Gen AI with Element

Surendra Panpaliya

PREREQUISITES

Participants should have:

Basic knowledge of **Python programming**

Familiarity with **APIs**, **JSON**, and **HTTP requests**

General understanding of **Machine Learning** and **NLP concepts**

PREREQUISITES

Awareness of **cloud platforms** (Azure or GCP preferred)

Prior exposure to **Jupyter Notebooks** or **VS Code**

Knowledge of **RESTful APIs**, **Docker**, and **Git**

Some experience with **LLMs** or **prompt engineering**

LAB SETUP REQUIREMENTS

Python 3.10+ installed (preferably in a virtual environment)

JupyterLab or **VS Code** with Python plugin

Access to:

Azure OpenAI API key (or)

Google GenAI credentials (Vertex AI Studio, PaLM/Gemini API)

📌 LEARNING OUTCOMES

Explain key concepts in

Generative AI and

Transformer-based LLMs

Build and interact with

OpenAI/Gemini models using Python

📌 LEARNING OUTCOMES

Design **RAG pipelines** integrated with

LangChain + Milvus

Apply structured prompt

engineering strategies in LLM apps

📌 LEARNING OUTCOMES

Utilize Walmart’s internal

LLM Gateway and **evaluation platforms**

Create simple **Agentic applications**

with planning, execution, and tools

📌 LEARNING OUTCOMES

Troubleshoot common LLM issues

such as hallucination or bias

Build and present a functional

Excel-based report builder GenAI app

Agenda

- DAY 1: GENAI FOUNDATION & ARCHITECTURE
- DAY 2: WALMART GENAI ECOSYSTEM
- DAY 3: APPLICATION DEVELOPMENT WITH GENAI
- DAY 4: HACKATHON & DEPLOYMENT

HACKATHON & DEPLOYMENT

Objective:

Build, test, evaluate, and present

a GenAI-powered prototype

focused on real-world data handling.

1. Natural Language Interface for Excel

Reading and interpreting Excel using Python

Text-to-Table and Table-to-Text generation with LLMs

Writing summaries and visualizations based on Excel data

Objective

- Build, test, evaluate, and present
- a GenAI-powered prototype
- focused on real-world
- data handling using Excel.

Natural Language Interface for Excel

GenAI team will learn to:

Read and interpret Excel using Python

Convert text queries into table summaries and vice versa

Use LLMs to write summaries and

create visualizations from Excel data

Real-World Use Case at Walmart

“A manager uploads a sales Excel file.

Can an AI agent generate a summary,

key trends, and charts

just by asking questions in plain English?”

This is where GenAI + Excel meets practical impact.

Hands-On GenAI Interface for Excel

Surendra Panpaliya

GKTCS Innovations

https://www.gktcs.com

1. Environment Setup

```
pip install pandas openpyxl langchain openai matplotlib seaborn python-docx
```

Also, load your .env file with your OpenAI API Key:

```
OPENAI_API_KEY=sk-xxxxx
```

2. Read and Interpret Excel using Python

```
import pandas as pd

# Load Excel
df = pd.read_excel("walmart_sales.xlsx")
print(df.head())
```

Sample Output

Date	Product	Sales	Store	Region
2024-01-01	Apple	1200	Store_A	West

3. Table-to-Text Summary with OpenAI

```
from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate

llm = OpenAI(temperature=0.3)
```

3. Table-to-Text Summary with OpenAI

```
 # Prompt Template
template = """
You are a data analyst. Summarize this table:
{dataframe}
"""
```

3. Table-to-Text Summary with OpenAI

```
prompt = PromptTemplate.from_template(template)

df_sample = df.head(10).to_string(index=False)
query = prompt.format(dataframe=df_sample)
response = llm(query)
print("Summary:", response)
```

Output:

“In the first 10 rows, Apple and Banana are
top-selling products in the West and Central regions.
Store_A had the highest revenue on Jan 1.”

Text-to-Table

```
query = "Which product sold the most in the West region?"

context = df[df["Region"] == "West"]

top_product = context.groupby("Product")["Sales"].sum().idxmax()

print("Top product in West:", top_product)
```

Text-to-Table

Extend this by integrating with **LangChain Tools** to allow LLM to trigger such queries dynamically.

What Developers Will Learn by End of Session ?

Capability	Skill Gained
Load and preprocess Excel	Real-world data handling with Pandas
Table-to-Text with LLM	Prompt engineering & LangChain integration
Querying Excel via language	Agentic querying (Text → Code → Result)
Auto-generate summaries	LLM-driven business insights
Visualize & report	Matplotlib + Word doc auto-reports

2. Agentic AI in Business Processes

Use case examples:

- Automated financial reporting
- Smart procurement assistants
- Customer support escalation systems

What is Agentic AI in Business Processes?

Agentic AI enables **autonomous systems** to:

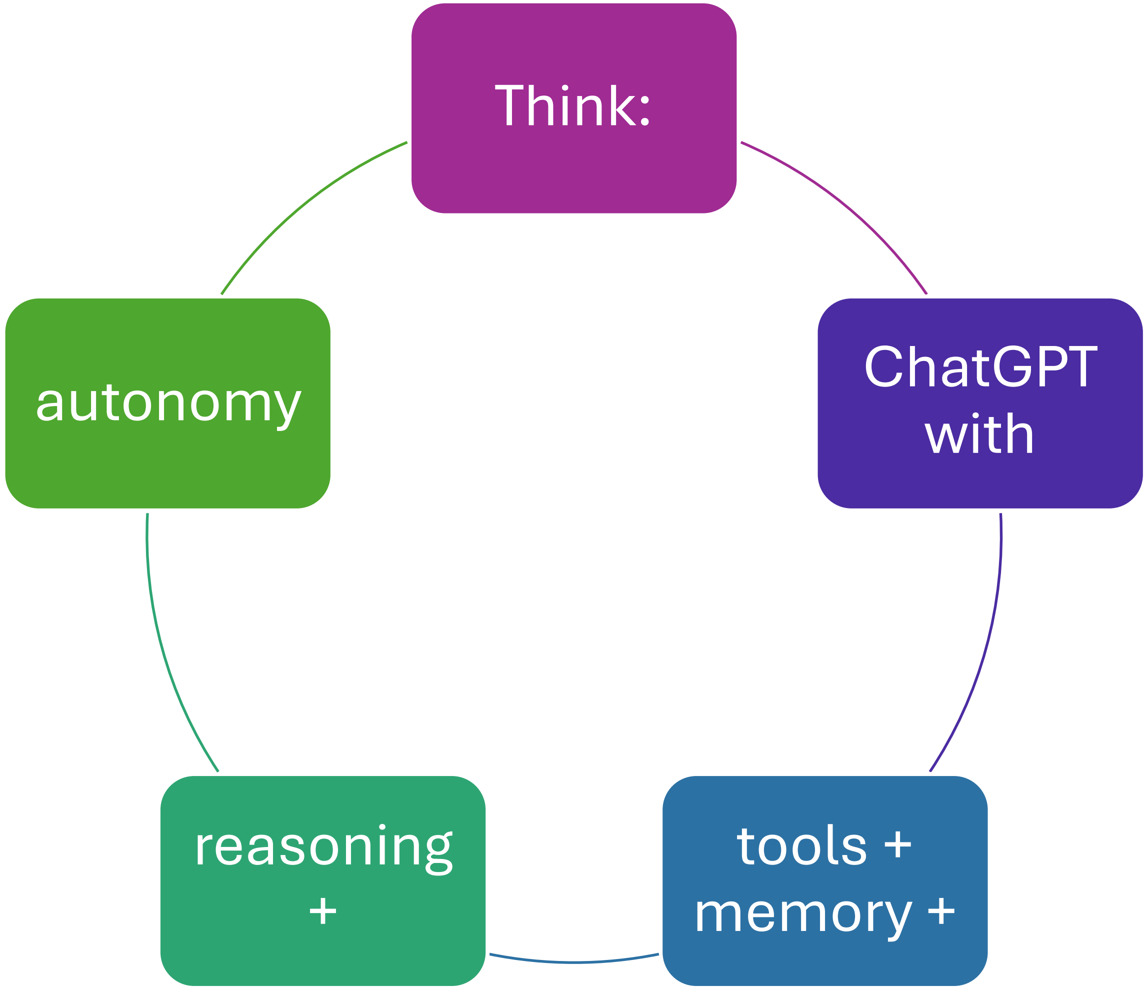
Understand goals from natural language

Plan sequences of actions

What is Agentic AI in Business Processes?

- Use tools/APIs autonomously
- Monitor outcomes and adjust
- Deliver end-to-end business outcomes

What is Agentic AI in Business Processes?



Think:

ChatGPT with

tools + memory +

reasoning +

autonomy

Why It Matters at Walmart?

Agentic AI streamlines and augments:

📊 **Financial intelligence**

🛒 **Procurement efficiency**

Why It Matters at Walmart?

- 📞 Customer support escalation
- 📦 Inventory management
- 📈 Retail analytics

Use Case1

Automated Financial Reporting Agent

Problem:

Finance teams spend hours every week

generating Excel-based monthly reports.

Agent Workflow

- Reads financial data from Excel/CSV
- Summarizes key trends
- Generates visualizations
- Emails auto-generated PDF/Word report to leadership

Hands-On Example

LangChain + Pandas + OpenAI

```
! pip install openai langchain pandas matplotlib python-docx
```

Hands-On Example

```
import pandas as pd
from langchain.llms import OpenAI
from docx import Document
import matplotlib.pyplot as plt
```

Use Case 2

Smart Procurement Assistant

Problem:

Buyers need help identifying shortages,

predicting demand, and placing orders.

Agent Capabilities

- Reads live inventory levels
- Predicts demand using AI
- Suggests purchase orders
- Integrates with procurement API

Use Case2

Goal: "Restock top 5 critical SKUs"

→ Agent → Planner → Inventory API + Demand Forecast Tool

→ Tool Execution → PO Suggestion → Email to buyer

Use Case 3:

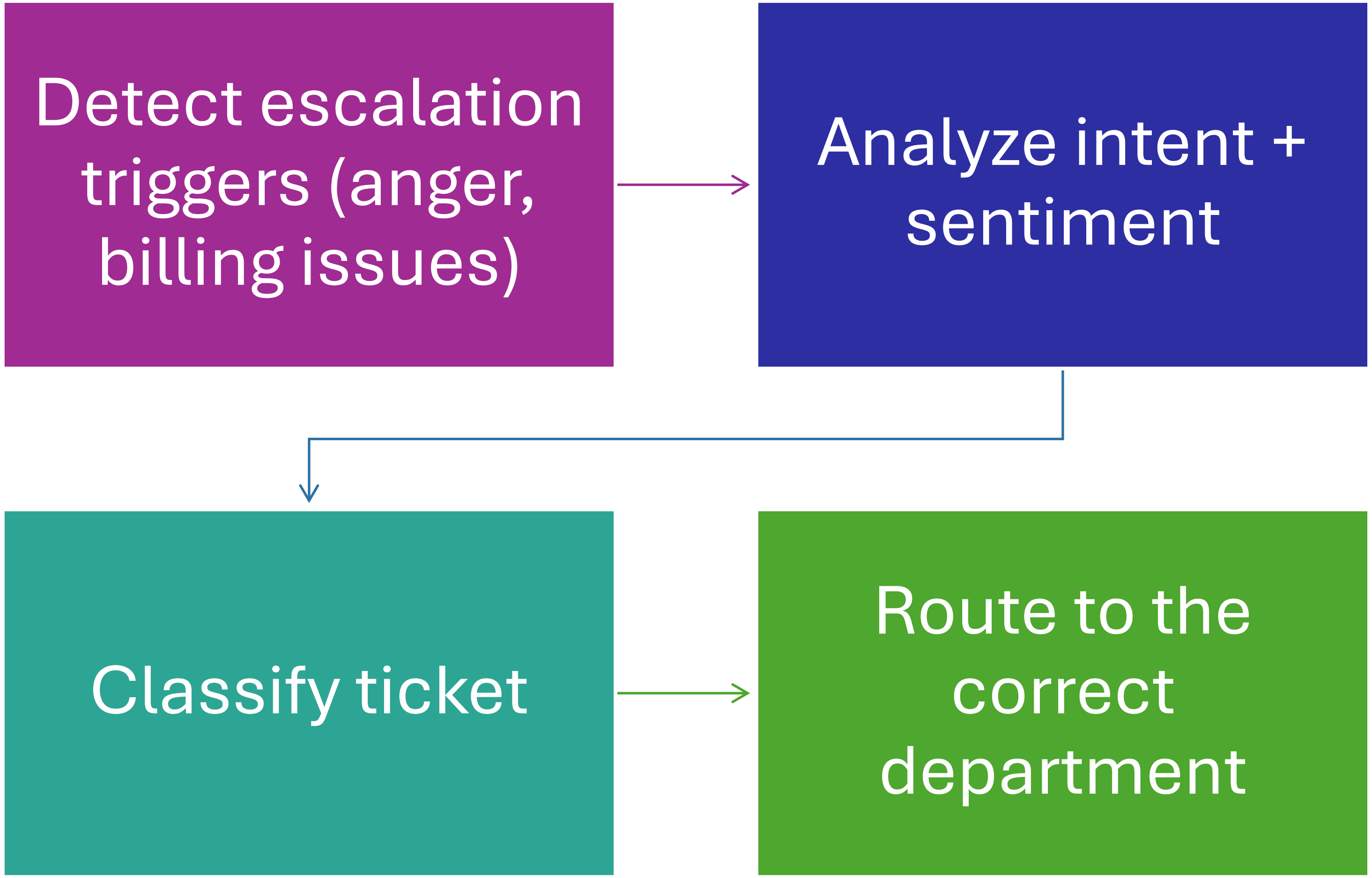
Customer Support Escalation Agent

Problem:

AI chatbots often struggle with complex queries

→ need smart handoff to human.

Agent Behavior


Detect escalation triggers (anger, billing issues)

Analyze intent + sentiment

Classify ticket

Route to the correct department

Expected Outcomes for Walmart Developers

Deliverable	Skill/Insight Gained
GenAI-powered business agent demo	End-to-end agent workflow implementation
LLM integration with business tools	APIs, file handling, summarization
Evaluation capability	Output quality, agent accuracy
Prototype Presentation	Enterprise-ready GenAI use case

3. Code Quality & Prompt Evaluation Metrics

Evaluating performance and

correctness of prompt-based workflows

Metrics:

Relevance

3. Code Quality & Prompt Evaluation Metrics

Fluency

Faithfulness

Toxicity

3. Code Quality & Prompt Evaluation Metrics

Objective:

Build, test, evaluate, and present a

GenAI-powered prototype focused on

real-world data handling.

Why This Matters at Walmart

- When using LLMs for business automation
- finance reports,
- procurement summaries,
- customer support responses

Why This Matters at Walmart

Must **evaluate prompt-generated outputs** for:

✍️ Quality

🔁 Reliability

🤖 Safety

✅ Business relevance

Prompt Evaluation: Key Metrics

Relevance

Fluency

Faithfulness

Toxicity

Relevance

- Does the output address the user’s intent?
- “Summarize sales for West region”
- → Should not include East region.

Fluency

- Is the output grammatically correct
- and natural-sounding?
- Essential for public-facing
- use cases (emails, reports).

Faithfulness

- Are the facts **actually derived**
- **from the source data**?
- Avoid hallucinations
- like imaginary products or metrics.

Toxicity

- Is the output **free from biased,**
- **offensive, or unethical content**?
- Critical in customer support or
- HR-facing prompts.

Hands-On

Evaluating Prompt Output with Python

openai (for LLM output)

transformers (for toxicity)

Manual checks (for relevance, fluency, faithfulness)

4. LLM Troubleshooting

- Common issues:
- hallucinations, verbosity, bias
- Debugging techniques for GenAI apps

LLM Troubleshooting

Objective:

Identify, debug, and mitigate **common issues in GenAI applications**

LLM Troubleshooting

such as:

🔍 **Hallucinations**

📢 **Verbosity**

⚖️ **Bias**

Common LLM Issues

Issue	Description	Example
Hallucination	LLM generates confident but incorrect or made-up facts	“Walmart opened a store on the moon.”
Verbosity	LLM provides overly long or repetitive answers	“Let me explain that in detail… again…”
Bias	LLM reflects unfair preferences or stereotypes	“Boys like trucks, girls like dolls.”

Why Troubleshooting is Critical at Walmart ?

- 🧾 Inaccurate financial summaries → poor decisions
- 📦 Wrong restocking suggestions → supply chain disruption
- 🤖 Biased chat responses → brand reputation risk

Hands-on Troubleshooting Walkthrough

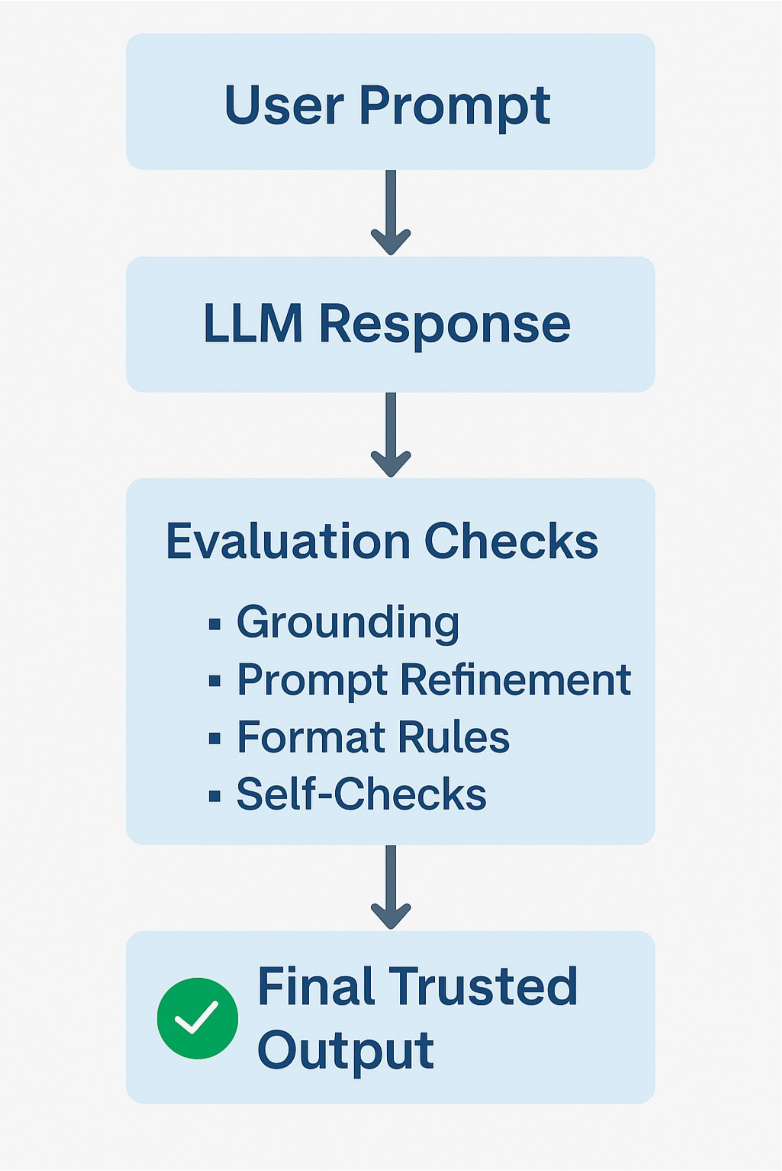
Step 1: Setup (for OpenAI + LangChain users)

```
pip install openai langchain transformers
```

Load your .env or configure your API key:

```
import openai
openai.api_key = "sk-..."
```

Debugging Flow: Visual



User Prompt

LLM Response

Evaluation Checks

- Grounding
- Prompt Refinement
- Format Rules
- Self-Checks

✓ Final Trusted Output

Summary

Issue	Detection Method	Fix/Technique
Hallucination	LLM Self-check, Schema mismatch	Ground with real data or tool calls
Verbosity	Lengthy/redundant output	Add constraints in prompt
Bias	Sentiment/Toxicity model	Toxic-BERT, Prompt rewriting

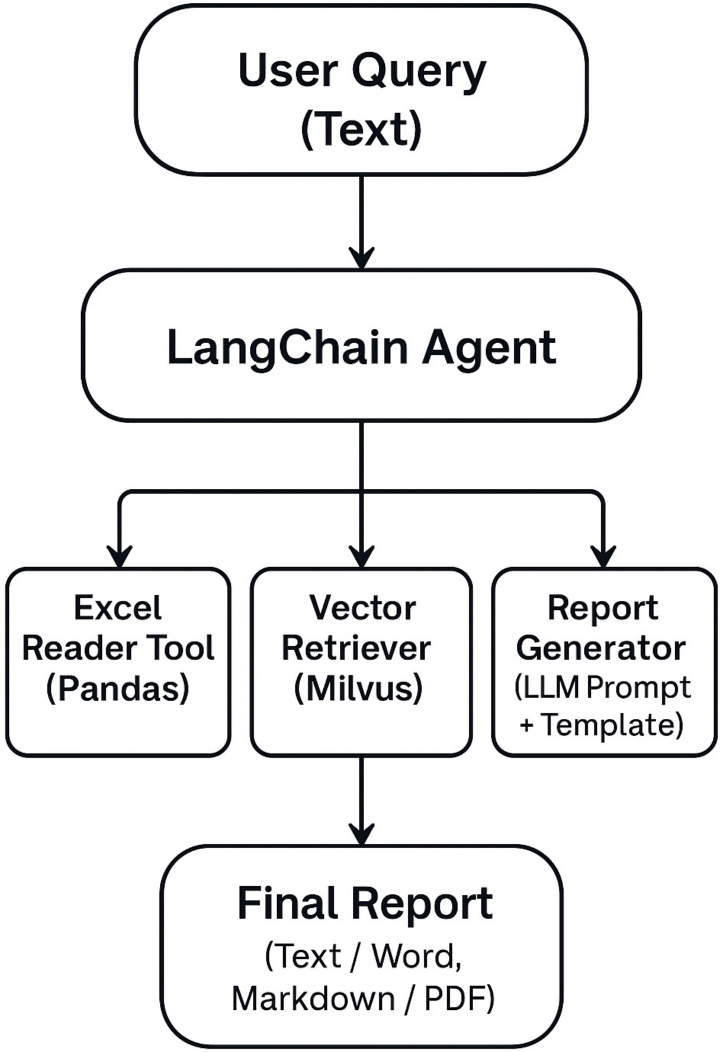
5. Recap: Ethical AI & Data Privacy

- Ethical concerns in LLM usage
- Data anonymization and compliance
- Secure usage of GenAI in enterprise environments

Hackathon Challenge: Report Builder Assistant

Project Objective:

- Build an LLM-based assistant that can:
- Query Excel files (via natural language)
- Generate structured reports from user prompts


User Query
(Text)
LangChain Agent
Excel
Reader Tool
(Pandas)
Vector
Retriever
(Milvus)
Report
Generator
(LLM Prompt
+ Template)
Final Report
(Text / Word,
Markdown / PDF)

Tools to Use

- **LLM Models**: Azure OpenAI or Google GenAI
- **Frameworks**: LangChain
- **Vector DB**: Milvus (mock or real)

Project Title

Report Builder Assistant – Your AI Reporting Partner

Project Objective

Build an LLM-powered assistant that:

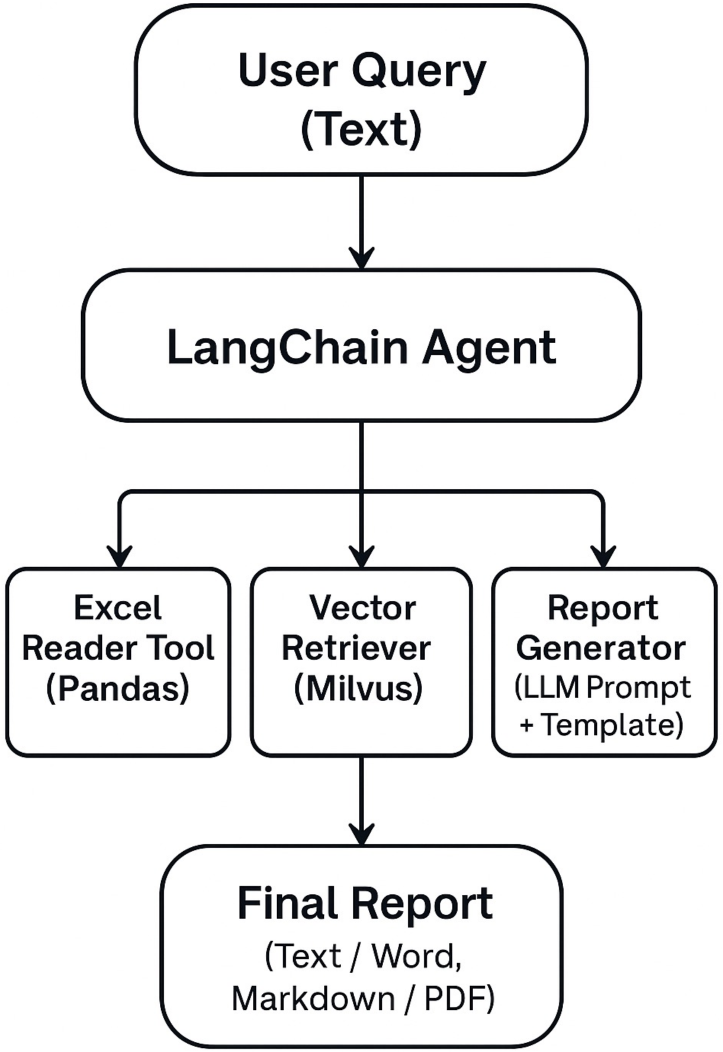
Accepts natural language queries

Reads Excel/CSV files dynamically

Generates structured, human-readable reports

Uses vector search (via Milvus)

for historical context or report templates


User Query (Text)

LangChain Agent

Excel Reader Tool (Pandas)

Vector Retriever (Milvus)

Report Generator (LLM Prompt + Template)

Final Report (Text / Word, Markdown / PDF)

🧰 Tools and Technologies

Component	Tech Stack
LLM Model	Azure OpenAI (gpt-35-turbo) or Google GenAI
Framework	LangChain for orchestration
Data Source	Excel/CSV files
Vector DB	Milvus (mock or real instance)
Libraries	pandas, openpyxl, langchain, pymilvus

✅ Step 1: Install Dependencies

```
pip install openai langchain pandas openpyxl pymilvus matplotlib
```

✅ Step 2: Create Excel File

```
Date,Product,Category,Sales,Store
2024-01-01,Laptop,Electronics,1200,Store_A
2024-01-02,TV,Electronics,3000,Store_B
2024-01-02,Shampoo,Health,200,Store_A
```

Step 3: Build LangChain Tool to Read Excel

```
from langchain.agents import Tool
import pandas as pd

def read_excel(query: str):
    df = pd.read_excel("walmart_sales.xlsx")
    if "total sales" in query.lower():
        return f"Total sales: ₹{df['Sales'].sum()}"
    elif "store" in query.lower():
        return df.groupby("Store")["Sales"].sum().to_string()
    else:
        return df.head().to_string()
```

Step 3: Build LangChain Tool to Read Excel

```
excel_tool = Tool(
   name="ExcelReader",
   func=read_excel,
   description="Use to query walmart_sales.xlsx based on user input"
)
```

Step 4: LangChain Agent (LLM + Tool)

```
from langchain.llms import OpenAI
from langchain.agents import initialize_agent, AgentType

llm = OpenAI(temperature=0.3)

agent = initialize_agent(
    tools=[excel_tool],
    llm=llm,
    agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
    verbose=True
)

agent.run("Show total sales and breakdown by store.")
```

#Step 5: Generate Report from Prompt

```
summary_prompt = f"""
You are a Walmart reporting assistant. Create a business summary based on this:
{read_excel("summary")}
"""

summary = llm(summary_prompt)
print(summary)
```

Step 6: Integrate Milvus for Context

- 📚 **Use Case:**
- Store past reports as embeddings in Milvus
- Retrieve similar summaries to help format/guide current response

Step 6: Integrate Milvus for Context

```
#Embedding + Retrieval Example:

from pymilvus import connections, Collection
from sentence_transformers import SentenceTransformer

# Connect to Milvus (mock or real)
connections.connect("default", host="localhost", port="19530")

# Embed and store past report
model = SentenceTransformer("all-MiniLM-L6-v2")
```

Step 6: Integrate Milvus for Context

```
text = "Store_A had highest sales last month. Recommend restocking."
embedding = model.encode(text).tolist()

# Insert into Milvus (use existing collection schema)
# Retrieve similar reports on demand using cosine similarity
```

Hackathon Deliverables

Item	Description
✅ Functional Agent	Accepts text query, generates report
✅ Excel Tool Integration	Reads and queries structured data
✅ Report Output	Summary in Word/Markdown format
✅ Optional Milvus Setup	Retrieves past report templates

Suggested Queries for Testing

“Summarize total sales by store.”

“What were top 3 selling products in Electronics?”

“Generate a report on January sales trends.”

“Show me health product sales across all stores.”

Judging Criteria

Criteria	Weight	Description
Functionality	40%	Agent works with prompts + Excel input
Innovation	20%	Use of Milvus, visuals, or templating
Usability	20%	Clean, easy-to-use interface (optional)
Code Quality	20%	Modular, readable, well-commented code

Happy Learning !! Thanks for Your Patience ☺

Surendra Panpaliya

GKTCS Innovations