E-Commerce Agentic Multimodal Sentiment Analysis System

Technical Documentation

By Pham Tran Yen Quyen

October 2, 2025

Contents

Contents 2

1 Overview 3
- 1.1 Key Objectives . . . 3

2 System Architecture 3
- 2.1 Conceptual Flow . . . 3
- 2.2 Microservice Diagram . . . 4
- 2.3 Service Roles . . . 4

3 Core Components & Agentic Systems 5
- 3.1 Intelligent Data Acquisition Agent . . . 5
- 3.2 Advanced Data Synthesis Agent . . . 5
- 3.3 Multimodal Sentiment Analysis with LLaVA . . . 6
  - 3.3.1 True Multimodal Understanding for Nuanced Sentiment . . . 6
  - 3.3.2 Advanced Visual Instruction Tuning . . . 6
- 3.4 Automated Labeling Agent . . . 7
- 3.5 Hybrid Retrieval System . . . 7

4 MLOps and Evaluation 8
- 4.1 Automated Training Pipeline . . . 8
- 4.2 Systematic Evaluation Framework . . . 8

5 Current Status and Implementation Gaps 9
- 5.1 Deployment Options . . . 9
  - 5.1.1 Option A: Minimal Demo . . . 9
  - 5.1.2 Option B: Full System . . . 9
- 5.2 Demo Sentiment Classification . . . 9
- 5.3 Implementation Status and Limitations . . . 10
- 5.4 Required Configuration . . . 10
- 5.5 Data and Model Bootstrapping . . . 10
- 5.6 Infrastructure Dependencies . . . 11

6 Technologies Stack 11

7 Setup and Usage 11
- 7.1 Prerequisites . . . 13
- 7.2 Local Development . . . 13

8 References 14

1 Overview

An end-to-end, production-ready agentic AI system for multimodal sentiment analysis. This system autonomously synthesizes review text and images, uses this data to train advanced Vision-Language Models, and provides an API for sentiment analysis and hybrid retrieval.

1.1 Key Objectives

The system is designed to achieve the following primary objectives:

1. **Autonomous Data Acquisition**: To intelligently scrape and collect multimodal review data from diverse public sources, learning over time which sources provide the most value.
2. **Advanced Data Synthesis**: To generate realistic, domain-specific review text and product images to augment the limited seed data, thereby creating a large-scale, balanced training dataset.
3. **Deep Multimodal Understanding**: To develop and train a Vision-Language Model (VLM) capable of jointly interpreting text and images to capture nuanced sentiment that unimodal models would miss.
4. **High-Precision Retrieval**: To implement a state-of-the-art hybrid retrieval system that can find semantically similar reviews based on text, images, or a combination of both, using a fusion of dense and sparse search techniques.
5. **End-to-End Automation**: To build a production-ready, microservice-based system that automates the entire pipeline, from data collection to model training, evaluation, and deployment, ensuring scalability and maintainability.

2 System Architecture

The system is designed as a distributed network of containerized microservices, orchestrated by Docker Compose for local development and designed for Kubernetes in a production environment. This architecture ensures scalability, resilience, and maintainability.

2.1 Conceptual Flow

The architecture is divided into two main parts: the online **Inference Stack** and the offline **Agentic Pipeline**.

- **Inference Stack**: Handles real-time user requests. A user interacts with the frontend, which sends requests to an API Gateway. The gateway routes the request to the appropriate service, such as the Inference Service for sentiment analysis, which in turn queries the vector and keyword databases.
- **Agentic Pipeline**: An automated, offline workflow orchestrated by a control agent (Prefect). This pipeline continuously runs in the background to gather new data, synthesize additional data, retrain models, and deploy improved versions without any human intervention.

2.2 Microservice Diagram

The diagram below illustrates the high-level interaction between the various microservices.

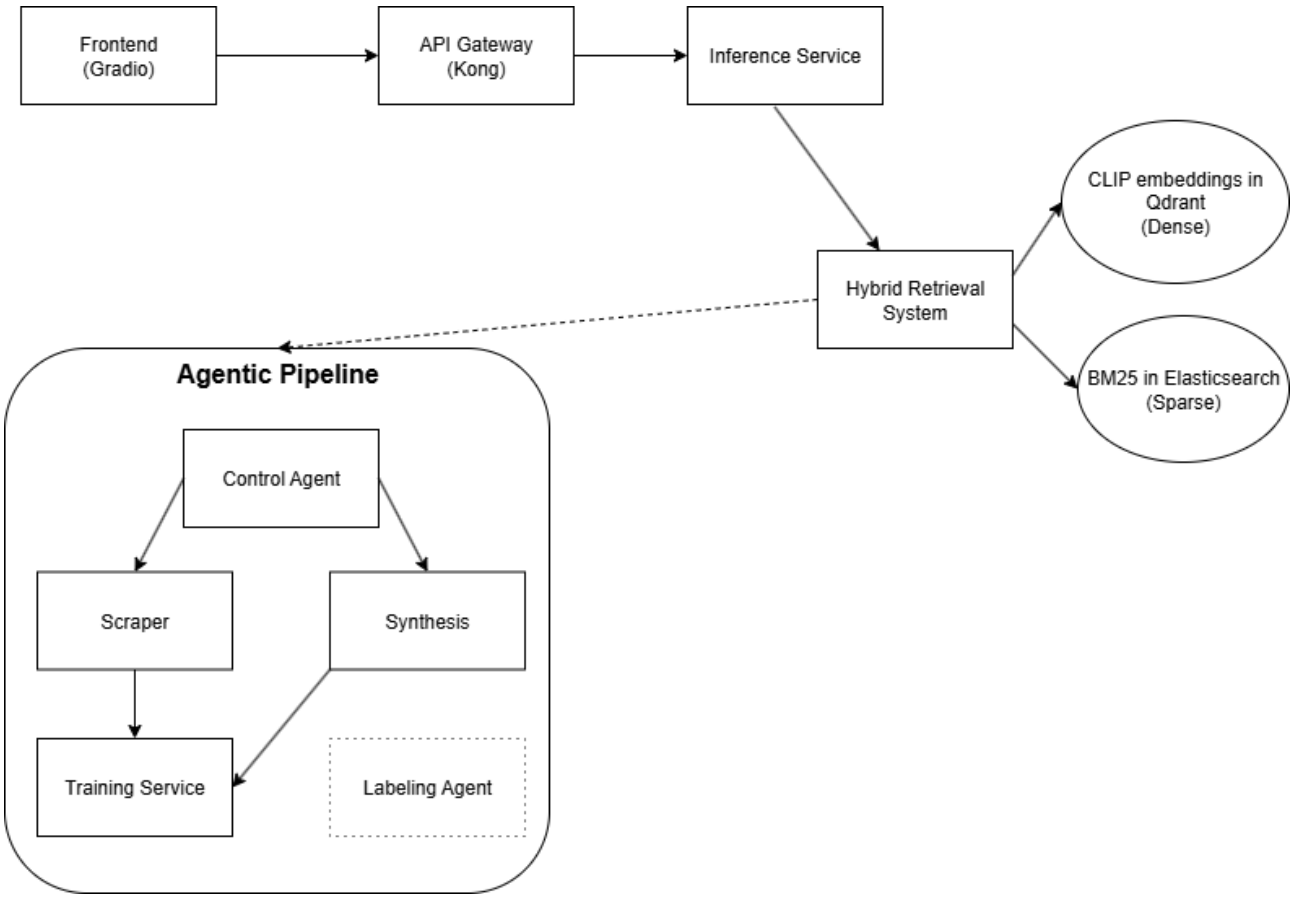

Figure 1: Microservice Diagram

2.3 Service Roles

Each component, as defined in the `docker-compose.yml`, has a distinct responsibility:

- **API Gateway (Kong)**: The single, unified entry point for all external traffic. It handles request routing, rate limiting, and authentication, directing traffic to the appropriate backend service.
- **Inference Service**: A FastAPI application that exposes the core `/analyze` endpoint. It orchestrates calls to the sentiment model and the retrieval system. For the local demo, this service uses a placeholder rule-based sentiment classifier and the CLIP model for embeddings, as loading a full-scale fine-tuned LLaVA model is too resource-intensive for a typical local machine.
- **Data Ingestion Agent**: An autonomous agent responsible for scraping multimodal data from the web. It uses an LLM to decide which sites to scrape and dynamically discovers data selectors.
- **Data Synthesis Agent**: Generates high-quality synthetic text and images to augment the training data, with controllable attributes like sentiment.
- **Training Service**: Manages the distributed training of the sentiment model using frameworks like DeepSpeed, tracks experiments with MLflow, and versions models.

- **Databases**: The system utilizes a polyglot persistence approach:
  - **Qdrant**: A vector database for high-speed dense retrieval of similar embeddings.
  - **Elasticsearch**: A search engine for sparse, keyword-based retrieval (BM25).
  - **PostgreSQL**: Used for storing structured metadata and serving as a backend for Kong and MLflow.
  - **Redis**: An in-memory cache to store frequent inference results, reducing latency.
- **Monitoring Stack**: Prometheus for collecting metrics from all services and Grafana for visualizing system health, performance, and model drift.

3 Core Components & Agentic Systems

The innovation of this project lies in its collection of intelligent agents and advanced ML components that work in concert to create a self-sufficient system.

3.1 Intelligent Data Acquisition Agent

The ingestion agent is designed to be more than a simple scraper. It is an autonomous system that learns and adapts, as implemented in `openai_webscraping_agent.py`.

- **LLM-Guided Strategy**: Before each run, the agent queries an LLM (OpenAI GPT-4o) with its current state, including past successes and failures stored in a 'strategy_memory.json' file on a MinIO S3 bucket. The LLM then decides the next best action, such as scraping a new site or deep-crawling a previously successful one.
- **Dynamic Selector Discovery**: For a given URL, the agent sends a snippet of the page's HTML to an LLM, asking it to identify the most likely CSS selectors for review containers. This allows the agent to adapt to different website structures without hardcoded rules, moving beyond brittle, manually defined selectors.
- **Resilient Scraping**: It uses Playwright to handle modern, JavaScript-heavy websites, ensuring that dynamically loaded content is captured correctly. This is crucial for scraping e-commerce sites that rely heavily on client-side rendering.
- **Data Provenance**: All scraped data is stored in an S3 bucket with a content-based hash as its ID to prevent duplication. Metadata, including the source URL and timestamp, is preserved for auditability.

3.2 Advanced Data Synthesis Agent

To overcome the data bottleneck, this agent generates high-fidelity synthetic data that is indistinguishable from real reviews, detailed in `controllable_synthetic_agent.py`.

- **Controllable Text Generation**: The agent uses a fine-tuned instruction-following LLM (Ex: Llama 3.1) to generate review text. The generation is guided by prompts that specify the target sentiment, product name, and desired review length. It leverages domain patterns extracted from seed data for in-context learning to ensure the style is authentic. The local demo provides a mock implementation of this feature to avoid the heavy computational cost.

- **Sentiment-Conditioned Image Generation**: Using a diffusion model like SDXL, the agent generates product images with visual cues that match the text's sentiment. For example, a positive review might be paired with an image of a pristine product in a studio setting, while a negative review could be accompanied by an image of a damaged item in poor lighting.
- **Adversarial Validation**: A key feature is the `AdversarialValidator`. This is a discriminator model (a BERT-based binary classifier) trained to distinguish between real and synthetic reviews. The quality of the synthetic data is measured by its ability to "fool" this discriminator. The goal is to generate data so realistic that the discriminator's accuracy is no better than chance ($\approx 50\%$), indicating that the synthetic data is of high quality.

3.3 Multimodal Sentiment Analysis with LLaVA

LLaVA (Large Language and Vision Assistant) model was chosen as the core component for sentiment analysis for several critical reasons:

3.3.1 True Multimodal Understanding for Nuanced Sentiment

The primary challenge in e-commerce sentiment analysis is that text alone does not always tell the full story. LLaVA is fundamentally designed to jointly interpret text and images, which is essential for accurately capturing customer sentiment.

- **Architecture**: LLaVA combines a powerful pre-trained vision encoder (like CLIP) with a large language model (LLM). The vision encoder processes the image and converts it into a format that the LLM can understand, effectively allowing the model to "see" and "read" at the same time.
- **Contextual Analysis**: This architecture enables the model to understand complex scenarios that would confuse a text-only model. For example:
  - **Sarcasm**: A user writes "This is perfect" alongside a photo of a shattered product. LLaVA can identify the contradiction between the positive text and the negative visual cue to correctly classify the sentiment as negative.
  - **Descriptive Mismatches**: A review states, "The color is not what I expected." LLaVA can analyze the product image to understand the context of the complaint.

As stated in the project's `README.md`, the goal is to achieve "joint text-image sentiment analysis" and capture "nuanced sentiment from text-image interactions," which is precisely what LLaVA's architecture enables.

3.3.2 Advanced Visual Instruction Tuning

LLaVA is not just a model that processes two types of data; it's an instruction-following model. This allows it to be fine-tuned for specific, targeted tasks in a highly effective way. The project's training pipeline, detailed in `distributed_training.py`, leverages this capability directly. The data is formatted into a conversational prompt before being fed to the model:

```
prompt = f"USER: <image>\nWhat is the sentiment of this product review:
'{item['text']}'?\nASSISTANT: {item['sentiment']}"
```

By fine-tuning on thousands of such examples, the model learns to perform the specific task of multimodal sentiment analysis, making it far more accurate than a general-purpose model.

3.4 Automated Labeling Agent

NOTE: THIS IS NOT IMPLEMENTED/PSUEDO IN THE CODE AND IS JUST A SUGGESTION FOR IMPROVEMENT

A component for processing the vast amount of scraped and synthesized data is the Automated Labeling Agent. Since manual labeling is not scalable, this agent programmatically assigns sentiment labels.

- **Zero-Shot LLM Classification**: The agent utilizes a powerful, general-purpose LLM (GPT-4o or Claude 3) in a zero-shot or few-shot setting. It provides the model with the review text and a directive to classify the sentiment as 'Positive', 'Negative', or 'Neutral', and to return a confidence score for its prediction.
- **Confidence Gating**: Not all machine-generated labels are treated equally. A confidence gating mechanism is employed:
  - Labels with high confidence (> 0.9) are automatically accepted.
  - Labels with medium confidence ($0.7 - 0.9$) are flagged and sent to a human review queue, forming a Human-in-the-Loop (HITL) system.
  - Labels with low confidence (< 0.7) are typically discarded to avoid introducing significant noise into the training set.
- **Active Learning Integration**: To make the human review process more efficient, the agent is designed to incorporate active learning. Instead of random sampling, the system identifies samples where the model is most uncertain. These high-impact samples are prioritized for human labeling, ensuring that human effort provides the maximum benefit to the model's performance.

3.5 Hybrid Retrieval System

The system's ability to retrieve similar reviews relies on a state-of-the-art hybrid search architecture that combines the strengths of multiple retrieval paradigms, as implemented in `hybrid_dense_parse_retrieval_approach.py`.

- **Dense Retrieval**: Utilizes CLIP embeddings stored in Qdrant. This captures the semantic and conceptual similarity between reviews ("this laptop is fast" is similar to "the performance is incredible"). It is ideal for understanding the intent behind a query.
- **Sparse Retrieval**: Uses the BM25 algorithm in Elasticsearch. This excels at matching exact keywords and product-specific jargon (model numbers or technical terms), providing high-precision results for specific queries.
- **Reciprocal Rank Fusion (RRF)**: Instead of simply taking the union of results, the system combines the ranked lists from both dense and sparse search using RRF. This algorithm prioritizes items that appear high up in both rankings, leading to more relevant and robust results than either method alone.
- **Cross-Encoder Reranking**: The fused list of candidates is then passed to a Cross-Encoder model (`cross-encoder/ms-marco-MiniLM-L-6-v2`). Unlike the initial retrieval (which compares a query to many documents), the cross-encoder performs a full-attention comparison between the query and each candidate, providing a highly accurate final re-ranking for maximum precision. This step is computationally expensive and is only applied to a small set of top candidates.

4 MLOps and Evaluation

A robust MLOps pipeline and a comprehensive evaluation framework are critical for ensuring the system's reliability and continuous improvement.

4.1 Automated Training Pipeline

The training process is fully automated and orchestrated by Prefect (`distributed_training.py`).

1. **Data Fetching**: The pipeline begins by fetching all high-quality labeled data (both real and synthetic) from the S3 data lake.
2. **Distributed Training**: It initiates a distributed training job using PyTorch and DeepSpeed. DeepSpeed's ZeRO optimization allows for fine-tuning very large models like LLaVA by efficiently partitioning the model's state (parameters, gradients, and optimizer states) across multiple GPUs, making it possible to train models that would not otherwise fit into a single GPU's memory.
3. **Experiment Tracking**: All training runs, parameters, and metrics are automatically logged to an MLflow server, which is configured to use a PostgreSQL backend for metadata and a MinIO S3 bucket for artifacts.
4. **Model Evaluation**: After training, the model is evaluated against a "golden" holdout test set using a comprehensive suite of metrics.
5. **Model Registration**: If the model's performance exceeds a predefined threshold (Macro F1 > 0.85), it is versioned and registered in the MLflow Model Registry and automatically promoted to the "Staging" environment.
6. **Automated Deployment**: The registration of a new model in Staging can trigger a CI/CD workflow (GitHub Actions) to initiate a rolling update of the inference service in the production Kubernetes environment.

4.2 Systematic Evaluation Framework

The system's quality is assessed not just on accuracy but on a wide range of criteria defined in the `ComprehensiveEvaluator` class within `systematic_eval_testing.py`.

- **Classification Performance**: Standard metrics including precision, recall, F1-score, and a confusion matrix are calculated to provide a baseline understanding of model performance.
- **Adversarial Robustness**: The model is tested against perturbed inputs, such as text with typos, images with Gaussian noise, and reviews with sarcastic or mixed sentiment, to measure its real-world resilience and prevent unexpected failures.
- **Multimodal Contribution**: The evaluator quantifies the exact performance uplift gained from using both text and images compared to using either modality alone. This is crucial for justifying the additional complexity of a multimodal system.
- **Latency and Throughput**: The inference service is load-tested to measure key performance indicators like mean/p50/p95/p99 latency and queries per second (QPS), ensuring it meets production service-level objectives (SLOs).

- **Automated Reporting**: The evaluation pipeline culminates in the generation of a comprehensive HTML report with visualizations like confusion matrices and robustness radar charts, providing a clear and accessible summary of the system's performance for all stakeholders.

5 Current Status and Implementation Gaps

TO DEPLOY THE DEMO OF THIS PROJECT: The user must perform several configuration and execution steps to make the system fully operational.
The following sections detail the current limitations and required actions.

5.1 Deployment Options

The system can be deployed in two modes using the 'setup.sh' script:

5.1.1 Option A: Minimal Demo

- Services: 8 essential services.
- Includes: API Gateway (Kong), Inference Service, Frontend (Gradio), Databases (PostgreSQL, Redis, Qdrant, Elasticsearch), and Storage (MinIO).
- Excludes: Training Service, Agents, and Monitoring stack.

5.1.2 Option B: Full System

- Services: All 13 services.
- Includes: Everything in the demo, plus the Scraping Agent, Synthetic Agent, Training Service, MLflow, monitoring stack.

5.2 Demo Sentiment Classification

The minimal demo is explicitly designed as a placeholder for the full, fine-tuned LLaVA model, which is too resource-intensive for a standard local deployment. The classification works as follows:

- **Predefined Keyword Lists**: The system has hardcoded lists of positive and negative words.
  - **Negative**: `['terrible', 'awful', 'bad', 'worst', 'hate', 'horrible', 'poor', 'disappointed', 'waste', 'broken']`
  - **Positive**: `['excellent', 'amazing', 'great', 'best', 'love', 'wonderful', 'perfect', 'fantastic', 'awesome', 'exceeded']`
- **Keyword Counting**: The code counts the occurrences of these words in the input text.
- **Classification Logic**: Sentiment is determined by comparing the counts. If positive words > negative words, the sentiment is "positive". If equal, it is "neutral."

5.3 Implementation Status and Limitations

- **Full ML Pipeline (Placeholder)**: The core ML components are provided as pseudo-code or simplified implementations.
  - **LLaVA Model**: The demo uses the rule-based classifier mentioned above. The full LLaVA model is $\approx 13GB$, requires a GPU, and a long download time. A production deployment requires fine-tuning this model.
  - **Training Pipeline**: The Prefect flow structure exists but requires real data and execution to produce model artifacts.
  - **Synthetic Agent**: The code structure is present, but running the full Llama 3.1 and SDXL models would require a high-end GPU (minimum 24GB VRAM).
- **Retrieval System (Not Integrated)**: The hybrid retrieval logic exists in the codebase but is not connected to the main '/analyze' inference endpoint in the demo.

5.4 Required Configuration

The system will not run without proper configuration of its services and credentials.

- **Environment Variables**: The project includes a `setup.sh` script that generates a default `.env` file. A user must edit this file and populate it with valid credentials for external services, including API keys for LLM providers (OpenAI), and potentially AWS credentials if deploying to the cloud.
- **API Gateway Configuration**: The Kong API gateway requires a configuration file (`kong/kong.yml`) that defines the upstream services and routing rules. This file must be correctly configured and mounted into the gateway container to enable communication between the frontend and the backend microservices. The provided file serves as a starting point. Sample of the 'kong.yml' file:

```
_format_version: "3.0"

services:
  - name: inference-api-service
    url: http://inference-service:8000
    # Pointing to the inference service
    # defined in docker-compose.yml
    routes:
      - name: inference-routes
        paths:
          - /api
        strip_path: true
        # Just for removing the `/api` from the path before forwarding to
        # the service (prevent dup path if the dev change the pathing)
```

5.5 Data and Model Bootstrapping

The system is designed to solve the "cold start" problem, which means it starts with no data or trained models.

- **Initial Data Collection**: A user must first run the Data Ingestion Agent to scrape an initial corpus of raw data. Subsequently, the Data Synthesis Agent must be run to augment this corpus. Without this data bootstrapping, the training pipeline has no data to process.
- **Model Training is Mandatory**: The core of the system's intelligence lies in fine-tuned models (LLaVA). The repository provides the pipelines to train these models (`distributed_training.py`) but does not include the final model artifacts due to their large size. A user must execute this entire training pipeline, which is a computationally intensive process requiring significant GPU resources and time, before the inference service can function with its intended intelligence. The local demo falls back to a simpler, non-fine-tuned model.

5.6 Infrastructure Dependencies

The project relies on a specific set of tools to function as designed.

- **Docker Environment**: The entire system is containerized and orchestrated with Docker Compose. A working installation of Docker and Docker Compose is a hard requirement.
- **GPU Requirement**: Key components, including the Inference Service, Training Service, and Synthesis Agent, are designed to run on NVIDIA GPUs for acceptable performance. While some parts may run in a CPU-only environment, performance will be severely degraded, and training large models will be impractical. The `docker-compose.yml` includes GPU reservations for the inference service.

6 Technologies Stack

Table 1: Technology Stack

Component	Technology
AI/ML Frameworks	PyTorch, Transformers, Diffusers
Backend Services	FastAPI (Python)
Orchestration	Prefect, Docker
Frontend	Gradio
Databases	PostgreSQL, Qdrant, Elasticsearch, Redis
Core Models	LLaVA-v1.6, CLIP, BERT
Web Scraping	Playwright, Scrapy
MLOps & Monitoring	MLflow, Prometheus, Grafana
Cloud Provider	AWS (S3, EKS, RDS), MinIO (local S3)

7 Setup and Usage

This section provides instructions for setting up the local development environment using Docker Compose.

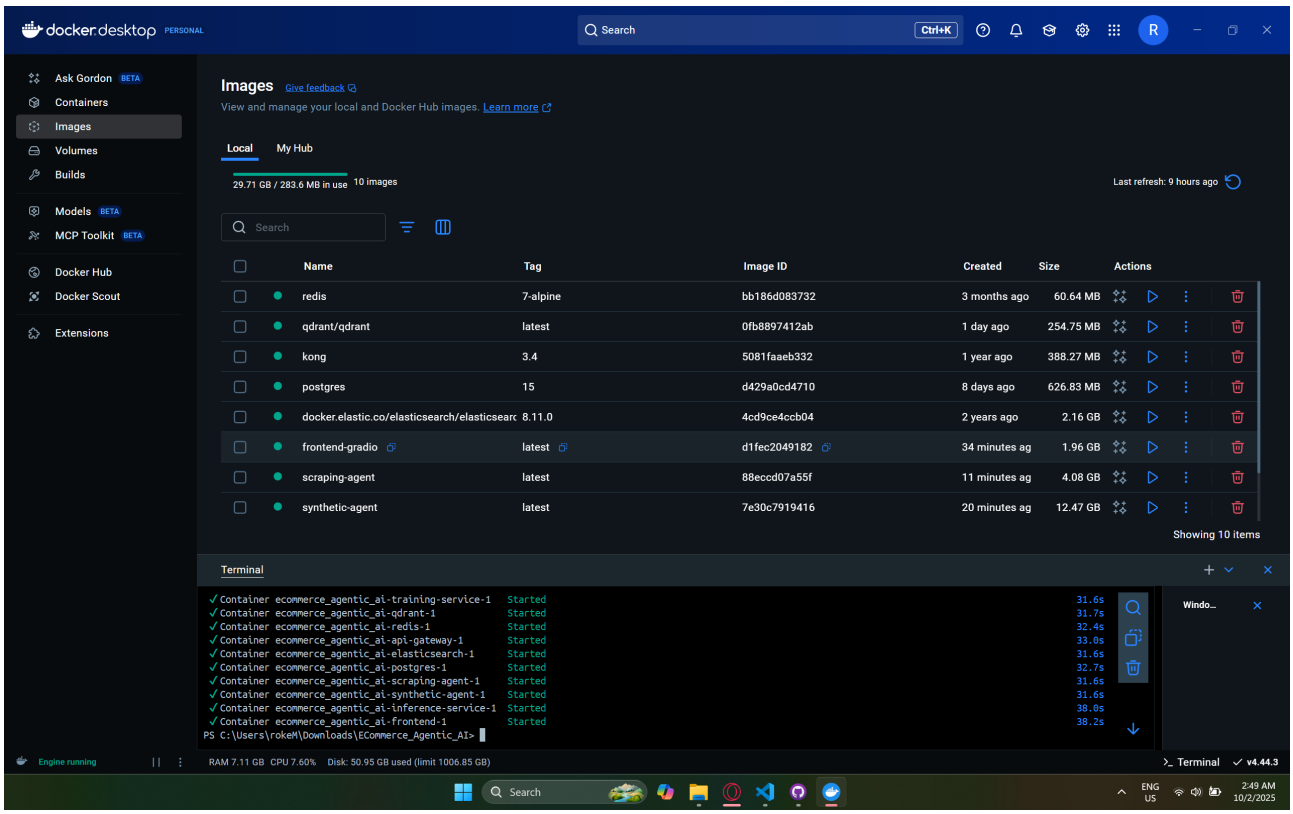

Figure 2: Docker Images

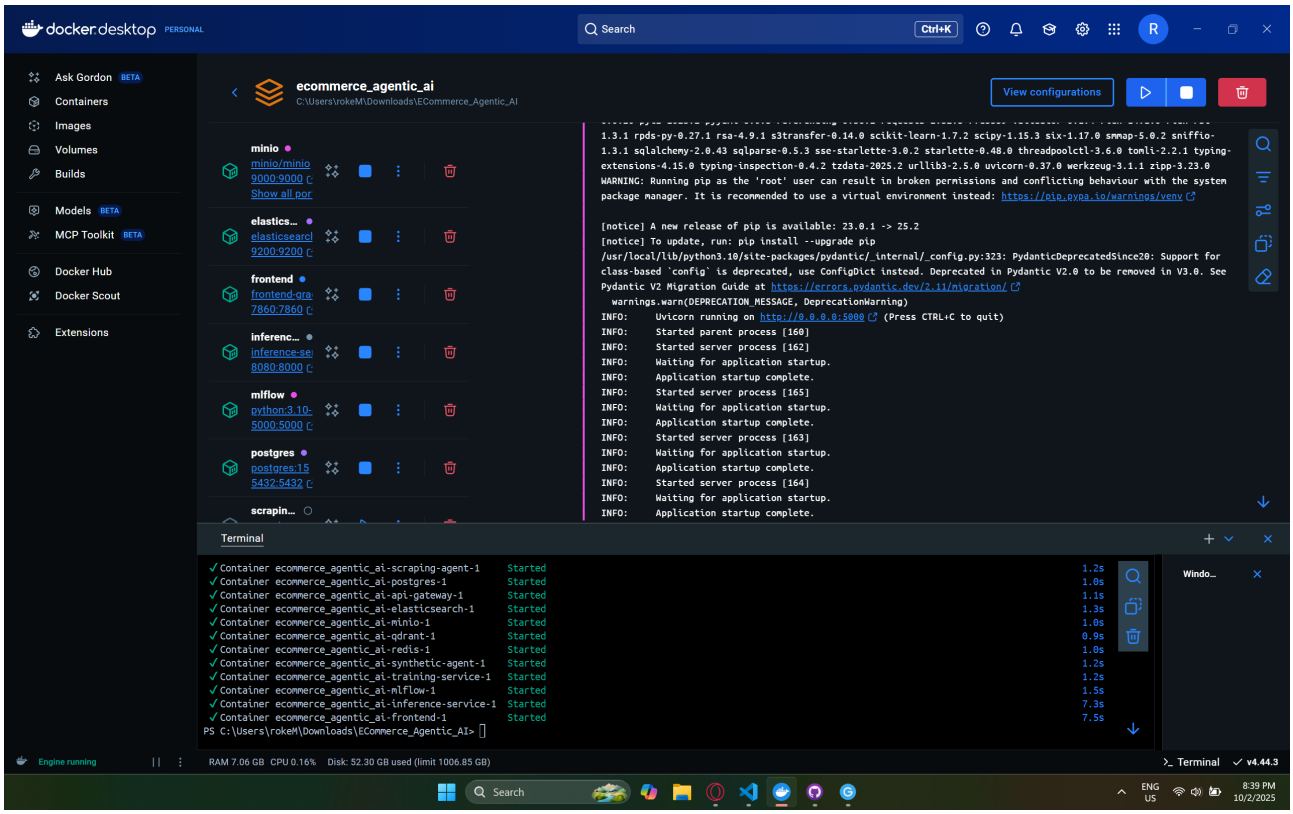

Figure 3: Services Status

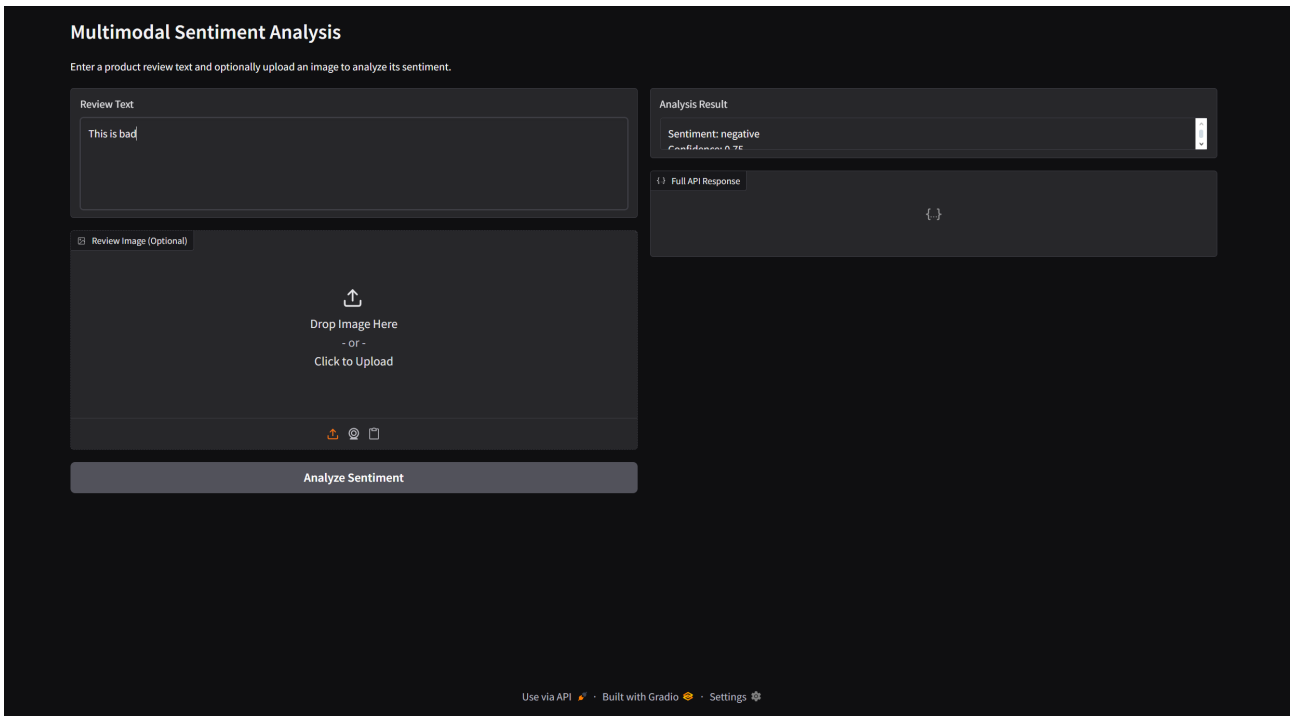

Figure 4: Frontend Usage

7.1 Prerequisites

- Docker Desktop
- NVIDIA GPU with appropriate drivers (optional but recommended for full functionality)
- Python (version 3.10.7, or change it in the Dockerfile)

7.2 Local Development

```
# Clone the repository
git clone https://github.com/rokemse/ecommerce_agentic_ai.git
cd ecommerce_agentic_ai

# Run the setup script to create configurations and .env file
./setup.sh

# The script will prompt you to edit the .env file with your API keys
nano .env
```

The setup script presents an option for a minimal or full deployment. For a first run, the minimal deployment is recommended.

```
# After the setup script finishes, it will start the selected services.
# To start them manually, you can run:
docker-compose up --build -d
```

The system's services will be accessible at the following local endpoints:

- **Frontend**: http://localhost:7860

- **API Gateway**: http://localhost:8000
- **MLflow UI**: http://localhost:5000
- **Grafana Dashboard**: http://localhost:3000
- **MinIO Console**: http://localhost:9001

8 References

- LLaVA: A large multi-modal language model
- Minio – Object storage server AWS S3
- MiniO - Documentations
- Hybrid Search: Combining Semantic and Keyword Approaches for Enhanced Information Retrieval
- A Practical Guide to Hybrid Search