# Preprocessing Steps

**Data Loading and Initial Exploration**

1. **Dataset Loading**:
    - The dataset marketing_sample_for_amazon_com-ecommerce__20200101_20200131__10k_data.csv is loaded from Google Drive.
2. **Initial Inspection**:
    - Columns with excessive null values (Marketplace, Shipping Option, Free Shipping, Fulfillment, Number of Offers, Discount Percentage, Item Condition, Merchant, Delivery Time, Product Variants, Search Keywords, Specifications, Technical Details) are identified and dropped.

**Data Cleaning**

1. **Cleaning Selling Price**: Extracts numeric values from the Selling Price column using a regex pattern and converts them to floats. Missing or invalid entries are replaced with NA.
2. **Cleaning Shipping Weight**:
    - Handles entries in Shipping Weight by:
        - Extracting numeric values for entries with "pounds".
        - Converting values from "ounces" to pounds.
        - Invalid entries are replaced with NA.
3. **Cleaning Is Amazon Seller**:
    - Converts values in the Is Amazon Seller column:
        - Y is mapped to True.
        - N is mapped to False.
        - Invalid entries are replaced with NA.
4. **Cleaning Text Attributes**:
    - The columns Product Name, Category, About Product, Product Specification, and Technical Details are cleaned by:
        - Converting text to lowercase.
        - Removing newline characters and pipes (|).
        - Stripping leading and trailing whitespace.
    - The cleaned values are stored in new columns with _Cleaned suffix, e.g., Product Name_Cleaned, Category_Cleaned.
5. **Validating Image URLs**: Ensures that all entries in the Image column start with http.

**Dataset Preparation for Embedding Generation**

1. **Final Dataset Selection**:
    - The following cleaned columns are selected for downstream processing: (Product Name_Cleaned, Category_Cleaned, Selling Price_Cleaned, About Product_Cleaned, Product Specification_Cleaned, Technical Details_Cleaned, Shipping Weight_Cleaned, Image, Product Url, Is Amazon Seller_Cleaned).
2. **Creating a Unified Text Field**:
    - A new column Combined_Text is created by concatenating: (Product Name_Cleaned, Category_Cleaned, About Product_Cleaned, Product Specification_Cleaned, Technical Details_Cleaned).

- Missing values in these columns are replaced with empty strings to ensure consistency.

# Model Architectures

Column Combined_Text and Image are converted to text_embeddings (10002, 512) and image_embeddings (10002, 512) using CLIP. combined_embeddings is generated by concatenating text_embeddings and image_embeddings.

The implementation leverages the CLIP (Contrastive Language-Image Pretraining) model, which integrates a transformer-based text encoder and a vision transformer (ViT)-based image encoder. The architecture enables effective multimodal understanding by aligning text and image embeddings into a shared latent space.

- **Text Encoder**: Based on a Transformer architecture, the text encoder maps textual input to a dense embedding space.
- **Image Encoder**: Built on Vision Transformer (ViT), the image encoder transforms visual input into a dense representation in the same space as text embeddings.
- **Joint Embedding Space**: CLIP uses contrastive learning during training to align similar text and image pairs while pushing apart dissimilar pairs, ensuring high retrieval performance for both modalities.

Key Features:

1. **Pretrained Backbone**: Utilizes "openai/clip-vit-base-patch32," a well-optimized and pretrained model for multimodal applications.
2. **Projection Layer**: Maps high-dimensional embeddings into a fixed 512-dimensional latent space.
3. **Fine-Grain Compatibility**: Text and image embeddings are directly comparable, enabling multimodal retrieval with minimal preprocessing.

# Integration Details

The project integrates several components to create a seamless pipeline for multimodal search and retrieval:

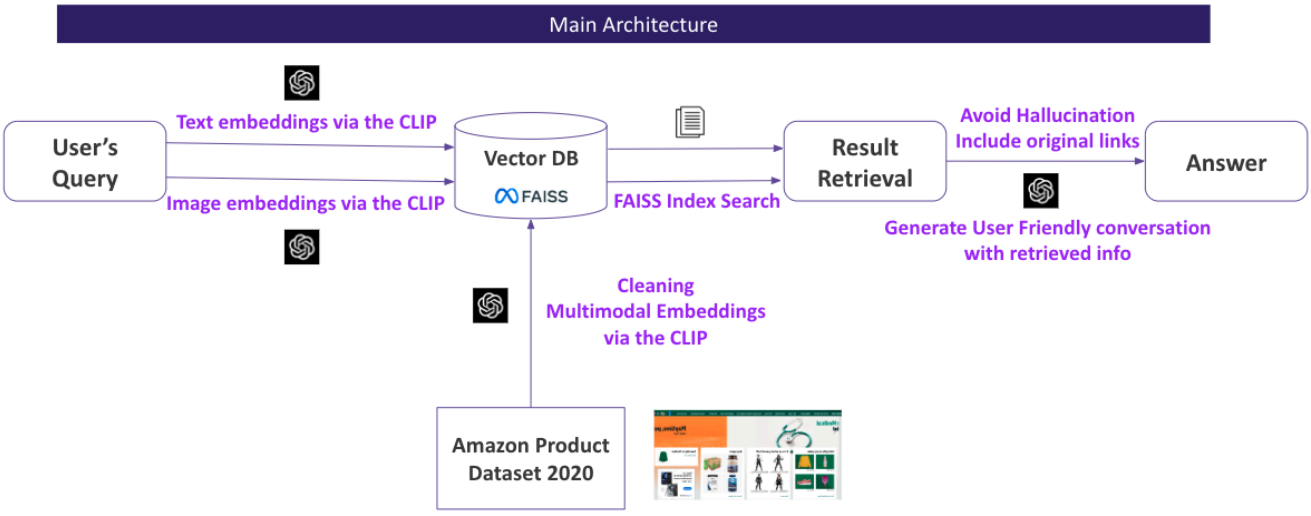
1. **Data Preparation:**
    - Text data from columns Product Name_Cleaned, Category_Cleaned, About Product_Cleaned, Product Specification_Cleaned, Technical Details_Cleaned is concatenated into a unified text representation, Combined_Text.
    - Selling Price_Cleaned, Shipping Weight_Cleaned, Product Url, Is Amazon Seller_Cleaned are kept to provide more information of the retrieved item.
    - Image URLs are processed and downloaded dynamically to generate embeddings.
2. **Embedding Generation:**
    - Text Embeddings: Generated using the CLIP text encoder with tokenized inputs using Combined_Text.
    - Image Embeddings: Extracted from the CLIP image encoder using preprocessed image tensors using Image.
3. **FAISS Index Creation:**
    - Text-only Index: Handles embeddings from textual descriptions.
    - Image-only Index: Handles embeddings from visual inputs.

- Multimodal Index: Combines both text and image embeddings for enhanced retrieval.

4. **Streamlit Application:**
   - Provides an intuitive user interface for text, image, and multimodal queries.
   - Utilizes GPT-4 for conversational recommendations, contextualized with retrieval results.
5. **Cloud Storage:**
   - FAISS indices are stored in Google Drive for persistent access.
   - Dataset files and embeddings are dynamically downloaded and cached for efficient runtime performance

# System Design

This system design emphasizes modularity, scalability, and user engagement, leveraging the strengths of CLIP, FAISS, and GPT-4 to deliver high-performance multimodal search and recommendation capabilities.




**1. Architecture**

The architecture incorporates the following components to support text, image, and hybrid queries:

- **Input Handling**:
  - Accepts **text queries** via an input box.
  - Allows **image uploads** for visual queries.
  - Supports **multimodal queries**, combining text and image inputs.
- **Preprocessing Pipeline**:
  - For text: Normalizes, tokenizes, and converts input queries into embeddings using CLIP's text encoder.
  - For images: Downloads and preprocesses images into tensors for embedding generation using CLIP's image encoder.
- **Search and Retrieval**:
  - Utilizes **FAISS indices** for approximate nearest-neighbor (ANN) search based on **L2 (Euclidean) distance** with low latency. Separate indices are maintained for: Text embeddings, Image embeddings, and Combined (multimodal) embeddings depending on the quey_type of Text, Image or Multimodal.

**2. Flow**

The system follows a clear and efficient flow:

1. **User Query**: Captures user input via text, image, or both.
2. **Embedding Generation**:
    - Text embeddings via the CLIP text encoder.
    - Image embeddings via the CLIP image encoder.
3. **FAISS Index Search**: Matches query embeddings against stored indices to retrieve relevant items.
4. **Result Retrieval**: Extracts product details, images, and links from the dataset.
5. **GPT-4 Recommendation**: Provides personalized product recommendations based on retrieved results.

**3. Components**

- **CLIP Model**:
    - Encodes both text and images into a shared embedding space, enabling multimodal understanding.
    - Efficiently processes user queries for embedding generation.
- **FAISS**:
    - Indexes embeddings for approximate nearest-neighbor search.
    - Ensures fast and scalable retrieval of relevant products.
- **GPT-4 Integration**:
    - **Context Generation from Retrieved Items**:
        - After retrieving relevant products using FAISS indices, the system compiles a detailed structured context from the retrieved items.
        - Each item's **name**, **description**, **category**, **price**, and **image URL** are formatted into a human-readable summary.
        - This context forms the backbone of GPT-4's recommendation process.

    **Prompt Construction**:
        - GPT-4 is given a well-structured prompt that includes:
          A clear description of its role ("helpful assistant recommending products").
          The context (details of retrieved items).
          The user query and a request to recommend suitable products based on the query

    **Generating Recommendations**:
        - GPT-4 processes the prompt and generates a response that:
            - Identifies the most relevant products.
            - Explains their suitability for the user's query.
            - Incorporates relevant product attributes (e.g., comfort, durability, or price).
            - Optionally includes image URLs if the query is image-centric.

    **Benefits of GPT-4 Integration**
        - **Personalization**: GPT-4 considers both the user query and retrieved product details to create responses that are highly tailored to the user's needs.
        - **Explainability**: The assistant not only lists relevant products but also explains why they are suitable, helping users make informed decisions.
        - **Versatility**: GPT-4 can handle a wide range of queries, from detailed product searches to broad category recommendations, ensuring comprehensive assistance.
        - **Enhanced Engagement**: Conversational responses make the user experience more engaging, as the system feels less mechanical and more like a helpful assistant.
- **Streamlit UI**:
    - Provides an interactive front-end for users to submit queries and view results.

- Displays retrieved products with details, images, and links.

# Implementation Process

The implementation process for the multimodal product search and recommendation system involves several interconnected steps, ranging from data preparation to building the user-facing application. Below is a detailed breakdown:

**1. Data Cleaning: Refer to Preprocessing Steps**

**2. Embedding Computation**

Efficient embedding generation for text and image data was a critical part of the pipeline. The following strategies were used:

- **Text Embeddings:**
  - Leveraged CLIP's text encoder to generate embeddings.
  - Text inputs were tokenized and padded using the CLIP processor.
- **Image Embeddings:**
  - Images were downloaded from URLs and preprocessed into tensors using CLIP's image processor.
  - A retry mechanism ensured robust handling of failed image downloads.
  - For missing or invalid images, zero embeddings were added to maintain consistency.

**3. Index Construction**

**The FAISS library was used to build high-performance indices for approximate nearest-neighbor (ANN) search.**

- **Separate Indices:**
  - Created individual FAISS indices for:
    - Text-only embeddings: Indexed embeddings from the text encoder.
    - Image-only embeddings: Indexed embeddings from the image encoder.
    - Multimodal embeddings: Combined text and image embeddings for hybrid queries.
  - Each index was constructed using the L2 distance metric for similarity computations.
- **Adding Embeddings:**
  - Generated embeddings were added to their respective indices.
- **Persistence:**
  - Indices were stored locally and backed up on Google Drive for future reuse.

**4. Retrieval and Evaluation**

- **Query Matching:**
  - User queries (text, image, or both) were converted into embeddings using CLIP.
  - Embeddings were matched against the corresponding FAISS index using nearest-neighbor search.
- **Evaluation Metrics:**
  - Retrieval accuracy and Recall@1 were used to measure performance, as we assume only one correct product is included in the dataset.

- Evaluation results are as follows. Overall, accurate responses were achieved for text-based queries, image-based queries, and requests for specific product images.
    - Accuracy: 0.875
    - Recall@1: 0.875

| | Questions | Evaluation |
|---|---|---|
| 1 | What are the features of the DB Longboards CoreFlex Crossbow 41? **(Text-Based Questions)** | Correct |
| 2 | Can you compare the DB Longboards CoreFlex Crossbow 41 with swagskate ng2 a.i.-powered electric longboard? **(Text-Based Questions)** | Half Correct |
| 3 | Can you identify the product in this image and describe its usage? (Image of a Pokemon Card) **(Image-Based Questions)** | Correct |
| 4 | Can you show me a picture of crossbow longboard? **(Request for a Product Image)** | Correct |

**5. Application Development**

**The application was built using Streamlit for a responsive and interactive user interface.**

- **UI Design:**
    - Provided options for query input via:
        - Text (text box).
        - Image (file uploader).
        - Multimodal (both text and image inputs).
- **Search Execution:**
    - Based on the query type, the appropriate FAISS index was selected for retrieval.
    - Retrieved results were displayed, including:
        - Product name, category, and price.
        - Image with a caption and a link to the product.
- **GPT-4 Integration:**
    - Leveraged GPT-4 to generate conversational recommendations based on retrieved results.
    - Responses included:
        - Product explanations tailored to the query.
        - Justifications for why the products meet user needs.

This comprehensive implementation process ensures the system is robust, scalable, and user-friendly, capable of delivering high-quality multimodal recommendations.

# Challenges Faced

- **Data Preprocessing**:

    - Handling missing values and malformed URLs required careful attention to maintain embedding consistency.
    - Ensuring text concatenation preserved semantic meaning without introducing noise.
- **Scalability**:
    - Embedding computation for larger datasets may lead to memory constraints, especially for images.
- **Index Loading**:
    - Managing FAISS indices across multiple modalities required optimized storage and retrieval logic for larger dataset.
- **Latency**:
    - Real-time query execution introduced latency challenges, especially with multimodal embeddings.
- **Model Limitations**:
    - CLIP's pretrained embeddings might not align perfectly with niche product descriptions, impacting retrieval precision.

- **GTP4 prompt:**
    - GTP4 does not provide correct information on retrieved item.

# Solutions Developed

1. **Fallback Mechanisms**:
    - Introduced zero embeddings for invalid images to maintain consistency in index sizes.
2. **Batch Processing**:
    - Reduced memory overhead during embedding computation by processing data in manageable batches.
3. **Pre-caching**:
    - Cached indices and datasets using st.cache_resource in Streamlit to minimize runtime delays.
4. **Hybrid Retrieval**:
    - Designed a multimodal index to balance the strengths of text and image embeddings.
5. **Enhanced Evaluation**:
    - Developed a robust evaluation pipeline to measure accuracy and Recall@k for fine-tuning retrieval parameters.
6. **Context Generation**
    - Used each item's name, description, category, price, and image URL as GTP4's context generation.

# Suggestions for Future Improvements

1. **Fine-tuning CLIP**:
    - Train the CLIP model on domain-specific data to improve alignment for niche product descriptions.

2. **FLAVA:**
    - Try FLAVA model to generating embeddings and compare results with CLIP.
3. **Index Optimization**:
    - Experiment with FAISS's HNSW (Hierarchical Navigable Small World) for faster, scalable ANN searches.
4. **Dynamic Index Updates**:
    - Implement dynamic updates to FAISS indices for new product additions without requiring full index rebuilds.
5. **Visual Enhancements**:
    - Integrate a carousel or grid display for retrieved product images in the Streamlit UI.
6. **Multilingual Support**:
    - Extend the system to support queries in multiple languages, leveraging translation models like MarianMT.
7. **Advanced Recommendations**:
    - Explore techniques like cross-attention or transformer-based recommender systems to provide personalized results.
8. **Cloud Deployment**:
    - Host the entire application on cloud platforms (e.g., AWS, GCP) to support larger datasets and user bases.