Agents of Change

Building AI Agents that Work (and Think) for Us

Jeff Prosise

@jprosise

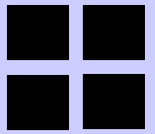
AI Agents

- New way of architecting AI solutions
  - "Microservices for AI"
- Build teams of autonomous agents, each designed to accomplish a single task extraordinarily well
- Use individual agents to solve problems or let them work as a team to solve more complex problems
  - You provide the building blocks
  - AI provides the workflow
- Frameworks provide the messaging infrastructure

Agentic Frameworks

Agents SDK

Python-first agentic framework from OpenAI. Supports **memory**, **RAG**, **audio/speech**, **tool use**, and more. Includes **Web Search**, **File Search**, and **Computer Use** tools, with Code Interpreter tool forthcoming.



AutoGen

Open-source framework from Microsoft for **multi-agent orchestration**. Supports **RAG**, diverse conversation patterns, integration with **popular LLMs**, and more. SDKs available for **Python** and **.NET**.

CrewAI

Open-source Python framework for "crews" of AI agents. **LLM-agnostic** with support for **RAG**, **tool use**, and more. Includes **dozens of built-in tools** and the **richest memory system** of all the frameworks.

Agno

Open-source Python framework that is **LLM-agnostic**. Includes a wide variety of built-in tools and supports **memory**, **persistence**, **RAG**, and more, and offers a playground for **building and testing workflows**.

Creating an Agno Agent

```python
from agno.agent import Agent
from agno.models.openai import OpenAIChat

agent = Agent(
    name='Sample Agent',
    description='You are a helpful agent who answers questions from users',
    model=OpenAIChat(id='gpt-4o-mini') # Use OpenAI's GPT-4o-mini model
)

response = agent.run('Why is the sky blue?')
print(response.content)
```

Streaming the Response

```python
from agno.agent import Agent
from agno.models.openai import OpenAIChat

agent = Agent(
    name='Sample Agent',
    description='You are a helpful agent who answers questions from users',
    model=OpenAIChat(id='gpt-4o-mini')
)

response = agent.run('Why is the sky blue?', stream=True)

for chunk in response:
    if chunk.event == 'RunContent':
        print(chunk.content, end='')
```

Enabling Chat History

```
from agno.agent import Agent
from agno.models.openai import OpenAIChat

agent = Agent(
    name='Sample Agent',
    description='You are a helpful agent who answers questions from users',
    model=OpenAIChat(id='gpt-4o-mini'),
    add_history_to_context=True,
    num_history_runs=12
)

response = agent.run('Why is the sky blue?')
```

Persisting Chat History in SQLite

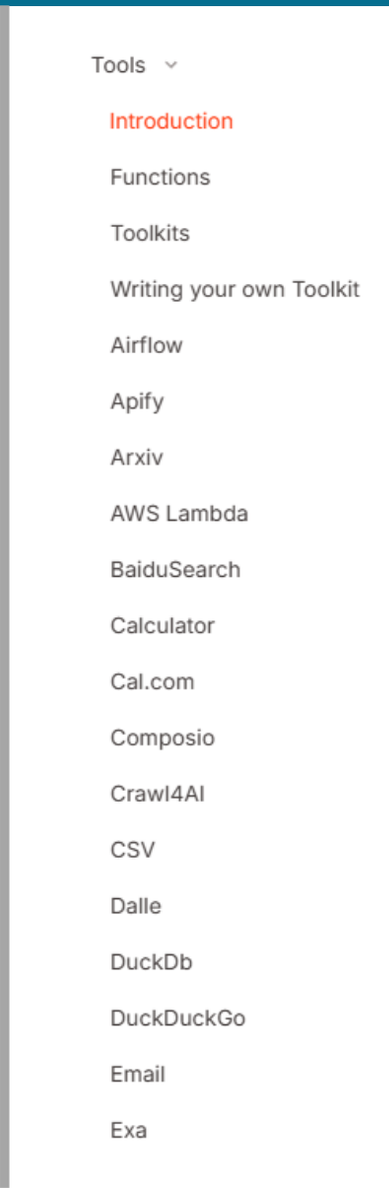
```
from agno.agent import Agent
from agno.models.openai import OpenAIChat
from agno.storage.sqlite import SqlLiteStorage

agent = Agent(
    name='Sample Agent',
    description='You are a helpful agent who answers questions from users',
    model=OpenAIChat(id='gpt-4o-mini'),
    add_history_to_context=True,
    num_history_runs=12,
    db=SqliteDb(db_file='data/sessions.db')
)

response = agent.run('Why is the sky blue?')
```

Using an Ollama Model

```
from agno.agent import Agent
from agno.models.ollama import Ollama

agent = Agent(
    name='Sample Agent',
    description='You are a helpful agent who answers questions from users',
    model=Ollama(id='llama-3.2:3b'), # Use the 3B-parameter version of Llama 3.2
)

response = agent.run('Why is the sky blue?')
```

Tools

- Agno comes with more than 50 toolkits containing hundreds of functions/tools
  - **DuckDuckGo**, **BaiduSearch**, **Arxiv**, and **Wikipedia** for search
  - **Google Calendar** and **Cal.com** for calendaring
  - **Python** for code generation and execution
  - **Slack**, **Twitter**, **Zoom**, and **GitHub** for app integration
  - **Yfinance** for financial data and many more
- Supports custom tools (custom functions) as well as custom toolkits (collections of related functions)

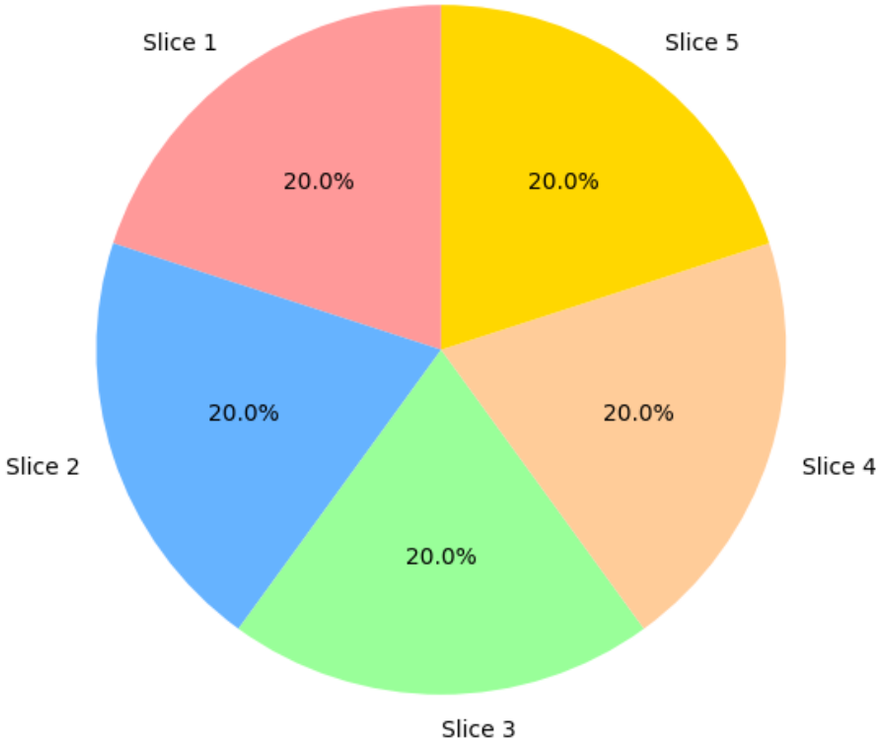


Using the DuckDuckGo Tool

```
from agno.tools.duckduckgo import DuckDuckGoTools

agent = Agent(
    name='Research Agent',
    description='You are a helpful agent who does research for users',
    model=OpenAIChat(id='gpt-4o-mini'),
    instructions='Always include links to sources',
    tools=[DuckDuckGoTools()]
)

response = agent.run('How do LLMs work?')
```

Using the Python Tool

```
from agno.tools.python import PythonTools

agent = Agent(
    name='Visualization Agent',
    description='You are a helpful data visualizer',
    model=OpenAIChat(id='gpt-4o-mini'),
    tools=[PythonTools()]
)

response = agent.run(
    'Generate a pie chart with five equal slices'
)
```



Writing a Custom Tool

```
def get_current_weather(location):
    '''
    Retrieves information about the weather at the specified location.

    Args:
        location (str): City or other location whose weather is to be retrieved.

    Returns:
        str: JSON string containing information about the current weather.
    '''
    response = requests.get(f'https://api.openweathermap.org/.../weather?q={location}')
    return json.dumps(response.json())
```

Using a Custom Tool

```
agent = Agent(
    name='Weather Agent',
    description='You are a helpful agent who provides weather information',
    model=OpenAIChat(id='gpt-4o-mini'),
    tools=[get_current_weather]
)

response = agent.run('Is it raining in Chicago?')
```

Demo

Agents with Tools

Model Context Protocol (MCP)

- Open standard developed by Anthropic to enable easy integration of language models and external data and APIs
  - Uses client-server architecture with JSON-RPC 2.0 messaging
  - "USB-C port for AI applications"
- Two types of servers
  - **stdio** servers, which run locally in subprocesses of the host
  - **Streamable HTTP** servers, which run remotely and are accessed over HTTP
- Official SDKs available for Python, TypeScript, Java, Rust, C#, and more
- Most agentic frameworks offer built-in support

Connecting Agno to a Stdio MCP Server

```
from agno.tools.mcp import MCPTools

mcp_tools = MCPTools(command=f'npx -y @openbnb/mcp-server-airbnb')
await mcp_tools.connect()

agent = Agent(
    name='Agno Agent',
    description='You are a helpful agent who has access to Airbnb listings',
    model=OpenAIChat(id='gpt-4o-mini'),
    tools=[mcp_tools]
)

response = await agent.arun('Where can I stay in Mostar?') # Call arun(), not run()
```

Setting Environment Variables

```
from agno.tools.mcp import MCPTools

mcp_tools = MCPTools(
    command=f'node C:/mcp/flightradar24-mcp-server/dist/index.js',
    env={
        'FR24_API_KEY': os.getenv('FR24_API_KEY'),
        'FR24_API_URL': 'https://fr24api.flightradar24.com'
    }
)

await mcp_tools.connect()
```

Connecting Agno to a Remote MCP Server

```
from agno.tools.mcp import MCPTools

mcp_tools = MCPTools(transport='streamable-http', url='MCP_SERVER_URL')
await mcp_tools.connect()

agent = Agent(
    name='Agno Agent',
    description='You are a helpful agent who has access to Airbnb listings',
    model=OpenAIChat(id='gpt-4o-mini'),
    tools=[mcp_tools]
)

response = await agent.arun('Where can I stay in Mostar?') # Call arun(), not run()
```

Passing HTTP Headers

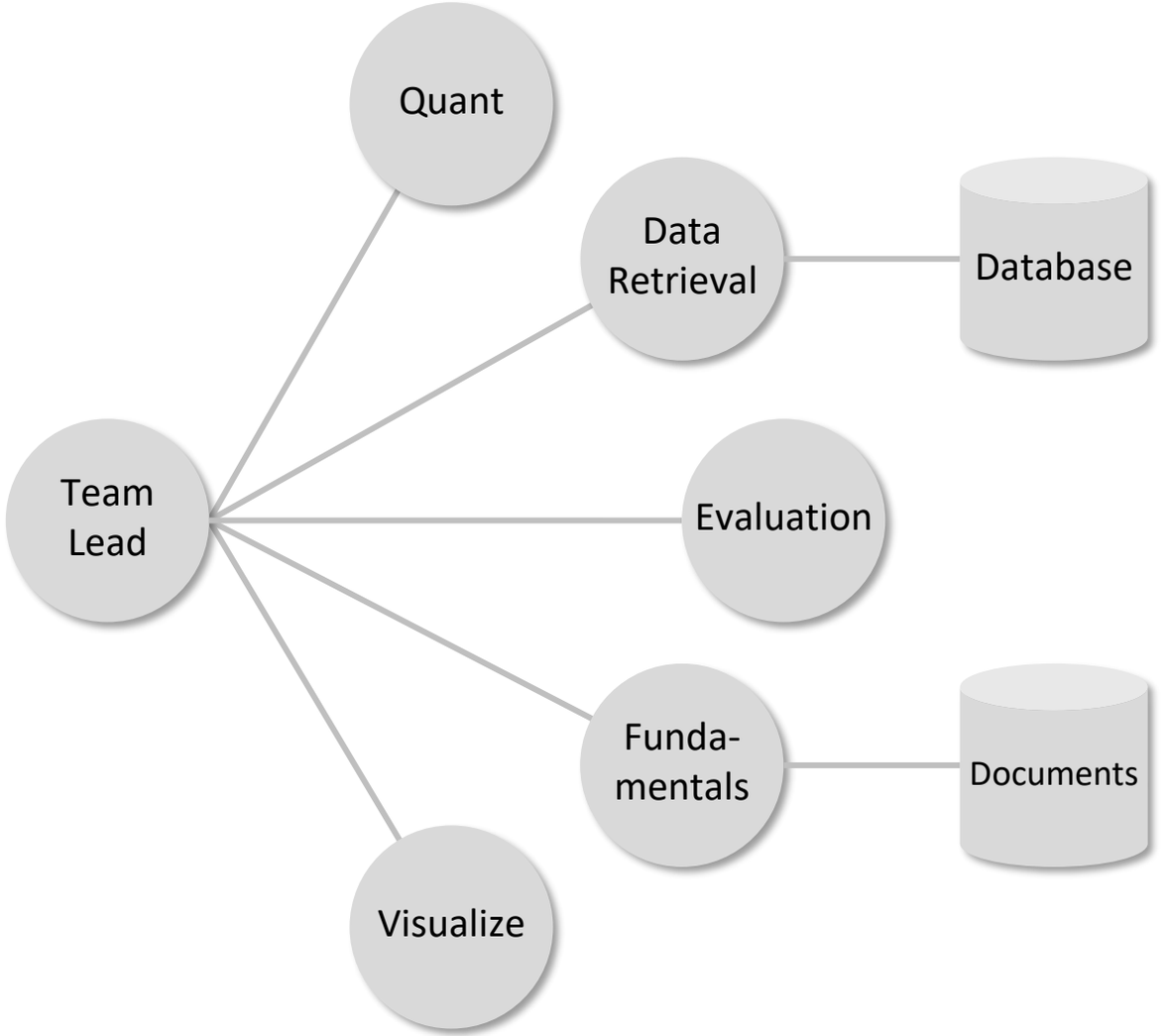
```
from agno.tools.mcp import StreamableHTTPClientParams, MCPTools

mcp_tools = MCPTools(
    transport='streamable-http', url='MCP_SERVER_URL'),
    server_params=StreamableHTTPClientParams(
        url='MCP_SERVER_URL',
        headers={
            'Authorization': 'Bearer YOUR_TOKEN_HERE',
            'X-Custom-Header': 'value'        }
    )
)

await mcp_tools.connect()
```

Demo

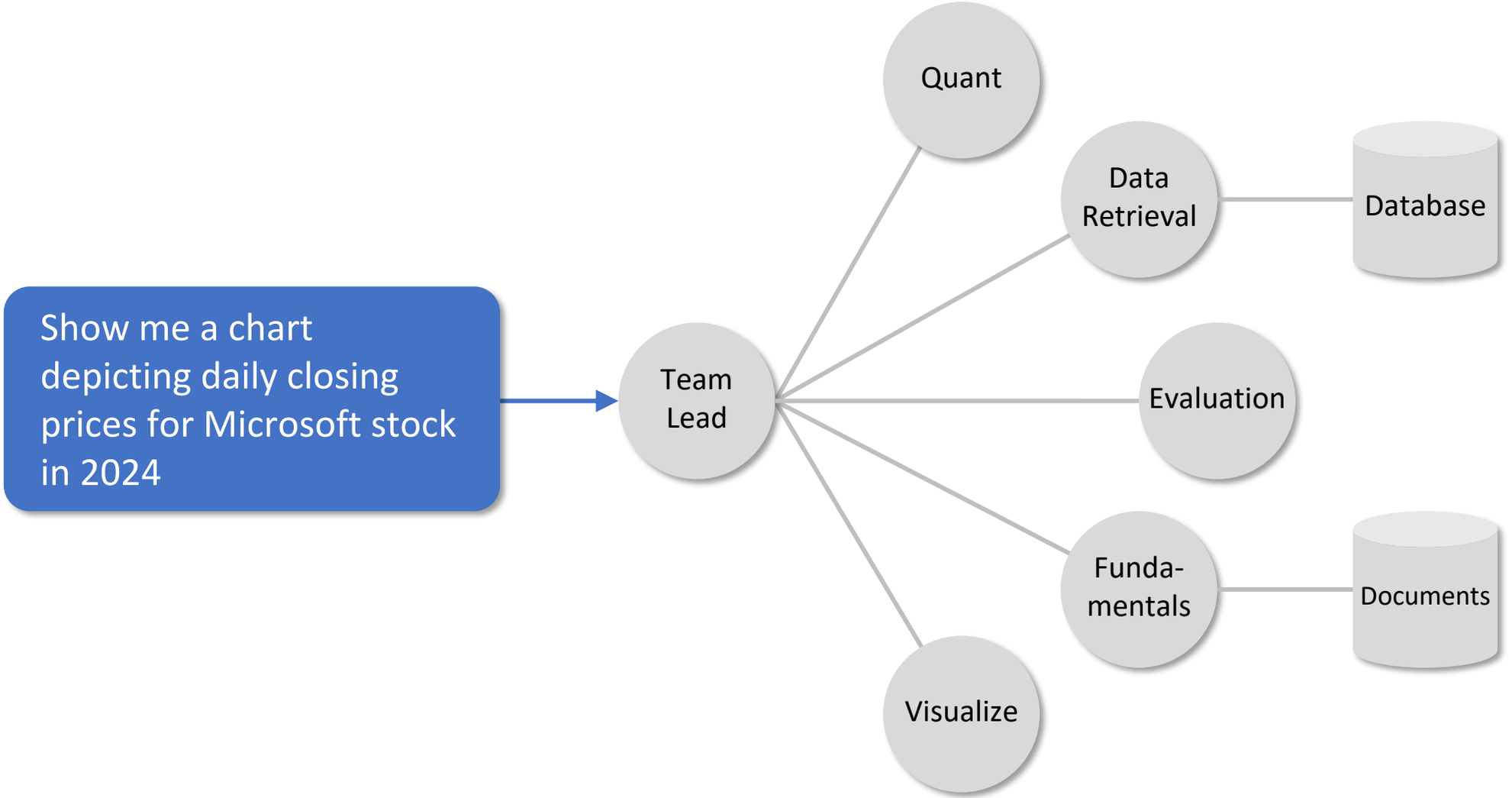
Model Context Protocol

Multi-Agent Orchestration

- Turn autonomous agents into collaborative agents that work as teams
  - Workflow can vary from one invocation to the next
  - Usually anchored by a "team lead" or supervisory agent
- Frameworks provide semantics for connecting agents and plumbing for messaging



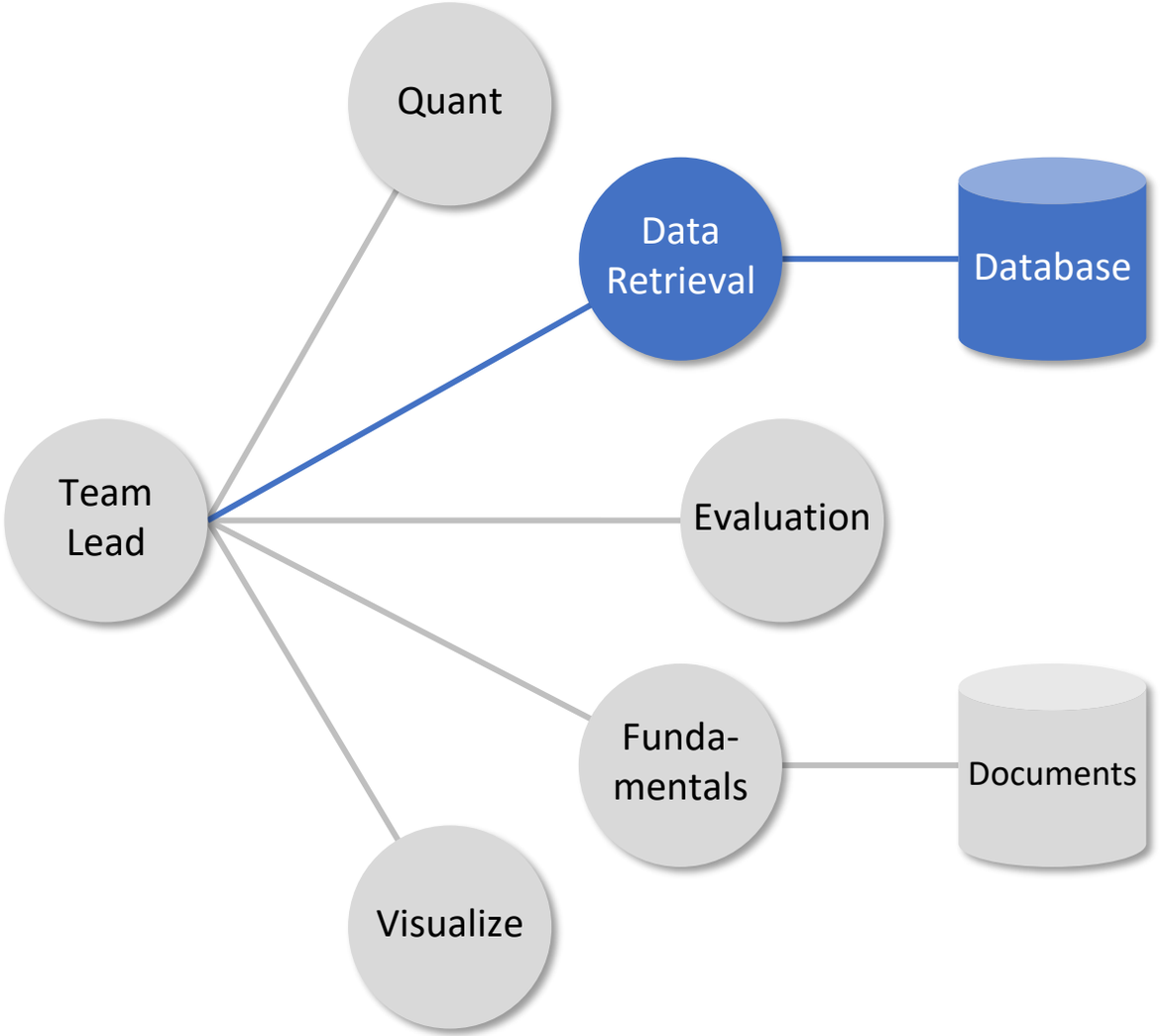
Agentic Workflow



Agentic Workflow, Cont.

Team Lead messages the **Data Retrieval** agent and asks it for MSFT's 2024 closing prices.

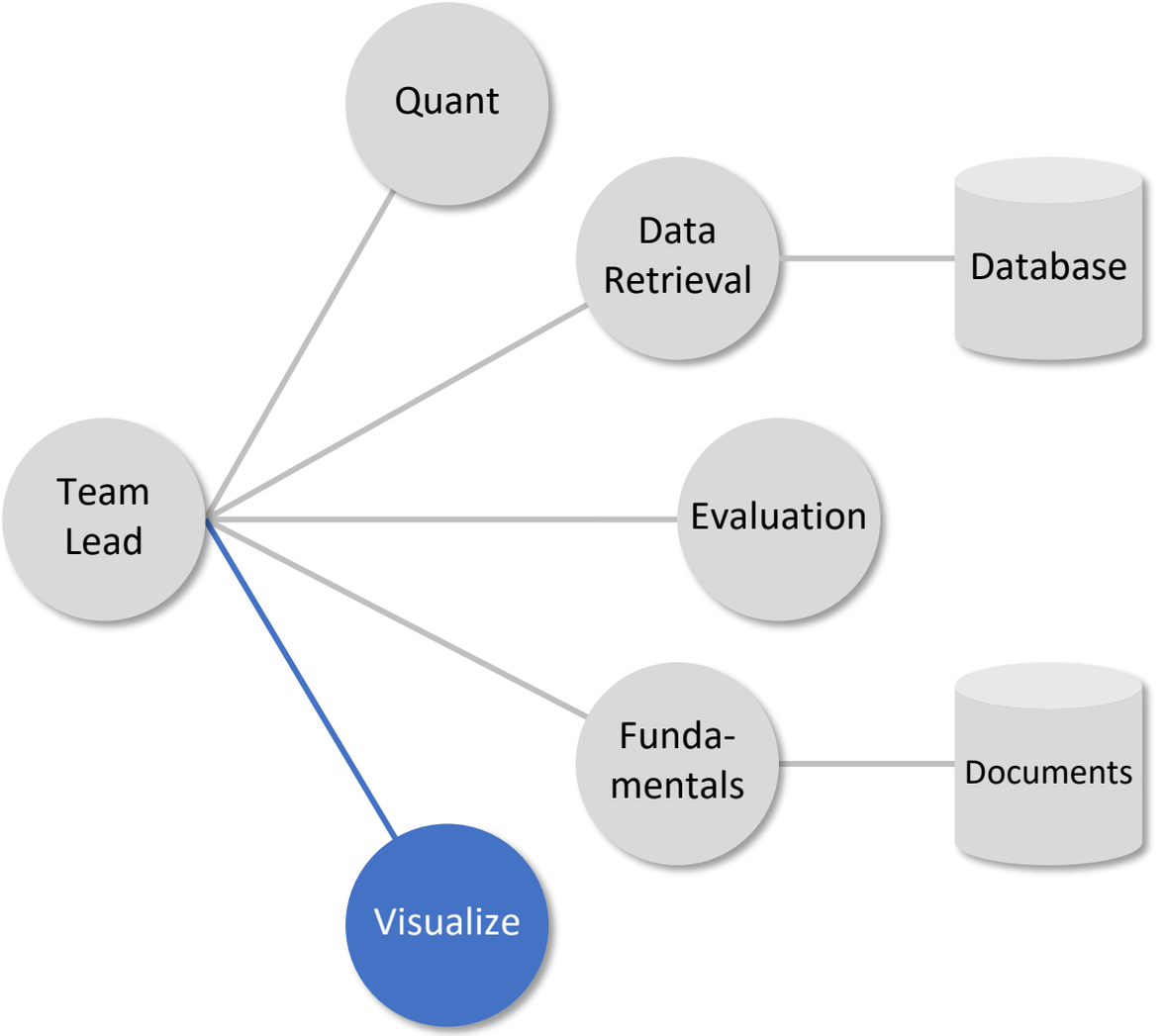
Data Retrieval agent converts the **Team Lead**'s request into a SQL query, queries the database, and returns the results to the **Team Lead**.



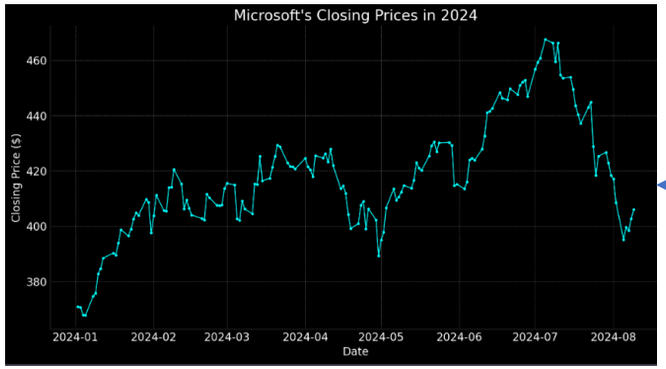
Agentic Workflow, Cont.

Team Lead passes the **Data Visualization** agent the closing prices provided by the **Data Retrieval** agent.

Data Visualization agent generates Python code to chart the data. Then it executes the code, saves the chart as an image, and returns the image URL to the **Team Lead**.



Agentic Workflow, Cont.



Team Lead

Quant

Data Retrieval

Database

Evaluation

Funda-mentals

Documents

Visualize

Composing a Team of Agents

```
agent_a = Agent(name='Agent A', ...)
agent_b = Agent(name='Agent B', ...)

team = Team(
    name='Team Lead',
    model=OpenAIChat(id='gpt-4o-mini'),
    share_member_interactions=True, # Share data among agents
    members=[agent_a, agent_b]
)

team.run('Generate a pie chart depicting annual sales')
```

Enabling Chat History

```
team = Team(
    name='Team Lead',
    mode='coordinate',
    model=OpenAIChat(id='gpt-4o-mini'),
    share_member_interactions=True,
    members=[agent_a, agent_b],
    add_history_to_context=True,
    num_history_runs=12
)
```

Demo

Multi-Agent Orchestration