data into a single quantitative score and investment recommendation.

```
from typing import Dict, Any
import logging

logger = logging.getLogger(__name__)

def calculate_investment_score(financial_data: Dict[str, Any], news_sentiment:
Dict[str, Any]) -> Dict[str, Any]:
    """
    Calculates a proprietary investment score based on financial metrics and news
sentiment.

    This is our custom tool. It takes structured data, applies a simple scoring
    logic, and returns a score and recommendation.
    """
    logger.info("CUSTOM TOOL: Calculating proprietary investment score...")
```

```
    scores = {}

    # 1. Valuation Score (based on P/E Ratio)
    pe_ratio = financial_data.get("pe_ratio")
    if isinstance(pe_ratio, (int, float)):
        if pe_ratio < 15:
            scores['valuation'] = 10  # Very Undervalued
        elif pe_ratio < 25:
            scores['valuation'] = 7   # Fairly Valued
        elif pe_ratio < 40:
            scores['valuation'] = 4   # Overvalued
        else:
            scores['valuation'] = 1   # Highly Overvalued
    else:
        scores['valuation'] = 3 # Default score if data is missing

    # 2. Profitability Score (based on Profit Margin)
    profit_margin = financial_data.get("profit_margin")
    if isinstance(profit_margin, (int, float)):
        if profit_margin > 0.20: # 20%
            scores['profitability'] = 10 # Excellent
        elif profit_margin > 0.10: # 10%
            scores['profitability'] = 7  # Good
        elif profit_margin > 0:
            scores['profitability'] = 4  # Average
        else:
            scores['profitability'] = 1  # Poor
    else:
        scores['profitability'] = 3

    # 3. Sentiment Score (based on news analysis)
    sentiment = news_sentiment.get('overall_sentiment', {}).get('sentiment',
'Neutral').lower()
    if sentiment == 'positive':
        scores['sentiment'] = 9
    elif sentiment == 'neutral':
        scores['sentiment'] = 5
    else: # Negative
        scores['sentiment'] = 1

    # Calculate final weighted score (out of 10)
    # Weighting: 40% Valuation, 40% Profitability, 20% Sentiment
    total_score = (scores['valuation'] * 0.4) + (scores['profitability'] * 0.4) +
(scores['sentiment'] * 0.2)
```

```
    # Determine final recommendation
    if total_score > 7.5:
        recommendation = "Strong Buy"
    elif total_score > 6.0:
        recommendation = "Buy"
    elif total_score > 4.0:
        recommendation = "Hold"
    else:
        recommendation = "Sell"

    result = {
        "total_score": round(total_score, 2),
        "recommendation": recommendation,
        "component_scores": scores
    }

    logger.info(f"CUSTOM TOOL: Investment score calculated: {result}")
    return result
```

tools/market_data_tools.py: Provides functions for fetching and processing stock market data (e.g., Yahoo Finance API).

```
"""
Financial Market Data Tools
Yahoo Finance integration for stock data retrieval
"""

import yfinance as yf
import pandas as pd
from datetime import datetime, timedelta
from typing import Dict, Any, Optional
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

class YahooFinanceTools:
    """
    Yahoo Finance data retrieval tools for quantitative analysis
    """

    def __init__(self):
        self.logger = logger

    def get_stock_data(self, ticker: str, period: str = "1y") -> Dict[str, Any]:
```

```
        """
        Fetch comprehensive stock data for analysis

        Args:
            ticker: Stock symbol (e.g., 'TSLA')
            period: Time period ('1d', '5d', '1mo', '3mo', '6mo', '1y', '2y',
'5y', '10y', 'ytd', 'max')

        Returns:
            Dict containing stock data and metrics
        """
        try:
            self.logger.info(f"Fetching stock data for {ticker}")

            # Create ticker object
            stock = yf.Ticker(ticker)

            # Get stock info
            info = stock.info

            # Get historical data
            history = stock.history(period=period)

            # Calculate additional metrics
            current_price = history['Close'].iloc[-1] if not history.empty else
None
            price_change = history['Close'].iloc[-1] - history['Close'].iloc[-2]
if len(history) > 1 else 0
            price_change_percent = (price_change / history['Close'].iloc[-2] *
100) if len(history) > 1 else 0

            # Organize data
            stock_data = {
                "ticker": ticker,
                "company_name": info.get("longName", "N/A"),
                "sector": info.get("sector", "N/A"),
                "industry": info.get("industry", "N/A"),
                "current_price": round(current_price, 2) if current_price else
None,
                "price_change": round(price_change, 2),
                "price_change_percent": round(price_change_percent, 2),
                "market_cap": info.get("marketCap", "N/A"),
                "pe_ratio": info.get("trailingPE", "N/A"),
                "eps": info.get("trailingEps", "N/A"),
                "dividend_yield": info.get("dividendYield", "N/A"),
```

```
                "52_week_high": round(history["High"].max(), 2) if not
history.empty else None,
                "52_week_low": round(history["Low"].min(), 2) if not
history.empty else None,
                "volume": history["Volume"].iloc[-1] if not history.empty else
None,
                "avg_volume": round(history["Volume"].mean(), 0) if not
history.empty else None,
                "beta": info.get("beta", "N/A"),
                "revenue": info.get("totalRevenue", "N/A"),
                "profit_margin": info.get("profitMargins", "N/A"),
                "data_retrieved_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            }

            self.logger.info(f"☑ Successfully retrieved data for {ticker}")
            return stock_data

        except Exception as e:
            self.logger.error(f"❌ Error fetching data for {ticker}: {str(e)}")
            return {
                "ticker": ticker,
                "error": str(e),
                "data_retrieved_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            }

    def get_technical_indicators(self, ticker: str, period: str = "3mo") ->
Dict[str, Any]:
        """
        Calculate basic technical indicators

        Args:
            ticker: Stock symbol
            period: Time period for calculation

        Returns:
            Dict containing technical indicators
        """
        try:
            self.logger.info(f"Calculating technical indicators for {ticker}")

            stock = yf.Ticker(ticker)
            history = stock.history(period=period)

            if history.empty:
                return {"error": "No historical data available"}
```

```
            # Calculate moving averages
            history['MA_20'] = history['Close'].rolling(window=20).mean()
            history['MA_50'] = history['Close'].rolling(window=50).mean()

            # Calculate RSI (simplified)
            delta = history['Close'].diff()
            gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()
            loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()
            rs = gain / loss
            rsi = 100 - (100 / (1 + rs))

            current_price = history['Close'].iloc[-1]
            ma_20 = history['MA_20'].iloc[-1]
            ma_50 = history['MA_50'].iloc[-1]
            current_rsi = rsi.iloc[-1]

            indicators = {
                "ticker": ticker,
                "current_price": round(current_price, 2),
                "moving_average_20": round(ma_20, 2) if not pd.isna(ma_20) else
None,
                "moving_average_50": round(ma_50, 2) if not pd.isna(ma_50) else
None,
                "rsi_14": round(current_rsi, 2) if not pd.isna(current_rsi) else
None,
                "price_vs_ma20": "Above" if current_price > ma_20 else "Below" if
not pd.isna(ma_20) else "N/A",
                "price_vs_ma50": "Above" if current_price > ma_50 else "Below" if
not pd.isna(ma_50) else "N/A",
                "rsi_signal": "Overbought" if current_rsi > 70 else "Oversold" if
current_rsi < 30 else "Neutral" if not pd.isna(current_rsi) else "N/A",
                "calculated_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            }

            self.logger.info(f"☑ Technical indicators calculated for {ticker}")
            return indicators

        except Exception as e:
            self.logger.error(f"❌ Error calculating indicators for {ticker}:
{str(e)}")
            return {
                "ticker": ticker,
                "error": str(e),
                "calculated_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
```

```
            }

# global instance
yahoo_finance_tools = YahooFinanceTools()
```

tools/news_tools.py: Handles fetching and summarizing financial news related to the target stock/company.

```
"""
News Research Tools
News API integration for market sentiment analysis
"""

import requests
from datetime import datetime, timedelta
from typing import Dict, Any, List, Optional
import logging
from config import NEWS_API_KEY

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

class NewsAPITools:
    """
    News API tools for market sentiment and news analysis
    """

    def __init__(self):
        self.api_key = NEWS_API_KEY
        self.base_url = "https://newsapi.org/v2"
        self.logger = logger

        if not self.api_key:
            self.logger.warning("⚠ NEWS_API_KEY not found. News tools will not
work.")

    def get_company_news(self, company_name: str, ticker: str, days_back: int =
7) -> Dict[str, Any]:
        """
        Fetch recent news articles about a company

        Args:
            company_name: Full company name (e.g., 'Tesla')
            ticker: Stock ticker (e.g., 'TSLA')
            days_back: Number of days to look back for news
```

```
        Returns:
            Dict containing news articles and analysis
        """
        try:
            if not self.api_key:
                return {"error": "NEWS_API_KEY not configured"}

            self.logger.info(f"Fetching news for {company_name} ({ticker})")

            # Calculate date range
            to_date = datetime.now()
            from_date = to_date - timedelta(days=days_back)

            # Search query - try both company name and ticker
            query = f'"{company_name}" OR "{ticker}"'

            params = {
                'q': query,
                'from': from_date.strftime('%Y-%m-%d'),
                'to': to_date.strftime('%Y-%m-%d'),
                'sortBy': 'publishedAt',
                'language': 'en',
                'pageSize': 10,
                'apiKey': self.api_key
            }

            # API request
            response = requests.get(f"{self.base_url}/everything", params=params)

            if response.status_code == 200:
                data = response.json()
                articles = data.get('articles', [])

                # Process articles
                processed_articles = []
                sentiment_scores = []

                for article in articles:
                    processed_article = {
                        'title': article.get('title', 'No Title'),
                        'description': article.get('description', 'No
Description'),
                        'source': article.get('source', {}).get('name',
'Unknown'),
```

```
                        'published_at': article.get('publishedAt', ''),
                        'url': article.get('url', ''),
                        'sentiment': self._analyze_sentiment(article.get('title',
'') + ' ' + article.get('description', ''))
                    }
                    processed_articles.append(processed_article)
                    sentiment_scores.append(processed_article['sentiment']['score
'])

                # Calculate overall sentiment
                overall_sentiment =
self._calculate_overall_sentiment(sentiment_scores)

                result = {
                    'company_name': company_name,
                    'ticker': ticker,
                    'total_articles': len(processed_articles),
                    'date_range': f"{from_date.strftime('%Y-%m-%d')} to
{to_date.strftime('%Y-%m-%d')}",
                    'overall_sentiment': overall_sentiment,
                    'articles': processed_articles,
                    'retrieved_at': datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                }

                self.logger.info(f"☑ Retrieved {len(processed_articles)}
articles for {company_name}")
                return result

            else:
                error_msg = f"API request failed with status
{response.status_code}"
                self.logger.error(f"❌ {error_msg}")
                return {"error": error_msg}

        except Exception as e:
            self.logger.error(f"❌ Error fetching news for {company_name}:
{str(e)}")
            return {
                "company_name": company_name,
                "ticker": ticker,
                "error": str(e),
                "retrieved_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            }
```

```python
    def get_market_news(self, category: str = "business", country: str = "us") ->
Dict[str, Any]:
        """
        Fetch general market/business news

        Args:
            category: News category ('business', 'technology', etc.)
            country: Country code ('us', 'gb', etc.)

        Returns:
            Dict containing market news
        """
        try:
            if not self.api_key:
                return {"error": "NEWS_API_KEY not configured"}

            self.logger.info(f"Fetching {category} news for {country}")

            params = {
                'category': category,
                'country': country,
                'pageSize': 5,
                'apiKey': self.api_key
            }

            response = requests.get(f"{self.base_url}/top-headlines",
params=params)

            if response.status_code == 200:
                data = response.json()
                articles = data.get('articles', [])

                processed_articles = []
                for article in articles:
                    processed_articles.append({
                        'title': article.get('title', 'No Title'),
                        'description': article.get('description', 'No
Description'),
                        'source': article.get('source', {}).get('name',
'Unknown'),
                        'published_at': article.get('publishedAt', ''),
                        'url': article.get('url', '')
                    })

                result = {
```

```
                    'category': category,
                    'country': country,
                    'total_articles': len(processed_articles),
                    'articles': processed_articles,
                    'retrieved_at': datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                }

                self.logger.info(f"☑ Retrieved {len(processed_articles)}
{category} articles")
                return result

            else:
                error_msg = f"API request failed with status
{response.status_code}"
                self.logger.error(f"❌ {error_msg}")
                return {"error": error_msg}

        except Exception as e:
            self.logger.error(f"❌ Error fetching market news: {str(e)}")
            return {"error": str(e)}

    def _analyze_sentiment(self, text: str) -> Dict[str, Any]:
        """
        Simple sentiment analysis based on keywords
        (In production, you'd use a proper sentiment analysis model)
        """
        positive_words = ['growth', 'profit', 'increase', 'rise', 'gain',
'positive', 'strong', 'good', 'bullish', 'up']
        negative_words = ['loss', 'decline', 'fall', 'drop', 'negative', 'weak',
'bad', 'bearish', 'down', 'crash']

        text_lower = text.lower()

        positive_count = sum(1 for word in positive_words if word in text_lower)
        negative_count = sum(1 for word in negative_words if word in text_lower)

        if positive_count > negative_count:
            sentiment = 'Positive'
            score = 1
        elif negative_count > positive_count:
            sentiment = 'Negative'
            score = -1
        else:
            sentiment = 'Neutral'
            score = 0
```

```
        return {
            'sentiment': sentiment,
            'score': score,
            'positive_indicators': positive_count,
            'negative_indicators': negative_count
        }

    def _calculate_overall_sentiment(self, sentiment_scores: List[int]) ->
Dict[str, Any]:
        """Calculate overall sentiment from individual scores"""
        if not sentiment_scores:
            return {'sentiment': 'Neutral', 'confidence': 'Low'}

        avg_score = sum(sentiment_scores) / len(sentiment_scores)

        if avg_score > 0.3:
            sentiment = 'Positive'
        elif avg_score < -0.3:
            sentiment = 'Negative'
        else:
            sentiment = 'Neutral'

        confidence = 'High' if abs(avg_score) > 0.5 else 'Medium' if
abs(avg_score) > 0.2 else 'Low'

        return {
            'sentiment': sentiment,
            'score': round(avg_score, 2),
            'confidence': confidence
        }

# global instance
news_api_tools = NewsAPITools()
```

utils/logging_setup.py: Sets up logging for tracking execution steps and saving logs to outputs/workflow.log.

```
import logging
import os

def setup_logging():
    """
    Configures a logger to output to both the console and a file.
    """
```

```
    os.makedirs("outputs", exist_ok=True)

    logger = logging.getLogger()
    logger.setLevel(logging.INFO)

    logger.propagate = False

    if logger.hasHandlers():
        logger.handlers.clear()

    formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s',
datefmt='%Y-%m-%d %H:%M:%S')

    console_handler = logging.StreamHandler()
    console_handler.setFormatter(formatter)
    logger.addHandler(console_handler)

    file_handler = logging.FileHandler("outputs/workflow.log", mode='w',
encoding='utf-8')
    file_handler.setFormatter(formatter)
    logger.addHandler(file_handler)

    return logger
```

2.4 Sample Structured Output

outputs/investment_report_aapl.md: This is an example of the generated report for Apple (AAPL).

Investment Report: Apple Inc. (AAPL)

Proprietary Investment Score: 7.4/10 (Buy)

Apple Inc. (AAPL) Investment Report

Date: September 24, 2025

Recommendation: Hold

Executive Summary:

Apple Inc. currently presents a compelling yet complex investment proposition. While the company boasts robust financial performance, a dominant market position, and positive market sentiment surrounding the iPhone 17 launch, its premium valuation, elevated RSI, and heavy reliance on a single product cycle warrant a cautious approach. Therefore, we recommend a **Hold** rating for AAPL, advising investors to carefully monitor the stock for potential short-term volatility and assess the long-term sustainability of its growth trajectory.

Quantitative Analysis:

Apple Inc. (AAPL), a prominent player in the Consumer Electronics segment of the Technology sector, closed at $254.43, reflecting a decline of 0.64% or $1.65. Trading volume stood at 60,232,300 shares, slightly above its average volume, suggesting modestly elevated trading activity.

1. Valuation:

- **Market Capitalization:** $3.775 trillion. *This signifies Apple's dominant position in the global equity markets and reflects investors' substantial confidence in the company's future prospects.*
- **Price-to-Earnings (P/E) Ratio:** 38.61. *This is relatively high, indicating that investors are willing to pay a premium for Apple's earnings, potentially driven by expectations of continued growth and strong brand loyalty.*
- **Earnings Per Share (EPS):** $6.59. *This solid EPS further supports the high valuation, suggesting robust profitability on a per-share basis.*

2. Profitability:

- **Profit Margin:** 24.30%. *This is a healthy margin, showcasing Apple's ability to effectively manage costs and generate significant profit from its revenue. It underscores the company's pricing power and operational efficiency.* Revenue of $408.62 billion further reinforces the scale of Apple's operations and profitability.

3. Technicals:

- **Moving Averages & Price:** The current price is trading above both its 20-day moving average ($237.15) and its 50-day moving average ($225.65). *This suggests positive short-term and medium-term momentum, with the stock potentially in an uptrend.*
- **Relative Strength Index (RSI):** 66.81. *While not yet a clear sell signal, this level warrants caution as it could suggest a potential pullback in the near term.*

Market Sentiment Analysis:

Market sentiment towards Apple is predominantly positive, driven by strong reported demand for the iPhone 17. Wedbush raised their price target for AAPL to $310 based on this perceived surging demand, with estimates suggesting a potential 20-30% increase over iPhone 16 sales. This positive momentum contributed to Apple's stock reaching record highs, coinciding with broader market gains. While some articles mention regulatory challenges in the EU, these concerns appear overshadowed by the optimism surrounding the iPhone 17 launch. Potential catalysts include the continued success of the iPhone 17 rollout and further updates on Apple's performance tied to this product cycle.

Risk Assessment:

1. Financial Risks: Valuation Risk (high P/E ratio); Financial Stability Risk (mitigated by strong financials)

2. Market Risks: Market Sentiment Risk (overreliance on iPhone 17); Broader Market Risk (correlation with market); Competitive Risk (intense competition in consumer electronics)

3. Operational & Business Model Risks: Product Concentration Risk (iPhone dependence); Regulatory Risk (EU challenges); Innovation Risk (pressure to maintain technological edge)

Overall Risk Rating: Moderate

Top 3 Key Risk Factors: Valuation Risk, Market Sentiment Risk, Product Concentration Risk

Investment Thesis and Rationale:

Apple's current financial strength is undeniable. The company's robust profitability, impressive EPS, and dominant market share justify a premium valuation to some extent. The positive market sentiment surrounding the iPhone 17 launch further supports the bullish narrative. However, the elevated P/E ratio and high RSI suggest the stock price may be overextended in the short term. The market's current enthusiasm hinges heavily on the success of a single product, creating vulnerability to any disappointment in actual sales figures or future product cycles. Moreover, while not currently a major headwind, the looming regulatory challenges in the EU warrant ongoing monitoring. Therefore, while the long-term outlook for Apple remains positive, a "Hold" recommendation is warranted given the potential for short-term volatility and the need to assess the long-term sustainability of its growth beyond the current iPhone cycle.

Conclusion and Next Steps:

Apple presents a compelling investment case supported by strong fundamentals and positive market sentiment. However, valuation and market sentiment risks warrant a cautious approach. Investors should closely monitor the following:

- **iPhone 17 sales figures:** Verify if actual sales align with the current optimistic projections.
- **Regulatory developments in the EU:** Assess the potential impact of ongoing regulatory scrutiny.
- **Developments in the competitive landscape:** Analyze how competitors are responding to Apple's latest offerings.
- **Apple's long-term innovation pipeline:** Evaluate the company's ability to maintain its technological edge and introduce compelling new products and services.

These factors will be crucial in determining whether Apple can justify its premium valuation and maintain its growth trajectory in the long term.

> ***Important Disclaimer:*** *This report is for informational purposes only and does not constitute investment advice. The information presented is based on publicly available data and believed to be reliable, but its accuracy cannot be guaranteed. Investing involves risk, including the potential loss of principal. Consult with a qualified financial advisor before making any investment decisions.*

outputs/workflow.log: Execution logs captured during the ADK workflow run.

```
2025-09-24 10:52:49 - INFO - ADK Investment Analysis System - Initiated
2025-09-24 10:52:53 - INFO - Fetching stock data for AAPL
2025-09-24 10:52:55 - INFO - ☑ Successfully retrieved data for AAPL
2025-09-24 10:52:55 - INFO - Starting analysis for: Apple Inc. (AAPL)
2025-09-24 10:52:55 - INFO - --- Starting Parallel Data Gathering ---
2025-09-24 10:52:55 - INFO - AGENT: Starting quantitative analysis for AAPL...
2025-09-24 10:52:55 - INFO - Fetching stock data for AAPL
2025-09-24 10:52:55 - INFO - AGENT: Starting market research for Apple Inc....
2025-09-24 10:52:55 - INFO - Fetching news for Apple Inc. (AAPL)
2025-09-24 10:52:56 - INFO - ☑ Retrieved 10 articles for Apple Inc.
2025-09-24 10:52:58 - INFO - ☑ Successfully retrieved data for AAPL
2025-09-24 10:52:58 - INFO - Calculating technical indicators for AAPL
2025-09-24 10:52:58 - INFO - ☑ Technical indicators calculated for AAPL
2025-09-24 10:53:01 - INFO - AGENT: Market research for Apple Inc. completed
successfully.
```

```
2025-09-24 10:53:09 - INFO - AGENT: Quantitative analysis for AAPL completed
successfully.
2025-09-24 10:53:09 - INFO - --- Parallel Data Gathering Complete ---
2025-09-24 10:53:09 - INFO - --- Starting Sequential Analysis & Synthesis ---
2025-09-24 10:53:09 - INFO - AGENT: Starting risk assessment...
2025-09-24 10:53:23 - INFO - AGENT: Risk assessment completed successfully.
2025-09-24 10:53:23 - INFO - AGENT: Starting final report synthesis...
2025-09-24 10:53:40 - INFO - AGENT: Draft report generated successfully.
2025-09-24 10:53:40 - INFO - AGENT: Starting compliance validation and fact-
checking...
2025-09-24 10:53:55 - INFO - AGENT: Compliance validation completed successfully.
2025-09-24 10:53:55 - INFO - --- Sequential Analysis & Synthesis Complete ---
2025-09-24 10:53:55 - INFO - --- Running Custom Tool ---
2025-09-24 10:53:55 - INFO - CUSTOM TOOL: Calculating proprietary investment
score...
2025-09-24 10:53:55 - INFO - CUSTOM TOOL: Investment score calculated:
{'total_score': 7.4, 'recommendation': 'Buy', 'component_scores': {'valuation':
4, 'profitability': 10, 'sentiment': 9}}
2025-09-24 10:53:55 - INFO - --- Custom Tool Execution Complete ---
2025-09-24 10:53:55 - INFO - 💾 Report saved to:
outputs/investment_report_aapl.md
2025-09-24 10:53:55 - INFO - 🎉 Workflow finished successfully!
```