

Official Documentation

# AI Shopping Assistant

## CICD / LLMOps

**Developed By:** Aditya Dinesh K (developer@aydie.in)

**Date of Completion:** Sep 2nd, 2025

# AI Shopping Assistant

A full-stack, LLM-powered laptop recommendation system that combines intelligent data ingestion pipelines with an interactive chatbot workflow. The system helps users find personalised laptop recommendations based on their preferences through natural language conversations.

## Project Overview

AI Shopping Assistant is a production-ready system that demonstrates advanced AI integration in e-commerce applications. The project features two distinct but interconnected workflows:

1. **Data Ingestion Pipeline:** An admin-triggered, automated workflow for processing and mapping laptop datasets
2. **Chatbot Interface:** An intelligent conversational agent that provides personalised laptop recommendations

The system is deployed on AWS Lambda and HuggingFace Spaces, integrating with Amazon S3 for storage, PostgreSQL (AIVEN) and Cloudflare D1 for data persistence, and OpenAI's GPT-4o-mini models for natural language processing.

- **Live Demo:** [AI Shopping Assistant](#)
- **Source Code:** [GitHub Repository](#)
- **Home Page:** [www.aydie.in](http://www.aydie.in)

## Tech Stack

### Backend Framework

- **Flask 3.0.3** - Main web framework (entry point: app.py)

### Databases

- **AIVEN PostgreSQL** - Primary storage for mapped laptop data
- **Cloudflare D1 SQL** - Alternative serverless database option
- **SQLAlchemy 2.0.36** - ORM for database abstraction
- **psycopg2 2.9.10** - PostgreSQL client

### Cloud & Storage

- **Amazon S3** - Raw CSV file storage
- **AWS Lambda** - Serverless backend deployment
- **HuggingFace Spaces** - Application hosting
- **boto3** - AWS SDK

## AI/LLM APIs

- **OpenAI API 1.102.0** - Primary language model (GPT-4o-mini)
- **Google Generative AI 0.7.2** - Optional Gemini integration
- **LangChain 0.2.16** - LLM orchestration (optional)
- **LlamaIndex 0.12.38** - Retrieval and indexing (optional)

## Data Processing

- **Pandas 2.2.2** - Data manipulation
- **NumPy 1.26.4** - Numerical computing
- **PandasSQL 0.7.3** - SQL query engine for pandas
- **PyArrow 17.0.0** - Parquet file handling
- **FastParquet 2024.5.0** - High-performance parquet operations

## Deployment & Production

- **Gunicorn 22.0.0** - WSGI server
- **Mangum 0.19.0** - AWS Lambda adapter

## Architecture & Workflow

The system implements two distinct pipelines that share core functionality through the orchestrator.py module:

### 1. Data Ingestion Pipeline (Admin Workflow)

**Purpose:** Maintain a fresh, pre-processed laptop recommendation database

#### Trigger Methods:

- Web interface via ingestor.py with file upload through upload.html
- Direct function call via Orchestrator.start\_internal\_data\_ingestion()

#### Process Flow:

CSV Upload → S3 Storage → Data Retrieval → LLM Mapping → Database Update

#### Key Components:

- 1. File Upload** (DataIngestion.start\_data\_ingestion())
  - Accepts CSV files containing laptop specifications
  - Uploads to Amazon S3 bucket via AWSConnection.upload\_file\_to\_s3()
  - Real-time progress tracking through /upload-status endpoint
- 2. Data Processing** (ProductMapper.start\_dataframe\_product\_mapping())
  - Reads structured data using parquet files for efficiency
  - Processes laptop descriptions through OpenAI GPT-4o-mini API
  - Handles batch processing with error recovery
- 3. Specification Mapping** (ProductMapper.do\_product\_mapping())
  - Maps laptop features to standardized categories using OpenAI GPT models:

- GPU Intensity: Low (integrated graphics) / Medium (mid-range dedicated) / High (high-end dedicated like RTX)
- Display Quality: Low (below Full HD) / Medium (Full HD+) / High (4K, HDR)
- Portability: High (<1.51kg) / Medium (1.51-2.51kg) / Low (>2.51kg)
- Multitasking: Low (8-12GB RAM) / Medium (16GB RAM) / High (32GB+ RAM)
- Processing Speed: Low (i3, Ryzen 3) / Medium (i5, Ryzen 5) / High (i7+, Ryzen 7+)

#### 4. **Database Update** (PostgresDataBaseUpdate.update\_to\_postgres\_database())

- Supports both PostgreSQL (AIVEN) and Cloudflare D1 databases
- Drops existing table for complete refresh
- Creates new schema based on DataFrame structure
- Batch inserts all processed data with JSON handling for mapped columns

## 2. Chatbot Workflow (User Interaction)

**Purpose:** Provide personalised laptop recommendations through conversational AI

### Conversation States:

- normal - Standard conversation flow
- awaiting\_feedback - Waiting for user satisfaction confirmation
- awaiting\_rating - Collecting user rating (1-5 scale)
- ended - Conversation terminated

### Process Flow:

User Input → Moderation → Profile Building → Intent Confirmation → Recommendation → Feedback

### Key Functions:

#### 1. **Input Processing** (/chat endpoint in app.py)

- Greeting detection via get\_chat\_completion() with OpenAI GPT-4o-mini
- Content moderation using moderation\_check() with omni-moderation-latest
- JSON filtering through filter\_json\_from\_response()

#### 2. **Profile Building** (Orchestrator.get\_chat\_completion())

- Extracts user preferences through guided conversation
- Uses system instruction prompts from SystemInstruction.system\_instruction
- Builds a comprehensive user profile dictionary:

python

```
{ 'GPU intensity': 'low/medium/high',
  'Display quality': 'low/medium/high',
  'Portability': 'low/medium/high',
  'Multitasking': 'low/medium/high',
  'Processing speed': 'low/medium/high',
```

```
'Budget': 'numerical_value'
}
```

3. **Intent Confirmation** (Orchestrator.intent\_confirmation\_check())
  - Validates profile completeness using IntentConfirmation.intent\_confirmation prompt
  - Ensures all 6 required attributes are captured
  - Returns JSON response with confirmation status
4. **Recommendation Engine** (ProductRecommendation.recommend\_product())
  - Queries database via LoadFromDatabase.fetch\_query\_engine\_data()
  - Filters by budget using QueryEngine.filter\_budget()
    - Calculates match scores via QueryEngine.filter\_by\_user\_score() and ProductMapper.map\_the\_score()
  - Returns top 3 recommendations with detailed specifications
5. **Response Generation**
  - Formats recommendations using ProductRecommender.system\_message prompt
  - Handles satisfaction feedback through /feedback endpoint
  - Routes to human agent via route\_to\_human\_agent() if user is unsatisfied
  - Collects ratings through /rate endpoint for continuous improvement

## Data Ingestor UI



## ChatBot UI

### About this demo

This is my portfolio project, this uses OpenAI API.

If this demo stops working then my API credit has been exhausted, since I am a college student I can't afford to run this on large scale.

**For source code visit:**

[github.com/aydiegithub/ai-shop-assistant.git](https://github.com/aydiegithub/ai-shop-assistant.git)

**Documentation:** [aydie.in/llm/ai-shop-assistant-documentation.pdf](https://aydie.in/llm/ai-shop-assistant-documentation.pdf)

**Website:** [www.aydie.in](https://www.aydie.in)

**My blogs:** [blogs.aydie.in](https://blogs.aydie.in)

**All projects:** [aydie.in/projects](https://aydie.in/projects)

**LinkedIn:** [linkedin.com/in/aydiemusic](https://linkedin.com/in/aydiemusic)

**Email:** [developer@aydie.in](mailto:developer@aydie.in) or [aditya@aydie.in](mailto:aditya@aydie.in)



### Shop Assist Bot

Laptop related only since it is a demo.

Clear

I want to buy a laptop for video editing

Video editing requires a multi-faceted approach to hardware, including the need for strong GPU performance, high display quality, and good processing speed.

Could you provide more details on your editing needs? Specifically, do you primarily work with high-resolution media files, such as 4K videos? Also, how important is multitasking for you when editing? Lastly, what is your budget for the laptop?

I do 4K video editing and also my budget is somewhere around one lakh rupees

Thank you for the information! Editing 4K videos will indeed require a high-performance laptop with a strong GPU and processing capabilities.

To summarize what we've established so far:

- GPU intensity: High, since you are working with 4K video editing.
- Processing speed: High, required for rendering and editing.
- Display quality: High, necessary for accurate color representation during editing.
- Multitasking: How often do you typically have multiple software applications open at the same

Type your message here...



### About this demo

This is my portfolio project, this uses OpenAI API.

If this demo stops working then my API credit has been exhausted, since I am a college student I can't afford to run this on large scale.

**For source code visit:**

[github.com/aydiegithub/ai-shop-assistant.git](https://github.com/aydiegithub/ai-shop-assistant.git)

**Documentation:** [aydie.in/llm/ai-shop-assistant-documentation.pdf](https://aydie.in/llm/ai-shop-assistant-documentation.pdf)

**Website:** [www.aydie.in](https://www.aydie.in)

**My blogs:** [blogs.aydie.in](https://blogs.aydie.in)

**All projects:** [aydie.in/projects](https://aydie.in/projects)

**LinkedIn:** [linkedin.com/in/aydiemusic](https://linkedin.com/in/aydiemusic)

**Email:** [developer@aydie.in](mailto:developer@aydie.in) or [aditya@aydie.in](mailto:aditya@aydie.in)



### Shop Assist Bot

Laptop related only since it is a demo.

Clear

Thank you for your responses! Based on the information you've provided, I'll summarize your preferences:

- GPU intensity: High (for 4K video editing)
- Display quality: Medium (as you use an external monitor)
- Portability: Low (as you travel frequently, but it's not a priority)
- Multitasking: High (you mentioned wanting strong multitasking capabilities)
- Processing speed: High (needed for rendering and editing)
- Budget: 100,000 (1 lakh rupees)

Now, I can confidently fill out your laptop profile.

Here are the top 2 recommended laptops based on the provided information:

1. **Acer Predator:** A powerhouse laptop designed for gaming and high-performance tasks, featuring a powerful processor, dedicated graphics card, and high refresh rate display, Rs 80,000.
2. **Acer Swift 3:** A lightweight and affordable laptop ideal for everyday use, equipped with a decent processor, good battery life, and a sleek design, Rs 50,000.

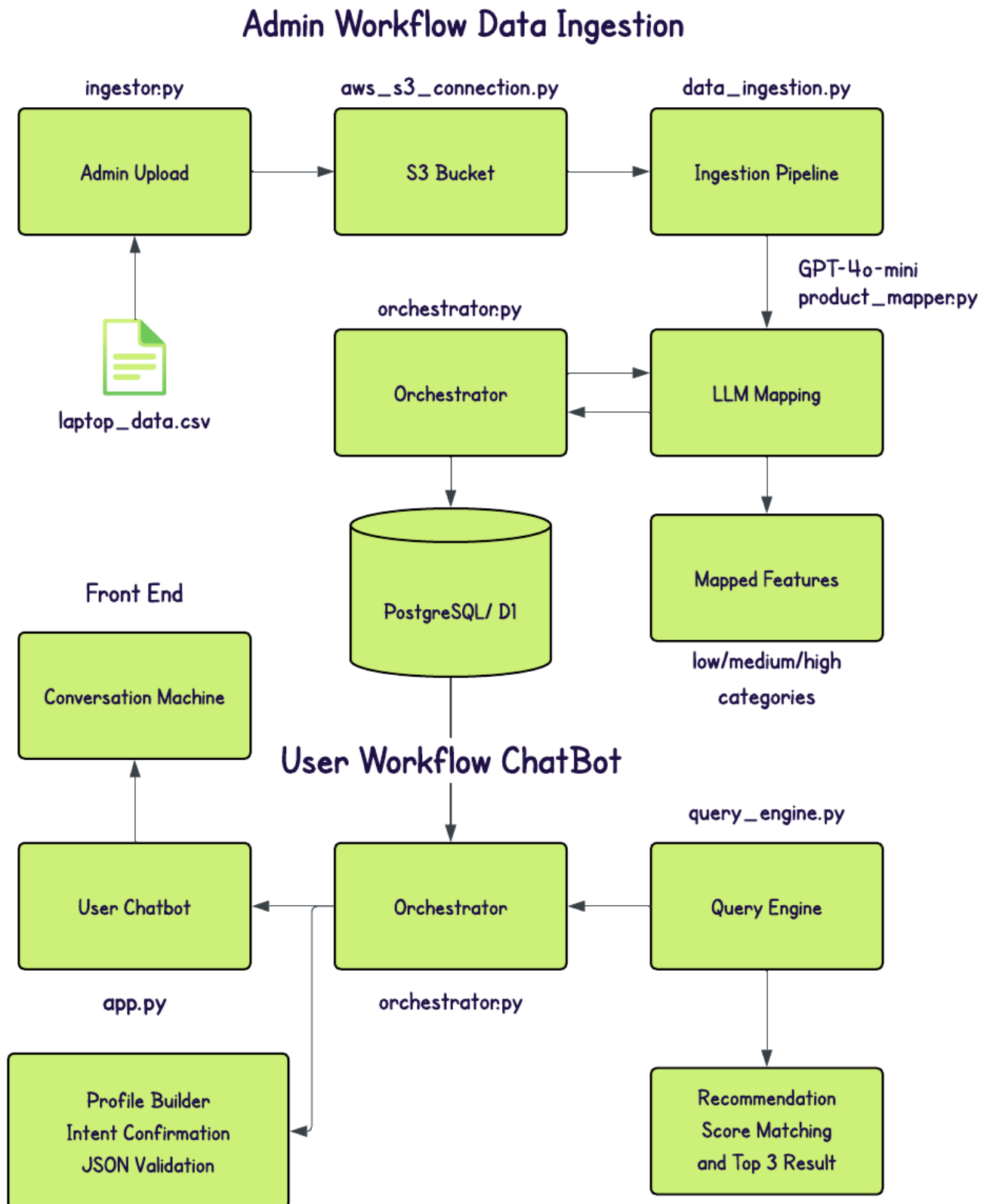
If you have any specific queries about these laptops or need further assistance, feel free to ask!

Hope I have solved your request. Did this help you? (yes/no)

Type your message here...



## Data Flow Diagram



# Setup Instructions

## Prerequisites

- Python 3.8+
- PostgreSQL database (AIVEN recommended) or Cloudflare D1
- AWS S3 bucket with appropriate permissions
- OpenAI API key with GPT-4o-mini access

## Installation

### 1. Clone the repository

```
git clone https://github.com/aydiegithub/ai-shopping-assistant.git
```

### 2. Install dependencies

```
pip install -r requirements.txt
```

### 3. Environment Configuration

Create a .env file with the following variables:

```
# OpenAI Configuration
OPENAI_API_KEY=your_openai_api_key

# AWS Configuration
AWS_ACCESS_KEY_ID=your_aws_access_key
AWS_SECRET_ACCESS_KEY=your_aws_secret_key
S3_BUCKET_NAME=your_s3_bucket_name
AWS_DEFAULT_REGION=your_region

# PostgreSQL Configuration (AIVEN)
POSTGRES_HOST=your_postgres_host
POSTGRES_PORT=your_port_number
POSTGRES_DB_NAME=your_database_name
POSTGRES_USER=your_postgres_user
POSTGRES_PASSWORD=your_postgres_password
POSTGRES_TABLE_NAME=table_name

# Cloudflare D1 Configuration (Optional)
CLOUDFLARE_ACCOUNT_ID=your_cloudflare_account_id
D1_SQL_DATABASE_ID=your_d1_database_id
CLOUDFLARE_API_TOKEN=your_cloudflare_api_token
D1_SQL_DATABASE_NAME=your_d1_database_name
```



## 4. Database Setup

```
from src.backend.orchestrator import Orchestrator

orchestrator = Orchestrator()

orchestrator.start_internal_data_ingestion(local_file_path,
                                           s3_file_name)
```

## 5. Run the application

```
python app.py          # UI for end user
python ingestor.py     # UI for admin for data ingestion
gunicorn -w 4 -b 0.0.0.0:8000 app:app # Production server
```

## Deployment

AWS Lambda Deployment:

The application includes the Mangum WSGI adapter for seamless Lambda deployment:

```
from mangum import Mangum
handler = Mangum(app)
```

HuggingFace Spaces:

Configure app.py with the appropriate port settings:

```
port = int(os.environ.get("PORT", 7860))
app.run(host='0.0.0.0', port=port, debug=True)
```

## Usage

### Data Ingestion (Admin)

1. Web Interface:
 

```
bash python ingestor.py # Navigate to http://localhost:5000 # Upload CSV files
                        # Monitor progress via real-time status updates
```
2. Programmatic Access:
 

```
python from src.backend.orchestrator import Orchestrator    orch = Orchestrator()
orch.start_internal_data_ingestion(    local_file_path="src/database/laptop_data.csv",
                                       s3_file_name="laptop_data.csv" )
```

### Chatbot Interaction (End User)

1. **Start Conversation:** Send greeting to /chat endpoint
2. **Profile Building:** Answer guided questions about laptop needs
3. **Receive Recommendations:** Get personalized top 3 suggestions
4. **Provide Feedback:** Rate assistance (1-5) and optionally escalate to human agent

## Example Conversation Flow

POST /chat

```
{
  "message": "Hi, I need help finding a laptop",
  "messages": [],
  "state": "normal"
}
```

Response:

```
{
  "message": "Hello there! I am here to help you. I am your personal laptop assistant. What kind of laptop are you looking for?",
  "messages": [...],
  "state": "normal"
}
```

POST /chat

```
{
  "message": "I'm a video editor working with 4K content in After Effects",
  "messages": [...],
  "state": "normal"
}
```

Response:

```
{
  "message": "Great! Working with After Effects and 4K content requires high GPU intensity and processing power. Do you need portability for travel work?",
  "messages": [...],
  "state": "normal"
}
```

POST /chat

```
{
  "message": "Mostly stationary work, budget around 150000 INR",
  "messages": [...],
  "state": "normal"
}
```

Response:

```
{
  "message": "{ 'GPU intensity': 'high', 'Display quality': 'high', 'Portability': 'low', 'Multitasking': 'high', 'Processing speed': 'high', 'Budget': '150000' }\n\nBased on your requirements for 4K video editing:\n\n1. **ASUS ROG Strix G15**": High-performance gaming laptop with
```

RTX 3070, 16GB RAM, AMD Ryzen 7, 15.6\" 144Hz display - Rs. 145,000\n2.  
\*\*MSI Creator 15\*\*:  
Content creation focused with RTX 3060, 32GB RAM, Intel i7, 15.6\" 4K display - Rs. 149,999\n3.  
\*\*HP OMEN 15\*\*:  
Gaming laptop with RTX 3060, 16GB RAM, AMD Ryzen 7, 15.6\" 165Hz display - Rs. 139,000\n\nHope I have solved your request. Did this help you? (yes/no)\",  
 "messages": [...],  
 "state": "awaiting\_feedback"  
}

## Dependencies

### Core Web Framework

flask==3.0.3  
python-dotenv==1.0.1

### Database & ORM

SQLAlchemy==2.0.36  
sqlalchemy-cloudflare-dl==0.1.0  
psycopg2==2.9.10

### Data Handling

pandas==2.2.2  
numpy==1.26.4  
pandasql==0.7.3  
pyarrow==17.0.0  
fastparquet==2024.5.0

### LLM/AI APIs

openai==1.102.0  
google-generativeai==0.7.2  
llama-index==0.12.38  
langchain==0.2.16  
langchain-community==0.2.16  
tenacity

### Cloud & AWS/Cloudflare Libraries

boto3>=1.26.0,<2  
cloudflare>=3.0.0,<4  
mangum==0.19.0

## HTTP Requests & Production

requests==2.32.3  
gunicorn==22.0.0

## Data Validation

pydantic>=2.7.4,<3  
pydantic-core>=2.16.3,<3

## Development Tools

black==24.8.0  
pytest==8.3.3

## API Endpoints

### Chatbot Endpoints (app.py)

- GET / - Main chatbot interface
  - Serves index.html template
  - Returns: HTML chatbot interface
- POST /chat - Main conversation endpoint
  - Body: {"message": "user\_input", "messages": [], "state": "normal"}
  - Returns: {"message": "bot\_response", "messages": [...], "state": "normal|awaiting\_feedback"}
  - Functions: Greeting detection, moderation, profile building, intent confirmation
- POST /feedback - User satisfaction feedback
  - Body: {"message": "yes/no", "messages": [...]}
  - Returns: Rating prompt or human agent escalation
  - Functions: Satisfaction assessment, routing logic
- POST /rate - User rating submission (1-5 scale)
  - Body: {"message": "1-5", "messages": [...]}
  - Returns: {"message": "Thank you for your feedback! Chat ended.", "state": "ended"}

## Data Ingestion Endpoints (ingestor.py)

- GET / - Upload interface
  - Serves upload.html template
  - Returns: File upload interface with progress tracking
- POST /upload-csv - CSV file upload and processing
  - Body: multipart/form-data with file
  - Returns: {"message": "Upload started", "status": "processing"}
  - Functions: File validation, S3 upload, LLM mapping, database update
- GET /upload-status - Real-time upload progress
  - Returns: {"status": "uploading|processing|completed|error", "progress": 0-100, "message": "...", "logs": [...]}
- GET /logs - Recent processing logs
  - Returns: {"logs": [{"timestamp": "HH:MM:SS", "message": "...", "level": "info|error"}]}

## Core Functions Reference

### Orchestrator (src/backend/orchestrator.py)

- initialise\_conversation() - Sets up system instruction for new chats
- get\_chat\_completion(input\_messages, json\_format=False) - OpenAI API wrapper with retry logic
- moderation\_check(input\_message) - Content safety validation using omni-moderation-latest
- intent\_confirmation\_check(input\_message) - Profile completeness validation
- start\_product\_recommendation(input\_message) - Triggers recommendation pipeline
- set\_user\_profile(message) - Extracts user preferences from conversation
- route\_to\_human\_agent(input\_message) - Escalation logic for unsatisfied users

### Product Mapper (src/backend/product\_mapper.py)

- do\_product\_mapping(laptop\_description) - Maps a single laptop description to categories using GPT-4o-mini
- start\_dataframe\_product\_mapping(df) - Batch processes entire DataFrame with parquet optimisation
- map\_the\_score(mapped\_column, user\_profile) - Calculates compatibility scores

## Query Engine (src/backend/query\_engine.py)

- filter\_budget(data, criteria) - Filters laptops by budget constraints
- filter\_by\_user\_score(data, user\_profile) - Ranks laptops by compatibility score

## Product Recommendation (src/backend/product\_recommender.py)

- recommend\_product(user\_profile) - End-to-end recommendation pipeline
- calculate\_score(mapping\_column, user\_profile) - Score calculation wrapper

## Contributing

1. Fork the repository
2. Create a feature branch (git checkout -b feature/AmazingFeature)
3. Commit your changes (git commit -m 'Add some AmazingFeature')
4. Push to the branch (git push origin feature/AmazingFeature)
5. Open a Pull Request

## Development Guidelines

- Follow PEP 8 style guidelines (use black==24.8.0 for formatting)
- Add comprehensive logging using the project's logging framework
- Write unit tests for new functionality using pytest==8.3.3
- Update documentation for API changes
- Ensure all environment variables are documented
- Test both ingestion and chatbot workflows before submitting

## Project Structure

ai-shopping-assistant/

```

├── app.py                # Main Flask chatbot application
├── ingestor.py           # Admin data ingestion interface
├── requirements.txt      # Python dependencies
├── .env                  # Environment variables (not tracked)
├── .gitignore            # Git ignore rules
├── LICENSE               # Apache 2.0 License
├── README.md             # Project documentation
├── Dockerfile            # Container configuration
├── setup.py              # Package setup
├── diagnostics.py        # System diagnostics
├── templates.py          # Template utilities
├── test.py               # Test utilities
├── tester.ipynb          # Jupyter notebook for testing
├── src/
│   ├── backend/
│   │   ├── __init__.py    # Backend module initialization
│   │   ├── orchestrator.py # Central coordination logic
│   │   ├── data_ingestion.py # Data pipeline management
│   │   ├── product_mapper.py # LLM-based feature mapping
│   │   ├── product_recommender.py # Recommendation engine
│   │   ├── query_engine.py # Database querying and filtering
│   │   └── prompts.py      # LLM prompt templates
│   ├── constants/
│   │   └── __init__.py    # Configuration constants and env var
│   ├── database/
│   │   ├── aws_s3_connection.py # S3 storage operations
│   │   ├── aiven_postgresql_update.py # PostgreSQL operations
│   │   ├── cloudflare_connection.py # Cloudflare D1 database ops
│   │   ├── load_from_database.py # Data retrieval operations
│   │   ├── laptop_data.csv      # Sample dataset
│   │   └── laptop_data_mapped.parquet # Pre-processed mapped data
│   ├── frontend/
│   │   ├── static/
│   │   │   ├── main.js        # Frontend JavaScript
│   │   │   └── style.css      # Frontend styling
│   │   └── templates/
│   │       ├── logo/          # Brand assets
│   │       ├── index.html     # Chatbot interface
│   │       └── upload.html     # Admin upload interface
│   ├── logging/
│   │   └── __init__.py        # Logging configuration
│   └── utils/
│       └── __init__.py        # Utility functions

```

## License

This project is licensed under the Apache 2.0 License. See the [LICENSE](#) file for details.

## Author & Contact

**Aditya (Aydie) Dinesh K**

- **LinkedIn:** [linkedin.com/in/aydiemusic](https://linkedin.com/in/aydiemusic)
- **Github:** [github.com/aydiegithub/](https://github.com/aydiegithub/)
- **Website:** [www.aydie.in](https://www.aydie.in)
- **Blogs:** [blogs.aydie.in](https://blogs.aydie.in)
- **Email:** [developer@aydie.in](mailto:developer@aydie.in) | [aditya@aydie.in](mailto:aditya@aydie.in)

*This project demonstrates advanced AI integration patterns in production environments, showcasing best practices for LLM-powered applications, data pipeline automation, and scalable cloud deployment. The system serves as a comprehensive example of modern AI-driven e-commerce solutions with real-world deployment considerations.*