Group-3_Final-Project

October 17, 2025

1 Final Team Project – Multi-Agent Financial Analysis System

Greg Bauer | Hassan Ali | Rebecca Cloe
AAI-520 | Group 3 | Submitted: Oct 20, 2025

1.0.1 Github link: https://github.com/DatasanAli/Multi_Agent_Financial_Analysts

---

---

2 ENVIRONMENT SETUP

---

[1]:
```python
import sys
print(sys.executable)
```

```
C:\Users\becky\anaconda3\envs\agents-dup\python.exe
```

[2]:
```python
import importlib.metadata as md
for p in ["langchain","langchain-core","langchain-openai","langgraph","openai"]:
    print(f"{p:18} {md.version(p)}")

from langchain_openai import ChatOpenAI
from langgraph.graph import StateGraph, END
from langgraph.graph.message import add_messages
from langgraph.prebuilt import create_react_agent
print("  All required imports succeeded.")
```

```
langchain          0.3.27
langchain-core     0.3.79
langchain-openai   0.3.35
langgraph          0.2.76
openai             2.3.0
  All required imports succeeded.
```

[3]:
```python
# === Core Python Utilities ===
import os                           # File system access and environment␣
↪variable management
import json                         # Memory and trace serialization
import re, ast                      # GPT output normalization and fallback␣
↪parsing
from pprint import pprint           # Structured debug output for memory and␣
↪trace inspection

# === Type Annotations and Models ===
from typing import List, Dict, Tuple, Union, TypedDict, Annotated, Optional,␣
↪Any  # Agent interfaces and LangGraph state typing
from pydantic import BaseModel                          # Input schema for␣
↪StructuredTool agents

# === IPython Display Utilities ===
from IPython.display import Markdown, display           # Inline rendering of␣
↪markdown-formatted reports and traces

# === External Data Access ===
import yfinance as yf                                   # Live financial␣
↪metadata for ResolverAgent

# === LangChain Core Modules ===
from langchain.prompts import PromptTemplate            # Prompt templates for␣
↪agent and chain interactions
from langchain.schema import AgentAction, AgentFinish   # Agent transitions for␣
↪custom orchestration

# === LangChain Tool Interface ===
from langchain_core.tools import Tool, StructuredTool   # Tool wrappers for␣
↪LangGraph-compatible agent functions
from langchain_core.prompts import PromptTemplate       # Prompt interface for␣
↪LangGraph nodes

# === LangChain OpenAI Integration ===
from langchain_openai import ChatOpenAI                 # Modern OpenAI␣
↪interface for LangGraph-compatible agents

# === LangGraph Orchestration ===
from langgraph.graph import StateGraph, END             # Graph construction␣
↪and terminal node
from langgraph.graph.message import add_messages        # Message state␣
↪management for LangGraph
from langgraph.prebuilt import create_react_agent       # Prebuilt ReAct agent␣
↪node for LangGraph
```

```
# === Environment Variable Loader ===
from dotenv import load_dotenv
load_dotenv()  # Loads variables from .env into os.environ

# === Libraries for http calls to SEC APIs and for date calculations in Finnhub
import datetime as dt
import requests

# === Instantiate Chat Model ===
llm = ChatOpenAI(
    model="gpt-4o-mini",
    temperature=0.3,
    openai_api_key=os.getenv("OPENAI_API_KEY"),
    max_tokens=800
)
```

3 MEMORY PERSISTENCE SETUP

Memory Initialization for Cross-Run Reproducibility and Rubric Traceability This cell scaffolds the persistent memory layer that underpins the entire agentic pipeline. It ensures that outputs—such as thesis drafts, evidence packs, and trace artifacts—can be retained across multiple runs, enabling reproducible analysis and rubric-aligned audit trails. By initializing and managing a lightweight JSON-based store, it supports:

- **Cross-run learning**: Agents can build on prior evaluations.
- **Rubric compliance**: Outputs are traceable to specific tickers and evidence.
- **Reproducibility**: Memory snapshots allow reviewers to inspect and reload results.

This memory system is accessed by orchestration logic, agents, and report generators throughout the notebook.

[4]:
```
# === Persistent Memory Store ===
# This section sets up a lightweight memory system that allows agents to␣
↪"remember" outputs
# across multiple runs of the pipeline. It supports rubric-aligned goals like:
# - Cross-run learning (agents retain prior analysis)
# - Reproducibility (outputs can be audited and reloaded)
# - Traceability (thesis and evidence are linked to specific tickers)

MEMORY_PATH = "agent_memory.json"  # File path for storing agent memory on disk
```

```
# Clear previous memory file at notebook startup to ensure a clean run
# This prevents stale or conflicting data from affecting current execution
if os.path.exists(MEMORY_PATH):
    os.remove(MEMORY_PATH)
    print(f"Deleted existing memory file: {MEMORY_PATH}")
else:
    print(f"No existing memory file found at: {MEMORY_PATH}")

def load_memory() -> Dict:
    """
    Loads memory from disk if the file exists.
    This is called at the beginning of the notebook to hydrate the global␣
 ↪`memory` variable,
    which stores prior agent outputs like thesis, metadata, and trace.
    """
    if os.path.exists(MEMORY_PATH):
        with open(MEMORY_PATH, "r") as f:
            return json.load(f)
    return {}  # If no file exists, start with an empty memory dictionary

def save_memory(memory: Dict):
    """
    Saves the current memory state to disk after each pipeline run.
    This ensures that agent outputs (e.g., thesis, trace, metadata) are␣
 ↪preserved
    for future inspection, reproducibility, and rubric validation.
    """
    with open(MEMORY_PATH, "w") as f:
        json.dump(memory, f, indent=2)

# === Initialize Memory at Startup ===
# This global `memory` variable is used throughout the pipeline to store and␣
 ↪retrieve
# agent outputs. It is accessed by orchestration logic, agents, and trace␣
 ↪renderers.
memory = load_memory()
```

```
No existing memory file found at: agent_memory.json
```

4 AGENT STATE SETUP

Shared State Initialization for Agent Coordination and Rubric-Aligned Output Flow This cell defines the global `state` dictionary and LangGraph-compatible `AgentState` schema that orchestrate data flow across the multi-agent pipeline. Each agent reads from and writes to this shared state, contributing structured outputs—such as evidence, analysis, thesis drafts, and critique—that support reproducibility, traceability, and rubric compliance. This foundational structure ensures consistent input/output handling across all pipeline stages.

[5]:
```
# === Immutable State Graph ===
# This dictionary defines the shared state that flows through the multi-agent␣
↪pipeline.
# Each agent reads from and writes to this state, contributing structured␣
↪outputs that
# support rubric-aligned goals like reproducibility, traceability, and modular␣
↪reasoning.

state = {
    "meta": {},  # Stores resolved company metadata (e.g., name, sector, market␣
↪cap)
                 # Populated by DataCollectionAgent via yfinance and SEC APIs.
                 # Used to personalize thesis and trace outputs.

    "evidence_pack": [],  # Holds preprocessed financial evidence from real␣
↪APIs.
                          # Generated by DataCollectionAgent from yfinance, SEC␣
↪EDGAR, and Finnhub.
                          # Routed to analysis agents for structured evaluation.

    "analysis_bundle": [],  # Contains outputs from specialized agents:
                            # - QualityAgent: evaluates moat, management,␣
↪concentration
                            # - ValuationAgent: assesses pricing and␣
↪justification
                            # - RiskAgent: identifies risks and counterpoints
                            # These insights feed directly into thesis␣
↪synthesis.

    "draft_thesis": {},   # Stores the initial investment thesis.
                           # Synthesized by ThesisWriterAgent using␣
↪analysis_bundle.
                           # Includes bull/bear case, confidence level, and␣
↪catalysts.

    "critic_patch": {},    # Holds suggested edits or improvements from␣
↪CriticAgent.
                           # Demonstrates evaluator-optimizer workflow pattern.
                           # Used to refine thesis for rubric compliance.
    # LangGraph / control
    "messages": [],             # required message channel for prebuilt agent
    "turns": 0,                 # loop counter used by router to stop the graph
}


# === LangGraph Message/State Definition ===
# Must list every key we want to persist across agent steps.
from typing import TypedDict, List, Dict, Any, Annotated
from langgraph.graph.message import add_messages

# === LangGraph Message State ===
# This class defines the message-passing structure used by LangGraph.
# It enables agents to communicate via structured messages and supports
# traceable reasoning across graph nodes.

class AgentState(TypedDict, total=False):
    # Required message channel for the prebuilt ReAct agent
    messages: Annotated[List, add_messages]

    # Persisted scalars/objects carried across steps
    turns: int
    meta: Dict[str, Any]
    evidence_pack: List[Dict[str, Any]]
    analysis_bundle: List[Dict[str, Any]]
    draft_thesis: Dict[str, Any]
    critic_patch: Dict[str, Any]
```

5 NON-CHATGPT BASED AGENTS

5.0.1 ADD NEW AGENTS HERE THAT ACCESS NON CHATGPT APIS

Real-World Financial Data Integration: yfinance, SEC EDGAR, and Finnhub This cell defines comprehensive data collection functions that integrate three real-world financial APIs:

1. **yfinance API**: Extracts company metadata (name, sector, market cap) and key metrics (P/E ratio, ROE, beta)
2. **SEC EDGAR API**: Pulls official 10-K financial statements including revenue, margins, cash flow, and debt ratios
3. **Finnhub API**: Fetches recent company news articles with headlines, summaries, and sources

The `collect_comprehensive_data` function orchestrates all three sources, resolving ticker symbols to CIK identifiers and converting raw API responses into rubric-aligned evidence for downstream agents. This multi-source approach ensures rich, authoritative financial analysis.

[6]:
```
# === API Configuration Constants ===
SEC_BASE = "https://data.sec.gov"
SEC_TICKER_CIK = "https://www.sec.gov/files/company_tickers.json"
FINNHUB_BASE = "https://finnhub.io/api/v1"
HEADERS_SEC = {"User-Agent": "Academic Research academic@university.edu",␣
↪"Accept-Encoding": "gzip, deflate"}
FINNHUB_API_KEY = os.getenv("FINNHUB_API_KEY", "").strip()

# === Core math libaries ===
import math
import statistics

# === Helper: HTTP Request with Error Handling ===
def fetch_json(url: str, params: Optional[Dict[str, str]] = None, headers:␣
↪Optional[Dict[str, str]] = None) -> Optional[Dict[str, Any]]:
    """Generic JSON fetcher with timeout and error handling."""
    try:
        r = requests.get(url, params=params, headers=headers, timeout=30)
        r.raise_for_status()
        return r.json()
    except Exception as e:
        print(f"[WARN] GET {url} failed: {e}")
        return None

# === 1. Ticker → CIK Resolution (SEC) ===
def resolve_ticker_to_cik(ticker: str) -> Optional[Dict[str, str]]:
    """
    Resolves stock ticker to SEC CIK identifier and company name.
    Handles both list- and dict-shaped payloads from SEC.
    """
    try:
        data = fetch_json(SEC_TICKER_CIK, headers=HEADERS_SEC)
        if not data:
            return None


        entries: List[Dict[str, Any]] = []
        if isinstance(data, list):
            entries = data
        elif isinstance(data, dict):
            # If it's a dict of objects { "0": {ticker, cik_str, title}, ... }␣
↪OR { "data": [...] }
            if "data" in data and isinstance(data["data"], list):
                entries = data["data"]
            else:
                try:
                    entries = list(data.values())  # best-effort fallback
                except Exception:
                    entries = []

        t_up = ticker.upper()
        for entry in entries:
            et = (entry.get("ticker") or entry.get("Ticker") or "").upper()
            if et == t_up:
                cik_val = entry.get("cik_str") or entry.get("cik") or entry.
↪get("CIK")
                if cik_val is None:
                    continue
                cik_str = str(cik_val).zfill(10)
                return {
                    "ticker": t_up,
                    "cik": cik_str,
                    "company_name": entry.get("title") or entry.get("name") or␣
↪"Unknown",
                }
    except Exception as e:
        print(f"[Resolver] Error fetching CIK for {ticker}: {e}")
    return None

# === 2. SEC EDGAR Financials ===
def latest_annual_value(values: List[Dict[str, Any]]) -> Optional[Dict[str,␣
↪Any]]:
    """Extracts most recent annual filing from SEC data."""
    if not values:
        return None
    annuals = [v for v in values if v.get("fp") == "FY" or v.get("form") in␣
↪{"10-K", "20-F"}]
    if not annuals:
        annuals = values[:]

    def parse_date(x):
        try:
            return dt.datetime.fromisoformat((x.get("end") or "").split("T")[0])
        except Exception:
            return dt.datetime.min

    # Prefer most recent end date, then fiscal year if present
    annuals.sort(key=lambda v: (parse_date(v), int(v.get("fy") or 0)),␣
↪reverse=True)
    return annuals[0] if annuals else None
```

```
def get_us_gaap(facts: Dict[str, Any], tag: str) -> Optional[float]:
    """Safely extracts US-GAAP metric from SEC facts."""
    try:
        tag_data = facts["facts"]["us-gaap"][tag]
        units = tag_data.get("units", {})
        usd_values = None
        # Try common unit keys; fallback to the first unit list found
        for unit_key in ["USD", "USD/shares", "pure"]:
            if unit_key in units:
                usd_values = units[unit_key]
                break
        if not usd_values and units:
            usd_values = next(iter(units.values()))
        latest = latest_annual_value(usd_values or [])
        return float(latest.get("val")) if latest and latest.get("val") is not␣
  ↪None else None
    except Exception:
        return None

def pull_sec_financials(cik: str) -> Dict[str, Any]:
    """Fetches SEC financial data and calculates key ratios."""
    if not cik:
        return {}
    url = f"{SEC_BASE}/api/xbrl/companyfacts/CIK{cik}.json"
    data = fetch_json(url, headers=HEADERS_SEC)
    if not data:
        return {}

    g = lambda tag: get_us_gaap(data, tag)

    revenue = g("RevenueFromContractWithCustomerExcludingAssessedTax") or␣
  ↪g("SalesRevenueNet") or g("Revenues")
    gross_profit = g("GrossProfit")
    operating_income = g("OperatingIncomeLoss") or g("OperatingIncome")
    net_income = g("NetIncomeLoss") or g("ProfitLoss")
    current_assets = g("AssetsCurrent")
    current_liabilities = g("LiabilitiesCurrent")
    equity =␣
  ↪g("StockholdersEquityIncludingPortionAttributableToNoncontrollingInterest")␣
  ↪or g("StockholdersEquity")
    cash = g("CashAndCashEquivalentsAtCarryingValue") or␣
  ↪g("CashCashEquivalentsAndShortTermInvestments")

    lt_debt = g("LongTermDebtNoncurrent") or g("LongTermDebt")
    current_portion_lt_debt = g("LongTermDebtCurrent")
    st_borrow = g("ShortTermBorrowings")
    commercial_paper = g("CommercialPaper")

    debt_parts = [v for v in [lt_debt, current_portion_lt_debt, st_borrow,␣
  ↪commercial_paper] if v is not None]
    total_debt = sum(debt_parts) if debt_parts else None

    # Defensive ratio math (avoid ZeroDivisionError and None-propagation)
    gross_margin = (gross_profit / revenue) if (revenue not in (None, 0) and␣
  ↪gross_profit is not None) else None
    operating_margin = (operating_income / revenue) if (revenue not in (None,␣
  ↪0) and operating_income is not None) else None
    current_ratio = (current_assets / current_liabilities) if (current_assets␣
  ↪and current_liabilities) else None
    debt_to_equity = (total_debt / equity) if (equity not in (None, 0) and␣
  ↪total_debt is not None) else None

    return {
        "revenue": revenue,
        "gross_profit": gross_profit,
        "operating_income": operating_income,
        "net_income": net_income,
        "cash": cash,
        "current_assets": current_assets,
        "current_liabilities": current_liabilities,
        "equity": equity,
        "total_debt": total_debt,
        "gross_margin": gross_margin,
        "operating_margin": operating_margin,
        "current_ratio": current_ratio,
        "debt_to_equity": debt_to_equity,
    }

# === 3. Finnhub News ===
def pull_finnhub_news(ticker: str, days: int = 30) -> Dict[str, Any]:
    """Fetches recent news articles from Finnhub."""
    if not FINNHUB_API_KEY:
        print("[WARN] Missing FINNHUB_API_KEY in .env")
        return {"error": "Missing FINNHUB_API_KEY"}

    to_date = dt.date.today()
    from_date = to_date - dt.timedelta(days=days)
    url = f"{FINNHUB_BASE}/company-news"
    params = {"symbol": ticker, "from": str(from_date), "to": str(to_date),␣
  ↪"token": FINNHUB_API_KEY}
    data = fetch_json(url, params=params)
    if not data or (isinstance(data, dict) and data.get("error")):
        return {"error": data.get("error") if isinstance(data, dict) else␣
↪"Failed to get news"}

    items = sorted(data, key=lambda x: x.get("datetime", 0), reverse=True)[:20]
    for it in items:
        try:
            ts = it.get("datetime", 0)
            it["datetime_iso"] = dt.datetime.fromtimestamp(ts, tz=dt.timezone.
↪utc).strftime("%Y-%m-%d %H:%M:%S")
        except Exception:
            pass
    return {"count": len(data), "sample": items}

# === 4. yfinance Price Trends ===
def pull_price_trend_yf(ticker: str, days: int = 60) -> Dict[str, Any]:
    """Fetches historical price data and calculates technical indicators."""
    try:
        stock = yf.Ticker(ticker)
        hist = stock.history(period=f"{days}d")
        if hist.empty:
            return {"error": "No price data returned"}

        closes_series = hist["Close"].dropna()
        closes = [(idx.strftime("%Y-%m-%d"), float(val)) for idx, val in␣
↪closes_series.items()]
        latest_close = closes[-1][1]
        oldest_close = closes[0][1] if closes else None
        pct_change = (latest_close / oldest_close - 1.0) if (oldest_close not␣
↪in (None, 0)) else None

        sma20 = float(closes_series.tail(20).mean()) if len(closes_series) >=␣
↪20 else None
        sma50 = float(closes_series.tail(50).mean()) if len(closes_series) >=␣
↪50 else None

        vals = list(closes_series.values)
        logrets = [math.log(vals[i] / vals[i-1]) for i in range(1, len(vals))␣
↪if vals[i-1]]
        vol = statistics.pstdev(logrets) * math.sqrt(252) if logrets else None

        return {
            "latest_close": latest_close,
            "oldest_close": oldest_close,
            "pct_change": pct_change,
            "sma20": sma20,
            "sma50": sma50,
            "annualized_vol": vol,
            "sample": closes[-5:],
            "days": days
        }
    except Exception as e:
        return {"error": str(e)}

# === 5. Orchestrated Data Collection ===
def collect_comprehensive_data(ticker: str) -> Dict[str, Any]:
    """
    Unified data collection from yfinance, SEC EDGAR, and Finnhub.
    Returns structured dictionary with metadata, SEC financials, price trends,␣
↪and news.
    """
    print(f"[INFO] Collecting data for {ticker}...")

    # 1. Resolve ticker to CIK and get yfinance metadata
    stock = yf.Ticker(ticker)
    try:
        info = stock.info or {}
    except Exception as e:
        print(f"[WARN] yfinance .info failed: {e}")
        info = {}

    resolved = resolve_ticker_to_cik(ticker)
    cik = resolved.get("cik") if resolved else None

    meta = {
        "ticker": ticker.upper(),
        "company_name": info.get("longName") or (resolved.get("company_name")␣
↪if resolved else "Unknown"),
        "sector": info.get("sector", "Unknown"),
        "industry": info.get("industry", "Unknown"),
        "marketCap": info.get("marketCap", "Unknown"),
        "price": info.get("currentPrice", "Unknown"),
        "exchange": info.get("exchange", "Unknown"),
        "cik": cik or "Unknown",
    }

    # 2. Fetch SEC financials
    sec_data = pull_sec_financials(cik) if cik else {}

    # 3. Fetch price trends
    price_data = pull_price_trend_yf(ticker)

    # 4. Fetch news
    news_data = pull_finnhub_news(ticker)

    # 5. Convert to rubric-aligned evidence
    evidence_pack: List[Dict[str, Any]] = []

    # From yfinance
    if info.get("trailingPE") is not None:
        evidence_pack.append({
            "source": "yfinance",
            "section_hint": "Valuation",
            "text": f"P/E ratio is {info['trailingPE']:.2f}",
            "score": 0.9
        })
    if info.get("returnOnEquity") is not None:
        evidence_pack.append({
            "source": "yfinance",
            "section_hint": "Quality",
            "text": f"Return on equity is {info['returnOnEquity']:.4f}",
            "score": 0.85
        })
    if info.get("beta") is not None:
        evidence_pack.append({
            "source": "yfinance",
            "section_hint": "Risk",
            "text": f"Beta is {info['beta']:.3f}",
            "score": 0.8
        })

    # From SEC financials
    if sec_data.get("gross_margin") is not None:
        evidence_pack.append({
            "source": "SEC EDGAR",
            "section_hint": "Quality",
            "text": f"Gross margin is {sec_data['gross_margin']:.1%}",
            "score": 0.9
        })
    if sec_data.get("operating_margin") is not None:
        evidence_pack.append({
            "source": "SEC EDGAR",
            "section_hint": "Quality",
            "text": f"Operating margin is {sec_data['operating_margin']:.1%}",
            "score": 0.9
        })
    if sec_data.get("current_ratio") is not None:
        evidence_pack.append({
            "source": "SEC EDGAR",
            "section_hint": "Risk",
            "text": f"Current ratio is {sec_data['current_ratio']:.2f}",
            "score": 0.85
        })
    if sec_data.get("debt_to_equity") is not None:
        evidence_pack.append({
            "source": "SEC EDGAR",
            "section_hint": "Risk",
            "text": f"Debt-to-equity ratio is {sec_data['debt_to_equity']:.2f}",
            "score": 0.85
        })

    # From price trends
    if price_data.get("pct_change") is not None:
        evidence_pack.append({
            "source": "yfinance",
            "section_hint": "Valuation",
            "text": f"{price_data['days']}-day price change is␣
↪{price_data['pct_change']:.1%}",
            "score": 0.8
        })
    if price_data.get("annualized_vol") is not None:
        evidence_pack.append({
            "source": "yfinance",
            "section_hint": "Risk",
            "text": f"Annualized volatility is {price_data['annualized_vol']:.
↪1%}",
            "score": 0.8
        })

    # From news (top 5 headlines)
    if not news_data.get("error") and news_data.get("sample"):
        for i, article in enumerate(news_data["sample"][:5]):
            if article.get("headline"):
                evidence_pack.append({
                    "source": "Finnhub",
                    "section_hint": "News",
                    "text": f"Recent news: {article['headline']}",
                    "score": 0.75,
                    "date": article.get("datetime_iso", "Unknown"),
                    "url": article.get("url", "")
                })

    print(f"[INFO] Collected {len(evidence_pack)} evidence items from 3␣
↪sources")

    return {
        "meta": meta,
        "sec": sec_data,
        "prices": price_data,
        "news": news_data,
        "evidence_pack": evidence_pack
    }
```

---

6 EVIDENCE PREPROCESSING UTILITIES

Evidence Normalization and Aggregation for Multi-Source API Data This cell defines preprocessing utilities that standardize and merge evidence from yfinance, SEC EDGAR, and Finnhub APIs. `normalize_evidence` ensures consistent formatting and filters out low-quality entries, while `aggregate_evidence` consolidates multiple evidence sources into a unified pack. These functions maintain rubric compliance and enable structured reasoning across all downstream agents.

[7]:
```
# === Evidence Preprocessing Utilities (Upgraded) ===
# Cleans, consolidates, and prioritizes evidence from multiple real APIs.
# Adds: canonical sections, de-duplication, date parsing, score normalization,
#       top-k filtering per section, and safe handling of odd inputs.

from typing import Optional, Iterable

_CANON_SECTIONS = {
    "valuation": "Valuation",
    "quality": "Quality",
    "risk": "Risk",
    "news": "News",
    "general": "General",
}

def _canonical_section(name: Optional[str]) -> str:
    if not name or not isinstance(name, str):
        return _CANON_SECTIONS["general"]
    key = name.strip().lower()
    return _CANON_SECTIONS.get(key, name.title())

def _parse_date(dt_like: Optional[str]) -> Optional[dt.datetime]:
    """Accepts 'YYYY-MM-DD HH:MM:SS' or 'YYYY-MM-DD' or ISO-ish; returns aware␣
 ↪UTC datetime when possible."""
    if not dt_like or not isinstance(dt_like, str):
        return None
    s = dt_like.strip()
    try:
        # Try full datetime
        return dt.datetime.fromisoformat(s).astimezone(dt.timezone.utc)
    except Exception:
        pass
    try:
        # Try date only
        return dt.datetime.fromisoformat(s.split("T")[0]).replace(tzinfo=dt.
↪timezone.utc)
    except Exception:
        return None

def _normalize_score(x: Optional[float], default: float = 0.85) -> float:
    try:
        v = float(x)
        if v != v:  # NaN check
            return default
        return max(0.0, min(1.0, v))
    except Exception:
        return default

def _text_from_evidence(e: Dict[str, Any]) -> str:
    return (e.get("text")
            or e.get("headline")
            or e.get("title")
            or str(e))

def normalize_evidence(evidence: List[Dict[str, Any]], *, drop_unknown: bool =␣
↪True) -> List[Dict[str, Any]]:
    """
    Cleans and standardizes raw evidence from yfinance, SEC EDGAR, and Finnhub.

    Parameters:
        evidence: Raw evidence entries from collect_comprehensive_data
        drop_unknown: If True, skip entries with placeholder or empty text.

    Returns:
        List[Dict]: Normalized evidence with consistent keys and source␣
↪tracking.
                    Keys: text, score, section_hint, source, date, url
    """
    if not evidence:
        return []

    normalized: List[Dict[str, Any]] = []
    for e in evidence:
        if not isinstance(e, dict):
            # best-effort wrap
            e = {"text": str(e), "source": "Unknown"}

        text = _text_from_evidence(e)
        if not isinstance(text, str):
            text = str(text)

        if drop_unknown:
            if (not text.strip()) or ("Unknown" in text):
                continue

        section_hint = _canonical_section(e.get("section_hint", "General"))
        date_val = e.get("date")
        parsed_date = _parse_date(date_val) if isinstance(date_val, str) else␣
↪None

        normalized.append({
            "text": text.strip(),
            "score": _normalize_score(e.get("score", 0.85)),
            "section_hint": section_hint,
            "source": (e.get("source") or "Unknown"),
            "date": date_val,
            "date_parsed": parsed_date,  # helper for sorting
            "url": e.get("url") or None,
        })

    return normalized

def _dedupe_evidence(evidence: List[Dict[str, Any]]) -> List[Dict[str, Any]]:
    """
    Deduplicate by (text,url,section). Keep the higher score and more recent␣
↪item.
    """
    if not evidence:
        return []

    best: Dict[Tuple[str, Optional[str], str], Dict[str, Any]] = {}
    for item in evidence:
        key = (item.get("text", ""), item.get("url"), item.get("section_hint",␣
↪"General"))
        prev = best.get(key)
        if prev is None:
            best[key] = item
            continue
        # Prefer higher score; tie-breaker: most recent date
        s_new, s_prev = item.get("score", 0), prev.get("score", 0)
        if s_new > s_prev:
            best[key] = item
        elif s_new == s_prev:
            d_new = item.get("date_parsed")
            d_prev = prev.get("date_parsed")
            if d_new and (not d_prev or d_new > d_prev):
                best[key] = item

    return list(best.values())

def _topk_per_section(evidence: List[Dict[str, Any]], k_map: Optional[Dict[str,␣
↪int]] = None) -> List[Dict[str, Any]]:
    """
    Take top-k per canonical section by score (desc), then recency.
    Default: Valuation:3, Quality:3, Risk:3, News:5, General:2
    """
    if not evidence:
        return []
    if k_map is None:
        k_map = {"Valuation": 3, "Quality": 3, "Risk": 3, "News": 5, "General":␣
↪2}

    buckets: Dict[str, List[Dict[str, Any]]] = {}
    for it in evidence:
        sec = it.get("section_hint", "General")
        buckets.setdefault(sec, []).append(it)

    picked: List[Dict[str, Any]] = []
    for sec, items in buckets.items():
        # Sort by score desc, then newest date
        items_sorted = sorted(
            items,
            key=lambda x: (
                float(x.get("score", 0.0)),
                x.get("date_parsed") or dt.datetime.min.replace(tzinfo=dt.
↪timezone.utc),
            ),
            reverse=True
        )
        picked.extend(items_sorted[: k_map.get(sec, 2)])

    # Stable overall order: Valuation, Quality, Risk, News, General
    order = ["Valuation", "Quality", "Risk", "News", "General"]
    picked.sort(key=lambda x: (order.index(x.get("section_hint", "General")) if␣
↪x.get("section_hint") in order else len(order),
                              -(x.get("score", 0.0))))
    # Strip helper field date_parsed
    for it in picked:
        it.pop("date_parsed", None)
    return picked

def aggregate_evidence(evidence_list: List[Union[List[Dict[str, Any]],␣
↪Dict[str, Any]]]) -> List[Dict[str, Any]]:
    """
    Flattens and merges evidence from multiple API sources.

    Parameters:
        evidence_list: Evidence from collect_comprehensive_data or list of␣
↪lists/dicts:
                      - a single list[dict]
                      - a list of lists[dict]
                      - dicts with "evidence_pack"
                      - dicts with "sample" (news shape)

    Returns:
        List[Dict]: Merged evidence entries (no normalization/dupes removed␣
↪here).
    """
    if not evidence_list:
        return []

    # If it's already a flat list of dicts
    if isinstance(evidence_list, list) and evidence_list and␣
↪isinstance(evidence_list[0], dict):
        return evidence_list

    merged: List[Dict[str, Any]] = []
    for group in evidence_list:
        if group is None:
            continue
        if isinstance(group, list):
            merged.extend([g for g in group if isinstance(g, dict)])
        elif isinstance(group, dict):
            if "evidence_pack" in group and isinstance(group["evidence_pack"],␣
↪list):
                merged.extend([g for g in group["evidence_pack"] if␣
↪isinstance(g, dict)])
            elif "sample" in group and isinstance(group["sample"], list):
                # Finnhub news shape; wrap minimally
                for a in group["sample"]:
                    if not isinstance(a, dict):
                        continue
                    merged.append({
                        "text": a.get("headline") or a.get("title") or "",
                        "score": 0.75,
                        "section_hint": "News",
                        "source": "Finnhub",
                        "date": a.get("datetime_iso"),
                        "url": a.get("url")
                    })
    return merged

# === Convenience: one-pass clean → dedupe → prioritize ===
def prepare_evidence_for_agents(
    evidence_list: List[Union[List[Dict[str, Any]], Dict[str, Any]]],
    *,
    topk_map: Optional[Dict[str, int]] = None,
    drop_unknown: bool = True
) -> List[Dict[str, Any]]:
    """
    Pipeline:
      aggregate → normalize → dedupe → top-k per section

    Returns a compact, high-signal evidence set ready for analysis/thesis␣
 ↪agents.
    """
    aggregated = aggregate_evidence(evidence_list)
    normalized = normalize_evidence(aggregated, drop_unknown=drop_unknown)
    deduped = _dedupe_evidence(normalized)
    prioritized = _topk_per_section(deduped, k_map=topk_map)
    return prioritized
```

7 CHATGPT BASED AGENTS

GPT-Based Agent Suite for Structured Financial Analysis and Thesis Synthesis This cell defines the core GPT-powered agents that drive the analytical reasoning and reporting stages of the pipeline. Each agent consumes normalized evidence and contributes structured outputs to the shared state:

- `gpt_quality_agent` evaluates competitive moat, customer concentration, and management track record.
- `gpt_valuation_agent` assesses valuation signals and justification.

- `gpt_risk_agent` identifies key risks and counterpoints to the bull case.
- `gpt_critic_agent` reviews the draft thesis for clarity and rubric compliance.
- `gpt_thesis_writer` synthesizes a markdown-formatted investment thesis from agent outputs.

These agents rely on prompt chaining and structured parsing to ensure reproducibility, rubric alignment, and traceable reasoning across the pipeline. The `safe_parse_gpt_output` utility ensures robust handling of GPT responses, enabling consistent downstream integration.

[8]:
```python
# =====================
# Evidence-Bounded Agents (APIs only)
# =====================
debug = True

import json
from typing import Any, Dict, List, Tuple, Union, Optional
from pydantic import BaseModel, Field, ValidationError

# ---------- Safe Parser ----------
def safe_parse_gpt_output(response: str) -> Dict:
    """Parses GPT output into JSON with resilient fallback handling."""
    if not isinstance(response, str):
        return {"error": "Non-string response", "raw": str(response)}

    refusal_phrases = [
        "I can't provide", "I need more data", "Please provide", "As an AI",␣
↪"I'm sorry"
    ]
    if any(phrase in response for phrase in refusal_phrases):
        return {"error": "Refusal", "raw": response}

    if "```json" in response:
        response = response.split("```json")[1].split("```")[0].strip()
    elif "```" in response:
        response = response.split("```")[1].split("```")[0].strip()

    if "{" in response:
        response = response[response.find("{"):]

    try:
        return json.loads(response.replace("'", '"'))
    except json.JSONDecodeError:
        try:
            import ast
            return ast.literal_eval(response)
        except Exception as e:
            return {"error": "Parse failure", "exception": str(e), "raw":␣
↪response}
```

```
# ---------- Evidence Utilities ----------
def _index_evidence(evidence: List[Dict[str, Any]]) -> Tuple[str, Dict[int,␣
↪Dict[str, Any]]]:
    """Formats evidence list for citation-safe prompts."""
    idx_map: Dict[int, Dict[str, Any]] = {}
    lines = []
    for i, e in enumerate(evidence, start=1):
        idx_map[i] = e
        sec = e.get("section_hint", "General")
        txt = (e.get("text") or "").strip()
        src = e.get("source", "Unknown")
        dt_s = e.get("date")
        lines.append(f"[{i}] ({sec}) {txt} - source={src}" + (f"  date={dt_s}"␣
↪if dt_s else ""))
    return "\n".join(lines), idx_map


# ---------- Pydantic Schemas ----------
def _ensure_min_list(lst: Optional[List[Any]], min_len: int, fallback:␣
↪List[Any]) -> List[Any]:
    if not isinstance(lst, list) or len(lst) < min_len:
        return list(fallback)
    return lst

class QualityOut(BaseModel):
    profitability_strength: str = Field(..., description="Summary of gross/
↪operating margins and ROE patterns.")
    efficiency_and_scale: str = Field(..., description="Commentary on asset␣
↪utilization, scale advantage, or cost control inferred from evidence.")
    financial_flexibility: str = Field(..., description="Assessment of␣
↪liquidity, leverage, and debt ratios; highlight resilience or vulnerability.
↪")
    evidence_gaps: List[str] = Field(
        default_factory=lambda: ["segment-level revenue data", "cash flow␣
↪trends", "peer comparison benchmarks"],
        description="Information missing for a deeper analysis."
    )
    citation_indices: List[int] = Field(default_factory=list)

    try:
        from pydantic import model_validator
        @model_validator(mode="after")  # pydantic v2
        def _fix_lists(self):
            self.evidence_gaps = _ensure_min_list(self.evidence_gaps, 1,␣
↪["additional disclosures required"])
```

```
            self.citation_indices = _ensure_min_list(self.citation_indices, 0,␣
↪[])
            return self
    except Exception:
        from pydantic import root_validator
        @root_validator  # pydantic v1
        def _fix_lists(cls, values):
            values["evidence_gaps"] = _ensure_min_list(values.
↪get("evidence_gaps"), 1, ["additional disclosures required"])
            values["citation_indices"] = _ensure_min_list(values.
↪get("citation_indices"), 0, [])
            return values

class ValuationOut(BaseModel):
    valuation_view: str = Field(..., description="'undervalued', 'fairly␣
↪valued', or 'overvalued' based on metrics.")
    justification: str = Field(..., description="Key ratios and evidence␣
↪supporting the valuation stance.")
    citation_indices: List[int] = Field(default_factory=list)

    try:
        from pydantic import model_validator
        @model_validator(mode="after")
        def _fix_lists(self):
            self.citation_indices = _ensure_min_list(self.citation_indices, 0,␣
↪[])
            return self
    except Exception:
        from pydantic import root_validator
        @root_validator
        def _fix_lists(cls, values):
            values["citation_indices"] = _ensure_min_list(values.
↪get("citation_indices"), 0, [])
            return values

class RiskOut(BaseModel):
    risks: List[str] = Field(..., description="Financial and operational risks␣
↪derived from evidence.")
    mitigants: List[str] = Field(default_factory=list)
    citation_indices: List[int] = Field(default_factory=list)

    try:
        from pydantic import model_validator
        @model_validator(mode="after")
        def _fix_lists(self):
            self.risks = _ensure_min_list(self.risks, 1,␣
↪["insufficient_evidence"])
            self.mitigants = _ensure_min_list(self.mitigants, 0, [])
            self.citation_indices = _ensure_min_list(self.citation_indices, 0,␣
↪[])
            return self
    except Exception:
        from pydantic import root_validator
        @root_validator
        def _fix_lists(cls, values):
            values["risks"] = _ensure_min_list(values.get("risks"), 1,␣
↪["insufficient_evidence"])
            values["mitigants"] = _ensure_min_list(values.get("mitigants"), 0,␣
↪[])
            values["citation_indices"] = _ensure_min_list(values.
↪get("citation_indices"), 0, [])
            return values

class ThesisOut(BaseModel):
    thesis: str = Field(..., description="1-2 sentence synthesis grounded in␣
↪valuation, quality, and risk.")
    bull_case: List[str] = Field(..., description="2-5 bullets")
    bear_case: List[str] = Field(..., description="2-5 bullets")
    catalysts: List[str] = Field(..., description="1-5 bullets")
    confidence: float = 0.6
    citation_indices: List[int] = Field(default_factory=list)

    try:
        from pydantic import model_validator  # type: ignore[attr-defined]
        @model_validator(mode="after")
        def _fix_lists(self):
            self.bull_case = _ensure_min_list(self.bull_case, 2,␣
↪["insufficient_evidence", "see analysis"])
            self.bear_case = _ensure_min_list(self.bear_case, 2,␣
↪["insufficient_evidence", "see analysis"])
            self.catalysts = _ensure_min_list(self.catalysts, 1,␣
↪["insufficient_evidence"])
            self.citation_indices = _ensure_min_list(self.citation_indices, 0,␣
↪[])
            return self
    except Exception:
        from pydantic import root_validator  # type: ignore
        @root_validator
        def _fix_lists(cls, values):
            values["bull_case"] = _ensure_min_list(values.get("bull_case"), 2,␣
↪["insufficient_evidence", "see analysis"])
```

```
            values["bear_case"] = _ensure_min_list(values.get("bear_case"), 2,␣
↪["insufficient_evidence", "see analysis"])
            values["catalysts"] = _ensure_min_list(values.get("catalysts"), 1,␣
↪["insufficient_evidence"])
            values["citation_indices"] = _ensure_min_list(values.
↪get("citation_indices"), 0, [])
            return values

class CriticOut(BaseModel):
    valid: bool = Field(..., description="Whether thesis meets quality␣
↪standards")
    summary: str = Field(..., description="A few sentences evaluating the␣
↪analysis results")
    patch: str = Field(..., description="Suggest improvements in analysis␣
↪results")


# ---------- LLM JSON helper ----------
def _llm_json(prompt_text: str, tries: int = 2) -> Dict[str, Any]:
    """Invokes LLM and parses JSON output with retry."""
    last = {}
    for t in range(tries):
        resp = llm.invoke(prompt_text)
        if debug:
            print(f"[LLM raw t={t+1}] {resp.content[:400]}")
        parsed = safe_parse_gpt_output(resp.content)
        if "error" not in parsed:
            return parsed
        last = parsed
    return last


# ---------- Helper: prepare evidence from mixed LangGraph inputs ----------
def _prepare_evidence_from_input(tool_input: Any) -> List[Dict]:
    """
    Accepts:
      - List[Dict] (already-evidence),
      - Dict with 'evidence' or '__arg1' (either a list or a ticker),
      - str or anything else (common in create_react_agent tool calls).

    Falls back to state["evidence_pack"] and normalizes if needed.
    """
    ev: List[Dict] = []

    # Already a list of evidence dicts
    if isinstance(tool_input, list):
        ev = tool_input

    # Dict input: common LangGraph tool call shapes
    elif isinstance(tool_input, dict):
        if isinstance(tool_input.get("evidence"), list):
            ev = tool_input["evidence"]
        elif isinstance(tool_input.get("__arg1"), list):
            ev = tool_input["__arg1"]
        elif isinstance(tool_input.get("__arg1"), str):
            # Sometimes __arg1 is a ticker; use global state evidence instead␣
↪of re-fetch
            ev = state.get("evidence_pack", [])

    # String or other → try global state evidence_pack
    else:
        ev = state.get("evidence_pack", [])

    # Normalize if looks raw
    if ev and ("section_hint" not in ev[0] or "text" not in ev[0]):
        try:
            ev = normalize_evidence(ev)
        except Exception:
            pass

    # Last-chance fallback to normalized state evidence
    if (not ev) and state.get("evidence_pack"):
        try:
            ev = normalize_evidence(state["evidence_pack"])
        except Exception:
            ev = state.get("evidence_pack", [])

    return ev if isinstance(ev, list) else []


# ---------- Agents  ----------
def gpt_quality_agent(tool_input: Any) -> Dict:
    """Evaluates profitability, efficiency, and flexibility - not qualitative␣
↪moat or management."""
    evidence = _prepare_evidence_from_input(tool_input)

    # If still no evidence, return structured fallback (no LLM call → no␣
↪refusals)
    if not evidence:
        result = {
            "profitability_strength": "insufficient_evidence",
            "efficiency_and_scale": "insufficient_evidence",
            "financial_flexibility": "insufficient_evidence",
            "evidence_gaps": ["API evidence missing; collect yfinance/SEC/
↪Finnhub first"],
            "citation_indices": []
        }
        state["analysis_bundle"].append({"agent": "QualityAgent", **result})
        return result

    ev_text, _ = _index_evidence(evidence)

    prompt_text = f"""
You are a CFA-level equity analyst. Evaluate corporate quality strictly from␣
↪the numbered evidence below.
Focus on what can be derived quantitatively or semi-qualitatively:
- Profitability strength (margins, ROE)
- Efficiency & scale (cost control, utilization)
- Financial flexibility (liquidity, leverage, stability)
Do not discuss qualitative factors like 'moat' or 'management'.

Evidence:
{ev_text}

Return STRICT JSON:
{{
  "profitability_strength": string,
  "efficiency_and_scale": string,
  "financial_flexibility": string,
  "evidence_gaps": [string, ...],
  "citation_indices": [int, ...]
}}
"""
    raw = _llm_json(prompt_text)
    try:
        obj = QualityOut(**raw)
    except ValidationError as e:
        if debug: print("[QualityAgent] validation error:", e)
        obj = QualityOut(
            profitability_strength="insufficient_evidence",
            efficiency_and_scale="insufficient_evidence",
            financial_flexibility="insufficient_evidence",
            evidence_gaps=["financial statement depth", "peer data missing"],
            citation_indices=[]
        )

    result = obj.model_dump()
    state["analysis_bundle"].append({"agent": "QualityAgent", **result})
    return result
```

```
def gpt_valuation_agent(tool_input: Any) -> Dict:
    """Assesses valuation stance and justification from quantitative ratios."""
    evidence = _prepare_evidence_from_input(tool_input)

    if not evidence:
        result = {
            "valuation_view": "unknown",
            "justification": "insufficient_evidence",
            "citation_indices": []
        }
        state["analysis_bundle"].append({"agent": "ValuationAgent", **result})
        return result

    ev_text, _ = _index_evidence(evidence)

    prompt_text = f"""
You are a valuation analyst. Use ONLY the numbered evidence.
Estimate valuation stance ('undervalued', 'fairly valued', 'overvalued')
based on P/E, ROE, margins, leverage, and price trend context.

Evidence:
{ev_text}

Return STRICT JSON:
{{
  "valuation_view": "undervalued" | "fairly valued" | "overvalued" | "unknown",
  "justification": string,
  "citation_indices": [int, ...]
}}
"""
    raw = _llm_json(prompt_text)
    try:
        obj = ValuationOut(**raw)
    except ValidationError as e:
        if debug: print("[ValuationAgent] validation error:", e)
        obj = ValuationOut(valuation_view="unknown",␣
 ↪justification="insufficient_evidence", citation_indices=[])

    result = obj.model_dump()
    state["analysis_bundle"].append({"agent": "ValuationAgent", **result})
    return result


def gpt_risk_agent(tool_input: Any) -> Dict:
    """Identifies financial and market risks (no speculative content)."""
    evidence = _prepare_evidence_from_input(tool_input)
    if not evidence:
        result = {
            "risks": ["insufficient_evidence"],
            "mitigants": [],
            "citation_indices": []
        }
        state["analysis_bundle"].append({"agent": "RiskAgent", **result})
        return result

    ev_text, _ = _index_evidence(evidence)

    prompt_text = f"""
You are a risk analyst. Use ONLY the numbered evidence.
Identify observable financial or operational risks (e.g., leverage, liquidity,␣
 ↪valuation risk).
Include mitigants if evidence supports them.

Evidence:
{ev_text}

Return STRICT JSON:
{{
  "risks": [string, ...],
  "mitigants": [string, ...],
  "citation_indices": [int, ...]
}}
"""
    raw = _llm_json(prompt_text)
    try:
        obj = RiskOut(**raw)
    except ValidationError as e:
        if debug: print("[RiskAgent] validation error:", e)
        obj = RiskOut(risks=["insufficient_evidence"], mitigants=[],␣
 ↪citation_indices=[])

    result = obj.model_dump()
    state["analysis_bundle"].append({"agent": "RiskAgent", **result})
    return result


def gpt_thesis_writer(analysis: List[Dict], evidence: Optional[List[Dict]] =␣
 ↪None) -> Dict:
    """Synthesizes a concise, evidence-grounded investment thesis."""
    evidence = evidence or state.get("evidence_pack", []) or []
    # Ensure normalized for consistent indexing
    if evidence and ("section_hint" not in evidence[0] or "text" not in␣
 ↪evidence[0]):
        try:
            evidence = normalize_evidence(evidence)
        except Exception:
            pass

    ev_text, _ = _index_evidence(evidence)

    quality = next((a for a in analysis if a.get("agent") == "QualityAgent"),␣
 ↪{})
    valuation = next((a for a in analysis if a.get("agent") ==␣
 ↪"ValuationAgent"), {})
    risk = next((a for a in analysis if a.get("agent") == "RiskAgent"), {})

    analysis_json = { "quality": quality, "valuation": valuation, "risk": risk }

    # If there is no evidence at all, avoid LLM call and return a minimal safe␣
 ↪thesis
    if not evidence:
        obj = ThesisOut(
            thesis="insufficient_evidence",
            bull_case=["insufficient_evidence", "see analysis"],
            bear_case=["insufficient_evidence", "see analysis"],
            catalysts=["insufficient_evidence"],
            confidence=0.5,
            citation_indices=[]
        )
        thesis_obj = obj.model_dump()
        state["draft_thesis"] = thesis_obj
        return thesis_obj

    prompt_text = f"""
You are an equity research writer. Create a short, balanced thesis strictly␣
 ↪from the following:
(a) the analysis JSON
(b) the numbered evidence
Do not add speculation or qualitative claims about management or strategy.

Evidence:
{ev_text}

Analysis JSON:
{json.dumps(analysis_json, indent=2)}

Return STRICT JSON:
{{
  "thesis": string,
  "bull_case": [string, ...],
  "bear_case": [string, ...],
  "catalysts": [string, ...],
  "confidence": 0.0,
  "citation_indices": [int, ...]
}}
"""
    raw = _llm_json(prompt_text)
    try:
        obj = ThesisOut(**raw)
    except ValidationError as e:
        if debug: print("[ThesisWriter] validation error:", e)
        obj = ThesisOut(
            thesis="insufficient_evidence",
            bull_case=["insufficient_evidence", "see analysis"],
            bear_case=["insufficient_evidence", "see analysis"],
            catalysts=["insufficient_evidence"],
            confidence=0.5,
            citation_indices=[]
        )

    thesis_obj = obj.model_dump()
    state["draft_thesis"] = thesis_obj
    return thesis_obj


def gpt_critic_agent(thesis: Dict, evidence: List[Dict]) -> Dict:
    """Reviews thesis completeness and realism using available evidence and␣
↪returns a one-sentence summary."""
    evidence = evidence or state.get("evidence_pack", []) or []
    if evidence and ("section_hint" not in evidence[0] or "text" not in␣
↪evidence[0]):
        try:
            evidence = normalize_evidence(evidence)
        except Exception:
            pass

    ev_text, _ = _index_evidence(evidence)
    thesis_text = json.dumps(thesis, indent=2)

    # If no evidence, return a graceful summary without LLM
    if not evidence:
        return {
            "valid": False,
            "summary": "No evidence available for review; collect and normalize␣
↪API data before critique.",
            "patch": "Run the data collection step, ensure evidence is␣
↪normalized, then re-synthesize the thesis."
        }

    prompt_text = f"""
You are a senior CFA reviewer. Evaluate if the thesis is:
- Balanced (bull/bear)
- Evidence-supported (cites available items)
- Reasonable in confidence

Use ONLY the numbered evidence.

Evidence:
{ev_text}

Thesis JSON:
{thesis_text}

Return STRICT JSON with keys:
{{
  "valid": boolean,
  "summary": string,
  "patch": string
}}
"""
    raw = _llm_json(prompt_text)
    try:
        obj = CriticOut(**raw)
        return obj.model_dump()
    except ValidationError as e:
        if debug: print("[CriticAgent] validation error:", e)
        # Fallback summary to avoid empty UI
        return {
            "valid": False,
            "summary": "Critic could not parse a proper response; provide␣
↪catalysts and align citations.",
            "patch": "Ensure the thesis includes at least one catalyst, clear␣
↪bull/bear bullets, and uses evidence indices."
        }
```

8 TOOL REGISTRATION

Tool Registration for Real-World API Integration This cell registers all GPT-powered agents and the comprehensive data collection function as LangChain-compatible tools. The tools integrate three real-world financial APIs (yfinance, SEC EDGAR, Finnhub) for modular orchestration and traceable reasoning. A Pydantic schema enforces structured input validation for the CriticAgent.

[9]:
```
# === Input Schemas ===
from typing import Any, Dict, List, Optional
from pydantic import BaseModel
from langchain_core.tools import StructuredTool  # Tool import not needed

class EvidenceInput(BaseModel):
    """Generic input for analysis tools; evidence is optional because agents␣
 ↪can fallback to state['evidence_pack']."""
    evidence: Optional[List[Dict]] = None

class CriticInput(BaseModel):
    thesis: Dict[str, Any]                    # structured thesis object from␣
 ↪gpt_thesis_writer
    evidence: Optional[List[Dict]] = None

class ThesisWriterInput(BaseModel):
    analysis: List[Dict]                      # state["analysis_bundle"] (outputs␣
 ↪from Quality/Valuation/Risk)
    evidence: Optional[List[Dict]] = None

class DataCollectionInput(BaseModel):
    ticker: str


# === Register GPT-Powered Analysis Agents (Structured; schema-aware) ===
quality_tool = StructuredTool.from_function(
    func=gpt_quality_agent,
    name="QualityAgent",
    description="Evaluates profitability strength, efficiency/scale, and␣
 ↪financial flexibility using API evidence.",
    args_schema=EvidenceInput,
    handle_tool_error=True,
)

valuation_tool = StructuredTool.from_function(
    func=gpt_valuation_agent,
    name="ValuationAgent",
    description="Assesses valuation stance (undervalued/fairly valued/
 ↪overvalued) and justification from ratios and trends.",
    args_schema=EvidenceInput,
    handle_tool_error=True,
)

risk_tool = StructuredTool.from_function(
    func=gpt_risk_agent,
    name="RiskAgent",
    description="Identifies observable financial/operational risks and␣
 ↪mitigants from SEC/yfinance/Finnhub evidence.",
    args_schema=EvidenceInput,
    handle_tool_error=True,
)

thesis_writer_tool = StructuredTool.from_function(
    func=gpt_thesis_writer,
    name="ThesisWriterAgent",
    description="Synthesizes a concise, evidence-grounded thesis JSON from␣
 ↪prior agent outputs (analysis) and optional evidence.",
    args_schema=ThesisWriterInput,
    handle_tool_error=True,
)

critic_tool = StructuredTool.from_function(
    func=gpt_critic_agent,
    name="CriticAgent",
    description="Reviews the structured thesis for balance, evidence support,␣
 ↪and reasonable confidence using provided evidence.",
    args_schema=CriticInput,
    handle_tool_error=True,
)

# === Register Comprehensive Data Collection Agent (schema-aware) ===
def collect_data(ticker: str):
    """Wrapper with explicit signature so args_schema maps cleanly."""
    return collect_comprehensive_data(ticker)

data_collection_tool = StructuredTool.from_function(
    func=collect_data,
    name="DataCollectionAgent",
    description="Collects real data from yfinance, SEC EDGAR, and Finnhub APIs;␣
 ↪returns meta, sec, prices, news, evidence_pack.",
    args_schema=DataCollectionInput,
    handle_tool_error=True,
)

# === Expose tools to the graph ===
def get_tools_for_state(state: Dict[str, Any]) -> List:
    has_evidence = bool(state.get("evidence_pack"))
    base = [quality_tool, valuation_tool, risk_tool, thesis_writer_tool,␣
↪critic_tool]
    if not has_evidence:
        base.append(data_collection_tool)
    return base
```

---

9 THESIS WRITING UTILIY

Agent Output Extraction for Modular Thesis Synthesis and Error Resilience This utility function enables targeted retrieval of agent outputs from the shared analysis bundle. By isolating results from specific agents—such as QualityAgent or RiskAgent—it supports modular thesis construction and rubric-aligned synthesis. The defensive fallback ensures runtime stability, allowing the pipeline to proceed even if an expected agent is missing. This function is essential for orchestrating structured reasoning without relying on fragile index-based access.

```
[10]: def extract_agent(agents: List[Dict], agent_type: str) -> Dict:
    """
    Retrieves a specific agent's output from a list of agent results.

    Parameters:
        agents (List[Dict]): A list of agent output dictionaries (e.g., from␣
↪state["analysis_bundle"])
        agent_type (str): The name of the agent to extract (e.g.,␣
↪"QualityAgent", "RiskAgent")

    Returns:
        Dict: The output dictionary for the specified agent, or an empty dict␣
↪if not found.

    Why this matters:
    - Enables modular access to agent outputs without hardcoding index positions
    - Supports rubric-aligned synthesis by isolating structured insights (e.g.,␣
↪for thesis generation)
    - Prevents runtime errors by safely returning an empty dict if the agent is␣
↪missing
    """
    for agent in agents:
        if agent.get("agent") == agent_type:
            return agent
    return {}  # Defensive fallback if agent not found
```

---

10 REACT AGENT INITIALIZATION

ReAct Agent Node Initialization for Tool-Driven Financial Reasoning This cell instantiates the LangGraph agent node using ReAct-style orchestration. It binds GPT-4 to a curated set of financial analysis tools—metadata resolution, quality assessment, valuation, risk analysis, and thesis critique—enabling structured, traceable reasoning across the pipeline. This node serves as the central planner, coordinating tool calls and message flow to produce rubric-aligned outputs.

[11]:
```
# === Research Planning (explicit, saved to memory) ===
# Purpose: Demonstrates "planning" and "learning across runs"
# Creates a visible step-by-step plan for researching a given stock
# and stores it persistently so the agent can refine future analyses

import os, json
from IPython.display import Markdown, display

def plan_research(symbol: str):
    steps = [
        f"Resolve {symbol} metadata (ticker → company profile; map to CIK if␣
 ↪needed)",
        "Pull historical prices & volume (yfinance) and basic stats",
        "Ingest recent news (Finnhub or internal source), with timestamps",
        "Preprocess: dedupe, normalize, strip boilerplate; keep source URLs",
        "Classify: earnings/guidance/litigation/macro/product/other",
        "Extract: entities, key metrics, sentiment, quoted claims",
        "Route: send to specialist analyzers (earnings/news/market) as needed",
        "Draft thesis: drivers, risks, valuation hooks, time horizon",
        "Evaluate: critic pass against rubric (coverage, recency, balance)",
        "Refine: patch thesis using feedback; record next-run notes",
    ]
    return steps

def save_plan(symbol: str, steps, path="memory/plan.json"):
    os.makedirs("memory", exist_ok=True)
    with open(path, "w") as f:
        json.dump({"symbol": symbol, "plan": steps}, f, indent=2)
```

```
def append_note(symbol: str, note: str, path="memory/notes.jsonl"):
    os.makedirs("memory", exist_ok=True)
    with open(path, "a") as f:
        f.write(json.dumps({"symbol": symbol, "note": note}) + "\n")

# Example usage (call this once before running the agent):
target_symbol = os.getenv("SYMBOL", "AAPL")
_steps = plan_research(target_symbol)
save_plan(target_symbol, _steps)
display(Markdown("### Research Plan for **{}**\n".format(target_symbol) + "\n".
 ↪join(f"1. {s}" for s in _steps)))
```

10.0.1 Research Plan for AAPL

1. Resolve AAPL metadata (ticker → company profile; map to CIK if needed)
2. Pull historical prices & volume (yfinance) and basic stats
3. Ingest recent news (Finnhub or internal source), with timestamps
4. Preprocess: dedupe, normalize, strip boilerplate; keep source URLs
5. Classify: earnings/guidance/litigation/macro/product/other
6. Extract: entities, key metrics, sentiment, quoted claims
7. Route: send to specialist analyzers (earnings/news/market) as needed
8. Draft thesis: drivers, risks, valuation hooks, time horizon
9. Evaluate: critic pass against rubric (coverage, recency, balance)
10. Refine: patch thesis using feedback; record next-run notes

[12]:
```
# === LangGraph Agent Node ===
# Create a ReAct-style agent node with conditional tool exposure.
# This prevents the planner from looping endlessly over DataCollectionAgent
# when evidence is already provided.

def create_financial_agent(state: Dict[str, Any]):
    """Dynamically create a ReAct-style agent node with conditional tools."""
    has_evidence = bool(state.get("evidence_pack"))
    active_tools = [
        quality_tool,
        valuation_tool,
        risk_tool,
        thesis_writer_tool,
        critic_tool,
    ]

    # Only include the data collection tool if evidence is missing
    if not has_evidence:
        active_tools.insert(0, data_collection_tool)

    return create_react_agent(
        model=llm,              # gpt-4o-mini model instance (ChatOpenAI)
        tools=active_tools,      # dynamic toolset based on state
        version="v1"             # standard ReAct-style planning
    )

# Initialize  ReAct agent node once (passing the current state)
agent_node = create_financial_agent(state)
```

LangGraph Execution Graph for Modular Agent Orchestration and Rubric Compliance This cell defines the LangGraph orchestration layer that governs agent execution across the pipeline. By initializing a node-based graph and compiling it into a runnable object, it enables structured message flow, modular reasoning, and traceable outputs. The graph wraps the ReAct-style agent node and sets clear entry and exit points, ensuring rubric-aligned coordination and extensibility for future nodes like memory or evaluators.

[13]:
```
# === LangGraph Orchestration (robust stop + auto-finalize) ===
from langgraph.graph import StateGraph, END

MAX_TURNS = 6           # keep small while debugging
NO_PROGRESS_MAX = 2     # stop if the agent makes no progress for 2 cycles

def _progress_signature(s: dict) -> tuple:
    msgs_len = len(s.get("messages", [])) if isinstance(s.get("messages"),␣
 ↪list) else 0
    thesis_ok = bool((s.get("draft_thesis") or {}).get("thesis"))
    critic_valid = bool((s.get("critic_patch") or {}).get("valid"))
    return (msgs_len, thesis_ok, critic_valid)

def is_complete(state: Dict[str, Any]) -> bool:
    thesis = (state or {}).get("draft_thesis") or {}
    critic = (state or {}).get("critic_patch") or {}
    return bool(thesis.get("thesis")) and critic.get("valid") is True

def route_after_agent(state: Dict[str, Any]) -> str:
    state["turns"] = int(state.get("turns", 0)) + 1
    sig = _progress_signature(state)
    prev = state.get("_last_sig")

    # no-progress tracker
    if prev == sig:
        state["_no_progress"] = int(state.get("_no_progress", 0)) + 1
    else:
        state["_no_progress"] = 0
    state["_last_sig"] = sig

    # hard stops
    if is_complete(state):
        return "end"
    # if nearing cap or stuck, jump to finalize to force completion
    if state["turns"] >= MAX_TURNS - 1 or state.get("_no_progress", 0) >=␣
↪NO_PROGRESS_MAX:
        return "finalize"

    return "again"

# --- Auto-finalize node: writes thesis/critic directly if agent didn't ---
def finalize_state(state: Dict[str, Any]) -> Dict[str, Any]:
    # Ensure normalized evidence
    try:
        normalized = normalize_evidence(state.get("evidence_pack", []))
    except Exception:
        normalized = state.get("evidence_pack", []) or []

    # Build minimal analysis bundle if none
    if not state.get("analysis_bundle"):
        try:
            bundle = build_bundle_from_evidence(normalized)
        except Exception:
            bundle = []
        state["analysis_bundle"] = bundle

    # Synthesize thesis if missing
    if not (state.get("draft_thesis") or {}).get("thesis"):
        state["draft_thesis"] = gpt_thesis_writer(state["analysis_bundle"],␣
↪normalized)

    # Critic pass if missing
    if not (state.get("critic_patch") or {}).get("valid"):
        state["critic_patch"] = gpt_critic_agent(state["draft_thesis"],␣
↪normalized)

    return state

# --- Build & compile the graph ---
graph = StateGraph(AgentState)
graph.add_node("AgentNode", agent_node)
graph.add_node("Finalize", finalize_state)

graph.add_conditional_edges(
    "AgentNode",
    route_after_agent,
    {"again": "AgentNode", "finalize": "Finalize", "end": END},
)
```

```
graph.add_edge("Finalize", END)
graph.set_entry_point("AgentNode")

runnable_graph = graph.compile()
print("Graph recompiled: robust stop rules + finalize node.")
```

```
Graph recompiled: robust stop rules + finalize node.
```

11 PIPELINE DEBUGGIN SANDBOX

Full Pipeline Execution with Real Financial Data This cell runs the complete agentic pipeline using real data from yfinance, SEC EDGAR, and Finnhub. It collects comprehensive evidence, normalizes it, routes it through Quality, Valuation, and Risk agents, synthesizes a thesis, and applies critique evaluation. All outputs are stored in memory for audit inspection.

[14]:
```
# === Execute Pipeline with Real Financial Data (direct function calls + critic␣
 ↪summary) ===
from pprint import pprint
from typing import Dict

ticker = "AAPL"

# Ensure `debug` exists to avoid NameError if not defined upstream
debug = bool(globals().get("debug", False))

print(f"\n{'='*60}")
print(f"COLLECTING DATA FOR {ticker}")
print(f"{'='*60}\n")

# Collect from real APIs only
result = collect_comprehensive_data(ticker)

# Persist into state
state["meta"] = result.get("meta", {})
state["evidence_pack"] = result.get("evidence_pack", [])
state["ticker"] = ticker  # handy for downstream nodes/logging

print(f"\n{'='*60}")
print(f"COLLECTED {len(state['evidence_pack'])} EVIDENCE ITEMS")
print(f"{'='*60}\n")
```

```python
print("Company Metadata:")
pprint(state["meta"])

# Normalize evidence and persist
normalized = normalize_evidence(state["evidence_pack"])
state["normalized_evidence"] = normalized

print(f"\n{'='*60}")
print(f"NORMALIZED {len(normalized)} EVIDENCE ITEMS")
print(f"{'='*60}\n")

print("Sample Evidence (first 5):")
for i, e in enumerate(normalized[:5], 1):
    print(f"{i}. [{e.get('source')}] {e.get('section_hint')}: {e.get('text')}")

# Run agents if sufficient evidence
if not normalized or len(normalized) < 3:
    print("\n[ERROR] Insufficient evidence. Check API keys.")
else:
    print(f"\n{'='*60}")
    print(f"RUNNING ANALYSIS AGENTS")
    print(f"{'='*60}\n")

    # Reset analysis bundle for this run
    state["analysis_bundle"] = []

    # Helper to tag agent name (downstream extractors rely on 'agent' key)
    def tag(agent_name: str, out: Dict) -> Dict:
        out = out or {}
        if isinstance(out, dict) and "agent" not in out:
            out["agent"] = agent_name
        return out if isinstance(out, dict) else {"agent": agent_name}

    print("[1/3] QualityAgent...")
    qa_out = gpt_quality_agent(normalized)   # positional to avoid kwarg issues
    state["analysis_bundle"].append(tag("QualityAgent", qa_out))

    print("\n[2/3] ValuationAgent...")
    va_out = gpt_valuation_agent(normalized)
    state["analysis_bundle"].append(tag("ValuationAgent", va_out))

    print("\n[3/3] RiskAgent...")
    ra_out = gpt_risk_agent(normalized)
    state["analysis_bundle"].append(tag("RiskAgent", ra_out))

    # Optional debug dumps
    if debug:
        print("\n[QualityAgent output]")
        pprint(qa_out)
        print("\n[ValuationAgent output]")
        pprint(va_out)
        print("\n[RiskAgent output]")
        pprint(ra_out)

    print(f"\n{'='*60}")
    print(f"SYNTHESIZING THESIS")
    print(f"{'='*60}\n")

    # Thesis expects analysis bundle + evidence (positional for safety)
    state["draft_thesis"] = gpt_thesis_writer(state["analysis_bundle"],␣
↪normalized)

    if debug:
        print("\n[Thesis Writer output]")
        pprint(state["draft_thesis"])

    print(f"\n{'='*60}")
    print(f"EVALUATING WITH CRITIC")
    print(f"{'='*60}\n")

    # Critic expects structured thesis + evidence (positional for safety)
    state["critic_patch"] = gpt_critic_agent(state["draft_thesis"], normalized)

    # Human-readable critic line
    critic_summary = (state["critic_patch"] or {}).get("summary")
    critic_valid = (state["critic_patch"] or {}).get("valid")
    critic_patch = (state["critic_patch"] or {}).get("patch")

    print("Critic verdict:", "VALID" if critic_valid else "NEEDS WORK")
    print("Critic says:", critic_summary or "(no summary returned)")
    if critic_patch:
        print("\nSuggested patch:\n", critic_patch)

    if debug:
        print("\n[Critic output]")
        pprint(state["critic_patch"])

    print("\n[COMPLETE] Pipeline finished.")
```

```
============================================================
COLLECTING DATA FOR AAPL
============================================================

[INFO] Collecting data for AAPL…
```

```
[INFO] Collected 14 evidence items from 3 sources

============================================================
COLLECTED 14 EVIDENCE ITEMS
============================================================

Company Metadata:
{'cik': '0000320193',
 'company_name': 'Apple Inc.',
 'exchange': 'NMS',
 'industry': 'Consumer Electronics',
 'marketCap': 3672254447616,
 'price': 247.45,
 'sector': 'Technology',
 'ticker': 'AAPL'}

============================================================
NORMALIZED 14 EVIDENCE ITEMS
============================================================

Sample Evidence (first 5):
1. [yfinance] Valuation: P/E ratio is 37.61
2. [yfinance] Quality: Return on equity is 1.4981
3. [yfinance] Risk: Beta is 1.094
4. [SEC EDGAR] Quality: Gross margin is 46.2%
5. [SEC EDGAR] Quality: Operating margin is 31.5%

============================================================
RUNNING ANALYSIS AGENTS
============================================================

[1/3] QualityAgent…
[LLM raw t=1] ```json
{
  "profitability_strength": "The company has a return on equity (ROE) of 1.4981,
which is relatively low, indicating limited profitability relative to
shareholder equity. However, the gross margin of 46.2% and operating margin of
31.5% suggest strong profitability at the operational level, indicating that the
company retains a significant portion of revenue as profit after covering its c

[2/3] ValuationAgent…
[LLM raw t=1] {
  "valuation_view": "overvalued",
  "justification": "The P/E ratio of 37.61 is significantly high compared to
historical averages, indicating that the stock may be overvalued. Additionally,
the return on equity (ROE) of 1.4981 is relatively low, which does not justify
such a high valuation. The high debt-to-equity ratio of 1.87 also indicates
increased financial risk, further supporting the ove
```

```
[3/3] RiskAgent…
[LLM raw t=1] {
  "risks": [
    "High valuation risk indicated by a P/E ratio of 37.61",
    "Liquidity risk due to a current ratio of 0.87",
    "Leverage risk indicated by a debt-to-equity ratio of 1.87",
    "Operational risk indicated by a beta of 1.094 and annualized volatility of
25.9%"
  ],
  "mitigants": [
    "Strong operational quality indicated by a gross margin of 46.2% and
operating margin of 31.5

[QualityAgent output]
{'agent': 'QualityAgent',
 'citation_indices': [2, 4, 5, 6, 7],
 'efficiency_and_scale': 'The evidence does not provide specific metrics '
                         'related to cost control or utilization rates. '
                         'However, the high gross and operating margins imply '
                         'effective cost management and operational '
                         'efficiency, allowing the company to maintain '
                         'substantial profitability despite potentially high '
                         'operational costs.',
 'evidence_gaps': ['Details on net income or total equity to better assess ROE '
                   'context',
                   'Information on cash flow metrics to evaluate liquidity '
                   'beyond the current ratio',
                   'Data on operational efficiency metrics such as inventory '
                   'turnover or asset utilization'],
 'financial_flexibility': 'The current ratio of 0.87 indicates potential '
                          'liquidity issues, as it is below the ideal '
                          'threshold of 1. This suggests that the company may '
                          'struggle to cover its short-term liabilities with '
                          'its short-term assets. The debt-to-equity ratio of '
                          '1.87 indicates a high level of leverage, which may '
                          'increase financial risk and reduce financial '
                          'flexibility. Overall, the company appears to have '
                          'limited financial flexibility due to its liquidity '
                          'position and high leverage.',
 'profitability_strength': 'The company has a return on equity (ROE) of '
                           '1.4981, which is relatively low, indicating '
                           'limited profitability relative to shareholder '
                           'equity. However, the gross margin of 46.2% and '
                           'operating margin of 31.5% suggest strong '
                           'profitability at the operational level, indicating '
                           'that the company retains a significant portion of '
                           'revenue as profit after covering its costs.'}
```

```
[ValuationAgent output]
{'agent': 'ValuationAgent',
 'citation_indices': [1, 2, 7],
 'justification': 'The P/E ratio of 37.61 is significantly high compared to '
                  'historical averages, indicating that the stock may be '
                  'overvalued. Additionally, the return on equity (ROE) of '
                  '1.4981 is relatively low, which does not justify such a '
                  'high valuation. The high debt-to-equity ratio of 1.87 also '
                  'indicates increased financial risk, further supporting the '
                  'overvaluation stance.',
 'valuation_view': 'overvalued'}

[RiskAgent output]
{'agent': 'RiskAgent',
 'citation_indices': [1, 6, 7, 3, 9, 4, 5, 8],
 'mitigants': ['Strong operational quality indicated by a gross margin of '
               '46.2% and operating margin of 31.5%',
               'Positive price momentum with a 60-day price change of 15.9%'],
 'risks': ['High valuation risk indicated by a P/E ratio of 37.61',
           'Liquidity risk due to a current ratio of 0.87',
           'Leverage risk indicated by a debt-to-equity ratio of 1.87',
           'Operational risk indicated by a beta of 1.094 and annualized '
           'volatility of 25.9%']}

============================================================
SYNTHESIZING THESIS
============================================================

[LLM raw t=1] {
  "thesis": "The company exhibits strong operational profitability with gross
and operating margins of 46.2% and 31.5%, respectively, but faces significant
valuation and financial risks, including a high P/E ratio of 37.61 and a debt-
to-equity ratio of 1.87, which may indicate overvaluation and increased
financial risk.",
  "bull_case": [
    "Strong operational quality with high gross and opera

[Thesis Writer output]
{'bear_case': ['High P/E ratio of 37.61 suggests potential overvaluation.',
               'Current ratio of 0.87 indicates liquidity issues.',
               'Debt-to-equity ratio of 1.87 reflects high leverage and '
               'financial risk.',
               'Beta of 1.094 and annualized volatility of 25.9% indicate '
               'operational risk.'],
 'bull_case': ['Strong operational quality with high gross and operating '
               'margins suggests effective cost management.',
               'Recent positive price momentum with a 60-day price change of '
```

```
              '15.9% indicates market interest.'],
 'catalysts': ['insufficient_evidence'],
 'citation_indices': [1, 2, 3, 4, 5, 6, 7, 8, 9],
 'confidence': 0.0,
 'thesis': 'The company exhibits strong operational profitability with gross '
           'and operating margins of 46.2% and 31.5%, respectively, but faces '
           'significant valuation and financial risks, including a high P/E '
           'ratio of 37.61 and a debt-to-equity ratio of 1.87, which may '
           'indicate overvaluation and increased financial risk.'}

============================================================
EVALUATING WITH CRITIC
============================================================

[LLM raw t=1] {
  "valid": true,
  "summary": "The thesis presents a balanced view by acknowledging both the
strong operational profitability of the company (bull case) and the significant
valuation and financial risks it faces (bear case). It is supported by relevant
evidence from the provided data, including margins, P/E ratio, and debt levels.
The confidence level is low, indicating caution in the assessment
Critic verdict: VALID
Critic says: The thesis presents a balanced view by acknowledging both the
strong operational profitability of the company (bull case) and the significant
valuation and financial risks it faces (bear case). It is supported by relevant
evidence from the provided data, including margins, P/E ratio, and debt levels.
The confidence level is low, indicating caution in the assessment, which is
reasonable given the risks highlighted.

Suggested patch:
 Increase confidence level to reflect a more nuanced understanding of the risks
and opportunities, perhaps to a value above 0.0, while maintaining the balance
between the bull and bear cases.

[Critic output]
{'patch': 'Increase confidence level to reflect a more nuanced understanding '
          'of the risks and opportunities, perhaps to a value above 0.0, while '
          'maintaining the balance between the bull and bear cases.',
 'summary': 'The thesis presents a balanced view by acknowledging both the '
            'strong operational profitability of the company (bull case) and '
            'the significant valuation and financial risks it faces (bear '
            'case). It is supported by relevant evidence from the provided '
            'data, including margins, P/E ratio, and debt levels. The '
            'confidence level is low, indicating caution in the assessment, '
            'which is reasonable given the risks highlighted.',
 'valid': True}

[COMPLETE] Pipeline finished.
```

12 PIPELINE RUN 1

LangGraph Pipeline Invocation for Multi-Agent Evaluation and Persistent State Update This cell executes the compiled LangGraph pipeline using a structured input that includes company metadata and preprocessed financial evidence. It activates multi-agent reasoning—triggering valuation, quality, risk, and critique tools—and updates the shared state with all outputs. By serializing messages and persisting the state to disk, it ensures reproducibility, rubric compliance, and traceability across notebook sessions. This marks the first full run of LangGraph-driven orchestration.

[15]:
```
# === Synthesize from existing evidence and save state safely (no agent␣
↪re-runs) ===
from pprint import pprint
from typing import Dict, Any, List
from datetime import datetime, date, time
import json
import math

# numpy / decimal shims if present
try:
    import numpy as np
except Exception:
    np = None

try:
    from decimal import Decimal
except Exception:
    Decimal = None

debug = bool(globals().get("debug", False))

# ---------- Helpers: evidence/bundle ----------
def safe_normalized(state: Dict[str, Any]) -> List[Dict[str, Any]]:
    """Prefer already-normalized evidence; otherwise normalize once."""
    if state.get("normalized_evidence"):
        return state["normalized_evidence"]
    ev = state.get("evidence_pack", [])
    norm = normalize_evidence(ev)
    state["normalized_evidence"] = norm
    return norm
```

```
def build_bundle_from_evidence(evidence: List[Dict[str, Any]]) ->␣
↪List[Dict[str, Any]]:
    """
    Build a minimal analysis bundle straight from evidence so thesis/critic can␣
↪run.
    Buckets by 'section_hint' keywords: Quality / Valuation / Risk.
    """
    buckets = {"QualityAgent": [], "ValuationAgent": [], "RiskAgent": []}
    idx_map = {"QualityAgent": [], "ValuationAgent": [], "RiskAgent": []}

    for i, e in enumerate(evidence):
        hint = (e.get("section_hint") or "").lower()
        line = e.get("text") or ""
        if "quality" in hint:
            buckets["QualityAgent"].append(line)
            idx_map["QualityAgent"].append(i)
        elif "valuation" in hint:
            buckets["ValuationAgent"].append(line)
            idx_map["ValuationAgent"].append(i)
        elif "risk" in hint:
            buckets["RiskAgent"].append(line)
            idx_map["RiskAgent"].append(i)

    bundle = []
    for agent in ("QualityAgent", "ValuationAgent", "RiskAgent"):
        if buckets[agent]:
            bundle.append({
                "agent": agent,
                "observations": buckets[agent],
                "citation_indices": idx_map[agent],
            })
    return bundle

# ---------- Helpers: message & state serialization ----------
def serialize_messages(messages: List[Any]) -> List[Dict[str, Any]]:
    """
    LangChain BaseMessage or plain dicts -> safe dicts {role, content}.
    Anything else becomes stringified to avoid save failures.
    """
    safe = []
    for m in messages or []:
        if hasattr(m, "role") and hasattr(m, "content"):
            safe.append({"role": m.role, "content": m.content})
        elif isinstance(m, dict) and "role" in m and "content" in m:
            safe.append({"role": m["role"], "content": m["content"]})
        else:
            safe.append({"role": "system", "content": str(m)})
    return safe

def to_jsonable(obj: Any) -> Any:
    """
    Recursively convert objects to JSON-serializable forms.
    Handles datetime/date/time, Decimal, numpy scalars/arrays, sets/tuples,␣
 ↪bytes, pydantic models, etc.
    """
    # Primitives
    if obj is None or isinstance(obj, (bool, int, float, str)):
        # normalize NaN/inf to None for strict JSON
        if isinstance(obj, float) and (math.isnan(obj) or math.isinf(obj)):
            return None
        return obj

    # datetime-like
    if isinstance(obj, (datetime, date, time)):
        try:
            return obj.isoformat()
        except Exception:
            return str(obj)

    # Decimal
    if Decimal is not None and isinstance(obj, Decimal):
        # convert to float; if not finite, use None
        f = float(obj)
        return None if (math.isnan(f) or math.isinf(f)) else f

    # numpy scalars / arrays
    if np is not None:
        if isinstance(obj, (np.integer,)):
            return int(obj)
        if isinstance(obj, (np.floating,)):
            f = float(obj)
            return None if (math.isnan(f) or math.isinf(f)) else f
        if isinstance(obj, (np.ndarray,)):
            return [to_jsonable(x) for x in obj.tolist()]

    # bytes
    if isinstance(obj, (bytes, bytearray)):
        try:
            return obj.decode("utf-8", errors="replace")
        except Exception:
            return str(obj)

    # dict
    if isinstance(obj, dict):
        return {str(to_jsonable(k)): to_jsonable(v) for k, v in obj.items()}

    # list / tuple / set
    if isinstance(obj, (list, tuple, set)):
        return [to_jsonable(x) for x in obj]

    # pydantic BaseModel
    try:
        from pydantic import BaseModel as _PydBase
        if isinstance(obj, _PydBase):
            return to_jsonable(obj.model_dump())
    except Exception:
        pass

    # fallback
    return str(obj)

def save_memory_safe(memory: Dict[str, Any], path: str = None):
    """
    Safe saver that JSON-serializes complex objects.
    If you already have a save_memory(...), this can replace it (same name),
    or you can call this instead.
    """
    # Prefer existing MEMORY_PATH if your earlier code defines it
    out_path = path or globals().get("MEMORY_PATH") or "memory.json"

    # Ensure messages are serializable
    if "messages" in memory:
        memory["messages"] = serialize_messages(memory["messages"])

    serializable = to_jsonable(memory)
    with open(out_path, "w") as f:
        json.dump(serializable, f, indent=2)
    print(f"[SAVE] State persisted to {out_path}")

# ======================== MAIN FLOW ========================

# 1) Gather normalized evidence (no new collection)
normalized = safe_normalized(state)

print(f"\n{'='*60}")
print(f"USING {len(normalized)} NORMALIZED EVIDENCE ITEMS (no agent re-run)")
print(f"{'='*60}\n")

# 2) Use existing analysis bundle if present; else derive a minimal one from␣
  ↪evidence
```

```python
if state.get("analysis_bundle"):
    analysis_bundle = state["analysis_bundle"]
    print("[INFO] Using existing analysis_bundle from state.")
else:
    analysis_bundle = build_bundle_from_evidence(normalized)
    state["analysis_bundle"] = analysis_bundle
    print("[INFO] Built minimal analysis_bundle directly from evidence.")

if debug:
    print("\n[Analysis bundle preview]")
    pprint(analysis_bundle[:2])

# 3) Synthesize thesis (positional calls to match your function signatures)
state["draft_thesis"] = gpt_thesis_writer(analysis_bundle, normalized)

if debug:
    print("\n[Thesis Writer output]")
    pprint(state["draft_thesis"])

# 4) Critic pass (positional)
state["critic_patch"] = gpt_critic_agent(state["draft_thesis"], normalized)

# Optional: apply simple critic JSON patch if it only updates 'confidence'
try:
    patch = state.get("critic_patch", {}).get("patch")
    if isinstance(patch, str):
        # if the critic returned a JSON string like {"confidence": 0.7}
        patch_obj = json.loads(patch)
        if isinstance(patch_obj, dict):
            state["draft_thesis"].update(patch_obj)
            print("[INFO] Applied critic patch to draft_thesis.")
except Exception:
    # non-fatal if patch parsing fails
    pass

# 5) Human-readable critic line
critic_summary = (state["critic_patch"] or {}).get("summary")
critic_valid = (state["critic_patch"] or {}).get("valid")
critic_patch = (state["critic_patch"] or {}).get("patch")

print("\n=== CRITIC RESULT ===")
print("Critic verdict:", "VALID" if critic_valid else "NEEDS WORK")
print("Critic says:", critic_summary or "(no summary returned)")
if critic_patch:
    print("\nSuggested patch:\n", critic_patch)

# 6) Persist safely (datetime and friends converted)
```

```
save_memory_safe(state)

print("\n[COMPLETE] Synthesis from existing evidence finished.")
```

```
============================================================
USING 14 NORMALIZED EVIDENCE ITEMS (no agent re-run)
============================================================

[INFO] Using existing analysis_bundle from state.

[Analysis bundle preview]
[{'agent': 'QualityAgent',
  'citation_indices': [2, 4, 5, 6, 7],
  'efficiency_and_scale': 'The evidence does not provide specific metrics '
                          'related to cost control or utilization rates. '
                          'However, the high gross and operating margins imply '
                          'effective cost management and operational '
                          'efficiency, allowing the company to maintain '
                          'substantial profitability despite potentially high '
                          'operational costs.',
  'evidence_gaps': ['Details on net income or total equity to better assess '
                    'ROE context',
                    'Information on cash flow metrics to evaluate liquidity '
                    'beyond the current ratio',
                    'Data on operational efficiency metrics such as inventory '
                    'turnover or asset utilization'],
  'financial_flexibility': 'The current ratio of 0.87 indicates potential '
                           'liquidity issues, as it is below the ideal '
                           'threshold of 1. This suggests that the company may '
                           'struggle to cover its short-term liabilities with '
                           'its short-term assets. The debt-to-equity ratio of '
                           '1.87 indicates a high level of leverage, which may '
                           'increase financial risk and reduce financial '
                           'flexibility. Overall, the company appears to have '
                           'limited financial flexibility due to its liquidity '
                           'position and high leverage.',
  'profitability_strength': 'The company has a return on equity (ROE) of '
                            '1.4981, which is relatively low, indicating '
                            'limited profitability relative to shareholder '
                            'equity. However, the gross margin of 46.2% and '
                            'operating margin of 31.5% suggest strong '
                            'profitability at the operational level, '
                            'indicating that the company retains a significant '
                            'portion of revenue as profit after covering its '
                            'costs.'},
 {'agent': 'QualityAgent',
  'citation_indices': [2, 4, 5, 6, 7],
  'efficiency_and_scale': 'The evidence does not provide specific metrics '
                          'related to cost control or utilization rates. '
                          'However, the high gross and operating margins imply '
                          'effective cost management and operational '
                          'efficiency, allowing the company to maintain '
                          'substantial profitability despite potentially high '
                          'operational costs.',
  'evidence_gaps': ['Details on net income or total equity to better assess '
                    'ROE context',
                    'Information on cash flow metrics to evaluate liquidity '
                    'beyond the current ratio',
                    'Data on operational efficiency metrics such as inventory '
                    'turnover or asset utilization'],
  'financial_flexibility': 'The current ratio of 0.87 indicates potential '
                           'liquidity issues, as it is below the ideal '
                           'threshold of 1. This suggests that the company may '
                           'struggle to cover its short-term liabilities with '
                           'its short-term assets. The debt-to-equity ratio of '
                           '1.87 indicates a high level of leverage, which may '
                           'increase financial risk and reduce financial '
                           'flexibility. Overall, the company appears to have '
                           'limited financial flexibility due to its liquidity '
                           'position and high leverage.',
  'profitability_strength': 'The company has a return on equity (ROE) of '
                            '1.4981, which is relatively low, indicating '
                            'limited profitability relative to shareholder '
                            'equity. However, the gross margin of 46.2% and '
                            'operating margin of 31.5% suggest strong '
                            'profitability at the operational level, '
                            'indicating that the company retains a significant '
                            'portion of revenue as profit after covering its '
                            'costs.'}]
[LLM raw t=1] ```json
{
  "thesis": "The stock appears overvalued with a P/E ratio of 37.61, which is
not justified by a relatively low return on equity of 1.4981. Additionally, the
company faces liquidity and leverage risks, as indicated by a current ratio of
0.87 and a debt-to-equity ratio of 1.87. However, strong operational margins
suggest effective cost management, and positive price momentum could provide

[Thesis Writer output]
{'bear_case': ['High valuation risk indicated by a P/E ratio of 37.61',
               'Liquidity risk due to a current ratio of 0.87',
               'Leverage risk indicated by a debt-to-equity ratio of 1.87',
               'Operational risk indicated by a beta of 1.094 and annualized '
               'volatility of 25.9%'],
 'bull_case': ['Strong operational quality indicated by a gross margin of '
               '46.2% and operating margin of 31.5%',
```

```
              'Positive price momentum with a 60-day price change of 15.9%'],
 'catalysts': ['insufficient_evidence'],
 'citation_indices': [1, 2, 4, 5, 6, 7, 8, 3, 9],
 'confidence': 0.0,
 'thesis': 'The stock appears overvalued with a P/E ratio of 37.61, which is '
           'not justified by a relatively low return on equity of 1.4981. '
           'Additionally, the company faces liquidity and leverage risks, as '
           'indicated by a current ratio of 0.87 and a debt-to-equity ratio of '
           '1.87. However, strong operational margins suggest effective cost '
           'management, and positive price momentum could provide some '
           'support.'}
[LLM raw t=1] ```json
{
  "valid": true,
  "summary": "The thesis is balanced, presenting both bull and bear cases
supported by relevant evidence. It cites specific metrics for valuation,
quality, and risk, providing a reasonable basis for the conclusions drawn. The
confidence level is low, indicating uncertainty, but the arguments are logically
structured.",
  "patch": "Consider increasing the confidence level

=== CRITIC RESULT ===
Critic verdict: VALID
Critic says: The thesis is balanced, presenting both bull and bear cases
supported by relevant evidence. It cites specific metrics for valuation,
quality, and risk, providing a reasonable basis for the conclusions drawn. The
confidence level is low, indicating uncertainty, but the arguments are logically
structured.

Suggested patch:
 Consider increasing the confidence level by providing more recent or additional
evidence to support the thesis, particularly regarding catalysts.
[SAVE] State persisted to agent_memory.json

[COMPLETE] Synthesis from existing evidence finished.
```

13 MEMORY INSPECTION

Memory Inspection for Verifying Saved Agent Outputs and Rubric Traceability This cell loads and prints the contents of the persistent memory file, allowing users to verify that key outputs—such as thesis drafts, trace artifacts, and metadata—have been successfully saved. It supports reproducibility, rubric validation, and audit trail inspection by exposing the serialized state after pipeline execution. This step confirms that cross-run memory retention is functioning as intended.

```
[16]: # === Inspect Persistent Memory ===
      # This cell loads and prints the current memory file in a readable format.
      # It confirms that agent outputs (e.g., thesis, trace, metadata) have been␣
       ↪successfully saved.
      # Useful for debugging, rubric validation, and audit trail inspection.

      print(json.dumps(load_memory(), indent=2))
```

```
{
  "meta": {
    "ticker": "AAPL",
    "company_name": "Apple Inc.",
    "sector": "Technology",
    "industry": "Consumer Electronics",
    "marketCap": 3672254447616,
    "price": 247.45,
    "exchange": "NMS",
    "cik": "0000320193"
  },
  "evidence_pack": [
    {
      "source": "yfinance",
      "section_hint": "Valuation",
      "text": "P/E ratio is 37.61",
      "score": 0.9
    },
    {
      "source": "yfinance",
      "section_hint": "Quality",
      "text": "Return on equity is 1.4981",
      "score": 0.85
    },
    {
      "source": "yfinance",
      "section_hint": "Risk",
      "text": "Beta is 1.094",
      "score": 0.8
    },
    {
      "source": "SEC EDGAR",
      "section_hint": "Quality",
      "text": "Gross margin is 46.2%",
      "score": 0.9
    },
    {
      "source": "SEC EDGAR",
      "section_hint": "Quality",
      "text": "Operating margin is 31.5%",
      "score": 0.9
    },
    {
      "source": "SEC EDGAR",
      "section_hint": "Risk",
      "text": "Current ratio is 0.87",
      "score": 0.85
    },
    {
      "source": "SEC EDGAR",
      "section_hint": "Risk",
      "text": "Debt-to-equity ratio is 1.87",
      "score": 0.85
    },
    {
      "source": "yfinance",
      "section_hint": "Valuation",
      "text": "60-day price change is 15.9%",
      "score": 0.8
    },
    {
      "source": "yfinance",
      "section_hint": "Risk",
      "text": "Annualized volatility is 25.9%",
      "score": 0.8
    },
    {
      "source": "Finnhub",
      "section_hint": "News",
      "text": "Recent news: China\u2019s Wentao blames US actions for trade
tensions",
      "score": 0.75,
      "date": "2025-10-16 15:40:30",
      "url": "https://finnhub.io/api/news?id=a14b719fb86e1cb63132beab35d5a1529d4
c53f6a3987f2853e0c63929ead2e1"
    },
    {
      "source": "Finnhub",
      "section_hint": "News",
      "text": "Recent news: Apple is reportedly making robots. Here\u2019s what
you need to know",
      "score": 0.75,
      "date": "2025-10-16 15:33:26",
      "url": "https://finnhub.io/api/news?id=b187cfd25dd2c5c717063df9350c487e4b9
```

```
576c32c4b0f15cf50f78aac6f2dbe"
    },
    {
      "source": "Finnhub",
      "section_hint": "News",
      "text": "Recent news: Apple loses another AI exec to Meta",
      "score": 0.75,
      "date": "2025-10-16 15:21:03",
      "url": "https://finnhub.io/api/news?id=ae711baa95b000d70d183924c1a76008f1e
6baea907b34f7dc06b0293de8a944"
    },
    {
      "source": "Finnhub",
      "section_hint": "News",
      "text": "Recent news: Nearly 40% jump in net profit: Chipmaker TSMC
capitalises on AI boom",
      "score": 0.75,
      "date": "2025-10-16 14:35:03",
      "url": "https://finnhub.io/api/news?id=9dc799e9ff88213ce7d0929179ce15791e2
2277a8977f57e1c969da2d054e9e3"
    },
    {
      "source": "Finnhub",
      "section_hint": "News",
      "text": "Recent news: Is Apple's Growth Story Over? Here's What History
Says.",
      "score": 0.75,
      "date": "2025-10-16 14:00:00",
      "url": "https://finnhub.io/api/news?id=2e9491d041e16147202e3e6290cffa8e25b
3a2431b4cbf5b65f2b2e8022bce4c"
    }
  ],
  "analysis_bundle": [
    {
      "agent": "QualityAgent",
      "profitability_strength": "The company has a return on equity (ROE) of
1.4981, which is relatively low, indicating limited profitability relative to
shareholder equity. However, the gross margin of 46.2% and operating margin of
31.5% suggest strong profitability at the operational level, indicating that the
company retains a significant portion of revenue as profit after covering its
costs.",
      "efficiency_and_scale": "The evidence does not provide specific metrics
related to cost control or utilization rates. However, the high gross and
operating margins imply effective cost management and operational efficiency,
allowing the company to maintain substantial profitability despite potentially
high operational costs.",
      "financial_flexibility": "The current ratio of 0.87 indicates potential
liquidity issues, as it is below the ideal threshold of 1. This suggests that
```

```
the company may struggle to cover its short-term liabilities with its short-term
assets. The debt-to-equity ratio of 1.87 indicates a high level of leverage,
which may increase financial risk and reduce financial flexibility. Overall, the
company appears to have limited financial flexibility due to its liquidity
position and high leverage.",
      "evidence_gaps": [
        "Details on net income or total equity to better assess ROE context",
        "Information on cash flow metrics to evaluate liquidity beyond the
current ratio",
        "Data on operational efficiency metrics such as inventory turnover or
asset utilization"
      ],
      "citation_indices": [
        2,
        4,
        5,
        6,
        7
      ]
    },
    {
      "profitability_strength": "The company has a return on equity (ROE) of
1.4981, which is relatively low, indicating limited profitability relative to
shareholder equity. However, the gross margin of 46.2% and operating margin of
31.5% suggest strong profitability at the operational level, indicating that the
company retains a significant portion of revenue as profit after covering its
costs.",
      "efficiency_and_scale": "The evidence does not provide specific metrics
related to cost control or utilization rates. However, the high gross and
operating margins imply effective cost management and operational efficiency,
allowing the company to maintain substantial profitability despite potentially
high operational costs.",
      "financial_flexibility": "The current ratio of 0.87 indicates potential
liquidity issues, as it is below the ideal threshold of 1. This suggests that
the company may struggle to cover its short-term liabilities with its short-term
assets. The debt-to-equity ratio of 1.87 indicates a high level of leverage,
which may increase financial risk and reduce financial flexibility. Overall, the
company appears to have limited financial flexibility due to its liquidity
position and high leverage.",
      "evidence_gaps": [
        "Details on net income or total equity to better assess ROE context",
        "Information on cash flow metrics to evaluate liquidity beyond the
current ratio",
        "Data on operational efficiency metrics such as inventory turnover or
asset utilization"
      ],
      "citation_indices": [
        2,
        4,
        5,
        6,
        7
      ],
      "agent": "QualityAgent"
    },
    {
      "agent": "ValuationAgent",
      "valuation_view": "overvalued",
      "justification": "The P/E ratio of 37.61 is significantly high compared to
historical averages, indicating that the stock may be overvalued. Additionally,
the return on equity (ROE) of 1.4981 is relatively low, which does not justify
such a high valuation. The high debt-to-equity ratio of 1.87 also indicates
increased financial risk, further supporting the overvaluation stance.",
      "citation_indices": [
        1,
        2,
        7
      ]
    },
    {
      "valuation_view": "overvalued",
      "justification": "The P/E ratio of 37.61 is significantly high compared to
historical averages, indicating that the stock may be overvalued. Additionally,
the return on equity (ROE) of 1.4981 is relatively low, which does not justify
such a high valuation. The high debt-to-equity ratio of 1.87 also indicates
increased financial risk, further supporting the overvaluation stance.",
      "citation_indices": [
        1,
        2,
        7
      ],
      "agent": "ValuationAgent"
    },
    {
      "agent": "RiskAgent",
      "risks": [
        "High valuation risk indicated by a P/E ratio of 37.61",
        "Liquidity risk due to a current ratio of 0.87",
        "Leverage risk indicated by a debt-to-equity ratio of 1.87",
        "Operational risk indicated by a beta of 1.094 and annualized volatility
of 25.9%"
      ],
      "mitigants": [
        "Strong operational quality indicated by a gross margin of 46.2% and
operating margin of 31.5%",
        "Positive price momentum with a 60-day price change of 15.9%"
      ],
      "citation_indices": [
        1,
        6,
        7,
        3,
        9,
        4,
        5,
        8
      ]
    },
    {
      "risks": [
        "High valuation risk indicated by a P/E ratio of 37.61",
        "Liquidity risk due to a current ratio of 0.87",
        "Leverage risk indicated by a debt-to-equity ratio of 1.87",
        "Operational risk indicated by a beta of 1.094 and annualized volatility
of 25.9%"
      ],
      "mitigants": [
        "Strong operational quality indicated by a gross margin of 46.2% and
operating margin of 31.5%",
        "Positive price momentum with a 60-day price change of 15.9%"
      ],
      "citation_indices": [
        1,
        6,
        7,
        3,
        9,
        4,
        5,
        8
      ],
      "agent": "RiskAgent"
    }
  ],
  "draft_thesis": {
    "thesis": "The stock appears overvalued with a P/E ratio of 37.61, which is
not justified by a relatively low return on equity of 1.4981. Additionally, the
company faces liquidity and leverage risks, as indicated by a current ratio of
0.87 and a debt-to-equity ratio of 1.87. However, strong operational margins
suggest effective cost management, and positive price momentum could provide
some support.",
    "bull_case": [
      "Strong operational quality indicated by a gross margin of 46.2% and
operating margin of 31.5%",
      "Positive price momentum with a 60-day price change of 15.9%"
    ],
    "bear_case": [
      "High valuation risk indicated by a P/E ratio of 37.61",
      "Liquidity risk due to a current ratio of 0.87",
      "Leverage risk indicated by a debt-to-equity ratio of 1.87",
      "Operational risk indicated by a beta of 1.094 and annualized volatility
of 25.9%"
    ],
    "catalysts": [
      "insufficient_evidence"
    ],
    "confidence": 0.0,
    "citation_indices": [
      1,
      2,
      4,
      5,
      6,
      7,
      8,
      3,
      9
    ]
  },
  "critic_patch": {
    "valid": true,
    "summary": "The thesis is balanced, presenting both bull and bear cases
supported by relevant evidence. It cites specific metrics for valuation,
quality, and risk, providing a reasonable basis for the conclusions drawn. The
confidence level is low, indicating uncertainty, but the arguments are logically
structured.",
    "patch": "Consider increasing the confidence level by providing more recent
or additional evidence to support the thesis, particularly regarding catalysts."
  },
  "messages": [],
  "turns": 0,
  "ticker": "AAPL",
  "normalized_evidence": [
    {
      "text": "P/E ratio is 37.61",
      "score": 0.9,
      "section_hint": "Valuation",
      "source": "yfinance",
      "date": null,
      "date_parsed": null,
      "url": null
    },
{
  "text": "Return on equity is 1.4981",
  "score": 0.85,
  "section_hint": "Quality",
  "source": "yfinance",
  "date": null,
  "date_parsed": null,
  "url": null
},
{
  "text": "Beta is 1.094",
  "score": 0.8,
  "section_hint": "Risk",
  "source": "yfinance",
  "date": null,
  "date_parsed": null,
  "url": null
},
{
  "text": "Gross margin is 46.2%",
  "score": 0.9,
  "section_hint": "Quality",
  "source": "SEC EDGAR",
  "date": null,
  "date_parsed": null,
  "url": null
},
{
  "text": "Operating margin is 31.5%",
  "score": 0.9,
  "section_hint": "Quality",
  "source": "SEC EDGAR",
  "date": null,
  "date_parsed": null,
  "url": null
},
{
  "text": "Current ratio is 0.87",
  "score": 0.85,
  "section_hint": "Risk",
  "source": "SEC EDGAR",
  "date": null,
  "date_parsed": null,
  "url": null
},
{
  "text": "Debt-to-equity ratio is 1.87",
  "score": 0.85,
      "section_hint": "Risk",
      "source": "SEC EDGAR",
      "date": null,
      "date_parsed": null,
      "url": null
    },
    {
      "text": "60-day price change is 15.9%",
      "score": 0.8,
      "section_hint": "Valuation",
      "source": "yfinance",
      "date": null,
      "date_parsed": null,
      "url": null
    },
    {
      "text": "Annualized volatility is 25.9%",
      "score": 0.8,
      "section_hint": "Risk",
      "source": "yfinance",
      "date": null,
      "date_parsed": null,
      "url": null
    },
    {
      "text": "Recent news: China\u2019s Wentao blames US actions for trade
tensions",
      "score": 0.75,
      "section_hint": "News",
      "source": "Finnhub",
      "date": "2025-10-16 15:40:30",
      "date_parsed": "2025-10-16T22:40:30+00:00",
      "url": "https://finnhub.io/api/news?id=a14b719fb86e1cb63132beab35d5a1529d4
c53f6a3987f2853e0c63929ead2e1"
    },
    {
      "text": "Recent news: Apple is reportedly making robots. Here\u2019s what
you need to know",
      "score": 0.75,
      "section_hint": "News",
      "source": "Finnhub",
      "date": "2025-10-16 15:33:26",
      "date_parsed": "2025-10-16T22:33:26+00:00",
      "url": "https://finnhub.io/api/news?id=b187cfd25dd2c5c717063df9350c487e4b9
576c32c4b0f15cf50f78aac6f2dbe"
    },
    {
      "text": "Recent news: Apple loses another AI exec to Meta",
      "score": 0.75,
      "section_hint": "News",
      "source": "Finnhub",
      "date": "2025-10-16 15:21:03",
      "date_parsed": "2025-10-16T22:21:03+00:00",
      "url": "https://finnhub.io/api/news?id=ae711baa95b000d70d183924c1a76008f1e
6baea907b34f7dc06b0293de8a944"
    },
    {
      "text": "Recent news: Nearly 40% jump in net profit: Chipmaker TSMC
capitalises on AI boom",
      "score": 0.75,
      "section_hint": "News",
      "source": "Finnhub",
      "date": "2025-10-16 14:35:03",
      "date_parsed": "2025-10-16T21:35:03+00:00",
      "url": "https://finnhub.io/api/news?id=9dc799e9ff88213ce7d0929179ce15791e2
2277a8977f57e1c969da2d054e9e3"
    },
    {
      "text": "Recent news: Is Apple's Growth Story Over? Here's What History
Says.",
      "score": 0.75,
      "section_hint": "News",
      "source": "Finnhub",
      "date": "2025-10-16 14:00:00",
      "date_parsed": "2025-10-16T21:00:00+00:00",
      "url": "https://finnhub.io/api/news?id=2e9491d041e16147202e3e6290cffa8e25b
3a2431b4cbf5b65f2b2e8022bce4c"
    }
  ]
}
```

14 REPORT GENERATION UTILITIES

Trace Rendering for Rubric-Aligned Documentation and Intermediate Reasoning Visibility This cell defines a markdown formatter for prompt chaining traces, converting structured evidence and reasoning steps into export-ready documentation. It displays raw news metadata, preprocessed signals, classification, extracted insights, and summary—all aligned with rubric dimensions. By rendering the trace from the latest pipeline response, it supports auditability, reproducibility, and reviewer inspection of intermediate agent reasoning.

```
[17]: # === Prompt Chaining Trace Display for Sample Ticker ===
      # Renders a markdown trace from the latest pipeline response.
      # Uses a fresh initial_state with a parameterized prompt to avoid mutating␣
       ↪global `state`.
      # Uses a fresh initial_state and clamps evidence to reduce token usage /␣
       ↪rate-limit risk.

      # === Prompt Chaining Trace Display for Sample Ticker (token-friendly + 429␣
       ↪backoff) ===
      # Fresh initial_state (no mutation of global `state`) + evidence clamp +␣
       ↪exponential backoff on rate limits.

      from typing import Dict
      from IPython.display import Markdown, display
      from langchain_core.messages import HumanMessage
      import os, time

      def format_trace_md(trace: Dict, company_name: str = "Unknown") -> str:
          """
          Converts a prompt chaining trace dictionary into a markdown-formatted␣
       ↪string.
          Accepts keys like: raw_news (list), preprocessed (list), classified (str),
          extracted (list), summary (str).
          """
          md = f"""### Prompt Chaining Trace - {company_name}

      **Raw News Source**: {trace.get('raw_news', [{}])[0].get('source', 'N/A')}
      **Title**: {trace.get('raw_news', [{}])[0].get('title', 'N/A')}

      **Preprocessed Evidence**:
      """
          for item in (trace.get("preprocessed", []) or []):
              md += f"- {item.get('section_hint', 'Unknown')}: {item.get('text', '')}␣
       ↪(score: {item.get('score', 'N/A')})\n"

          md += f"""\n**Classification**: {trace.get('classified', 'N/A')}

      **Extracted Signals**:
      """
          for signal in (trace.get("extracted", []) or []):
              md += f"- {signal}\n"

          md += f"""\n**Summary**: {trace.get('summary', 'N/A')}"""
          return md


      # --- Build a fresh initial_state (don't mutate global `state`) ---
```

```
ticker = os.getenv("SYMBOL", "AAPL")
prompt = f"Concise, evidence-bounded financial analysis for {ticker}. If␣
↪evidence is missing, collect briefly."

initial_state = dict(state)  # shallow copy only
initial_state["messages"] = [HumanMessage(content=prompt)]

# --- Clamp evidence to reduce token usage (Option A) ---
def clamp_evidence(s: dict, *, topk_map=None, hard_cap=12):
    try:
        # Prefer your helper for top-k per section (keeps signal density high).
        if topk_map is None:
            topk_map = {"Valuation": 2, "Quality": 2, "Risk": 2, "News": 3,␣
↪"General": 1}
        prioritized = prepare_evidence_for_agents(
            [s.get("evidence_pack", [])],
            topk_map=topk_map,
            drop_unknown=True
        )
        s["evidence_pack"] = prioritized[:hard_cap]
    except Exception:
        ev = s.get("evidence_pack", []) or []
        s["evidence_pack"] = ev[:hard_cap]

# initial clamp
clamp_evidence(initial_state, hard_cap=12)

# --- Invoke the graph with 429 backoff & progressive clamping ---
# (Avoid importing openai exceptions directly to keep the cell self-contained.)
def invoke_with_backoff(s: dict, max_attempts=4, base_sleep=3):
    sleep = base_sleep
    for attempt in range(1, max_attempts + 1):
        try:
            return runnable_graph.invoke(s)
        except Exception as e:
            msg = str(e)
            # Treat OpenAI 429/rate-limit as retryable
            retryable = ("rate limit" in msg.lower()) or ("429" in msg)
            if not retryable or attempt == max_attempts:
                print(f"[ERROR] Invoke failed (attempt {attempt}/
↪{max_attempts}): {e}")
                raise
            # tighten prompt further between retries
            hard_cap = max(4, 12 - attempt * 3)   # 12 -> 9 -> 6 -> 4
            clamp_evidence(s, hard_cap=hard_cap)
            print(f"[WARN] Rate limited. Retrying in {sleep}s with evidence␣
↪cap={hard_cap} …")
```

```
            time.sleep(sleep)
            sleep = min(sleep * 2, 30)

response = runnable_graph.invoke(initial_state, config={"recursion_limit":␣
↪MAX_TURNS})

print("Response keys:", list(response.keys()))

# --- Render trace (if present) ---
trace_md = format_trace_md(
    trace=response.get("trace", {}) or {},
    company_name=response.get("meta", {}).get("company_name", "Unknown")
)
display(Markdown(trace_md if trace_md.strip() else "*(No trace available from␣
↪this run.)*"))
```

```
[INFO] Collecting data for AAPL…
[INFO] Collected 14 evidence items from 3 sources
Response keys: ['messages', 'turns', 'meta', 'evidence_pack', 'analysis_bundle',
'draft_thesis', 'critic_patch']
```

14.0.1 Prompt Chaining Trace - Apple Inc.

Raw News Source: N/A **Title**: N/A

Preprocessed Evidence:

Classification: N/A

Extracted Signals:

Summary: N/A

Final Report Builder for Rubric-Aligned Thesis Documentation and Export This cell defines the `build_report` function, which compiles all pipeline outputs—thesis, agent assessments, evidence, and trace—into a markdown-formatted investment report. It supports rubric scoring, reproducibility, and auditability by organizing insights into clearly labeled sections. This function is typically invoked at the end of the pipeline.

[18]:
```
def build_report(
    thesis: Dict,
    evidence: List[Dict],
    trace: Dict,
    analysis_bundle: List[Dict],
    company_name: str = "Unknown"
) -> str:
    """
    Constructs a markdown-formatted investment report from thesis, evidence,␣
↪trace, and agent outputs.
```

```
    This function supports rubric-aligned documentation and reproducible audit␣
↪trails.
    It is typically called at the end of the pipeline to generate a final␣
↪export-ready report.
    """

    report = f"# Investment Thesis Report - {company_name}\n"

    # === Thesis Summary ===
    # Presents the core thesis components: bull/bear case, confidence level,␣
↪and catalysts.
    # These are extracted from the thesis dictionary returned by␣
↪ThesisWriterAgent.
    report += "\n## Thesis Summary\n"
    report += f"**Bull Case**: {thesis.get('bull_case', 'N/A')}\n"
    report += f"**Bear Case**: {thesis.get('bear_case', 'N/A')}\n"
    report += f"**Confidence**: {thesis.get('confidence', 'N/A')}\n"
    report += f"**Catalysts**: {', '.join(thesis.get('catalysts', []))}\n"

    # === Agent Contributions ===
    # Summarizes structured outputs from each analysis agent.
    # Includes assessments and citations for rubric scoring and traceability.
    report += "\n## Agent Contributions\n"

    for agent in analysis_bundle:
        agent_type = agent.get("agent", "UnknownAgent")

        if agent_type == "QualityAgent":
            report += "\n### QualityAgent\n"
            # New schema (preferred)
            if any(k in agent for k in ["profitability_strength",␣
↪"efficiency_and_scale", "financial_flexibility"]):
                report += f"- **Profitability strength**: {agent.
↪get('profitability_strength', 'N/A')}\n"
                report += f"- **Efficiency & scale**: {agent.
↪get('efficiency_and_scale', 'N/A')}\n"
                report += f"- **Financial flexibility**: {agent.
↪get('financial_flexibility', 'N/A')}\n"
                gaps = agent.get("evidence_gaps", [])
                if gaps:
                    report += f"- **Evidence gaps**: {', '.join(gaps)}\n"
            else:
                # Back-compat (older schema)
                for key in ["moat", "customer_concentration",␣
↪"management_track_record"]:
                    if key in agent:
                        value = agent[key]
                        if isinstance(value, dict):
                            assessment = value.get("assessment", "N/A")
                            citations = value.get("citations", [])
                        else:
                            assessment = str(value)
                            citations = []
                        report += f"- **{key.replace('_', ' ').title()}**:␣
↪{assessment}\n"
                        if citations:
                            report += f"  - Citations: {', '.join(citations)}\n"

        elif agent_type == "ValuationAgent":
            report += "\n### ValuationAgent\n"
            # New schema (preferred)
            if ("valuation_view" in agent) or ("justification" in agent):
                report += f"- **Valuation View**: {agent.get('valuation_view',␣
↪'N/A')}\n"
                report += f"- **Justification**: {agent.get('justification', 'N/
↪A')}\n"
                cidx = agent.get("citation_indices", [])
                if cidx:
                    report += f"- **Citations (indices)**: {', '.join(map(str,␣
↪cidx))}\n"
            else:
                # Back-compat (older schema)
                report += f"- **Valuation**: {agent.get('valuation', 'N/A')}\n"
                report += f"- **Justification**: {agent.get('justification', 'N/
↪A')}\n"
                citations = agent.get("citations", [])
                if citations:
                    report += f"- **Citations**: {', '.join([c if isinstance(c,␣
↪str) else c.get('citation', '') for c in citations])}\n"

        elif agent_type == "RiskAgent":
            report += "\n### RiskAgent\n"
            # New schema (preferred)
            if ("risks" in agent) or ("mitigants" in agent):
                risks = agent.get("risks", [])
                mitigants = agent.get("mitigants", [])
                report += f"- **Risks**: {', '.join([r if isinstance(r, str)␣
↪else str(r) for r in risks])}\n"
                if mitigants:
                    report += f"- **Mitigants**: {', '.join([m if isinstance(m,␣
↪str) else str(m) for m in mitigants])}\n"
                cidx = agent.get("citation_indices", [])
                if cidx:
                    report += f"- **Citations (indices)**: {', '.join(map(str,␣
↪cidx))}\n"
            else:
                # Back-compat (older schema)
                risks = agent.get("risks", [])
                counterpoints = agent.get("counterpoints", [])
                report += f"- **Risks**: {', '.join([r if isinstance(r, str)␣
↪else r.get('description', '') for r in risks])}\n"
                report += f"- **Counterpoints**: {', '.join([c if isinstance(c,␣
↪str) else c.get('counterpoint', '') for c in counterpoints])}\n"
                citations = agent.get("citations", [])
                if citations:
                    report += f"- **Citations**: {', '.join([c if isinstance(c,␣
↪str) else c.get('citation', '') for c in citations])}\n"

    # === Supporting Evidence ===
    # Lists all normalized evidence used by agents.
    # Useful for rubric reviewers to trace signal origin and scoring.
    report += "\n## Supporting Evidence\n"
    for item in evidence:
        report += f"- {item.get('section_hint', 'Unknown')}: {item.get('text',␣
↪'')} (score: {item.get('score', 'N/A')})\n"

    # === Prompt Chaining Trace ===
    # Renders the trace from collect_comprehensive_data.
    # Includes raw news metadata, preprocessed signals, classification, and␣
↪summary.
    if trace:
        report += "\n## Prompt Chaining Trace\n"
        report += format_trace_md(trace, company_name=company_name)

    return report
```

[19]:
```
# === Final Report Assembly (with de-duplication) ===
from IPython.display import Markdown, display

# Safely extract components from the graph response
thesis = (response.get("draft_thesis") or {})
evidence = (response.get("evidence_pack") or [])
trace = (response.get("trace") or {})
analysis_bundle = (response.get("analysis_bundle") or [])
company = response.get("meta", {}).get("company_name", "Unknown")

# --- Deduplicate repeated agent outputs (in case the graph looped) ---
def _dedupe_analysis_bundle(bundle):
    seen = set()
    unique = []
    for item in bundle:
        agent_type = item.get("agent")
        if agent_type and agent_type not in seen:
            seen.add(agent_type)
            unique.append(item)
    return unique

analysis_bundle = _dedupe_analysis_bundle(analysis_bundle)

# Generate the full report markdown
final_report_md = build_report(
    thesis=thesis,
    evidence=evidence,
    trace=trace,
    analysis_bundle=analysis_bundle,
    company_name=company
)

# Display the markdown report inline
display(Markdown(final_report_md))
```

15 Investment Thesis Report – Apple Inc.

15.1 Thesis Summary

Bull Case: ['Strong operational quality indicated by a gross margin of 46.2% and operating margin of 31.5%', 'Positive price momentum with a 60-day price change of 15.9%'] **Bear Case**: ['High valuation risk indicated by a P/E ratio of 37.61', 'Liquidity risk due to a current ratio of 0.87', 'Leverage risk indicated by a debt-to-equity ratio of 1.87', 'Operational risk indicated by a beta of 1.094 and annualized volatility of 25.9%'] **Confidence**: 0.0 **Catalysts**: insufficient_evidence

15.2 Agent Contributions

15.2.1 QualityAgent

- **Profitability strength**: The company has a return on equity (ROE) of 1.4981, which is relatively low, indicating limited profitability relative to shareholder equity. However, the gross margin of 46.2% and operating margin of 31.5% suggest strong profitability at the operational level, indicating that the company retains a significant portion of revenue as profit after covering its costs.
- **Efficiency & scale**: The evidence does not provide specific metrics related to cost control or utilization rates. However, the high gross and operating margins imply effective cost management and operational efficiency, allowing the company to maintain substantial profitability despite potentially high operational costs.
- **Financial flexibility**: The current ratio of 0.87 indicates potential liquidity issues, as it is below the ideal threshold of 1. This suggests that the company may struggle to cover its short-term liabilities with its short-term assets. The debt-to-equity ratio of 1.87 indicates

a high level of leverage, which may increase financial risk and reduce financial flexibility. Overall, the company appears to have limited financial flexibility due to its liquidity position and high leverage.
- **Evidence gaps**: Details on net income or total equity to better assess ROE context, Information on cash flow metrics to evaluate liquidity beyond the current ratio, Data on operational efficiency metrics such as inventory turnover or asset utilization

15.2.2 ValuationAgent

- **Valuation View**: overvalued
- **Justification**: The P/E ratio of 37.61 is significantly high compared to historical averages, indicating that the stock may be overvalued. Additionally, the return on equity (ROE) of 1.4981 is relatively low, which does not justify such a high valuation. The high debt-to-equity ratio of 1.87 also indicates increased financial risk, further supporting the overvaluation stance.
- **Citations (indices)**: 1, 2, 7

15.2.3 RiskAgent

- **Risks**: High valuation risk indicated by a P/E ratio of 37.61, Liquidity risk due to a current ratio of 0.87, Leverage risk indicated by a debt-to-equity ratio of 1.87, Operational risk indicated by a beta of 1.094 and annualized volatility of 25.9%
- **Mitigants**: Strong operational quality indicated by a gross margin of 46.2% and operating margin of 31.5%, Positive price momentum with a 60-day price change of 15.9%
- **Citations (indices)**: 1, 6, 7, 3, 9, 4, 5, 8

15.3 Supporting Evidence

- Valuation: P/E ratio is 37.61 (score: 0.9)
- Valuation: 60-day price change is 15.9% (score: 0.8)
- Quality: Gross margin is 46.2% (score: 0.9)
- Quality: Operating margin is 31.5% (score: 0.9)
- Risk: Current ratio is 0.87 (score: 0.85)
- Risk: Debt-to-equity ratio is 1.87 (score: 0.85)
- News: Recent news: China's Wentao blames US actions for trade tensions (score: 0.75)
- News: Recent news: Apple is reportedly making robots. Here's what you need to know (score: 0.75)
- News: Recent news: Apple loses another AI exec to Meta (score: 0.75)

---

16 PIPELINE RUN 2

Full Pipeline Execution for Alternative Ticker This cell demonstrates complete pipeline execution for NVDA, showcasing multi-source integration (yfinance, SEC EDGAR, Finnhub). It collects real evidence, runs all agents, synthesizes a thesis, and applies critique. The state is serialized and saved for reproducibility.

```
[20]: # === Alternate ticker Pipeline Run ===
from typing import Any, Dict, List
from pprint import pprint
import json, math
from datetime import datetime, date, time

# ---- Config ----
ticker = "NVDA"    # for Nvidia stock analysis
debug = bool(globals().get("debug", False))

print(f"\n{'='*60}")
print(f"PIPELINE RUN: {ticker}")
print(f"{'='*60}\n")

# 1) Fresh collection
result = collect_comprehensive_data(ticker)
meta: Dict[str, Any] = result.get("meta", {})
evidence: List[Dict[str, Any]] = result.get("evidence_pack", [])

# 2) Normalize once (local)
normalized: List[Dict[str, Any]] = normalize_evidence(evidence)

print(f"[INFO] Normalized evidence items: {len(normalized)}\n")
if debug:
    print("Metadata:")
    pprint(meta)
    print("\nSample Evidence (first 5):")
    for i, e in enumerate(normalized[:5], 1):
        print(f"{i}. [{e.get('source')}] {e.get('section_hint')}: {e.
↪get('text')}")

# 3) Helper to tag agent outputs
def tag(agent_name: str, out: Dict) -> Dict:
    out = out or {}
    if isinstance(out, dict) and "agent" not in out:
        out["agent"] = agent_name
    return out if isinstance(out, dict) else {"agent": agent_name}

# 4) Run the agent functions directly (positional args only)
analysis_bundle: List[Dict[str, Any]] = []
if normalized and len(normalized) >= 3:
    print("Running agents...")

    qa_out = gpt_quality_agent(normalized)
    va_out = gpt_valuation_agent(normalized)
    ra_out = gpt_risk_agent(normalized)

    analysis_bundle.append(tag("QualityAgent", qa_out))
    analysis_bundle.append(tag("ValuationAgent", va_out))
    analysis_bundle.append(tag("RiskAgent", ra_out))

    if debug:
        print("\n[QualityAgent output]"); pprint(qa_out)
        print("\n[ValuationAgent output]"); pprint(va_out)
        print("\n[RiskAgent output]"); pprint(ra_out)

    # 5) Synthesize thesis + run critic (positional calls for safety)
    draft_thesis = gpt_thesis_writer(analysis_bundle, normalized)
    critic_patch = gpt_critic_agent(draft_thesis, normalized)


    try:
        patch_raw = critic_patch.get("patch")
        if isinstance(patch_raw, str):
            patch_obj = json.loads(patch_raw)
            if isinstance(patch_obj, dict):
                draft_thesis.update(patch_obj)
                print("[INFO] Applied critic patch to draft_thesis.")
    except Exception:
        pass

    print("\n=== CRITIC RESULT ===")
    print("Critic verdict:", "VALID" if critic_patch.get("valid") else "NEEDS␣
↪WORK")
    print("Critic says:", critic_patch.get("summary") or "(no summary␣
↪returned)")
    if critic_patch.get("patch"):
        print("\nSuggested patch:\n", critic_patch.get("patch"))

    # 6) Persist under per-ticker namespace and set current pointers
    state.setdefault("runs", {})
    state["runs"][ticker] = {
        "ticker": ticker,
        "meta": meta,
        "evidence_pack": evidence,           # this run's raw evidence
        "normalized_evidence": normalized,   # this run's normalized evidence
        "analysis_bundle": analysis_bundle,
        "draft_thesis": draft_thesis,
        "critic_patch": critic_patch,
    }
    state["ticker"] = ticker
    state["meta"] = meta
    state["evidence_pack"] = evidence
    state["normalized_evidence"] = normalized
    state["analysis_bundle"] = analysis_bundle
    state["draft_thesis"] = draft_thesis
    state["critic_patch"] = critic_patch

    print("\n[COMPLETE] Pipeline finished.\n")
else:
    print("[ERROR] Insufficient evidence to run agents.")

# 7) Safe message serialization
def _serialize_messages(messages: List) -> List[Dict]:
    out = []
    for m in messages or []:
        if hasattr(m, "role") and hasattr(m, "content"):
            out.append({"role": m.role, "content": m.content})
        elif hasattr(m, "type") and hasattr(m, "content"):
            role = "user" if m.type in ("human", "user") else "assistant"
            out.append({"role": role, "content": m.content})
        elif isinstance(m, dict) and "role" in m and "content" in m:
            out.append({"role": m["role"], "content": m["content"]})
        else:
            out.append({"role": "system", "content": str(m)})
    return out

if "messages" in state:
    state["messages"] = _serialize_messages(state["messages"])

# 8) JSON-safe save
def _to_jsonable(obj: Any) -> Any:
    try:
        from decimal import Decimal
    except Exception:
        Decimal = None
    try:
        import numpy as np
    except Exception:
        np = None

    if obj is None or isinstance(obj, (bool, int, float, str)):
        if isinstance(obj, float) and (math.isnan(obj) or math.isinf(obj)):
            return None
        return obj
    if isinstance(obj, (datetime, date, time)):
        return obj.isoformat()
    if Decimal is not None and isinstance(obj, Decimal):
        f = float(obj);  return None if (math.isnan(f) or math.isinf(f)) else f
    if np is not None:
        if isinstance(obj, (np.integer,)):  return int(obj)
        if isinstance(obj, (np.floating,)):
            f = float(obj);  return None if (math.isnan(f) or math.isinf(f))␣
 ↪else f
        if isinstance(obj, (np.ndarray,)):  return [_to_jsonable(x) for x in␣
 ↪obj.tolist()]
    if isinstance(obj, (bytes, bytearray)):  return obj.decode("utf-8",␣
 ↪errors="replace")
    if isinstance(obj, dict):                return {str(_to_jsonable(k)):␣
 ↪_to_jsonable(v) for k, v in obj.items()}
    if isinstance(obj, (list, tuple, set)):  return [_to_jsonable(x) for x in␣
 ↪obj]
    # pydantic
    try:
        from pydantic import BaseModel as _BM
        if isinstance(obj, _BM):
            return _to_jsonable(obj.model_dump())
    except Exception:
        pass
    return str(obj)

def _save_memory_safe(memory: Dict[str, Any], path: str = None):
    out_path = path or globals().get("MEMORY_PATH") or "memory.json"
    serializable = _to_jsonable(memory)
    with open(out_path, "w") as f:
        json.dump(serializable, f, indent=2)
    print(f"[SAVE] State persisted to {out_path}")

# Use existing save_memory_safe if available; else fallback to our local one
if "save_memory_safe" in globals() and callable(globals()["save_memory_safe"]):
    save_memory_safe(state)  # your existing robust saver
else:
    _save_memory_safe(state)

print("State saved to memory")
```

```
============================================================
PIPELINE RUN: NVDA
============================================================

[INFO] Collecting data for NVDA…
[INFO] Collected 14 evidence items from 3 sources
[INFO] Normalized evidence items: 14
```

```
Metadata:
{'cik': '0001045810',
 'company_name': 'NVIDIA Corporation',
 'exchange': 'NMS',
 'industry': 'Semiconductors',
 'marketCap': 4426527932416,
 'price': 181.81,
 'sector': 'Technology',
 'ticker': 'NVDA'}

Sample Evidence (first 5):
1. [yfinance] Valuation: P/E ratio is 51.80
2. [yfinance] Quality: Return on equity is 1.0942
3. [yfinance] Risk: Beta is 2.123
4. [SEC EDGAR] Quality: Gross margin is 363.6%
5. [SEC EDGAR] Quality: Operating margin is 302.6%
Running agents…
[LLM raw t=1] {
  "profitability_strength": "The company exhibits exceptionally high
profitability strength with a gross margin of 363.6% and an operating margin of
302.6%. However, the return on equity (ROE) is relatively low at 1.0942,
indicating that while the company is generating high margins, it may not be
effectively utilizing its equity to generate returns.",
  "efficiency_and_scale": "The evidence does
[LLM raw t=1] {
  "valuation_view": "overvalued",
  "justification": "The P/E ratio of 51.80 indicates a high valuation compared
to typical market levels, suggesting overvaluation. Additionally, while the
return on equity is relatively high at 1.0942, the extreme gross margin of
363.6% and operating margin of 302.6% may not be sustainable in the long term.
The low debt-to-equity ratio of 0.11 indicates low leve
[LLM raw t=1] ```json
{
  "risks": [
    "High valuation risk indicated by a P/E ratio of 51.80",
    "High operational risk indicated by a beta of 2.123",
    "High annualized volatility of 30.1% indicating potential price
fluctuations"
  ],
  "mitigants": [
    "Strong liquidity position indicated by a current ratio of 4.44",
    "Low leverage indicated by a debt-to-equity ratio of 0.11",
    "High gross marg

[QualityAgent output]
{'agent': 'QualityAgent',
 'citation_indices': [2, 4, 5, 6, 7],
```

```
 'efficiency_and_scale': 'The evidence does not provide specific metrics on '
                         'cost control or utilization rates, making it '
                         'difficult to assess efficiency and scale directly. '
                         'However, the high margins suggest that the company '
                         'may have strong pricing power or operational '
                         'efficiency.',
 'evidence_gaps': ['Lack of detailed information on cost control measures and '
                   'operational efficiency metrics.',
                   'Absence of net income or total equity figures to better '
                   'assess ROE context.',
                   'No information on cash flow metrics or long-term debt '
                   'levels.'],
 'financial_flexibility': 'The company demonstrates strong financial '
                          'flexibility with a current ratio of 4.44, '
                          'indicating ample liquidity to cover short-term '
                          'obligations. Additionally, a low debt-to-equity '
                          'ratio of 0.11 suggests minimal leverage, which '
                          'enhances financial stability.',
 'profitability_strength': 'The company exhibits exceptionally high '
                           'profitability strength with a gross margin of '
                           '363.6% and an operating margin of 302.6%. However, '
                           'the return on equity (ROE) is relatively low at '
                           '1.0942, indicating that while the company is '
                           'generating high margins, it may not be effectively '
                           'utilizing its equity to generate returns.'}

[ValuationAgent output]
{'agent': 'ValuationAgent',
 'citation_indices': [1, 2, 4, 5, 7, 3, 8],
 'justification': 'The P/E ratio of 51.80 indicates a high valuation compared '
                  'to typical market levels, suggesting overvaluation. '
                  'Additionally, while the return on equity is relatively high '
                  'at 1.0942, the extreme gross margin of 363.6% and operating '
                  'margin of 302.6% may not be sustainable in the long term. '
                  'The low debt-to-equity ratio of 0.11 indicates low '
                  'leverage, which is positive, but the high beta of 2.123 '
                  'suggests increased risk. The recent price change of 4.7% '
                  'does not indicate strong upward momentum, further '
                  'supporting the overvaluation stance.',
 'valuation_view': 'overvalued'}

[RiskAgent output]
{'agent': 'RiskAgent',
 'citation_indices': [1, 3, 9, 6, 7, 4, 5],
 'mitigants': ['Strong liquidity position indicated by a current ratio of 4.44',
               'Low leverage indicated by a debt-to-equity ratio of 0.11',
               'High gross margin of 363.6% and operating margin of 302.6% '
               'indicating strong operational efficiency'],
 'risks': ['High valuation risk indicated by a P/E ratio of 51.80',
           'High operational risk indicated by a beta of 2.123',
           'High annualized volatility of 30.1% indicating potential price '
           'fluctuations']}
[LLM raw t=1] {
  "thesis": "The company exhibits high profitability with exceptional gross and
operating margins, but faces valuation concerns due to a high P/E ratio and
increased risk indicated by its beta and volatility. While strong liquidity and
low leverage provide some financial stability, the sustainability of its margins
and overall valuation remain in question.",
  "bull_case": [
    "Exceptional gro
[LLM raw t=1] ```json
{
  "valid": true,
  "summary": "The thesis is balanced, presenting both bullish and bearish
arguments supported by relevant evidence. It cites specific metrics that
highlight profitability, risk, and valuation concerns, providing a well-rounded
perspective. The confidence level is reasonable given the mixed signals from the
data.",
  "patch": ""
}
```

=== CRITIC RESULT ===
Critic verdict: VALID
Critic says: The thesis is balanced, presenting both bullish and bearish
arguments supported by relevant evidence. It cites specific metrics that
highlight profitability, risk, and valuation concerns, providing a well-rounded
perspective. The confidence level is reasonable given the mixed signals from the
data.

[COMPLETE] Pipeline finished.

[SAVE] State persisted to agent_memory.json
State saved to memory
```

## 17 FINAL REPORT

**Final Report Rendering for Rubric Scoring and Export-Ready Documentation** This cell compiles all pipeline outputs into a markdown-formatted investment report using
```

build_report. It integrates the thesis, agent assessments, supporting evidence, and trace into a single cohesive artifact. Designed for rubric alignment and reproducibility, the report can be reviewed inline and audited for traceable reasoning. This marks the final synthesis step of the agentic pipeline.

[21]:
```
# === Render Final Investment Report(s) ===
# Builds markdown reports for ticker in state["runs"]

from pprint import pprint

def render_single_report(run_state: dict) -> str:
    thesis = run_state.get("draft_thesis", {})                    # Synthesized␣
 ↪thesis
    evidence = run_state.get("normalized_evidence", [])           # Normalized␣
 ↪evidence used by agents
    trace = run_state.get("trace", {})
    analysis_bundle = run_state.get("analysis_bundle", [])        # Outputs from␣
 ↪Quality/Valuation/Risk
    company_name = run_state.get("meta", {}).get("company_name", "Unknown")

    return build_report(
        thesis=thesis,
        evidence=evidence,
        trace=trace,
        analysis_bundle=analysis_bundle,
        company_name=company_name
    )

reports_md = []

if isinstance(state.get("runs"), dict) and state["runs"]:
    # Stable ordering by ticker symbol
    for ticker_key in sorted(state["runs"].keys()):
        run = state["runs"][ticker_key] or {}
        header = f"\n\n# === Final Investment Report: {ticker_key} ===\n"
        try:
            report_md = render_single_report(run)
        except Exception as e:
            report_md = f"_Error building report for {ticker_key}: {e}_"
        reports_md.append(header + str(report_md))
else:
    # Fallback: render from top-level state (single ticker context)
    header = f"\n\n# === Final Investment Report: {state.
 ↪get('ticker','Unknown')} ===\n"
    try:
        report_md = render_single_report(state)
    except Exception as e:
        report_md = f"_Error building report: {e}_"
    reports_md.append(header + str(report_md))

# Display reports inline (concatenated)
final_output = "\n".join(reports_md)
print(final_output)
```

```
# === Final Investment Report: NVDA ===
# Investment Thesis Report - NVIDIA Corporation

## Thesis Summary
**Bull Case**: ['Exceptional gross margin of 363.6% and operating margin of
302.6% indicate strong pricing power.', 'Strong liquidity position with a
current ratio of 4.44 suggests the ability to meet short-term obligations.',
'Low debt-to-equity ratio of 0.11 enhances financial stability.']
**Bear Case**: ['High P/E ratio of 51.80 suggests overvaluation compared to
market norms.', 'Beta of 2.123 indicates increased operational risk and
potential for significant price fluctuations.', 'Annualized volatility of 30.1%
raises concerns about price stability.']
**Confidence**: 0.0
**Catalysts**: Recent funding news may enhance growth prospects and operational
capabilities.

## Agent Contributions

### QualityAgent
- **Profitability strength**: The company exhibits exceptionally high
profitability strength with a gross margin of 363.6% and an operating margin of
302.6%. However, the return on equity (ROE) is relatively low at 1.0942,
indicating that while the company is generating high margins, it may not be
effectively utilizing its equity to generate returns.
- **Efficiency & scale**: The evidence does not provide specific metrics on cost
control or utilization rates, making it difficult to assess efficiency and scale
directly. However, the high margins suggest that the company may have strong
pricing power or operational efficiency.
- **Financial flexibility**: The company demonstrates strong financial
flexibility with a current ratio of 4.44, indicating ample liquidity to cover
short-term obligations. Additionally, a low debt-to-equity ratio of 0.11
suggests minimal leverage, which enhances financial stability.
- **Evidence gaps**: Lack of detailed information on cost control measures and
operational efficiency metrics., Absence of net income or total equity figures
to better assess ROE context., No information on cash flow metrics or long-term
debt levels.

### ValuationAgent
- **Valuation View**: overvalued
- **Justification**: The P/E ratio of 51.80 indicates a high valuation compared
to typical market levels, suggesting overvaluation. Additionally, while the
return on equity is relatively high at 1.0942, the extreme gross margin of
363.6% and operating margin of 302.6% may not be sustainable in the long term.
The low debt-to-equity ratio of 0.11 indicates low leverage, which is positive,
but the high beta of 2.123 suggests increased risk. The recent price change of
4.7% does not indicate strong upward momentum, further supporting the
overvaluation stance.
- **Citations (indices)**: 1, 2, 4, 5, 7, 3, 8

### RiskAgent
- **Risks**: High valuation risk indicated by a P/E ratio of 51.80, High
operational risk indicated by a beta of 2.123, High annualized volatility of
30.1% indicating potential price fluctuations
- **Mitigants**: Strong liquidity position indicated by a current ratio of 4.44,
Low leverage indicated by a debt-to-equity ratio of 0.11, High gross margin of
363.6% and operating margin of 302.6% indicating strong operational efficiency
- **Citations (indices)**: 1, 3, 9, 6, 7, 4, 5

## Supporting Evidence
- Valuation: P/E ratio is 51.80 (score: 0.9)
- Quality: Return on equity is 1.0942 (score: 0.85)
- Risk: Beta is 2.123 (score: 0.8)
- Quality: Gross margin is 363.6% (score: 0.9)
- Quality: Operating margin is 302.6% (score: 0.9)
- Risk: Current ratio is 4.44 (score: 0.85)
- Risk: Debt-to-equity ratio is 0.11 (score: 0.85)
- Valuation: 60-day price change is 4.7% (score: 0.8)
- Risk: Annualized volatility is 30.1% (score: 0.8)
- News: Recent news: Unikraft Raises $6M From Heavybit And Vercel Ventures To
Revolutionize AI Cloud Speed With Millisecond-Native Platform (score: 0.75)
- News: Recent news: Stock Market Today: Nasdaq Up, Snowflake Tests Entry;
NuScale Leads Nukes (Live Coverage) (score: 0.75)
- News: Recent news: Nvidia partners on $2.9 billion AI data center project in
Australia (score: 0.75)
- News: Recent news: OpenAI would have to spend over $1 trillion to deliver its
promised computing power. It may not have the cash. (score: 0.75)
- News: Recent news: ABB CEO 'very confident' of demand for data centers
powering AI (score: 0.75)
```

---

18 RUBRIC REQUIREMENTS

Rubric Checklist

- Agent Functions: planning, tool usage, self-reflection, memory
- Workflow Patterns: prompt chaining, routing, evaluator–optimizer
- Code: modular agents, reproducible state, error handling
- API Integration: live financial metadata via yfinance
- Final Report: markdown-formatted investment thesis
- Prompt Chaining Trace: visible and structured