AI-Powered E-commerce Analytics Agent

Project Report

1. Project Overview

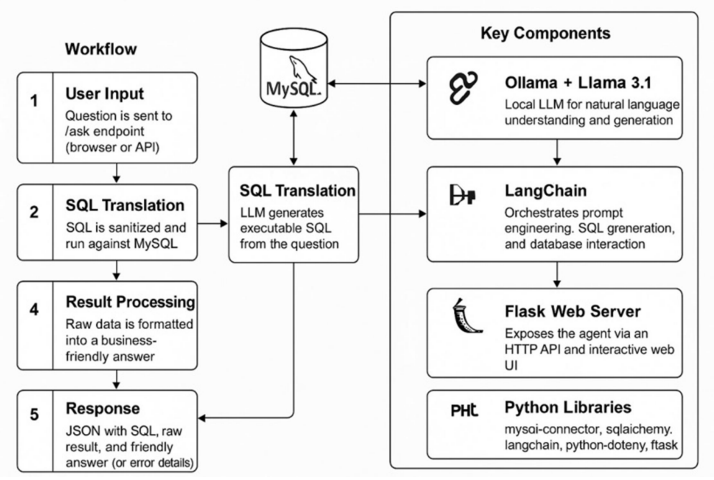
This project develops an **AI-powered analytics agent** that allows business users to query e-commerce data using **natural language**. The system translates questions into SQL automatically, runs queries against a live MySQL database, and returns both **raw results** and a **human-friendly summary**. The solution is built in **Python**, using **Llama 3.1:8b (via Ollama)** for natural language understanding and **LangChain** for database integration. The **Flask API** provides a web interface and REST endpoint, making the agent accessible for both browser and programmatic use.

2. System Architecture

Key Components

- **MySQL Database**: Stores e-commerce metrics (sales, ad spend, impressions, eligibility, etc.).
- **Ollama + Llama 3.1:8b**: Local LLM for natural language understanding and generation.
- **LangChain**: Orchestrates prompt engineering, SQL generation, and database interaction.
- **Flask Web Server**: Exposes the agent via an HTTP API and interactive web UI.
- **Python Libraries**: `mysql-connector`, `sqlalchemy`, `langchain`, `python-dotenv`, `flask`.

Workflow:



1. **User Input**: Question is sent to `/ask` endpoint (browser or API).
2. **SQL Translation**: LLM generates executable SQL from the question.
3. **Query Execution**: SQL is sanitized and run against MySQL.
4. **Result Processing**: Raw data is formatted into a business-friendly answer.
5. **Response**: JSON with SQL, raw result, and friendly answer (or error details).

3. Implementation

Development Steps

- **MySQL Setup**: Imported e-commerce schema (tables: `adsalesandmetrics`, `totalsalesandmetrics`, `eligibility`).
- **Prompt Engineering**: Designed instructions for the LLM to output **plain, executable SQL**.
- **LangChain Integration**: Built a chain that injects schema context and enforces strict SQL-only output.
- **Sanitization Logic**: Removed LLM commentary/non-SQL text from responses.
- **Humanization of Answers**: Used the LLM to rewrite SQL results as **clear, business-ready summaries**.
- **Flask API**: Created endpoints for `/ask` (POST), `/health` (GET), and a homepage with a live query form.

Key Code Files

- **ecommerce_agent.py**: Core logic for SQL generation, sanitization, and answer humanization.
- **api_server.py**: Flask app with API endpoints and web UI.

4. Outcomes

Functionality

- **Natural Language Queries**: Users can ask questions like *"What is the total sales?"* or *"Which product had the highest CPC?"*.
- **Technical and Business Answers**: Returns **raw SQL/json** for analysts and **friendly answers** for business users.
- **Interactive Web UI**: Browser-based interface for testing and exploration.
- **REST API**: Integration-ready endpoint for dashboards or automation.

Sample Output

- **Question**: *"List all eligible products."*
- **SQL**: `SELECT DISTINCT item_id FROM eligibility WHERE eligibility = 'Yes';`
- **Raw Result**: `[('P1001',), ('P1002',)]`
- **Friendly Answer**: *"The eligible products are P1001 and P1002."*

5. Challenges & Solutions

Challenge	Solution
LLM outputs non-SQL text	Strict prompt template + regex-based sanitization
Special chars in MySQL passwords	URL-encode special characters in connection string
Schema changes	LangChain injects live schema; prompt has minimal hardcoded examples
Unknown table/column references	Used LangChain's schema introspection for dynamic context
Lack of conversational answers	Added a separate LLM call to humanize raw results
Flask API usability	Added interactive homepage with clear usage instructions and error handling

6. Technical Highlights

- **Dynamic Schema Awareness**: No need to hardcode table/column names in the prompt.
- **Robust Error Handling**: Clear error messages for users and developers.
- **Debug Output**: Terminal logging for SQL, LLM output, and sanitization steps.
- **Modular Design**: Easy to extend for new databases, LLMs, or output formats.

7. Future Enhancements

- **Visualization**: Detect chart requests, generate and return graphs (matplotlib/plotly).
- **Streaming Answers**: Progressive display of LLM output for long-running queries.
- **Authentication**: Add API keys or OAuth for secure access.
- **Docker Deployment**: Package the app for easy cloud/local deployment.

- **Advanced Prompting**: Support multi-step reasoning or follow-up questions.
- **Integration**: Connect to live e-commerce platforms (Shopify, WooCommerce, etc.).

8. Conclusion

This project demonstrates the feasibility and business value of **natural language interfaces for database analytics**. By combining **local LLMs**, **schema-aware prompting**, and **user-centric answer formatting**, it delivers a **scalable, maintainable AI agent** for e-commerce data exploration. The system is **ready for expansion** into visualization, automation, and enterprise integration.

Appendix A: Environment Variables

(For local setup, add these to `.env`)

```
MYSQL_HOST=localhost
MYSQL_PORT=3306
MYSQL_USER=your_username
MYSQL_PASSWORD=your_password
MYSQL_DATABASE=your_database
OLLAMA_BASE_URL=http://localhost:11434
```

Appendix B: Example API Usage

```
curl -X POST http://localhost:8000/ask \
  -H "Content-Type: application/json" \
  -d '{"question": "What is the total sales across all products?"}'
```

For source code, setup instructions, or further technical details, see the GitHub repository or contact the project team.