```python
import sys
import os
from typing import List
import yfinance as yf

from langchain.chains import SequentialChain
from langchain_openai import AzureChatOpenAI
from langchain.prompts import PromptTemplate
from langchain.output_parsers import PydanticOutputParser
from pydantic import BaseModel, Field
from langchain.tools import BaseTool

# --- Langfuse import and initialization ---
from langfuse import Langfuse
LANGFUSE_PUBLIC_KEY = os.environ["LANGFUSE_PUBLIC_KEY"]
LANGFUSE_SECRET_KEY = os.environ["LANGFUSE_SECRET_KEY"]
langfuse = Langfuse(
    public_key=LANGFUSE_PUBLIC_KEY,
    secret_key=LANGFUSE_SECRET_KEY,
)

def require_env(varname):
    value = os.environ.get(varname)
    if not value:
        print(f"Error: Set the environment variable {varname}")
        sys.exit(1)
    return value

AZURE_OPENAI_ENDPOINT = require_env("AZURE_OPENAI_ENDPOINT")
AZURE_OPENAI_API_KEY = require_env("AZURE_OPENAI_API_KEY")

class YahooFinanceTickerTool(BaseTool):
    name = "YahooFinanceTicker"
    description = "Get the ticker symbol for a company name using yfinance."
    def _run(self, company_name: str):
        ticker = yf.Ticker(company_name)
        info = ticker.info
        symbol = info.get("symbol", None)
        return symbol if symbol else company_name.upper()
    def _arun(self, company_name: str):
        raise NotImplementedError("Async not supported")

class YahooFinanceNewsTool(BaseTool):
    name = "YahooFinanceNews"
    description = "Get recent news summaries for a ticker symbol using yfinance."
    def _run(self, ticker_symbol: str):
        ticker = yf.Ticker(ticker_symbol)
        news = ticker.news or []
        summaries = [item.get("summary") or item.get("title", "") for item in news[:5]]
        return "\n".join(summaries)
    def _arun(self, ticker_symbol: str):
        raise NotImplementedError("Async not supported")

class SentimentProfile(BaseModel):
    company_name: str
    stock_code: str
    newsdesc: str
    sentiment: str = Field(..., description="Positive/Negative/Neutral")
    people_names: List[str]
    places_names: List[str]
    other_companies_referred: List[str]
    related_industries: List[str]
    market_implications: str
    confidence_score: float

output_parser = PydanticOutputParser(pydantic_object=SentimentProfile)
output_format_instructions = output_parser.get_format_instructions()

llm = AzureChatOpenAI(
    openai_api_key=AZURE_OPENAI_API_KEY,
    azure_endpoint=AZURE_OPENAI_ENDPOINT,
    openai_api_version="2025-01-01-preview",
    model="gpt-4o-mini",
    temperature=0
)

ticker_tool = YahooFinanceTickerTool()
news_tool = YahooFinanceNewsTool()

def get_stock_code(company_name, trace_id=None):
    span = langfuse.span(name="StockCodeExtraction", parent_id=trace_id)
    span.update(input={"company_name": company_name})
    stock_code = ticker_tool.run(company_name)
    span.update(output={"stock_code": stock_code})
    span.end()
    return stock_code

def get_news(ticker, trace_id=None):
    span = langfuse.span(name="NewsFetching", parent_id=trace_id)
    span.update(input={"ticker": ticker})
    news = news_tool.run(ticker)
    span.update(output={"news": news})
    span.end()
    return news

sentiment_prompt = PromptTemplate(
    input_variables=["company_name", "stock_code", "newsdesc", "format_instructions"],
    template="""
You are a financial analyst.

Given the following company name: {company_name}
Stock code: {stock_code}
News summaries: {newsdesc}

Classify overall market sentiment (Positive/Negative/Neutral), extract people/places/other companies/industries, market implications, and give a confidence score.
Respond ONLY in this JSON format:
{format_instructions}
"""
)

def get_sentiment(company_name, stock_code, newsdesc, trace_id=None):
    span = langfuse.span(name="SentimentParsing", parent_id=trace_id)
    prompt_text = sentiment_prompt.format(
        company_name=company_name,
        stock_code=stock_code,
        newsdesc=newsdesc,
        format_instructions=output_format_instructions
    )
    span.update(input={"prompt": prompt_text})
    result = llm.invoke([
        {"role": "system", "content": "You are a helpful financial analyst."},
        {"role": "user", "content": prompt_text}
    ])
```

```
    content = getattr(result, "content", None)
    if not content and hasattr(result, "choices"):
        content = result.choices[0].message.content
    span.update(output={"llm_response": content})
    span.end()
    return output_parser.parse(content)

def analyze_market_sentiment(company_name: str):
    trace = langfuse.trace(name="market_sentiment_analysis")
    trace_id = trace.id
    print("Step 1: Extracting stock code...")
    stock_code = get_stock_code(company_name, trace_id=trace_id)
    print("Step 2: Fetching news...")
    newsdesc = get_news(stock_code, trace_id=trace_id)
    print("Step 3: Analyzing sentiment...")
    result = get_sentiment(company_name, stock_code, newsdesc, trace_id=trace_id)
    # Ensure fields present
    if not result.company_name:
        result.company_name = company_name
    if not result.stock_code:
        result.stock_code = stock_code
    if not result.newsdesc:
        result.newsdesc = newsdesc
    print("Analysis complete! (See Langfuse for full trace)")
    return result

def main(company):
    result = analyze_market_sentiment(company)
    print(result.model_dump_json(indent=2))

if __name__ == "__main__":
    if len(sys.argv) == 2:
        main(sys.argv[1])
    else:
        for company in ["Microsoft", "Apple"]:
            print(f"\n=== Testing {company} ===")
            main(company)
```