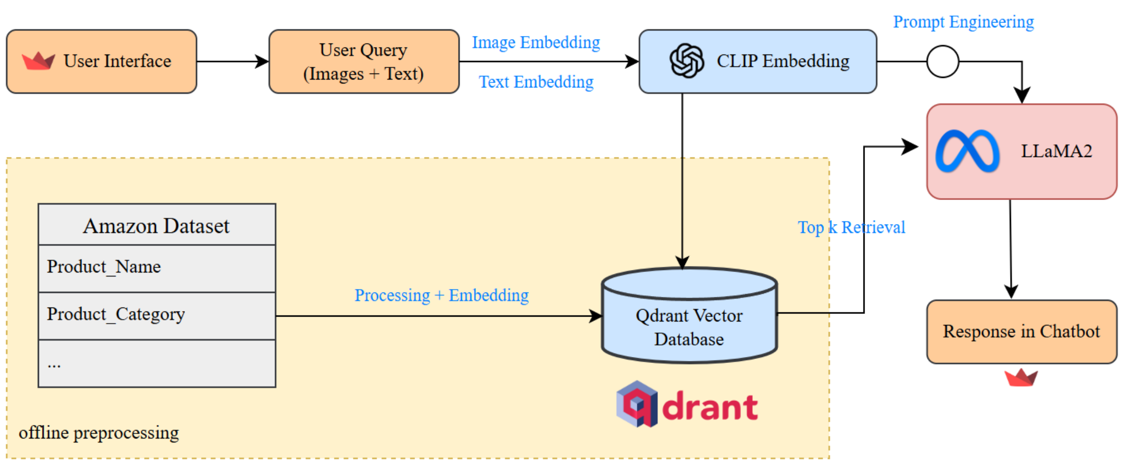
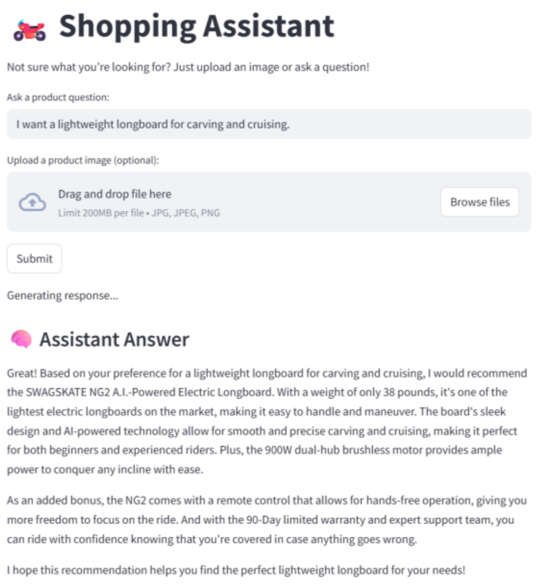
System Architecture Diagram



Flow

1. User Interface: The user interacts through a chatbot interface, submitting texts/images
2. Query Embedding (CLIP)
   - Text is embedded using CLIP's text encoder
   - Image is embedded using CLIP's image encoder
   - If both are present, embeddings are concatenated and L2-normalized
3. Vector Search (Qdrant)
   - The normalized query embedding is sent to Qdrant
   - Qdrant searches against stored product embeddings and returns the top-K most similar products with metadata (name, price, description, image URL)
4. Prompt Construction
   - Retrieved product data is formatted into a structured prompt
   - Instructions and fallback examples are included to ground the LLM response in context only
5. Answer Generation (LLaMA 2)
   - LLaMA generates a natural-language response using only the retrieved product context



Shopping Assistant

Not sure what you're looking for? Just upload an image or ask a question!

Ask a product question:

I want a lightweight longboard for carving and cruising.

Upload a product image (optional):

Generating response...

Assistant Answer

Great! Based on your preference for a lightweight longboard for carving and cruising, I would recommend the SWAGSKATE NG2 A.I.-Powered Electric Longboard. With a weight of only 38 pounds, it's one of the lightest electric longboards on the market, making it easy to handle and maneuver. The board's sleek design and AI-powered technology allow for smooth and precise carving and cruising, making it perfect for both beginners and experienced riders. Plus, the 900W dual-hub brushless motor provides ample power to conquer any incline with ease.

As an added bonus, the NG2 comes with a remote control that allows for hands-free operation, giving you more freedom to focus on the ride. And with the 90-Day limited warranty and expert support team, you can ride with confidence knowing that you're covered in case anything goes wrong.

I hope this recommendation helps you find the perfect lightweight longboard for your needs!

System Architecture

1. Preprocessing Pipeline (Offline)
- Product metadata is cleaned and transformed (text normalization, unit standardization)
- CLIP is used to embed:
- CLIP_Text_Embedding → 512-dim
- CLIP_Image_Embedding → 512-dim
- CLIP_Combined_Embedding → 1024-dim (text + image)
- Embeddings are stored in Qdrant with full product payloads

2. Real-Time Query System (Online)

Component	Description
User Interface	Powered by Streamlit; allow user to submit text/image query
CLIP (ViT-B/32)	Generates embeddings for text/image/both at query time
Qdrant	Vector DB used to store and retrieve top-k product embeddings via cosine similarity
Prompt Builder	Converts top-k retrieved results into a formatted input for the LLM
LLaMA-2 7B Chat	Generates natural-language answers grounded in retrieved product context
User Interface	Displays response and supporting product visuals in Chatbot powered by Streamlit

Documentation

I. Data Cleaning

Handling Numerical Columns

1. Shipping Weights:
    a. Strip commas/periods and parse as numeric
    b. Convert all units to pounds
    c. Standardize to an integer (e.g. “1,070 pounds” → 1070)
    d. Mark unparseable or missing as NaN
    e. Drop the original Shipping Weight column

2. Selling Prices
    a. Remove commas and “$” signs
    b. Extract all \d+\.\d{2} patterns; if multiple, take the lowest
    c. Convert entries like “$ 14 94” to 14.94 via regex
    d. Treat “currently unavailable,” “free shipping,” “from ...”, etc. as NaN
    e. Fill any remaining parse failures with NaN
    f. Drop the original Selling Price column

3. Is_Amazon_Seller
    a. .str.strip() + .str.lower() on the raw values
    b. Vectorized .isin({'y','yes','true','1'}) → True/False → 1/0
    c. Cast to integer and drop the original column

Handling Non-Numeric Columns

1. Base cleaning for all text fields (base_clean)
    a. Unicode normalization: converts full-width punctuation and accented letters into their standard forms (NFKC)
    b. Whitespace cleanup: replaces \n and \t with a single space, then collapses any runs of 2+ spaces down to one
    c. Non-printable removal: strips out control characters (e.g. bell, backspace) so only visible characters remain
    d. Final trim: strips leading/trailing spaces
2. Product name (clean_product_name)
    a. Run base_clean
    b. Decompose any remaining accents (NFKD) and drop the combining marks(e.g. turns “café” to “cafe”)
    c. Standardize punctuation: convert “ ’ ” to “ ' “
    d. Turn pipe separators into sentence breaks: replace | (and variants like |) with “. ”
3. About product blurb (clean_about)
    a. Run base_clean
    b. Remove boilerplate copy such as “Make sure this fits by entering your model number.” (case-insensitive)

    c. Strip out emojis or other non-ASCII symbols ([^\x00-\x7F]+)
    d. Normalize separators: collapse all | into ". "
    e. Insert a space between digits and letters ((\d)([A-Za-z]) → \1 \2), so "2XL" becomes "2 XL"
    f. Collapse repeated periods (... or ..) into a single "."
4. Specification details (clean_spec)
    a. Run base_clean
    b. Normalize separators and digit-letter spacing as above
    c. Guarantee a trailing period if missing
5. Technical details (clean_technical)
    a. Run base_clean
    b. Remove user-guidance and marketing text:
        i. Phrases like "Go to your orders...Ship it!"
        ii. "From the Manufacturer...." section
        iii. Strip any non-ASCII, then apply the same digit-letter spacing and sentence-ending fixes
6. Category path formatting (format_category)
    a. Split the raw category string on |, trimming whitespace from each level
    b. If there's only one level, output: "It is part of the X category."
    c. If there are multiple levels, join with commas and output: "This product appears in the following categories: Level 1, Level 2, Level 3."

Feature Selection

1. Remove the raw text columns cleaned
2. Eliminate any column that has *100%* missing values
3. Drop identifier and code columns that aren't predictive
4. Features Selected:
    - Image
    - Product Url
    - Shipping_Weight_Lbs
    - Selling_Price
    - Is_Amazon_Seller_Flag
    - Product_Name
    - Product_Category
    - About_Product
    - Product_Specification
    - Technical_Details

II. Embeddings

To generate text embeddings, we use the OpenAI CLIP model with the ViT-B/32 architecture. We first convert product information—such as name, category, price, weight, and specifications—into natural-language sentences that are easier for language models to understand. These sentences are then tokenized using CLIP's built-in tokenizer and processed in batches through model.encode_text(...), which outputs 512-dimensional fixed-size embeddings for each input. This approach aligns with how CLIP was trained and ensures compatibility with cosine-based similarity search.

Text Embedding

1. Integrate numerical and text features into a natural sentence
    a. Always include Product_Name
    b. Append price, weight, category, etc.
        i. "It belongs to the X category."
           "It is priced at $Y."
        ii. "It weighs around Z pounds."
        iii. "Specifications include: ..."
        iv. "Additional details: ..."
2. Generate CLIP text embeddings with a dimension of 512
3. Save embeddings to disk

Image Embeddings

1. Precheck images' validity
    a. Return NaN on any error
2. Batch-wise embedding
    a. Loop in chunks of batch_size
    b. For each URL in the batch:
        i. Stack all successfully-loaded tensors and encode the batch; final shape: (number of valid images on that batch × 512)
        ii. Re-insert zero-vectors (also 512 dimensions) for any URLs that failed, preserving original order
3. Save embeddings to disk

III. Vectorbase Integration (Qdrant)

Build a per-point payload

For each product row, We construct a metadata dictionary (payload) containing relevant fields such as product name, category, price, description, specifications, and image URL. This payload is stored alongside each embedding in Qdrant, enabling rich metadata filtering and human-readable retrieval.

Store each embedding set

1. Text-only: The natural language product descriptions are embedded using CLIP's text encoder (model.encode_text). The resulting 512-dimensional vectors are L2-normalized for calculating cosine similarity for retrieval.

1. Image-only: Product images are fetched from their URLs, preprocessed using CLIP's vision pipeline, and passed through model.encode_image to generate 512-dimensional image embeddings. These are also normalized for cosine similarity retrieval.
2. Multimodal: The text and image embeddings are concatenated into a single 1024-dimensional vector, normalized, and stored in a third Qdrant collection named "combined_products". This is also normalized.

IV. Evaluate Retrieval Quality (Text Search)

To evaluate the quality of semantic retrieval using CLIP text embeddings stored in Qdrant, a test is performed using a sample product query:

"DB Longboards CoreFlex Crossbow 41 Bamboo Fiberglass Longboard Complete".

After embedding the query, Qdrant collection is queried to retrieve the top 10 most similar products. Several key retrieval metrics and all values return 1, which means **ground truth is the top-1 result**:

- Accuracy (whether the top-1 result is exactly the ground truth) = 1
- Recall@1 (whether the ground truth appears in the top 1 result) = 1
- Recall@5 (whether the ground truth appears in the top 5 results) = 1
- Recall@10 (whether the ground truth appears in the top 10 results) = 1

V. Retrieval-Augmented Generation (RAG) with CLIP + LLaMA

This pipeline integrates CLIP embeddings, Qdrant retrieval, and LLaMA-2 to build a multimodal shopping assistant capable of answering user queries using both product text and image inputs. Here is how a user query will be processed:

1. Embed the query with CLIP and normalize to unit length
2. Send the normalized embeddings to Qdrant's collection and returns the top-K nearest neighbors and their payloads
3. Build LLM prompt
   a. Instructions are added for properly answer generation
   b. A few-shot fallback example is also added in case if no relevant information is returned
4. Generate answer through LLaMA-2