EarningsCalendarProvider

earnings_calendar_provider.ipynb:

EarningsCalendarProvider: tells us when the event happens (the next earnings dates). It's a thin wrapper around the Finnhub API that fetches an upcoming earnings calendar for a given stock (e.g., AAPL) within a configurable lookahead window (default 120 days). You initialize EarningsCalendarProvider with an API key and ticker, it builds a Finnhub client, computes a time window from "now" to "now + lookahead," calls earnings_calendar(...), and returns whatever Finnhub sends back. The class is meant to be a plug-in style "provider" you can swap or extend later (e.g., add a user-supplied function, multiple data sources, or logic to pick the single next earnings date).

Why EarningsCalendarProvider matters ?

Knowing the next earnings date is crucial for automation, enabling workflows like setting alerts and backtesting strategies. It also enhances user experience with timely notifications ("AAPL reports in 5 days"), calendar overlays, and conditional tasks (e.g., scrape news the day before) and facilitates data quality through

In [1]:

```python
# import necessary libraries
from __future__ import annotations
from datetime import datetime, timedelta, timezone
from typing import Any

import os
import finnhub

class EarningsCalendarProvider:
    """
    Earnings date lookup with sensible fallbacks.
    Free Tier: 1 month of historical earnings and new updates...

    Order of operations:
      Try an optional user-supplied provider function (if given).

    A "provider function" should be:  (ticker: str, now_utc: datetime, lookahead_days: int) -> Iterable[datetime]
    It returns one or more candidate datetimes (past or future). We'll pick the next upcoming one.
    """

    def __init__(
            self,
            api_key: str,
            ticker: str,
            lookahead_days: int = 120,
            prefer_window: bool = True,
    ):
        self.client = finnhub.Client(api_key=api_key)
        self.lookahead_days = int(lookahead_days)
        self.ticker = ticker
        self.prefer_window = prefer_window  # prefer a date within lookahead window if multiple upcoming exist

    # ---------- public API ----------
    def future_earnings(self) -> Any | None:
        try:
            now = datetime.now(timezone.utc)
```

```python
            window_end = now + timedelta(days=self.lookahead_days)

            # user provider
            dt = self.client.earnings_calendar(_from=now, to=window_end, symbol=self.ticker, international=False)
            if dt:
                return dt
        except Exception as e:
            print("Error fetching next_earnings error.", repr(e))

ct = EarningsCalendarProvider(ticker="AAPL", api_key="api_key_here")
calendar = ct.future_earnings()
print(f"Future earnings calendar: {calendar}")
print(calendar)
```

```
Future earnings calendar: {'earningsCalendar': [{'date': '2026-01-28', 'epsActual': None, 'epsEstimate': 2.5411, 'hour': 'amc', 'quarter': 1, 'revenueActual': None, 'revenueEstimate
': 133684531371, 'symbol': 'AAPL', 'year': 2026}, {'date': '2025-10-30', 'epsActual': None, 'epsEstimate': 1.7924, 'hour': 'amc', 'quarter': 4, 'revenueActual': None, 'revenueEstima
te': 103706233519, 'symbol': 'AAPL', 'year': 2025}]}
{'earningsCalendar': [{'date': '2026-01-28', 'epsActual': None, 'epsEstimate': 2.5411, 'hour': 'amc', 'quarter': 1, 'revenueActual': None, 'revenueEstimate': 133684531371, 'symbol':
'AAPL', 'year': 2026}, {'date': '2025-10-30', 'epsActual': None, 'epsEstimate': 1.7924, 'hour': 'amc', 'quarter': 4, 'revenueActual': None, 'revenueEstimate': 103706233519, 'symbol'
: 'AAPL', 'year': 2025}]}
```

Market News Provider

market_news_provider.ipynb

It builds a Market News Provider that fetches news from Finnhub for a chosen category (general, forex, crypto, merger). Each article is processed through a three-node pipeline (per article):

- sentiment analysis. predicts overall sentiment (positive/negative/neutral)
- topic categorization. assigns a topical label (inflation, rates, fed, macro, other)
- summarization (produces bullet-point summaries of each article). It then uses map-reduce summarization (summary of the summaries) to distill a final executive summary.

Finally, the end product is exposed as market_report.

Why MarketNewsProvider matters ?

From raw headlines to decision-ready insights, this tool performs classification, summarization, and theming, resulting in a concise executive-level brief that can be read in minutes. It follows an agentic workflow pattern, demonstrating core patterns such as tool use (using Finnhub), prompt chaining (classifying, categorizing, and summarizing), and a simple map-reduce condenser, which illustrates how modern “research agents” are assembled and provides actionable metrics for trend tracking and routing.

In [1]:
```
import os
from langchain_ollama import ChatOllama

# Initialize an Ollama Client for our generative Llm model

text_model = "llama3.2"

def llm_client_loader():
    """This function serves an Ollama-hosted text generator model, to be used by our graphs."""
    try:
        llm = ChatOllama(
            model=text_model,
            temperature=0.2
        )
        return llm
    except Exception as e:
        print(f"Error {e} instantiating the Ollama client, is the Ollama server running?.")

# could add something to verify Ollama is running
```

```
/Users/cortizmontesdeoca/Documents/usd/aai-520-group7-final-project/.venv/lib/python3.13/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and i
pywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [2]:
```
import os
import finnhub
from langgraph.constants import START
from langgraph.graph import StateGraph
from langgraph.graph import END
from pydantic import Field, BaseModel
from typing import Literal, Any, TypedDict
from langchain_core.messages import HumanMessage, SystemMessage
```

```python
class MarketNewsProvider:
    """
    Market News Provider of a specific given category.
    """
    def __init__(
            self,
            api_key: str,
            category: Literal["general","forex","crypto", "merger"],
    ):
        self.client = finnhub.Client(api_key=api_key)
        self.category = category
        self.news_repository = self.fetch()
        self.llm_client = llm_client_loader()
        self.graph = self.graph_builder()
        self.sentiment_by_category = {"inflation":[],"rates":[],"fed":[],"macro":[], "other":[]}
        self.market_report = self.summarizer()

    def fetch(self) -> Any | None:
        try:
            if self.category:
                news_repository = self.client.general_news(category=self.category)
                return news_repository
            else:
                print(f"Missing news category parameter.")
        except Exception as e:
            print("Error fetching Market news error.", repr(e))

    def graph_builder(self) -> Any | None:
        try:
            class State(TypedDict):
                news_article: str
                sentiment: str  # result sentiment
                label: str  # result label
                bullets: str

            # Internal State definitions
            class News_Sentimenter(BaseModel):
                sentiment : Literal["positive", "negative", "neutral"] = Field(
                    description="The sentiment detected for the news article.")

            sentimenter_llm = self.llm_client.with_structured_output(News_Sentimenter)

            # Nodes / Tools
            def sentimenter(state: TypedDict):
                try:
                    """Classifier that classifies the news articles with an overall perceived sentiment.
                    """

                    # Generate queries...
                    news_articles = sentimenter_llm.invoke(
                        [
                            SystemMessage(
                                content="You are a Sentiment Analysis expert. Classify the following news article, "
                                        "assess the overall sentiment and categorize accordingly."),
                            HumanMessage(
```

```python
                    content=f"News article: {state["news_article"]}."),
                ]
            )

            return {"sentiment": news_articles.sentiment}

        except Exception as e:
            print(f"Error {e} during the classifier process.")

    # Internal State definitions
    class News_Categorizer(BaseModel):
        label: Literal["inflation", "rates", "fed", "macro", "other"] = Field(
            description="The category that resembles the news article the most.")

    categorizer_llm = self.llm_client.with_structured_output(News_Categorizer)

    # Nodes / Tools
    def categorizer(state: TypedDict):
        try:
            """Classifier that classifies the news articles with a news category.
            """

            # Generate queries...
            news_articles = categorizer_llm.invoke(
                [
                    SystemMessage(
                        content="You are a News Categorization expert. Classify the following news article, "
                                "accordingly."),
                    HumanMessage(
                        content=f"News article: {state["news_article"]}."),
                ]
            )

            return {"label": news_articles.label}

        except Exception as e:
            print(f"Error {e} during the classifier process.")

    # map reduce (later)
    def synthesizer(state: TypedDict):
        """Synthesizer that summarizes the news article into a couple of bullet points."""
        try:
            # Generate queries...
            bullet_points = self.llm_client.invoke(
                [
                    SystemMessage(
                        content="Provide a bullet-point summary of the main arguments in the following news "
                                "article. Keep it concise while at the same time not losing any of the most"
                                "important information."),
                    HumanMessage(
                        content=f"News article: {state["news_article"]}."),
                ]
            )

            return {"bullets": bullet_points.content}
        except Exception as e:
```

```python
                print(f"Error {e} during the synthesizer process.")

        # Build workflow
        graph_constructor = StateGraph(State)

        # Add the nodes
        graph_constructor.add_node("sentimenter", sentimenter)
        graph_constructor.add_node("categorizer", categorizer)
        graph_constructor.add_node("synthesizer", synthesizer)

        # Add edges to connect nodes
        graph_constructor.add_edge(START, "sentimenter")
        graph_constructor.add_edge(START, "categorizer")
        graph_constructor.add_edge(START, "synthesizer")
        graph_constructor.add_edge("sentimenter", END)
        graph_constructor.add_edge("categorizer", END)
        graph_constructor.add_edge("synthesizer", END)

        # Compile the workflow
        graph = graph_constructor.compile()
        return graph
    except Exception as e:
        print(f"Error {e} during the graph building process.")

def summarizer(self):
    try:
        bullet_points = []

        for news in self.news_repository:
            headline = news["headline"]
            news_summary = news["summary"]
            prompt = HumanMessage(f"{headline}: {news_summary}")

            state = self.graph.invoke({"news_article": f"{prompt.content}"})

            self.sentiment_by_category[state['label']].append(state["sentiment"])
            bullet_points.append(state["bullets"])

        def group_bullets(summaries, max_summaries):
            mag = []
            current_mag = []
            for summary in summaries:
                current_mag.append(summary)
                if len(current_mag) >= max_summaries:
                    mag.append(current_mag)
                    current_mag = []
            if len(current_mag) > 0:
                mag.append(current_mag)

            return mag

        buckets = group_bullets(bullet_points, 7)
        final_buckets = []

        for bucket in buckets:
            response = self.llm_client.invoke(f"The following is set of bullet-point summaries: {bucket} Take these "
```

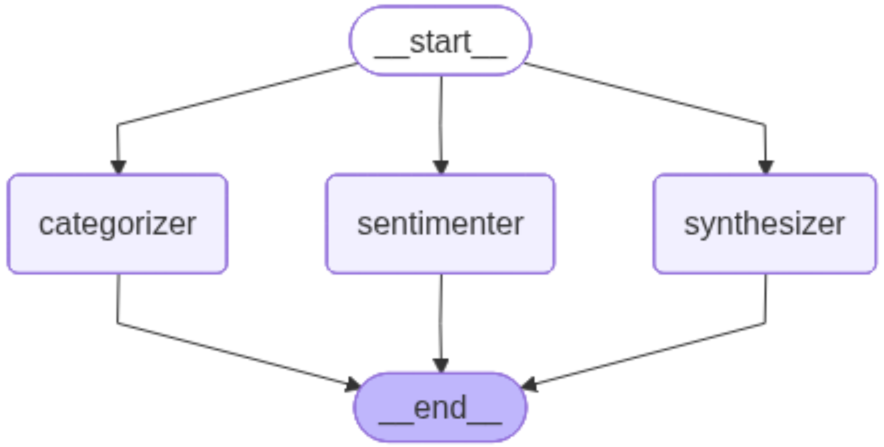
```python
                                                     f"and distill it into a consolidated bullet-point summary of the main "
                                                     f"themes. Remove the bullet points that are not relevant to the whole "
                                                     f"text. The consolidated summary cannot be more than 11 bullet points.")

                final_buckets.append(response.content)

            final_response = self.llm_client.invoke(f"The following is set of bullet-point summaries: {final_buckets} "
                                                       f"Take these and distill it into a final executive summary of the "
                                                       f"main themes, capturing the key ideas without missing critical "
                                                       f"points. Ensure the summary touches upon all of main themes "
                                                       f"found, and be sure to include important details.")

            return final_response.content

        except Exception as e:
            print(f"Error {e} during the graph building process.")

# Invoke
mn = MarketNewsProvider(category="general", api_key="api key here")
print(mn.market_report)
```

```
Here is a consolidated bullet-point summary of the main themes:

* The stock market will be closely watched for updates on five stocks in a portfolio, with investors experiencing market volatility and uncertainty.
* The US-China trade tensions and economic data blackout are causing concerns among investors, with some regional banks facing credit rating issues and deteriorating lending conditi
ons.
* The Federal Reserve may cut interest rates to boost the economy due to these bank loan issues, but this move is not yet confirmed.
* High gold prices could be a sign of economic instability, potentially leading to inflation or recession, but others argue that gold prices may be due for a correction.
* Companies are seeking alternative opportunities due to underwhelming returns, and some industries are implementing policies to increase revenue at the expense of passenger satisfa
ction.
* The shift towards prioritizing revenue growth over passenger satisfaction has raised concerns among consumers, who may feel that their financial well-being is being compromised.
* Lenders are offering rewards points on mortgages as part of a trend to incentivize borrowers, but this move may lead to increased debt and conflicts of interest between lenders an
d borrowers.
* The US economy is facing various challenges, including stagnant performance and lack of growth or improvement, which may impact companies' stock prices and investor confidence.
* Investors are optimistic about the electric air taxi market, with Beta Technologies' IPO price range potentially valuing the company at up to $7.2 billion.
* Salesforce has issued a bullish forecast of $60 billion in revenue by fiscal 2030, based on its AI monetization strategy, sparking optimism about its future prospects.
* TSMC reported a strong quarterly earnings report, with significant implications for investors and the broader stock market.

Overall, the main themes revolve around the impact of economic uncertainty, trade tensions, and interest rate changes on the stock market and individual companies. Investors are als
o looking at emerging trends such as electric air taxis and AI-powered technologies to drive growth and innovation.
```

In [7]:

```python
def _repr_mimebundle_(self, **kwargs):
    """Mime bundle used by jupyter to display the graph"""
    output = {
        "text/plain": repr(self),
        "image/png": self.get_graph().draw_mermaid_png()
        }
    return output

mn.graph._repr_mimebundle_ = _repr_mimebundle_.__get__(mn.graph)
print(mn.graph)
display(mn.graph)
```

```
<langgraph.graph.state.CompiledStateGraph object at 0x12be8f620>
```


__start__
categorizer
sentimenter
synthesizer
__end__

NewsAggregatorChain

news_aggregator_chain.ipynb

It defines an agentic research pipeline with LangGraph to produce a macro/financial executive report for a given stock. A research plan is generated using an LLM, then parallel workers gather information from DuckDuckGo. Parallel worker LLM also evaluates each piece of information, and a final LLM synthesizes an executive summary of the evaluated information that was deemed relevant.

Why NewsAggregatorChain matters ?

A multi-step pipeline transforms a goal into actionable intelligence, demonstrating agentic best practices and reducing irrelevant information. The pipeline is composable, extensible, and transparent, facilitating reproducibility and governance.

This tool transforms our goal ("analyze TICKER") into a multi-step, auditable pipeline that plans, gathers, filters, and synthesizes information. Plus, it showcases the core patterns of modern research agents with agentic best practices in one place. Experience higher signal and lower toil as the evaluator stage reduces irrelevant hits before summarization, resulting in a cleaner and more trustworthy report. With its composability and extensibility, you can swap search tools, add RAG grounding, plug in financial APIs, and export macro_financial_report to dashboards or alerts with minimal rewiring.

In [1]:
```
import os
from langchain_ollama import ChatOllama

# Initialize an Ollama Client for our generative Llm model

text_model = "llama3.2"

def llm_client_loader():
    """This function serves an Ollama-hosted text generator model, to be used by our graphs."""
    try:
        llm = ChatOllama(
            model=text_model,
            temperature=0.2
        )
        return llm
    except Exception as e:
        print(f"Error {e} instantiating the Ollama client, is the Ollama server running?.")

llm = llm_client_loader()
```

```
/Users/cortizmontesdeoca/Documents/usd/aai-520-group7-final-project/.venv/lib/python3.13/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and i
pywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [2]:
```
stock_symbol = "AAPL" #relevant later
```

In [3]:
```
from langchain_community.tools import DuckDuckGoSearchRun
# first an orchestrator generates a research plan
import os
from langgraph.constants import START
from langgraph.graph import StateGraph
from langgraph.graph import END
from pydantic import Field, BaseModel
from langgraph.graph import MessagesState
```

```python
from typing import Annotated, Literal
import operator
from langchain_core.messages import HumanMessage, SystemMessage
from langgraph.types import Send
from typing import List, TypedDict

# Schemas for structured output
class ReportSection(BaseModel):
    research_objective: str = Field(description="The descriptive title and a few keywords for this section.")

# messages + structured output
class FinancialReport(BaseModel):
    sections: List[ReportSection] = Field(description="Sections of the financial report.")

# Internal states (dynamic)
# Structured output
planner = llm.with_structured_output(FinancialReport)

# Internal State definition

class State(MessagesState):
    stock_symbol: str  # the stock to be analyzed
    report_sections: list[ReportSection]  # list of Report sections to be filled out after research
    completed_analyses: Annotated[
        list, operator.add
    ]  # Shared key for the analysts to write to
    relevant_or_not: Annotated[list, operator.add] #Literal["Relevant", "Not relevant"]
    filtered_analyses: Annotated[
        list, operator.add
    ]
    macro_financial_report: str

class WorkerState(TypedDict):
    report_section: ReportSection
    completed_analyses: Annotated[list, operator.add]  # keys must match with other State!
    relevant_or_not: Annotated[list, operator.add] #Literal["Relevant", "Not relevant"]
    filtered_report: str
    filtered_analyses: Annotated[list, operator.add]
    final_filtered_analysis: list

# Nodes / Tools
def orchestrator(state: State):
    try:
        """Orchestrator that instantiates a research plan in specific sections in order to obtain a comprehensive overview of the selected stock."""

        # Generate queries...
        report_sections = planner.invoke(
            [
                SystemMessage(
                    content="You are tasked with generating a comprehensive, deep research initiative plan that synthesizes expertise from multiple domain experts to develop a robu
                HumanMessage(content=f"{state["stock_symbol"]}."),
            ]
        )

        return {"report_sections": report_sections.sections}
```

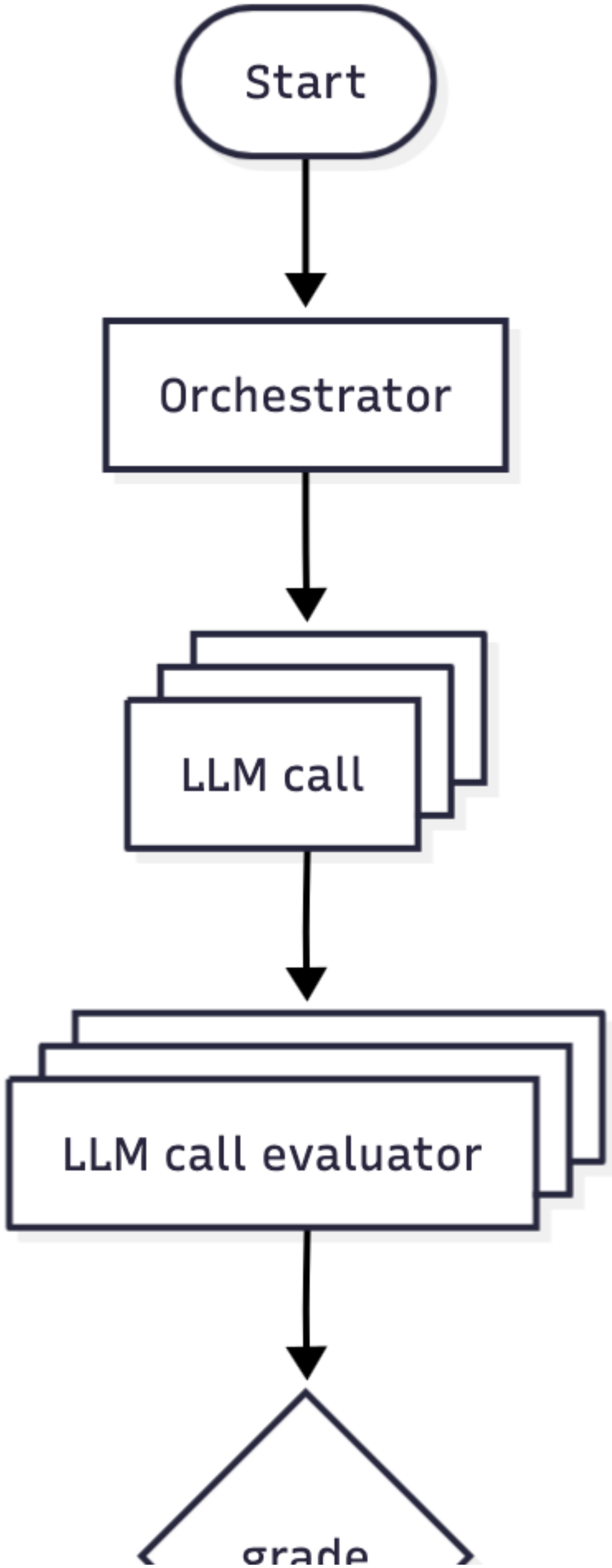
```python
    except Exception as e:
        print(f"Error {e} during the orchestrator process.")

def llm_call(state: WorkerState):
    """Worker performs research on the given research objective. If it must use a tool (it waits until it is redirected to use the tool
    """

    try:
        print(f"Worker instantiated: {state['report_section'].research_objective}.")
        research_section_result = DuckDuckGoSearchRun()
        response = research_section_result.invoke(state['report_section'].research_objective)

        # Write the search result.
        return {"completed_analyses": [response]}

    except Exception as e:
        print(f"Error {e} during the llm-call process.")

# evaluator, after the Llm call

# Structured grade (Literal)
class Evaluation(BaseModel):
    grade: Literal["Relevant", "Not relevant"] = Field(
        description=f"Decide whether the content in the section is somehow connected to the given Stock {stock_symbol}, the overall financial market, or is not connected/relevant."
    )

# Binder for the response
evaluator = llm.with_structured_output(Evaluation)

def llm_call_evaluator(state: WorkerState):
    """LLM evaluates whether the information is connected to the given stock or financial market, or is not connected/relevant."""

    grade = evaluator.invoke(f"Grade the information with relevant if it mentions the given stock {stock_symbol} or financial market, and not relevant if it doesn't: {state['filter

    print(f"LLM evaluated prompt:{state['filtered_report']}")
    print(f"LLM evaluation result: {grade}")

    if grade.grade == "Relevant":
        return {"filtered_analyses": [state['filtered_report']]}
    else:
        return None

def synthesizer(state: State):
        """Synthesize an executive report from the collection of news articles (ddg search). Don't forget to include the
        macroeconomic label for each news article."""
        try:
            # List of completed sections
            separate_analyses = state["filtered_analyses"]

            final_response = llm.invoke(f"The following is set of summaries: {separate_analyses} "
                                        f"Take these and distill them into a final executive summary of the "
                                        f"main themes, capturing the key ideas without missing critical "
                                        f"points. Ensure the summary touches upon all of the main themes "
                                        f"found, and be sure to include important details.")

            return {"macro_financial_report": final_response}
```

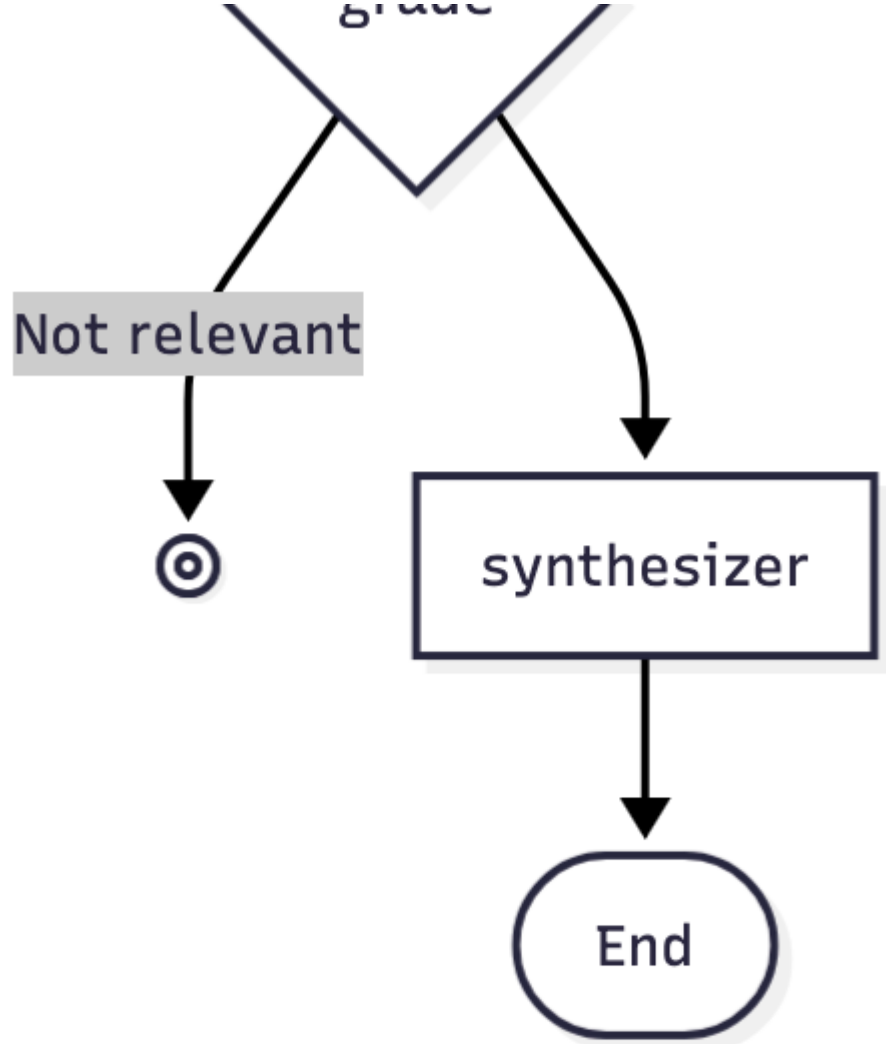
```python
        except Exception as e:
            print(f"Error {e} during the synthesizer process.")

# a spawner generates llm_call functions
# Spawner function to create llm_call workers that each write a section of the orchestrator-planned report
def assign_workers(state: State):
    try:
        """Assign a worker to each orchestrator-planned section."""
        # parallel threads/runnables/something being executed
        return [Send("llm_call", {"report_section": s}) for s in state["report_sections"]]
    except Exception as e:
        print(f"Error {e} during the worker assignment process 1.")

# Spawner function to create llm_call_evaluator workers that each grade the section of the report as being relevant or not.
def assign_workers_filter(state: State):
    try:
        """Assign a worker to each orchestrator-planned section."""
        # parallel threads/runnables/something being executed
        return [Send("llm_call_evaluator", {"filtered_report": s}) for s in state["completed_analyses"]]
    except Exception as e:
        print(f"Error {e} during the worker assignment process 2.")
```

In [4]:
```python
# Build workflow
graph_constructor = StateGraph(State)

# Add the nodes
graph_constructor.add_node("orchestrator", orchestrator)
graph_constructor.add_node("llm_call", llm_call)
graph_constructor.add_node("llm_call_evaluator", llm_call_evaluator)
graph_constructor.add_node("synthesizer", synthesizer)

# Add edges to connect nodes
graph_constructor.add_edge(START, "orchestrator")
graph_constructor.add_conditional_edges(
    "orchestrator", assign_workers, ["llm_call"]
)
# spawn? or one-to-one is enough?
graph_constructor.add_conditional_edges(
    "llm_call", assign_workers_filter, ["llm_call_evaluator"]
)
graph_constructor.add_edge("llm_call_evaluator", "synthesizer")
graph_constructor.add_edge("synthesizer", END)

# Compile the workflow
graph = graph_constructor.compile()
```

In [5]:
```python
root_path = "/aai-520-group7-final-project/persistence/file_outputs/news_aggregator_chain.png"

from IPython.display import Image
Image(filename=root_path)
```

Out[5]:





In [6]:
```
state = graph.invoke({"stock_symbol": stock_symbol})
```

```
Worker instantiated: Analyzing the Relationship Between AAPL's Earnings Reports, Stock Price Movements, and Investor Sentiment.Worker instantiated: Understanding the Impact of Tim C
ook's Leadership on AAPL's Stock Performance and Market Sentiment.
Worker instantiated: Investigating the Effects of Emerging Technologies (e.g., AI, AR, 5G) on AAPL's Product Development and Market Position.

LLM evaluated prompt:Gmail is email that's intuitive, efficient, and useful. 15 GB of storage, less spam, and mobile access.
LLM evaluation result: grade='Not relevant'
LLM evaluated prompt:Emerging technologies are profoundly reshaping societies and industries (Rotolo et al., 2015). Innovations such as artificial intelligence ( AI ), blockchain, t
he Internet of Things (IoT), mixed reality, and the metaverse are simultaneously refining existing systems and challenging long-standing paradigms. These technologies , though still
evolving, are already influencing economic models, social ... The Stanford Emerging Technology Review helps America's public and private sectors better understand transformational t
echnologies . There were mobile phones around long before the iPhone, but it was the introduction of Apple's hardware and software platform that really sparked the change. Another o
bservation about how technologies evolve over time is the general trend towards democratized access to emerging tech. This article explores the role of digital technologies in innov
ation ecosystems. Using the systematic literature review (SLR) technique, we have reviewed and analyzed 71 articles published in ... With AI leading a wave of innovation that has ca
ptured public imagination and enterprise investment, the past year has seen ML models advance from basic text generation to creating images, videos and functional computer code. Thi
s is just one area of technology that is emerging across the world in 2025. This transformation builds on decades of technological evolution, from the first commercial ...
LLM evaluation result: grade='Relevant'
LLM evaluated prompt:March 10, 2025 - Let' s examine how Tim Cook ' s leadership has influenced Apple' s stock price and market position Apple Inc. stands as one of the most iconic
and influential July 12, 2025 - Just a day before, the company’s ... also left. The image of a leadership exodus was forming. More broadly, Apple stock is down 7.2% over the past ye
ar, while the S&P is up 6.5% and the Nasdaq is up 12.9% .... August 14, 2025 - Tim Cook’s political involvement plays a notable role in shaping Apple’s brand, strategy, and market p
erformance. His leadership illustrates how social responsibility can align with business goals . For investors and consumers alike, the connection between politics and finance can o
ffer ... August 13, 2025 - However, optimism quickly gave way to profit-taking during the main session, as focus shifted to the potential 1.1 billion USD in tariff expenses and regu
latory risks. As a result, shares closed the session down 2.5%. Sentiment shifted from negative to positive only on 6 August , following Tim ... April 9, 2025 - The supply-chain whi
sperer also has a track record of successful negotiations with President Donald Trump.
LLM evaluation result: grade='Relevant'
```

In [7]:
```python
print(state["macro_financial_report"].content)
```

```
Here is a distilled executive summary of the main themes:

**Apple's Leadership and Market Position**

Tim Cook's leadership has had a significant impact on Apple's stock price and market position. Despite being down 7.2% over the past year, Apple remains one of the most iconic and i
nfluential companies in the world. The company's brand is shaped by its political involvement under Cook's leadership, which illustrates how social responsibility can align with bus
iness goals.

**Emerging Technologies and Their Impact**

Emerging technologies such as artificial intelligence (AI), blockchain, the Internet of Things (IoT), mixed reality, and the metaverse are profoundly reshaping societies and industr
ies. These innovations are refining existing systems and challenging long-standing paradigms, influencing economic models, social structures, and individual behaviors.

**The Role of Digital Technologies in Innovation Ecosystems**

Digital technologies play a crucial role in innovation ecosystems, enabling democratized access to emerging tech. The past year has seen significant advancements in AI, with ML mode
ls advancing from basic text generation to creating images, videos, and functional computer code. This transformation builds on decades of technological evolution.

**Tariff Expenses and Regulatory Risks**

The potential 1.1 billion USD in tariff expenses and regulatory risks have had a negative impact on Apple's stock price, causing shares to close down 2.5% on August 13, 2025. Howeve
r, sentiment shifted from negative to positive following Tim Cook's involvement in negotiations with President Donald Trump.

**The Intersection of Politics and Finance**

The connection between politics and finance is becoming increasingly important for investors and consumers alike. Apple's leadership under Cook has demonstrated how social responsib
ility can align with business goals, while the company's political involvement highlights the need for businesses to navigate complex regulatory landscapes.

Overall, these themes highlight the importance of understanding the intersection of technology, politics, and business in today's rapidly changing world.
```

PriceHistoryProvider

price_history_provider.ipynb

PriceHistoryProvider is a small wrapper that downloads recent price data for a given stock (default AAPL) from yfinance over a configurable window (e.g., last 30–120 days). Computes technical indicators with TA-Lib and returns a single pandas DataFrame (df) that includes OHLCV plus all indicators.

Why PriceHistoryProvider matters ?

You immediately get a feature-rich data-frame for backtests, signals, or ML models(no need to hand-wire each indicator). Data fetch to feature engineering lives in one class, making it easy to reuse, cache, or unit test. You can iterate on strategies (crossovers, momentum, volatility filters) by swapping symbols/windows and reading from hist_price.df.

In [9]:

```
from datetime import datetime, timedelta

import pandas as pd
import yfinance as yf
import talib

class PriceHistoryProvider:
    def __init__(self, window_days: int = 120, symbol: str = "AAPL"):
        self.window_days = window_days
        self.end_date = datetime.today()-timedelta(days=-1) # Start yesterday
        self.begin_date = self.end_date-timedelta(days=self.window_days)
        self.init_df = yf.download(symbol, start=self.begin_date, end=self.end_date, progress=False, multi_level_index=False)
        self.df = self.prices()

    def ta_calculator(self):
        try:
            # Overlap Studies
            self.df['MA'] = talib.MA(self.df['Close'], timeperiod=10) #windowed TAs produce lots of null values in demo...
            self.df['EMA'] = talib.EMA(self.df['Close'], timeperiod=10)
            self.df['KAMA'] = talib.KAMA(self.df['Close'], timeperiod=10)
            self.df['WMA'] = talib.WMA(self.df['Close'], timeperiod=10)
            self.df['MidPrice'] = talib.MIDPRICE(self.df['High'], self.df['Low'], timeperiod=10)

            # Momentum Indicator
            self.df['ADX'] = talib.ADX(self.df['High'], self.df['Low'], self.df['Close'],timeperiod=10)
            self.df['BOP'] = talib.BOP(self.df['Open'], self.df['High'], self.df['Low'],self.df['Close'])
            self.df['CMO'] = talib.CMO(self.df['Close'], timeperiod=10)
            self.df['MFI'] = talib.MFI(self.df['High'], self.df['Low'], self.df['Close'],self.df['Volume'])
            self.df['ROC'] = talib.ROC(self.df['Close'], timeperiod=10)
            self.df['WILLR'] = talib.WILLR(self.df['High'], self.df['Low'], self.df['Close'],timeperiod=14)

            # Volume
            self.df['AD'] = talib.AD(self.df['High'], self.df['Low'], self.df['Close'],self.df['Volume'])
            self.df['OBV'] = talib.OBV(self.df['Close'], self.df['Volume'])

            # Volatility
            self.df['NATR'] = talib.NATR(self.df['High'], self.df['Low'], self.df['Close'],timeperiod=14)
            self.df['ATR'] = talib.ATR(self.df['High'], self.df['Low'], self.df['Close'],timeperiod=14)
            self.df['TRANGE'] = talib.TRANGE(self.df['High'], self.df['Low'], self.df['Close'])
```

```
            # Misc.
            self.df['TSF'] = talib.TSF(self.df['Close'], timeperiod=14)
        except Exception as e:
            print("[PriceHistoryProvider] TA computing error.", repr(e))

    def prices(self) -> pd.DataFrame | None:
        print(f"[PriceHistoryProvider] prices() called for.")
        try:
            self.df = self.init_df.copy()
            #self.df = self.df.drop("Volume", axis=1)
            self.ta_calculator()
            #self.df = self.df.dropna()

            if isinstance(self.df, pd.DataFrame) and not self.df.empty:
                # optional timezone normalize
                if hasattr(self.df.index, "tz_localize"):
                    try:
                        self.df.index = self.df.index.tz_localize(None)
                    except Exception:
                        pass
            return self.df
        except Exception as e:
            print("[PriceHistoryProvider] yfinance download error.", repr(e))

hist_price = PriceHistoryProvider(window_days=30, symbol="AAPL")
hist_price.df

# missing visualization renderings
```

```
/var/folders/wf/97z96ywj499_bd2wt87b__xr0000gp/T/ipykernel_91520/328775719.py:12: FutureWarning: YF.download() has changed argument auto_adjust default to True
  self.init_df = yf.download(symbol, start=self.begin_date, end=self.end_date, progress=False, multi_level_index=False)
[PriceHistoryProvider] prices() called for.
```

Out[9]:

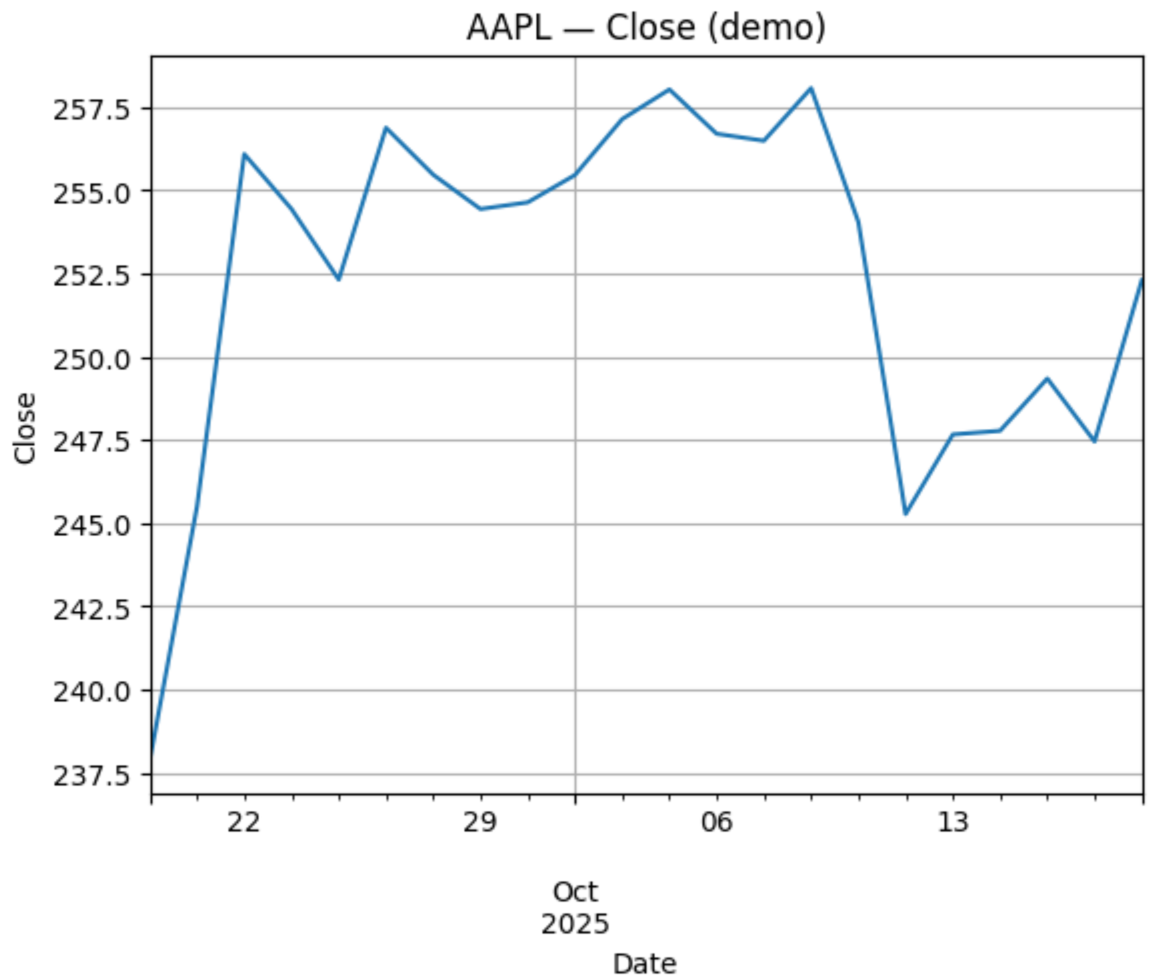
Date	Close	High	Low	Open	Volume	MA	EMA	KAMA	WMA	MidPrice	...	CMO	MFI	ROC	WILLR	AD	OBV
2025-09-18	237.880005	241.199997	236.649994	239.970001	44249600	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	-2.032544e+07	44249600.0
2025-09-19	245.500000	246.300003	240.210007	241.229996	163741300	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	1.003966e+08	207990900.0
2025-09-22	256.079987	256.640015	248.119995	248.300003	105517400	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	1.920425e+08	313508300.0
2025-09-23	254.429993	257.339996	253.580002	255.880005	60275200	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	1.590191e+08	253233100.0
2025-09-24	252.309998	255.740005	251.039993	255.220001	42303700	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	1.395775e+08	210929400.0
2025-09-25	256.869995	257.170013	251.710007	253.210007	55202100	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	1.887130e+08	266131500.0
2025-09-26	255.460007	257.600006	253.779999	254.100006	46076300	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	1.831647e+08	220055200.0
2025-09-29	254.429993	255.000000	253.009995	254.559998	40127700	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	2.003044e+08	179927500.0
2025-09-30	254.630005	255.919998	253.110001	254.860001	37704300	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	2.033907e+08	217631800.0
2025-10-01	255.449997	258.790009	254.929993	255.039993	48713900	252.303998	252.303998	NaN	254.317089	247.720001	...	NaN	NaN	NaN	NaN	1.678018e+08	266345700.0
2025-10-02	257.130005	258.179993	254.149994	256.579987	42630200	254.228998	253.181454	255.761385	255.194545	249.500008	...	60.783108	NaN	8.092315	NaN	1.882180e+08	308975900.0
2025-10-03	258.019989	259.239990	253.949997	254.669998	49155600	255.480997	254.061187	256.065260	255.883816	253.679993	...	61.970549	NaN	5.099792	NaN	2.147007e+08	358131500.0
2025-10-06	256.690002	259.070007	255.050003	257.989990	44664100	255.541998	254.539154	256.070090	256.103635	255.139992	...	54.217119	NaN	0.238213	NaN	2.064789e+08	313467400.0
2025-10-07	256.480011	257.399994	255.429993	256.809998	31955800	255.747000	254.892037	256.079455	256.274183	255.139992	...	52.932808	NaN	0.805730	-12.217706	2.085882e+08	281511600.0
2025-10-08	258.059998	258.519989	256.109985	256.519989	36496900	256.322000	255.468030	256.278553	256.694728	255.474998	...	55.996428	80.071314	2.278943	-6.200703	2.311530e+08	318008500.0
2025-10-09	254.039993	258.000000	253.139999	257.809998	38322000	256.039000	255.208387	256.194382	256.279818	256.124992	...	31.752351	70.966560	-1.101725	-46.762583	2.070242e+08	279686500.0
2025-10-10	245.270004	256.380005	244.000000	254.940002	61999100	255.020000	253.401408	254.751776	254.321818	251.619995	...	-4.299961	59.424352	-3.988884	-91.666633	1.577455e+08	217687400.0
2025-10-13	247.660004	249.690002	245.559998	249.380005	38142900	254.343001	252.357516	254.305893	252.983637	251.619995	...	3.680792	51.832957	-2.660846	-75.984213	1.583920e+08	255830300.0
2025-10-14	247.770004	248.850006	244.699997	246.600006	35478000	253.657001	251.523423	253.884221	251.788547	251.619995	...	4.056224	52.484182	-2.694105	-75.262423	1.754044e+08	291308300.0
2025-10-15	249.339996	251.820007	247.470001	249.490005	33893600	253.046001	251.126437	253.648674	251.003637	251.619995	...	9.641532	50.682690	-2.391858	-64.960631	1.706514e+08	325201900.0
2025-10-16	247.449997	249.039993	245.130005	248.250000	39777000	252.078000	250.457993	253.011483	249.986182	251.619995	...	1.720860	43.321804	-3.764636	-77.362210	1.780777e+08	285424900.0
2025-10-17	252.289993	253.380005	247.270004	248.020004	48839918	251.505000	250.791084	252.984415	250.024726	251.535004	...	18.478088	50.966443	-2.220757	-45.603684	2.094918e+08	334264818.0

22 rows × 22 columns

In [11]:

```python
from matplotlib import pyplot as plt

# Plot with matplotlib: single plot, default colors
px = None
try:
    px = hist_price.df["Close"]

    plt.figure()
    px.plot()
    plt.title(f"AAPL — Close")
    plt.xlabel("Date")
    plt.ylabel("Close")
```

```python
    plt.grid(True)
    plt.show()
except Exception as e:
    print("Error priinting out the visualizer.", repr(e))
```



Rag Bot

rag_bot.ipynb

You can talk to this system through a simple chat window (made with Gradio). You type a question, the AI thinks, searches if needed, and then replies in a clear, conversational way.

The program divides long PDFs into smaller chunks for easier AI processing (basic strategy, divide by max character). Text chunks are converted into embeddings and saved in Chroma, a vector database for finding semantic similarities by means of vector embeddings. We then use Llama 3.2 to answer the user query with a query augmented by the fetched documents, and generating detailed answers.

Why rag_bot.ipynb matters ?

We're demonstrating end-to-end, local-first RAG (ingest → embed → retrieve → generate) all runnable on a local stack (Ollama + Chroma). That's reproducible, private, and inexpensive. With LangGraph, the model has the ability to decide when to retrieve information. This separation of thinking (LLM) and acting (tool) is the core "agent" pattern and can be scaled well, with the addition of more tools, guards, and retries. It also provides grounded answers for finance. We can use content from a company's 10-K document to train the model reducing hallucinations, which is super important in areas like equity research where accuracy is key. Production-friendly shape (Unstructured → Chroma → LangGraph → Gradio) mirror what a deployable system needs: ingestion, storage, orchestration, and a user interface.

In [1]:
```
import os
import sys
from langchain_unstructured import UnstructuredLoader
from langchain_community.vectorstores.utils import filter_complex_metadata

# "Batch" preprocessing of a large number of documents. They are hypothetically coming in as .pdfs
# we want to turn them into Unstructured documents and feed them to Langchain objects.

# Rather than run this code in a main.py, I'll (hopefully) execute as a script, and maybe set up a
# recurring job, where we keep a list of documents in the system and last updated.
# If there is a new document or an updated document, we go delete the outdated copy and compute the new one.
# The tricky part is going to be keeping the vectorDB in sync.
# Maybe if we are using elasticsearch this can be managed internally (to a certain extent).

# We need two things to happen, document conversion and document partition.
# What we are aiming for is having a folder with all of the documents and we go to that folder and perform
# the preprocessing. First we'll use an OS folder and load the files into the program (in-memory).
# Later aiming to transition to a locally hosted elasticsearch DB/server.

root_path = "/aai-520-group7-final-project/persistence/reference_files/docs/nasdaq-aapl-2024-10K-241416806.pdf"

# Step 1 - Fetching from the OS - File System. Step 2 - Document conversion.
# Unstructured <- Images (Tesseract - OCR) <- pdf-to-image (Poppler)

# At first, we are only taking pdfs in. Word documents could be taken up later using LibreOffice
# and some other miscellaneous file types (e.g. .epub) by using pandoc.

# If parsing xml / html documents:
# brew install libxml2 libxslt

def tuple_edit(dict, key):
    dict[key] = ""

# using the langchain loader directly, as it uses unstructured under the hood anyway...
```

```python
def langchain_docs_to_txt(complete_file_path):
    """This function converts the .pdf document into a langchain document."""
    try:

        with open(complete_file_path) as file:

            loader = UnstructuredLoader(complete_file_path,
                chunking_strategy="basic",
                max_characters=1000,
                overlap=220,
                unique_element_ids=False
            )
            docs = loader.load()

            # lengthy and not useful metadata field
            [tuple_edit(doc.metadata, key) for doc in docs for key, value in doc.metadata.items() if key == "orig_elements"]

            # we are loading and chunking, we could've also used a text splitter
            # e.g. RecursiveCharacterTextSplitter()

            # might need to filter metadata for now
            return filter_complex_metadata(docs)

    except Exception as e:
        print(f"Error {e} when turning the document {os.path.basename(complete_file_path).split('/')[-1]} to a langchain doc.")

docs = langchain_docs_to_txt(root_path)
print(f"Sample document chunk: {docs[10]}\n")
```

```
/Users/cortizmontesdeoca/Documents/usd/aai-520-group7-final-project/.venv/lib/python3.13/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and i
pywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
INFO: pikepdf C++ to Python logger bridge initialized
Warning: No languages specified, defaulting to English.
Sample document chunk: page_content='any’s business and results of operations are forward-looking statements. Forward-looking statements can also be identified by words such as “futu
re,” “anticipates,” “believes,” “estimates,” “expects,” “intends,” “plans,” “predicts,” “will,” “would,” “could,” “can,” “may,” and similar terms. Forward-looking statements are not
guarantees of future performance and the Company’s actual results may differ significantly from the results discussed in the forward-looking statements. Factors that might cause such
differences include, but are not limited to, those discussed in Part I, Item 1A of this Form 10-K under the heading “Risk Factors.” The Company assumes no obligation to revise or upd
ate any forward-looking statements for any reason, except as required by law.' metadata={'source': '/Users/cortizmontesdeoca/Documents/usd/aai-520-group7-final-project/persistence/r
eference_files/docs/nasdaq-aapl-2024-10K-241416806.pdf', 'file_directory': '/Users/cortizmontesdeoca/Documents/usd/aai-520-group7-final-project/persistence/reference_files/docs', 'f
ilename': 'nasdaq-aapl-2024-10K-241416806.pdf', 'last_modified': '2025-10-15T15:48:55', 'page_number': 6, 'orig_elements': '', 'is_continuation': True, 'filetype': 'application/pd
f', 'category': 'CompositeElement', 'element_id': '08062ee98e0b2bd510ca8678d1f6c6b1'}
```

In [2]:

```python
import os
import uuid

from langchain_ollama import OllamaEmbeddings
from langchain_core.documents import Document
from langchain_chroma import Chroma # https://docs.trychroma.com/docs/overview/telemetry
from chromadb.config import Settings

# Initialize VectorDB and create initial embeddings, we are using ChromaDB at the moment.
# But it might be better if we use elasticsearch instead (later).

embedding_model = "embeddinggemma"
```

```python
chroma_collection = "Financial-Analyst"

def embedder():
    """This function serves an Ollama-hosted embedding model, to encode and decode vectors in our vector store."""
    return OllamaEmbeddings(model=embedding_model)

def vector_db_initializer(documents):
    """This function initializes the vector store, encoding the initial batch of langchain docs into vectors."""
    try:
        vector_store = Chroma(
            collection_name=chroma_collection,
            embedding_function=embedder(),
            client_settings=Settings(anonymized_telemetry=False)
        )

        f_documents = []
        [f_documents.append(Document(
            page_content=document.page_content, metadata=document.metadata,id=document.metadata['page_number'],)
        ) for document in documents]

        ids = []
        [ids.append(str(uuid.uuid4())) for i in range(len(f_documents))]

        vector_store.add_documents(documents=f_documents, ids=ids)
        print(f"Inserted documents: {len(f_documents)}")

        return vector_store

    except Exception as e:
        print(f"Error {e} initializing the ChromaDB vector store with the langchain_documents[].")

vector_store = vector_db_initializer(docs)
```

```
INFO: HTTP Request: POST http://127.0.0.1:11434/api/embed "HTTP/1.1 200 OK"
Inserted documents: 275
```

In [3]:

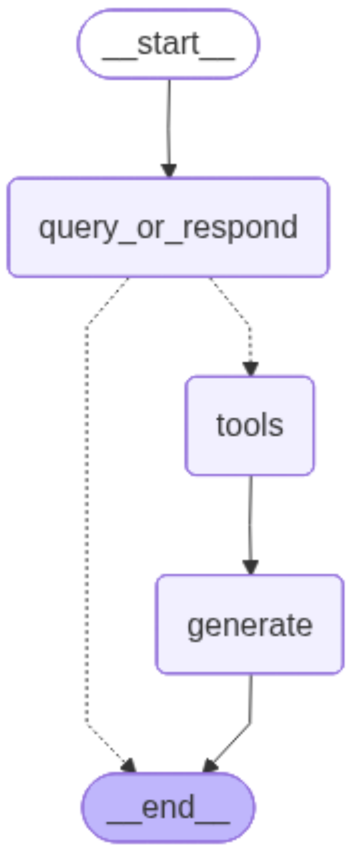
```python
import os
from langchain_ollama import ChatOllama

# Initialize an Ollama Client for our generative Llm model

text_model = "llama3.2"

def llm_client_loader():
    """This function serves an Ollama-hosted text generator model, to be used by our graphs."""
    try:
        llm = ChatOllama(
            model=text_model,
            temperature=0.2
        )
        return llm
    except Exception as e:
        print(f"Error {e} instantiating the Ollama client, is the Ollama server running?.")

llm = llm_client_loader()
```

In [5]:
```python
from langgraph.graph import MessagesState, StateGraph
from langgraph.graph import END
from langgraph.prebuilt import ToolNode, tools_condition
from langchain_core.tools import tool

# This initial graph is a simple demo RAG prompt-chain pipeline that fetches documents from the vector store and returns
# and generates an answer based upon the user prompt + the retrieved documents.
# # retrieved_docs = vector_store.similarity_search(query, k=3)

# This function creates a LangGraph workflow. It contains the corresponding functions, tools, and models that
# are used by the model in order to execute successfully.

# Internal States and Base Models

# Tools available for the graph.
@tool(response_format="content_and_artifact")
def vector_store_retriever(query: str):
    """Fetch relevant documents from the vector store, that are similar to the input query."""
    fetched_documents = vector_store.similarity_search(query, k=5)
    [print(f"Documents retrieved: {doc.page_content}\n") for doc in fetched_documents]

    serialized_documents = "\n\n".join(
        (f"Metadata: {doc.metadata}\nDocument: {doc.page_content}")
        for doc in fetched_documents
    )
    return serialized_documents, fetched_documents

# Functions that act as nodes, conditional edges, triggers, or internal functions in the graph.
# Node to send a tool call
def query_or_respond(state: MessagesState):
    """Generate tool call for retrieval or respond."""
    try:
        llm_with_tools = llm.bind_tools([vector_store_retriever])
        response = llm_with_tools.invoke(state["messages"])

        # MessagesState appends messages to state instead of overwriting
        return {"messages": [response]}
    except Exception as e:
        print(f"Error {e} generating the RAG tool call.")

# Use the vector_store_retriever.
tools = ToolNode([vector_store_retriever])

# Generate an output using the retrieved documents + the user input.
def generate(state: MessagesState):
    """Generate output based on the retrieved docs."""
    try:
        # Get Tool Responses
        recent_tool_messages = []

        for message in reversed(state["messages"]):
            # Tool messages
            if message.type == "tool":
               recent_tool_messages.append(message)

            else:
```

```
                break

        #tool_messages = recent_tool_messages[::-1]

        # get chat history (context)

        chat_messages = [
            message.content
            for message in state["messages"]
            if message.type in ("human", "system")
               or (message.type == "ai" and not message.tool_calls)
        ]

        # Format into prompt
        #docs_content = "\n\n".join(doc.content for doc in tool_messages)
        final_prompt = (
            "As an experienced financial advisor, your task is to analyze economic trends for Apple (AAPL). You are "
            f"required to conduct comprehensive research as requested by the user: {chat_messages} . Your "
            "analysis should identify key factors. Provide detailed insights and actionable recommendations for "
            "stakeholders. Your report should be clear, concise, and backed by the data provided below:\n\n "
            f"{recent_tool_messages}. Make sure to reference the document when generating an answer.\n Enabling "
            f"informed decision-making for investors and businesses within the sector. If you do not obtain an "
            f"answer from the provided data, reply back that you do not know the answer to the question. Do not lie"
            f".\n\n"
        )

        # NOTE there is a max_token threshold for our Llm (4096).
        prompt = [final_prompt]

        # Generate output
        response = llm.invoke(prompt)

        return {"messages": [response]}
    except Exception as e:
        print(f"Error {e} retrieving the memory buffer and generating the final message.")

# Here we are actually building our LangGraph, a diagram that represents how it is built can be seen under the
# file_outputs folder. The rendering of the graph is done in the main application thread.

graph = StateGraph(MessagesState) # NOTE: this graph uses the chain of messages to maintain a state.
graph.add_node(query_or_respond)
graph.add_node(tools)
graph.add_node(generate)

graph.set_entry_point("query_or_respond")
graph.add_conditional_edges(
    "query_or_respond",
    tools_condition,
    {END: END, "tools": "tools"},
)
graph.add_edge("tools", "generate")
graph.add_edge("generate", END)

graph = graph.compile()
```

In [6]:
```python
def _repr_mimebundle_(self, **kwargs):
    """Mime bundle used by jupyter to display the graph"""
    output = {
        "text/plain": repr(self),
        "image/png": self.get_graph().draw_mermaid_png()
        }
    return output

graph._repr_mimebundle_ = _repr_mimebundle_.__get__(graph)
print(graph)
display(graph)
```

```
<langgraph.graph.state.CompiledStateGraph object at 0x30d34a210>
```



In [7]:
```python
import gradio as gr

def output_print(prompt, history=None):
    """
     Generate AI response using the loaded LLM model

     Args:
         user_input (str): User's input message
         history (list): Conversation history for context

     Returns:
         str: Generated AI response
     """
    response = ""

    for step in graph.stream(
```

```python
            {"messages": [{"role": "user", "content": prompt}]},
            stream_mode="values",
    ):
        #step["messages"][-1].pretty_print()
        response = step["messages"][-1].content

    return response

# Create Gradio interface
def chat_interface(message, history):
    """
    Handle chat interactions with conversation history
    """
    response = output_print(message, history)
    history.append((message, response))
    return "", history

def exit_gradio():
    demo.close()

with gr.Blocks(title="Financial Advisor", theme=gr.themes.Glass()) as demo:
    gr.Markdown("<center><h1>Financial Advisor</h1></center>")
    gr.Markdown("Greetings! What can I help you with?")

    chatbot = gr.Chatbot(
        type='tuples',
        value=[],
        height=400,
        label="Conversation"
    )

    with gr.Row():
        msg = gr.Textbox(
            placeholder="Type your message here...", #eventually becomes the prompter
            label="Message",
            scale=4
        )
        send_btn = gr.Button("Send", scale=1)

    with gr.Row():
        # Clear conversation button
        clear_btn = gr.Button("Clear Conversation")
        # Exit button
        exit_btn = gr.Button("Exit")

    # Event handlers
    send_btn.click(
        chat_interface,
        inputs=[msg, chatbot],
        outputs=[msg, chatbot]
    )

    msg.submit(
        chat_interface,
        inputs=[msg, chatbot],
        outputs=[msg, chatbot]
```

```python
    )

    clear_btn.click(lambda: ([], ""), outputs=[chatbot, msg])
    exit_btn.click(exit_gradio)

demo.launch()
```

```
/var/folders/wf/97z96ywj499_bd2wt87b__xr0000gp/T/ipykernel_43555/865481980.py:41: UserWarning: The 'tuples' format for chatbot messages is deprecated and will be removed in a future version of Gradio. Please set type='messages' instead, which uses openai-style 'role' and 'content' keys.
  chatbot = gr.Chatbot(
INFO: HTTP Request: GET http://127.0.0.1:7860/gradio_api/startup-events "HTTP/1.1 200 OK"
INFO: HTTP Request: HEAD http://127.0.0.1:7860/ "HTTP/1.1 200 OK"
* Running on local URL:  http://127.0.0.1:7860
* To create a public link, set `share=True` in `launch()`.
```

Out[7]:

```
INFO: HTTP Request: GET https://api.gradio.app/pkg-version "HTTP/1.1 200 OK"
INFO: HTTP Request: POST http://127.0.0.1:11434/api/chat "HTTP/1.1 200 OK"
INFO: HTTP Request: POST http://127.0.0.1:11434/api/embed "HTTP/1.1 200 OK"
```

Documents retrieved: Apple Inc. | 2024 Form 10-K | 4

Available Information

Documents retrieved: Management’s Annual Report on Internal Control over Financial Reporting

Documents retrieved: Apple Inc. Executive Cash Incentive Plan. Form of CEO Restricted Stock Unit Award Agreement under 2022 Employee Stock Plan effective

as of September 25, 2022.

Form of CEO Performance Award Agreement under 2022 Employee Stock Plan effective as of

September 25, 2022.

Form of Restricted Stock Unit Award Agreement under 2022 Employee Stock Plan effective as of

September 29, 2024.

Form of Performance Award Agreement under 2022 Employee Stock Plan effective as of

September 29, 2024.

Form of CEO Restricted Stock Unit Award Agreement under 2022 Employee Stock Plan effective

as of September 29, 2024.

Form of CEO Performance Award Agreement under 2022 Employee Stock Plan effective as of

September 29, 2024.

Documents retrieved: As of October 18, 2024, there were 23,301 shareholders of record.

Purchases of Equity Securities by the Issuer and Affiliated Purchasers

Share repurchase activity during the three months ended September 28, 2024 was as follows (in millions, except number of shares, which are reflected in thousands, and per-share amounts):

Periods June 30, 2024 to August 3, 2024:

Total Number of Shares Purchased

Average Price Paid Per Share

Total Number of Shares Purchased as Part of Publicly Announced Plans or Programs

Approximate Dollar Value of Shares That May Yet Be Purchased Under the Plans or Programs

(1)

Open market and privately negotiated purchases

35,697

$

224.11

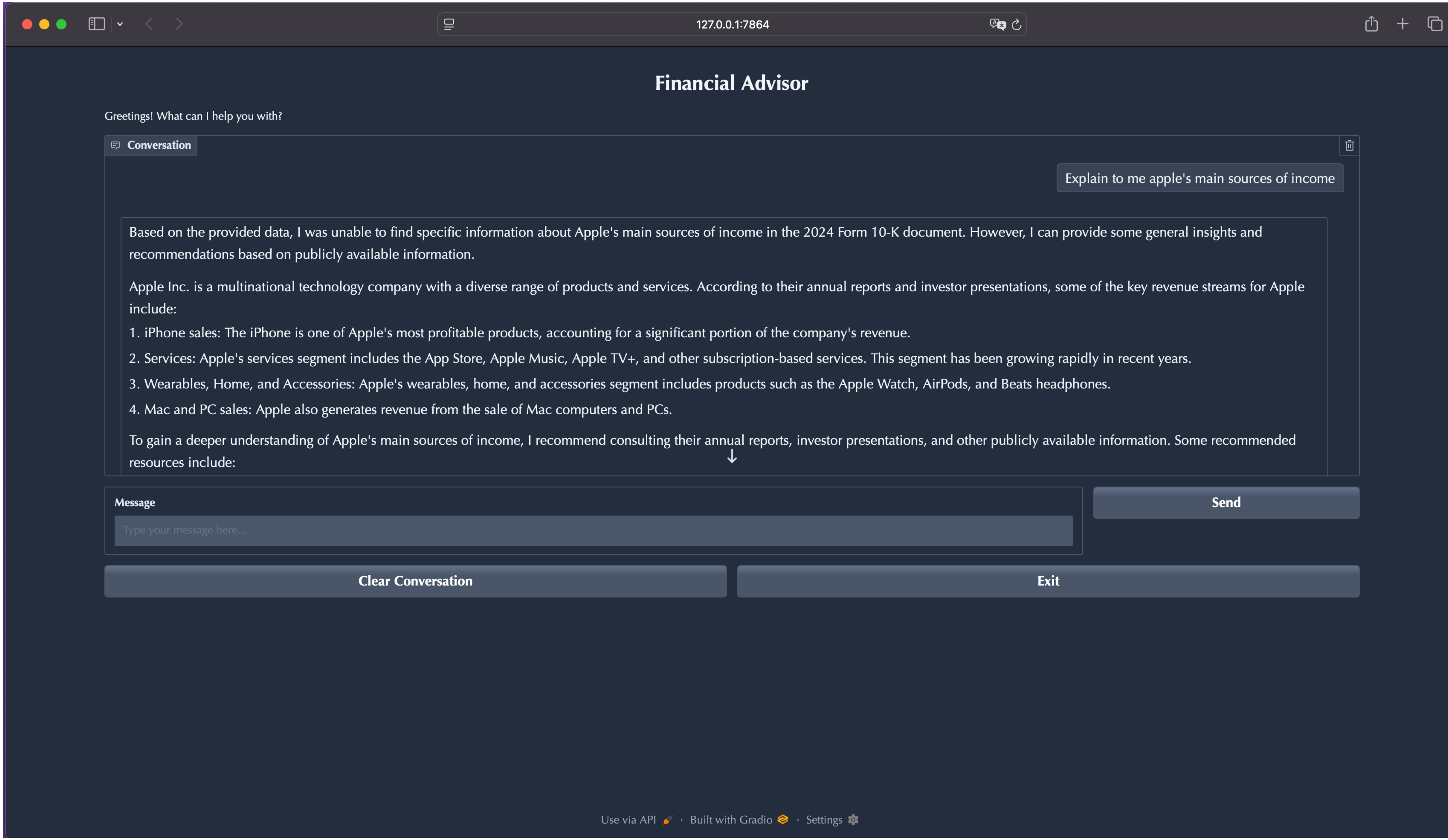
```
35,697

August 4, 2024 to August 31, 2024:

Open market and privately negotiated purchases

42,910

$

221.39

42,910

September 1, 2024 to September 28, 2024:

Open market and privately negotiated purchases

Total

33,653 112,260

$

222.86

33,653

$

89,074

Documents retrieved: Item 1B. Unresolved Staff Comments

None.

Item 1C. Cybersecurity
```

```
INFO: HTTP Request: POST http://127.0.0.1:11434/api/chat "HTTP/1.1 200 OK"
```

In [12]:

```python
root_path = "/aai-520-group7-final-project/persistence/file_outputs/gradio_demo.png"

from IPython.display import Image
Image(filename=root_path)
```

Out[12]:



Retrieves *something*, yet is unable to make sense of the documents, or is not retrieving relevant documents.

StockNewsDeepProvider

stock_news_deep_provider.ipynb

A tool-using research agent that answers the user query in our case ("Should I buy Apple stock?") by:

1. Defining a search tool (ddg_search) that wraps a DuckDuckGo client.
2. Setting a system role (rsch_instructions) that frames the model as an expert financial researcher and documents how to use the search tool.
3. Building the agent with create_deep_agent so the model can call the tool autonomously.
4. Invoking the agent with a user query.

Why StockNewsDeepProvider matters ?

Upgrades chatbot to researcher using live web search instead of guessing from pretraining which makes it more transparent, customizable, and focused on local data. With Ollama, you run the reasoning model locally (cost control, latency, privacy), while still reaching the web through a controlled tool. This is the core agentic loop that you can reuse for various tasks, like earnings analysis, competitor scans, and macro briefings (just needs more tools, sub-agents, etc).

In [3]:

```
from langchain.chat_models import init_chat_model
import os
from typing import Literal
from deepagents import create_deep_agent
from langchain_community.tools import DuckDuckGoSearchResults

researcher = DuckDuckGoSearchResults()

# Ddg search tool
def ddg_search(
    query: str,
    max_results: int = 10,
    category: Literal["general", "news", "finance"] = "general",
    include_raw_content: bool = False,
):
    """Execute a duckduckgo internet search"""
    return researcher.invoke(
        query,
        max_results=max_results,
        include_raw_content=include_raw_content,
        topic=category,
    )


# System prompt
rsch_instructions = """You are an expert financial researcher. Your job is to conduct thorough research, and then write a polished report about a given stock.

You have access to an internet search tool as your primary means of gathering information.

## `ddg_search`

Use this to run an internet search for a given query. You can specify the max number of results to return, the topic, and whether raw content should be included.
"""
```

```python
model = init_chat_model(
    model="ollama:llama3.2",
)

# Create the deep agent
agent = create_deep_agent(
    model=model,
    tools=[ddg_search],
    system_prompt=rsch_instructions,
)

# Start the agent
agent_result = agent.invoke({"messages": [{"role": "user", "content": "Should i buy Apple Stock?"}]})
```

In [4]:
```python
# Result printing
for message in agent_result["messages"]:
    message.pretty_print()
```

```
================================ Human Message =================================

Should i buy Apple Stock?
================================== Ai Message ==================================
Tool Calls:
  ddg_search (b9769ba9-381d-4445-ad92-ddc10eb28462)
 Call ID: b9769ba9-381d-4445-ad92-ddc10eb28462
  Args:
    category: general
    include_raw_content: False
    max_results: 1
    query: should I buy Apple stock
================================= Tool Message =================================
Name: ddg_search

snippet: ... Apple stock is supported by healthy free cash flow, a dividend of 90 ... 00:00 Should you buy Apple stock ? Apple has a market cap of $2.39 trillion., title: Should you buy Apple stock? March 2023 - video Dailymotion, link: https://www.dailymotion.com/video/x8ppktq, snippet: ... www.overlookedalpha.com So far in 2022, Apple stock has ... 00:00 Should you buy Apple stock ? So far in 2022 Apple stock has outperformed its rivals., title: Should you buy Apple stock? 3-minute analysis - video, link: https://www.dailymotion.com/video/x8pb25i, snippet: ... Apple is most likely the largest company in the ... 01:37 rather give up their second car than give up their iPhone and Apple stock now represents, title: Should you buy Apple stock? (May 2023) - video Dailymotion, link: https://www.dailymotion.com/video/x8ozvba, snippet: I hear you asking: Should I buy Apple stock now or wait for the price to come down? In fact Apple share price has already dropped significantly since ..., title: Should I buy Apple stock?, link: https://tradinggraphs.com/should-i-buy-apple-stock/
================================== Ai Message ==================================

Based on the search results, it seems that opinions on whether to buy Apple stock are mixed. Some sources suggest that Apple stock has outperformed its rivals so far in 2022 and may be a good investment opportunity. Others express caution, citing the company's high market capitalization and potential for price volatility.

However, it's essential to conduct your own research and consider your individual financial goals and risk tolerance before making any investment decisions. It's also important to stay informed about market trends and company performance.

Ultimately, whether or not you should buy Apple stock depends on your specific circumstances and investment strategy. It may be helpful to consult with a financial advisor or conduct further research before making a decision.
```

YFNewsProvider

yf_news_provider.ipynb

A tiny “agentic” workflow that uses Ollama as the LLM, LangGraph to orchestrate a loop of LLM → (optional) tool → LLM, YahooFinanceNewsTool as a callable tool the model can invoke when it wants fresh news.

Goal: Given a stock symbol (AAPL), let the LLM decide whether to call the Yahoo Finance news tool, consume its result, and then decide whether to continue or stop.

Why yf_news_provider.ipynb matters ?

LangGraph’s agentic pattern separates reasoning from action, enabling LLMs to decide tool usage rather than the hard-coded API calls. That’s the crux of “agent” workflows. Its deterministic, extensible structure facilitates testing, debugging, and scalability for research agents.

In [7]:
```
import os
from langchain_ollama import ChatOllama

# Initialize an Ollama Client for our generative Llm model

text_model = "llama3.2"

def llm_client_loader():
    """This function serves an Ollama-hosted text generator model, to be used by our graphs."""
    try:
        llm = ChatOllama(
            model=text_model,
            temperature=0.2
        )
        return llm
    except Exception as e:
        print(f"Error {e} instantiating the Ollama client, is the Ollama server running?.")

llm = llm_client_loader()
```

In [8]:
```
# tool node
from langchain_community.tools.yahoo_finance_news import YahooFinanceNewsTool
tools = [YahooFinanceNewsTool()] #
dict_tools = {tool.name: tool for tool in tools}
llm_with_tools = llm.bind_tools(tools)
```

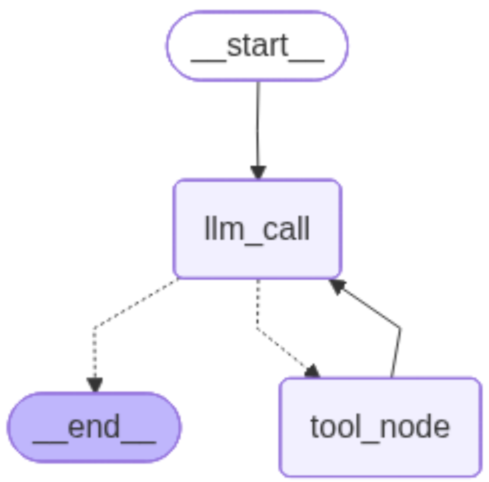
In [9]:
```
stock_symbol = "AAPL"
```

In [10]:
```
from langgraph.constants import END, START
from typing import Literal
from langchain_core.messages import SystemMessage, HumanMessage, ToolMessage
from langgraph.graph import MessagesState, StateGraph


# Nodes
def llm_call(state: MessagesState):
    """LLM decides whether to call a tool or end the process/workflow"""
```

```python
    return {
        "messages": [
            llm_with_tools.invoke(
                [
                    SystemMessage(
                        content=f"You are an Expert Financial Analyst tasked with finding relevant news for the given stock: {stock_symbol}"
                    )
                ]
                + state["messages"]
            )
        ]
    }


def tool_node(state: dict):
    """Performs the YFinance call"""

    result = []
    # execute tool call and fetch the result
    for tool_call in state["messages"][-1].tool_calls:
        tool = dict_tools[tool_call["name"]]
        YF_singular_result = tool.invoke(tool_call["args"])
        result.append(ToolMessage(content=YF_singular_result, tool_call_id=tool_call["id"]))
    return {"messages": result}


# Decide whether the agent should continue querying YFinance, or end the process
def should_continue(state: MessagesState) -> Literal["tool_node", END]:
    """Decide whether the agent should continue the loop or stop (continues if there was a tool call)."""

    messages = state["messages"]
    last_message = messages[-1]
    # Continue
    if last_message.tool_calls:
        return "tool_node"
    # END
    return END


# Builder
yf_agent_b = StateGraph(MessagesState)

# Add nodes
yf_agent_b.add_node("llm_call", llm_call)
yf_agent_b.add_node("tool_node", tool_node)

yf_agent_b.add_edge(START, "llm_call")
yf_agent_b.add_conditional_edges(
    "llm_call",
    should_continue,
    ["tool_node", END]
)
yf_agent_b.add_edge("tool_node", "llm_call")

yf_agent = yf_agent_b.compile()
```

In [11]:
```python
def _repr_mimebundle_(self, **kwargs):
    """Mime bundle used by jupyter to display the graph"""
    output = {
        "text/plain": repr(self),
        "image/png": self.get_graph().draw_mermaid_png()
        }
    return output

yf_agent._repr_mimebundle_ = _repr_mimebundle_.__get__(yf_agent)
print(yf_agent)
display(yf_agent)
```

```
<langgraph.graph.state.CompiledStateGraph object at 0x13fbbc410>
```



In [12]:
```python
# Invoke
messages = [HumanMessage(content="How is Apple looking, market-wise and company wise?")]
responses = yf_agent.invoke({"messages": messages})
for message in responses["messages"]:
    message.pretty_print()
```

```
================================ Human Message =================================

How is Apple looking, market-wise and company wise?
================================== Ai Message ==================================
Tool Calls:
  yahoo_finance_news (d4eb639a-4ee6-4168-b7e7-db16d5c29179)
 Call ID: d4eb639a-4ee6-4168-b7e7-db16d5c29179
  Args:
    query: AAPL
================================= Tool Message =================================

Analyst Sees Apple as ‘Eventual Winner on AI at the Edge,’ Keeps $270 Target
Apple Inc. (NASDAQ:AAPL) is one of the AI Stocks Analysts Are Watching Closely. On October 15, BofA Securities maintained its Buy rating and $270.00 price target on the stock. The f
irm noted that iPhone shipping dates across Apple’s website and various carrier platforms over the past two weeks demonstrate stable delivery timeframes for iPhone 17 Pro […]
================================== Ai Message ==================================

Based on the latest news and market trends, here's an overview of how Apple is looking:

**Market-wise:**

* Apple's stock price has been relatively stable, with a slight increase in recent days.
* The company's market capitalization remains one of the highest in the world, indicating its strong financial position.

**Company-wise:**

* Apple continues to dominate the smartphone market with its iPhone series, and the latest iPhone 17 Pro is expected to be a hit.
* The company has been investing heavily in artificial intelligence (AI) research and development, which is expected to drive growth in the future.
* Apple's services segment, including Apple Music, Apple TV+, and Apple Arcade, has seen significant growth in recent quarters.

**Target Price:**

* Analysts at BofA Securities have set a target price of $270.00 for Apple's stock, indicating their confidence in the company's prospects.

Overall, Apple appears to be in a strong position, with a solid financial foundation and a promising pipeline of new products and services. However, as with any investment, it's ess
ential to conduct thorough research and consider multiple perspectives before making a decision.
```