1) Columns to keep based on non-nulls + usefulness

- **Uniq Id** — stable key for joins.
- **Product Name** — essential for search, display, and prompts.
- **Category** — filtering and context.
- **Selling Price (object → numeric)** — for display/filters.
- **Model Number** — strong for dedup/entity resolution.
- **About Product** — primary free text.
- **Product Specification** — parse to key–value text.
- **Technical Details** — more specs; same parsing as above.
- **Shipping Weight** — normalize to a canonical unit; optional for answers.
- **Image** — URL(s) for CLIP image embeddings + UI thumbnail.
- **Variants** — useful (sizes/colors) if present.
- **Product Url** — provenance link in answers.
- **Is Amazon Seller** — optional to use as a filter as it is 100% non-null.

Rationale: prioritize fields that actually exist at scale and materially improve retrieval/generation. This aligns with the "define optimal combination of attributes" requirement.

---

2) Data cleaning & normalization

1. **Types**
    - Selling Price: strip currency symbol/commas → float for easy filtering etc.
    - Shipping Weight: regex → value + unit → convert to grams/cm for consistency in the unit used for easy comparison
2. **Text sanitization**
    - Lowercase (for indexing), but keep original for display.
    - Remove HTML tags, bullets, excess whitespace.
    - For Product Specification / Technical Details, parse common "key: value" patterns; flatten into readable bullet text (e.g., "• Battery: 4000 mAh").
        i. If the column is **already structured** (has things like `Material: Plastic | Weight: 2 lbs`), we extract **key: value pairs**.

    ii. If it's **just free text** (like a paragraph from marketing copy), we keep it as-is instead of forcing it into key:value format.

3. **Category standardization**
    - Split on separators (>, |, /), keep top-1 and full path as two separate columns; map common aliases (e.g., "Cell Phones & Accessories" → "Cell Phones").
4. **Deduplication**
    - Candidate key: (Product Name normalized, Model Number) + fuzzy similarity.
    - Prefer records with image present; merge texts/specs across dups.

    We use **product identity clues**:

        1. **Product Name** (normalized — lowercase, remove punctuation, extra spaces).
        2. **Model Number** (if available) — usually the most reliable unique ID in retail data.
        3. **Fuzzy string matching** between names — catches cases like:
           arduino
           CopyEdit
            a. `"Sony WH-1000XM4 Wireless Headphones"`
            b. `"Sony WH1000XM4 Wireless Noise Cancelling Headphones"`

    If `Model Number` is missing, you can:

    - Compare `Product Name` similarity (fuzzy match score > threshold, like 90%).
    - Optionally compare `Selling Price` and `Category` as extra checks.

    Once duplicates are found:

    - **Keep**: The version with the best data (image, most complete specs, About Product).
    - **Merge**: If one has specs and another has extra details, combine them into a single final record.
    - **Drop**: The other duplicate rows.
5. **Missing images**
    - Validate Image URLs (HEAD request); flag broken; you can still index text-only.

Output: a **canonical product table** + a **long-form "context_text"** field per product (see next step).

---

3) Build the canonical "context_text" for RAG

Concatenate, in this order (with labels so the LLM can reason):

Title: {Product Name}
Category: {Category_path_or_top}
Model Number: {Model Number}
Price: ${Selling Price}
Key Features:
{bulleted summary from About Product}
Specifications:
{flattened key:value lines from Product Specification + Technical Details or free text}
Shipping Weight: {normalized}
Variants: {parsed if present}
Link: {Product Url}

- Keep it ~1–2k tokens max. If longer, **chunk** into ~200–300 token passages with overlap (keep chunk boundaries at sentence ends). Store chunk IDs with the parent product ID.

---

4) Image preprocessing

- Download images to local cache; skip if unreachable.
- CLIP expects ~224×224 with its specific normalization; keep original URL for UI.
- If multiple images per product (if present), embed each; store per-image vector + product_id.

---

5) Embeddings & vector store

The brief suggests CLIP as the backbone and a vector DB (e.g., Vertex AI Vector Search); FAISS/Milvus/Weaviate are fine alternatives if Vertex isn't available. Store **two indices**:

1. **Text index (chunk level)**
    - Use **CLIP text encoder** for each context_text chunk → L2-normalized vectors.
    - Store: chunk_id → vector, product_id, fields (title, url, category, price).
2. **Image index (image level)**
    - Use **CLIP image encoder** per image → normalized vectors.
    - Store: image_id → vector, product_id, image_url.

Why CLIP? It aligns vision and text into a **shared embedding space** so text↔image queries work out-of-the-box—exactly what your spec asks.

---

6) Retrieval logic (unimodal + cross-modal)

Support 3 modes:

- **Text query** → Encode with CLIP-text → search **text index** (primary) and **image index** (secondary) → merge by product_id (score max or learned weight), optional **category filter** if the query mentions it.
- **Image query** → Encode with CLIP-image → search **image index** and **text index** → merge by product_id.
- **Text + image query** → Encode both; fuse scores (e.g., weighted sum, default 0.6 text / 0.4 image).

Optional: **re-ranking** top-20 text chunks with a cross-encoder (text-only) for better faithfulness; for image re-ranking stick to CLIP cosine.

Return: top-k (e.g., k=5) **chunks grouped by product** with metadata + image URLs.

---

7) Generation (LLM integration)

Your brief suggests an open-source LLM (e.g., Llama-3.1 or Mixtral). Keep the generator **grounded** by passing only retrieved chunks + metadata.

System prompt (core rules):

- Answer **only** using the provided product context.
- If missing info, say what's missing and avoid guessing.
- Prefer bullet points and cite the product title + link in the answer.

User prompt template (text-only query):

less
CopyEdit
User question:
{raw_user_query}

Top retrieved product snippets (do not hallucinate beyond these):
[1] {title_1} — {url_1}
{chunk_text_1}

[2] {title_2} — {url_2}
{chunk_text_2}
...
Instruction: Provide a concise, accurate answer. If a comparison is requested, build a small table from the snippets.

Image query path: Show the top retrieved product image(s) in the UI and pass the **associated text chunks** to the LLM. If you use a **vision-language LLM**, you can also feed the image directly; the brief mentions using a vision-language model in combination with retrieval.

---

8) Streamlit UI (MVP)

---

9) Evaluation

Your brief calls for **Accuracy of Retrieval** and **Recall@K (1/5/10)**. Build a lightweight eval set:

- **Text queries:** auto-generate prompts from product titles (e.g., “What are the features of {title}?”), plus a few human-written comparisons. The **positive** is the same product_id. Compute Recall@K at the **product level**.
- **Image queries:** for each product with an image, use that image as the query; positives are the same product_id’s text chunks/images.
- **Response quality (optional):** small human rubric (correctness, completeness, citation) or LLM-as-judge with strict instructions to grade against the provided context.