Multimodal Conversational AI for E-Commerce

Alex Tsourmas (alextsourmas@uchicago.edu), Jane Lee (gylee@uchicago.edu), Yeochan Youn (yyoun5@uchicago.edu), Kaylie Nguyen (adqnguyen@uchicago.edu), Danylo Sovgut (sovgut@uchicago.edu)

Introduction

This system implements a multimodal RAG application for e-commerce using CLIP embeddings for unified text-image search. Three core modules: Chainlit web interface (`app/main.py`), CLIP-based retrieval system (`multimodal_rag.py`), and data processing pipeline (`data_processor.py`) handling Amazon Product Dataset 2020.

Data Retrieval and Preprocessing

Dataset: Amazon Product Dataset 2020 (10,000+ products) via Kaggle containing product names, descriptions, specifications, categories, pricing, and image URLs.

Text Processing: Concatenates product fields into unified text, implements query enhancement based on product vocabulary, preprocesses for CLIP's 77-token limit.

Image Processing: Async downloads using aiohttp with sync fallback, validates images (min 50x50px), standardizes to 512x512 JPEG format, handles timeouts and corrupted files.

Data Storage & Embeddings

CLIP Implementation: Uses `openai/clip-vit-base-patch32` via transformers, generates 512-dim embeddings in shared semantic space, CUDA/CPU device detection.

Storage: Three embedding types (text, image, multimodal-averaged), stored as NumPy `.npy` files with JSON fallback, in-memory cosine similarity search using scikit-learn.

Retrieval System

Core Class:

```
class MultimodalEcommerceRAG:
    def get_clip_query_embedding(self, text_query, image_path)
    def retrieve_similar_products(self, query_embedding, search_mode, top_k)
    def generate_response(self, user_query, retrieved_products, chat_history)
```

Search Modes: Text search (CLIP text encoder), image search (CLIP image encoder), multimodal search (averaged embeddings). Returns top-k products ranked by cosine similarity.

LLM Integration & Modeling

Model Stack: CLIP `clip-vit-base-patch32` for embeddings, GPT-4o for text generation, GPT-4 Vision for image analysis.

Process: Retrieve products via CLIP → format context → generate response with GPT-4o → track metrics via `SimpleAnalyticsTracker`.

Evaluation Methodology

LLM-as-Judge: GPT-4o-mini rates product relevance (1-5 scale) for 15 realistic queries, calculates Precision@5, MRR, NDCG with threshold ≥3 for relevance.

Analytics: Tracks response time, embedding time, retrieval time, search mode distribution via `analytics_tracker.py`.

Performance Results

LLM Evaluation (15 queries):
- Success Rate: 100%
- Precision@5: 0.587 (58.7%)
- MRR: 0.802
- NDCG: 0.837
- Average Relevance: 3.25/5

System Performance:
- Response time: 2.1s average
- Throughput: 0.48 queries/second
- Examples: "puzzle for kids" (P=0.8, MRR=1.0), "board games" (P=0.0, MRR=0.0)

System Architecture

Data Flow:
Raw CSV → Cleaning → Image Download → CLIP Embedding → Storage
User Query → CLIP Encoding → Vector Search → Product Retrieval → GPT Response

Tech Stack: PyTorch, transformers, OpenAI API, Chainlit, pandas, numpy, scikit-learn, aiohttp.

File Structure:

```
data/
├── amazon_products.csv
├── processed_products.csv
├── images/
└── embeddings/ (*.npy files)
```

Conclusion

Functional multimodal RAG system achieving 58.7% precision on realistic queries. CLIP creates unified embedding space for text-image search. Modular architecture separates data processing, retrieval, and generation. LLM-as-judge provides practical evaluation without artificial benchmarks.

Limitations: Dataset quality affects performance, API dependencies, requires valid images/descriptions, OpenAI API latency constraints.