Building AI-Powered QA Workflows

Streamlining Test Case Generation with LangGraph

Yogesh Haribhau Kulkarni (PhD)
Published in Technology Hits
4 min read · Just now

Photo by Campaign Creators on Unsplash

The integration of Large Language Models (LLMs) into software development has revolutionized how we approach coding tasks. While much attention focuses on using LLMs as coding assistants, there's untapped potential in applying these models to quality assurance processes. The provided code exemplifies how LangGraph can orchestrate LLM-powered test case generation from requirement documents — creating a structured, predictable workflow while leveraging AI capabilities.

Understanding LangGraph's Role

LangGraph, an orchestration framework for LLMs, stands at the heart of this application. It provides a directed graph structure that determines how data flows through various processing stages. The code demonstrates several key

LangGraph concepts:

1. State Management through TypedDict

The foundation starts with defining a state structure that flows through the graph:

```
class GraphState(TypedDict):
    user_request: str
    requirements_docs_content: str
    requirements_docs_summary: str
    testcases_format: str
    testcases: str
    answer: str
```

This TypedDict provides a contract for the data structure passed between nodes, ensuring consistency throughout the workflow.

2. Node Definition

Each processing step is encapsulated as a node function that transforms the state:

```
def generate_summary_node_function(state: GraphState) -> GraphState:
    """Uses LLM to generate summary of `requirements_docs_content`."""
    requirements_docs_content = state.get("requirements_docs_content", "")

    prompt = (
        "You are an expert in generating QA testcases for any known formats. \n" +
        "Study the given 'Requirements Documents Content' carefully and generate summary of about 5 lines\n" +
        f"Requirements Documents Content: {requirements_docs_content}\n" +
        "Answer:"
    )

    response = st.session_state.llm.invoke(prompt)

    state['requirements_docs_summary'] = response.content
    state['answer'] = response.content
    return state
```

Each function receives the current state, performs its specific task, and returns an updated state — maintaining the functional programming paradigm that makes workflows predictable and testable.

3. Conditional Routing with Edge Definition

One of LangGraph's most powerful features is its ability to implement conditional logic through routing functions:

```
def route_user_request(state: GraphState) -> str:
    user_request = state["user_request"]
    # Routing logic that determines whether to generate Gherkin or Selenium tests
    # ...
    return tool  # Returns "gherkin" or "selenium"
```

This router dictates which path the workflow takes next, creating dynamic, context-aware processing.

4. Graph Construction

The workflow is assembled by connecting nodes with directed edges:

```
workflow = StateGraph(GraphState)
workflow.add_node("summary_node", generate_summary_node_function)
workflow.add_node("gherkin_node", generate_gherkin_testcases_node_function)
workflow.add_node("selenium_node", generate_selenium_testcases_node_function)

workflow.set_entry_point("summary_node")
workflow.add_conditional_edges(
    "summary_node",
    route_user_request,
    {
        "gherkin": "gherkin_node",
        "selenium": "selenium_node"
    }
)
workflow.add_edge("gherkin_node", END)
workflow.add_edge("selenium_node", END)
```

This structure creates a clear processing pipeline with branching logic, allowing for multiple possible execution paths while maintaining overall control of the workflow.

Streamlit Integration for User Interaction

The application uses Streamlit to provide an interactive interface, managing state through Streamlit's session management:

```
# Stream the output
total_answer = ""
for output in app.stream(inputs):
    for node_name, state in output.items():
        if 'answer' in state:
            total_answer += state['answer']
response_placeholder.markdown(total_answer)
```

This streaming capability provides real-time feedback as the LLM processes information through different nodes of the graph.

Key Architectural Benefits

1. **Separation of Concerns**: Each node has a focused responsibility — summarizing requirements, generating specific test formats, or making routing decisions.
2. **Controlled AI Decision-Making**: While leveraging LLM capabilities, the application maintains predictable execution paths through the graph structure.
3. **Flexibility in LLM Selection**: The code supports multiple Groq-hosted models, allowing users to select the most appropriate LLM for their requirements.
4. **Stateful Processing**: The graph maintains state throughout the workflow, ensuring data consistency between different processing stages.

The Power of Structured LLM Workflows

What makes this application particularly notable is how it addresses a common challenge with LLM applications — balancing the free-form generative capabilities of models with the need for structured, predictable application behavior.

By embedding LLM calls within a directed graph, developers gain several advantages:

1. **Transparency**: The workflow structure makes it clear how information flows from requirements to test cases.
2. **Debuggability**: Each node's behavior can be tested independently, making it easier to isolate and fix issues.

3. **Extensibility:** New test formats or processing steps can be added by simply defining new nodes and connecting them to the graph.

Conclusion: Beyond Test Generation

While this application focuses on test case generation, the architectural pattern demonstrated here has far-reaching implications. The combination of LangGraph for orchestration, LLMs for intelligence, and Streamlit for interaction creates a powerful template for building AI-augmented workflows across domains.

As we continue exploring the integration of LLMs into software development, frameworks like LangGraph will be crucial in creating applications that harness AI capabilities while maintaining the predictability and reliability required for professional tools. The challenge moving forward will be to expand these patterns to more complex workflows while maintaining the clarity and control demonstrated in this proof of concept.

The code exemplifies a new paradigm in software development — one where AI components are first-class citizens in application architecture, but their behavior is channeled through structures that ensure reliability and predictability. As LLMs continue to evolve, frameworks that successfully manage their integration into traditional software will become increasingly valuable.

Langgraph Workflow Automation Generative Ai Tools Large Language Models QA

Following

Published in Technology Hits

3.3K Followers · Last published just now

Important, high-impact, informative, and engaging stories on all aspects of technology.

Edit profile

Written by Yogesh Haribhau Kulkarni (PhD)

1.7K Followers · 2.1K Following

PhD in Geometric Modeling | Google Developer Expert (Machine Learning) | Top Writer 3x (Medium) | More at https://www.linkedin.com/in/yogeshkulkarni/

No responses yet

Yogesh Haribhau Kulkarni (PhD)

What are your thoughts?

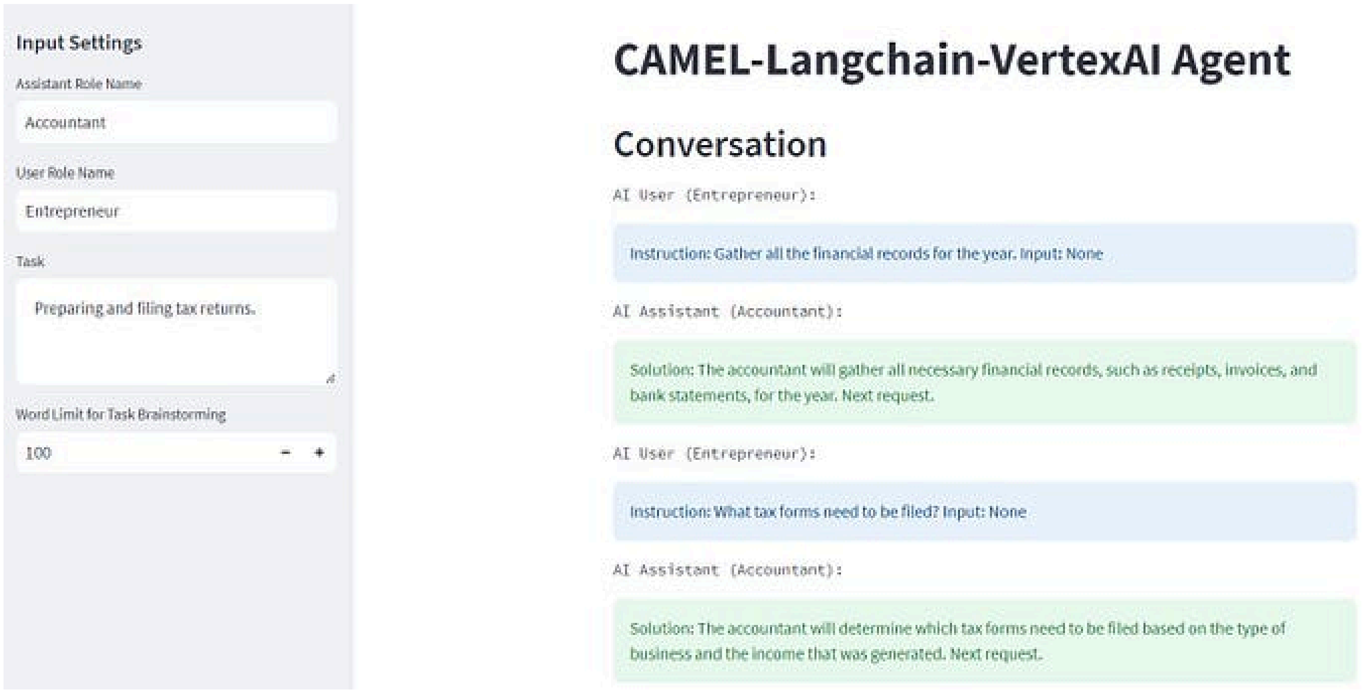
More from Yogesh Haribhau Kulkarni (PhD) and Technology Hits

In Technology Hits by Yogesh Haribhau Kulkarni (PhD)

Unveiling Manifold learning

What a neural network is really doing?

Feb 23, 2024 281 1



Input Settings

Assistant Role Name: Accountant

User Role Name: Entrepreneur

Task: Preparing and filing tax returns.

Word Limit for Task Brainstorming: 100

CAMEL-Langchain-VertexAI Agent

Conversation

AI User (Entrepreneur):

Instruction: Gather all the financial records for the year. Input: None

AI Assistant (Accountant):

Solution: The accountant will gather all necessary financial records, such as receipts, invoices, and bank statements, for the year. Next request.

AI User (Entrepreneur):

Instruction: What tax forms need to be filed? Input: None

AI Assistant (Accountant):

Solution: The accountant will determine which tax forms need to be filed based on the type of business and the income that was generated. Next request.

In Google Developer Experts by Yogesh Haribhau Kulkarni (PhD)

A CAMEL ride

A Story of AI Role-Playing using CAMEL, Langchain and VertexAI

Oct 2, 2023 56

In ILLUMINATION Videos and Podcasts by Yogesh Haribhau Kulkarni (PhD)

The Equation of True Happiness

Based on talks by Arthur C Brooks

Sep 17, 2023 164 1

In Analytics Vidhya by Yogesh Haribhau Kulkarni (PhD)

Introduction to DBpedia

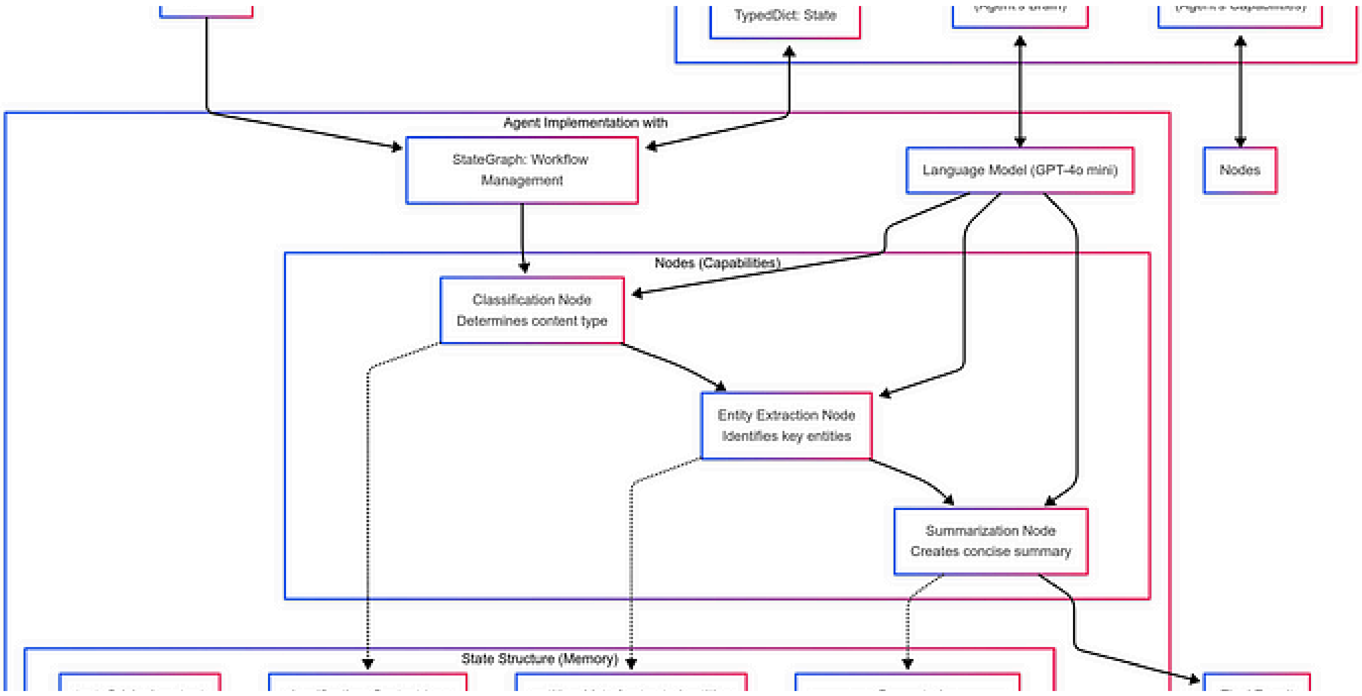
Making Wikipedia Query-able

Apr 11, 2024 25 1

See all from Yogesh Haribhau Kulkarni (PhD)

See all from Technology Hits

Recommended from Medium

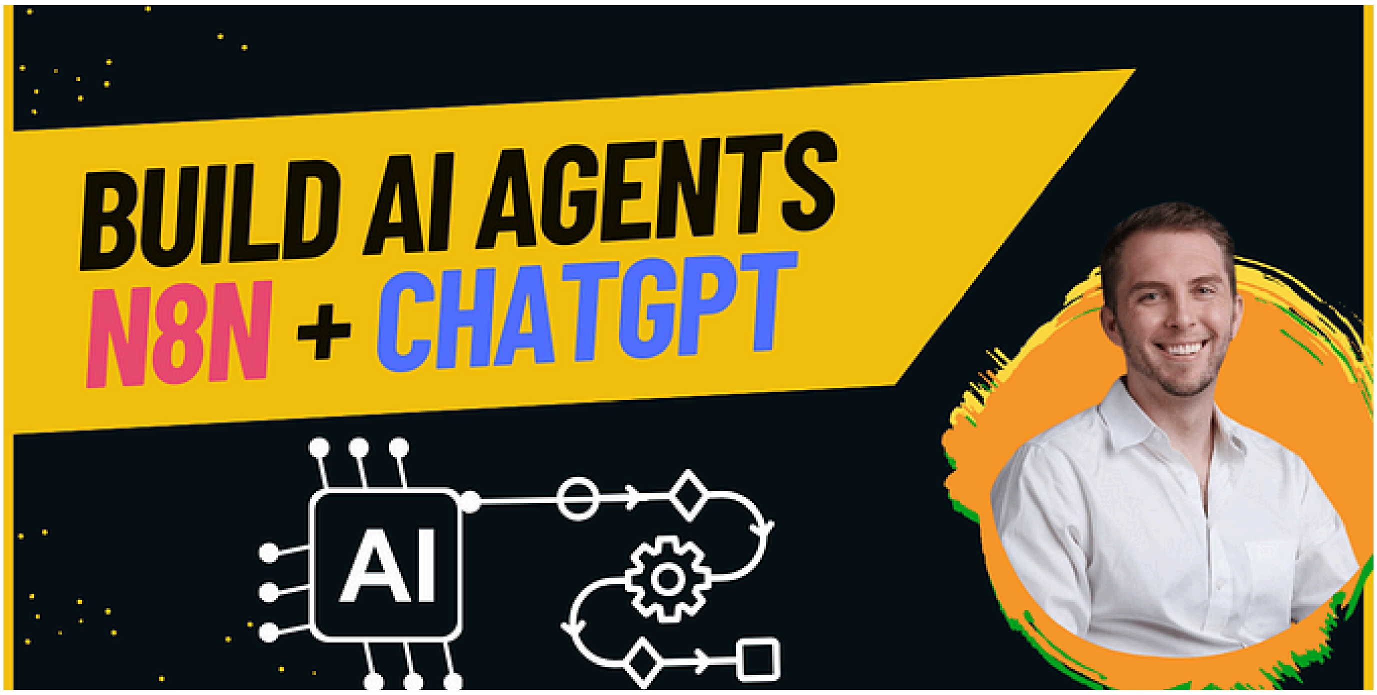

TypedDict: State
Agent Implementation with
StateGraph: Workflow Management
Language Model (GPT-4o mini)
Nodes
Nodes (Capabilities)
Classification Node Determines content type
Entity Extraction Node Identifies key entities
Summarization Node Creates concise summary
State Structure (Memory)

In Data Science Collective by Paolo Perrone

The Complete Guide to Building Your First AI Agent (It's Easier Than You Think)

Three months into building my first commercial AI agent, everything collapsed during the client demo.

Mar 11 1.8K 41

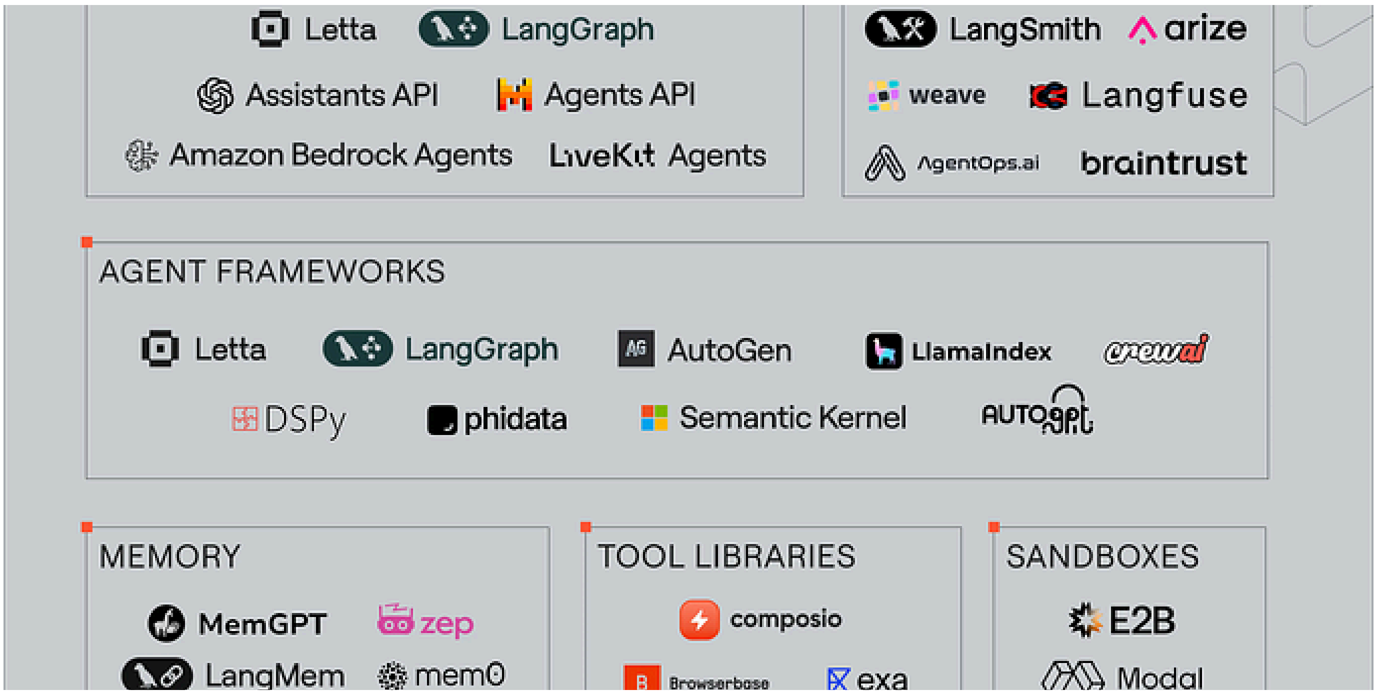


Nick Canfield

How to Build Simple AI Agents With n8n & ChatGPT (OpenAI)

Learn how to build an AI agent with n8n + ChatGPT. Automate your business processes with an intelligent workflow design using AI.

4d ago 33

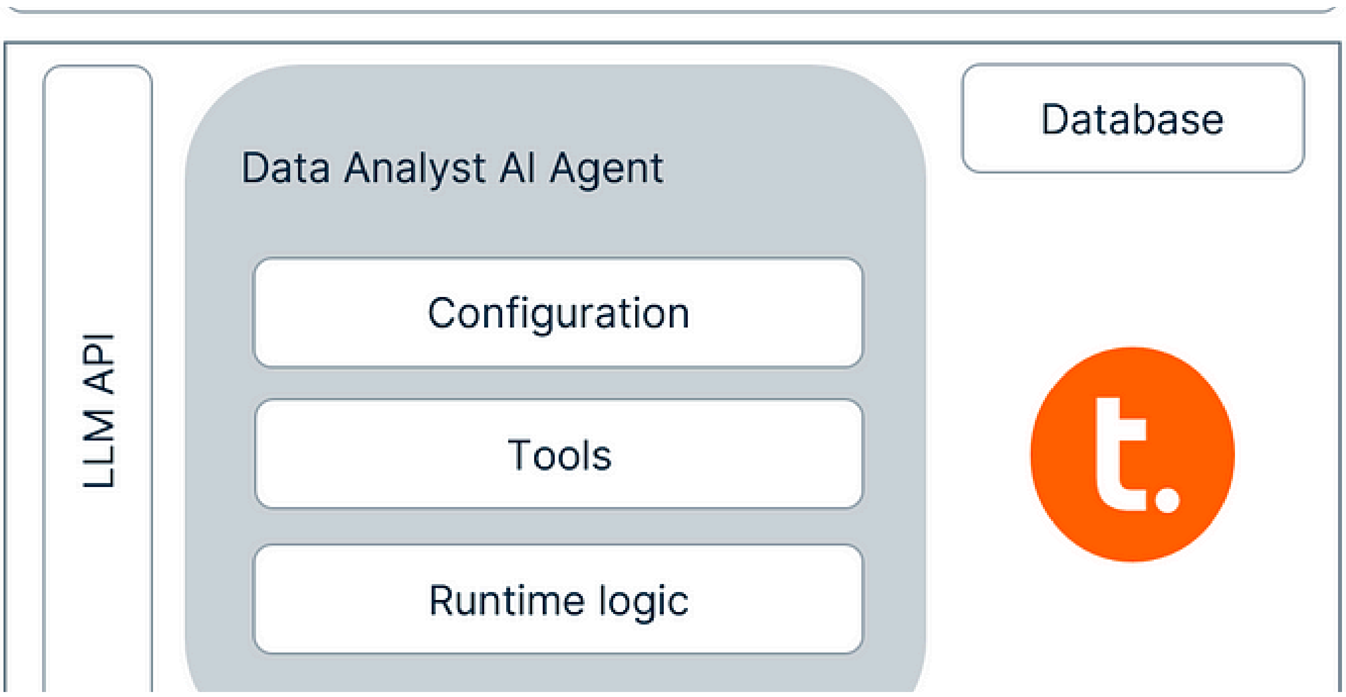


Vipra Singh

AI Agents: Introduction (Part-1)

Discover AI agents, their design, and real-world applications.

Feb 3 1.4K 30

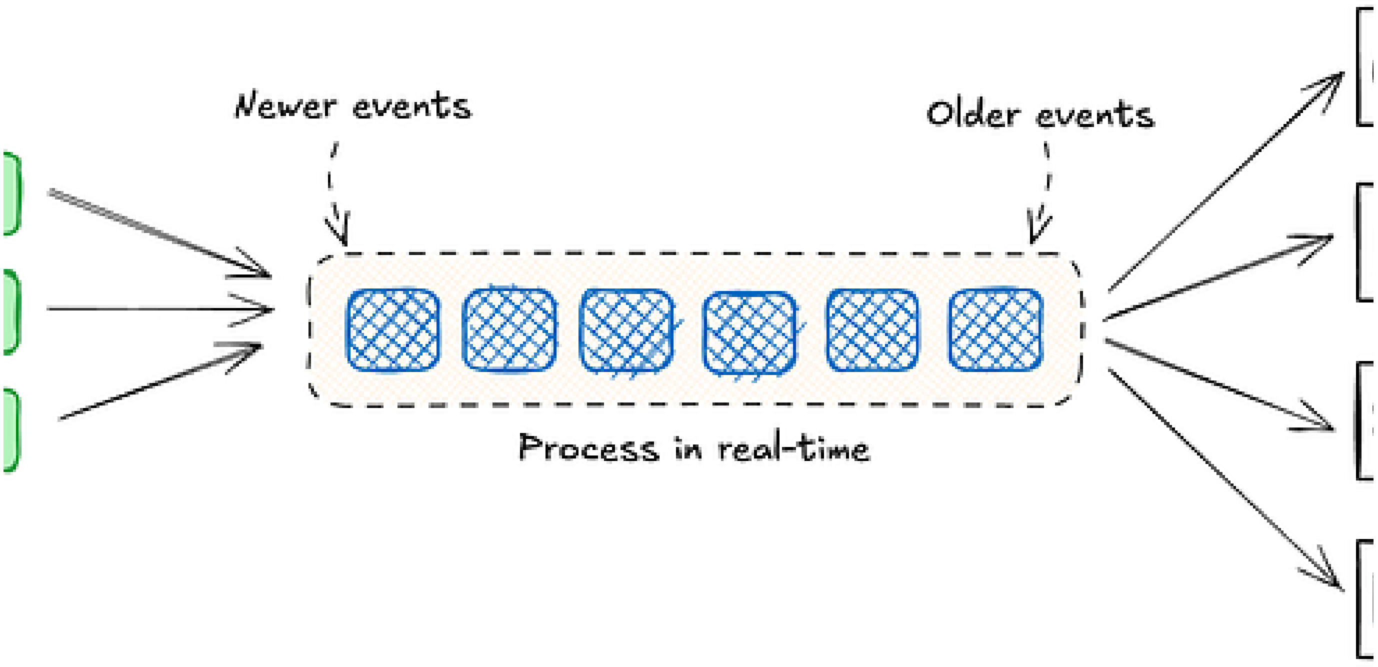


In Teradata by Daniel Herrera

Build a Data Analyst AI Agent from Scratch

As presented in the Open Data Science Conference AI Builders Summit 2025

Feb 7 280 4

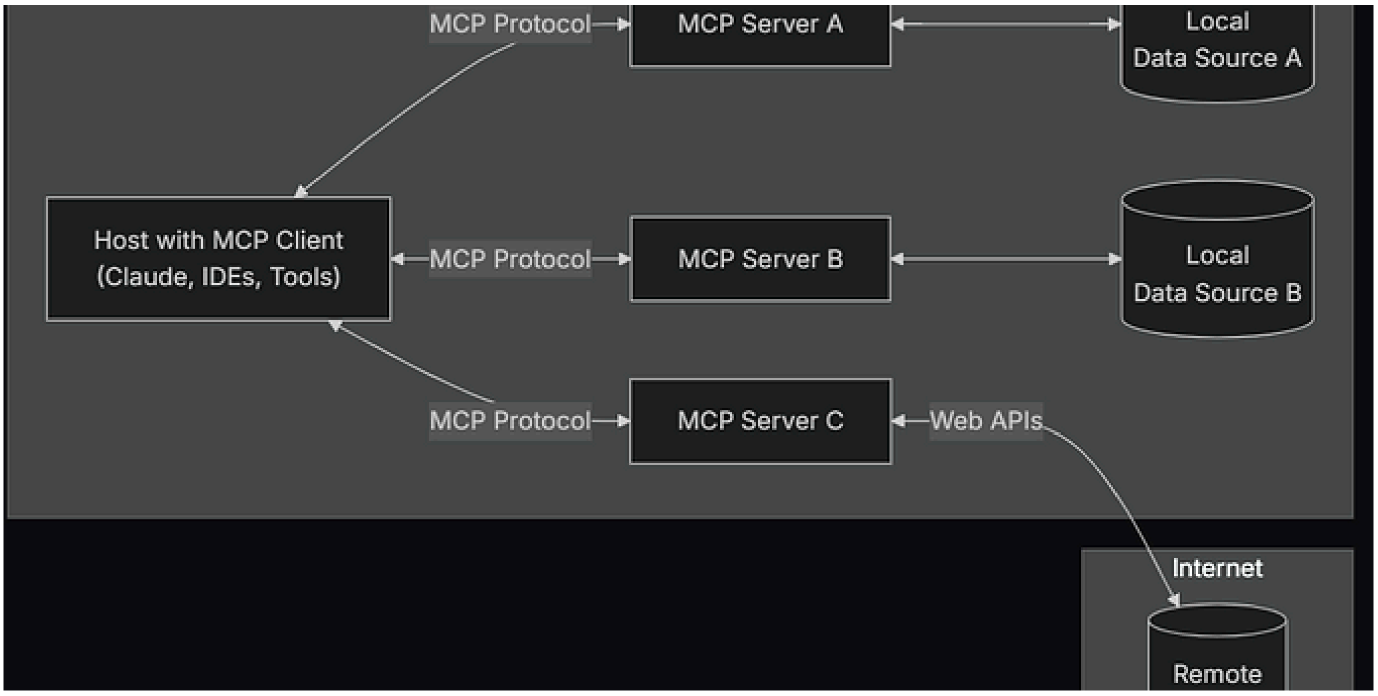


Sean Falconer

The Future of AI Agents is Event-Driven

AI agents promise autonomy and adaptability. Event-driven architecture provides the backbone for these systems to scale and evolve.

Mar 12 752 19



In Data And Beyond by TONI RAMCHANDANI

The Model Context Protocol (MCP): The Ultimate Guide

Introduction to the Model Context Protocol (MCP)

Mar 10 132 3

See more recommendations