# **A Multimodal Deep Learning Approach to Lost & Found Item Retrieval**

**Authors:** Ashhad Ahmed Kamran, Xintong Zhu, Qazi Maaz Pasha Syed **Course:** AI Lab **Date:** June 3, 2025

## **1. Introduction**

### **1.1 The Ubiquitous Problem of Lost & Found**

The loss of personal belongings is a common and often distressing experience, leading to significant inconvenience for individuals and operational challenges for organizations. Traditional lost and found systems, prevalent in universities, airports, public transport hubs, and large event venues, are typically inefficient and rely heavily on manual processes. These systems often involve physical sorting of found items, fragmented communication channels, and subjective textual descriptions that may not adequately capture the unique characteristics of a lost object. For instance, a simple description like "a bag" is inherently ambiguous, failing to distinguish between a backpack, a handbag, or a grocery bag, let alone its color, material, or unique features. This reliance on incomplete or imprecise information frequently results in a low retrieval rate for lost items, leading to frustration for owners and a backlog of unclaimed property for institutions.

### **1.2 Introduction to Artificial Intelligence in Information Retrieval**

Artificial Intelligence (AI) has revolutionized information retrieval by enabling systems to understand and process data in ways that go beyond simple keyword matching. Early retrieval systems primarily relied on exact textual matches, which limited their effectiveness when faced with natural language's inherent variability and ambiguity. The advent of sophisticated AI techniques, particularly in Natural Language Processing (NLP) and Computer Vision (CV), has paved the way for "semantic search," where systems can grasp the underlying meaning and context of queries rather than just literal terms. This shift allows for more intuitive and effective interactions, as AI can now interpret nuances in human language and visual information.

### **1.3 The Power of Multimodal AI for Lost & Found**

While advancements in individual AI modalities (e.g., text processing or image recognition) are significant, a purely text-based or image-based system remains insufficient for the complexities of lost and found. A text-only system struggles with vague user descriptions or the inability to describe unique visual patterns. Conversely, an image-only system would require users to upload a perfect match, which is often not possible. The true power emerges from **multimodal AI**, which integrates and processes information from multiple distinct modalities simultaneously.

For a lost and found system, combining text and image modalities allows for a richer, more robust understanding of an item, mirroring how humans perceive and remember objects. A user might describe "a blue backpack with a unicorn keychain," providing both semantic (backpack, unicorn) and attribute (blue) information. If they also upload a photo of a similar backpack, the system can leverage both inputs to achieve a more precise match. This synergistic approach enables the system to:

* Understand semantic relationships between words and visual features.
* Handle ambiguities inherent in natural language.
* Process visual queries for items difficult to describe verbally.
* Improve the overall accuracy and efficiency of item retrieval.

This report details the design, implementation, and evaluation of a novel multimodal AI system for lost and found item retrieval. Our system enables users to search for items using either text descriptions or image uploads, leveraging deep learning models to extract and align features from both modalities for highly accurate matching.

### **1.4 Related Work and Background**

The development of our multimodal lost and found system builds upon significant advancements in several areas of artificial intelligence:

* **Content-Based Image Retrieval (CBIR):** Early systems focused on retrieving images based on their visual content (e.g., color, texture, shape) rather than metadata. While effective for visual similarity, these often lacked semantic understanding and the ability to be queried by natural language.
* **Natural Language Processing (NLP):** The field of NLP has seen tremendous progress with the advent of **word embeddings** (e.g., Word2Vec, GloVe) and, more recently, **transformer-based models** (e.g., BERT, RoBERTa). These models can capture the semantic meaning and contextual relationships of words and sentences, enabling tasks like sentiment analysis, machine translation, and semantic search.
* **Computer Vision (CV):** Deep Convolutional Neural Networks (CNNs) like ResNet and EfficientNet have revolutionized image recognition, enabling highly accurate object detection, image classification, and feature extraction from visual data.

The most direct and influential predecessor to our system's core matching mechanism is **Contrastive Language-Image Pre-training (CLIP)**. Developed by OpenAI, CLIP demonstrated a groundbreaking approach to learning transferable visual models from natural language supervision.

* **CLIP's Core Idea:** CLIP learns to align images and text in a shared, high-dimensional embedding space. It does this by training on a massive dataset of (image, text) pairs, where the model learns to associate an image with its correct text description while distinguishing it from incorrect descriptions. This is achieved through a **contrastive learning objective** that maximizes the similarity between correct image-text pairs and minimizes the similarity between incorrect pairs.
* **Reference:** Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Dhariwal, G., Mcgrew, M., ... & Sutskever, I. (2021). Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning* (pp. 8748-8763). PMLR.

The concept of **embeddings** is fundamental here. An embedding is a dense numerical vector representation of an object (e.g., a word, a sentence, an image) that captures its semantic meaning and relationships in a continuous vector space. In this space, similar items are located close to each other, while dissimilar items are far apart. This allows for efficient **similarity search** using techniques like **FAISS (Facebook AI Similarity Search)**, which can quickly find the nearest neighbors (most similar items) to a given query embedding in a large database.

While these individual technologies have been extensively researched, a robust, accessible multimodal system specifically tailored for the Lost & Found domain, integrating both text and image input types with user feedback, presents a valuable application opportunity. Our project aims to bridge this gap by creating a practical and intelligent solution.

## **2. Method**

This section details the methodology employed in developing the Multimodal Lost & Found System. The system is designed to provide a flexible and intelligent search experience by processing both textual and visual queries, leveraging deep learning for multimodal feature extraction and similarity-based retrieval.

### **2.1 System Architecture Overview**

The proposed system operates through a series of interconnected components, as illustrated in the high-level block diagram below.

[Insert Diagram Here: A block diagram showing the overall system flow. **Left Side (Input):** \* User Text Query (Text Input field in Streamlit) \* User Image Query (Currently, the Streamlit UI only supports text input for search. Image input is for adding items to the database. This would be a future enhancement.) **Middle (Processing & Database):** \* Text Encoder (receives Text Input from st.text\_input) -> Text Embedding \* Image Encoder (used for indexing database images and for processing user image queries, though image search input is not yet implemented) -> Image Embedding \* Found Item Image Database (represented by dataset/amazon\_products.csv and dynamically added items) \* Pre-computed Image Embeddings (stored in LostFoundApp.image\_features as a PyTorch tensor) \* Item Descriptions (stored in LostFoundApp.original\_texts and LostFoundApp.image\_urls) **Right Side (Matching & Output):** \* Similarity Search (Direct tensor dot product @ in LostFoundApp.search\_lost\_item, not FAISS) \* Re-ranking Logic (Implicitly handled by direct text-to-image similarity if the text encoder is robust, but no explicit re-ranking with image features is present in search\_lost\_item) \* Top-K Results Display (Streamlit st.image and st.caption) \* User Feedback Loop (Simple "Did you find your desired item?" prompt) **Arrows:** Show flow from text input to Text Encoder, then to similarity search. Show database images going through Image Encoder to pre-computed embeddings. Show similarity search leading to results display, and then the feedback loop. Indicate the training phase vs. inference phase. **Title:** Figure 1: Multimodal Lost & Found System Architecture (Streamlit Implementation)]

The system's workflow can be summarized as follows:

1. **Data Ingestion:** A database of found item images is established, initially populated from dataset/amazon\_products.csv and dynamically expandable via user-provided image URLs. Each image is associated with a textual description (its 'title' from the CSV or 'N/A' for newly added items).
2. **Model Training:** Dedicated deep learning models (Image Encoder and Text Encoder) are trained using a contrastive learning approach to project both images and their corresponding text descriptions into a shared, semantically meaningful embedding space. Due to project constraints, this is a "minimal training" phase.
3. **Database Indexing:** Once models are initialized/trained, the Image Encoder generates embeddings for a subset of images from the found item database. These embeddings are stored in memory as a PyTorch tensor within the LostFoundApp instance.
4. **User Query:** A user submits a query via a text description in the Streamlit interface. (While the overall idea includes image input, the current Streamlit UI primarily supports text search.)
5. **Query Embedding:** The TextEncoder transforms the user's text query into an embedding in the shared vector space.
6. **Similarity Search:** The query embedding is directly compared (using dot product for cosine similarity) against all pre-computed image embeddings in the in-memory index. The top-K most similar items are identified.
7. **Result Retrieval & Display:** The images corresponding to the top matches, along with their original titles/descriptions, are retrieved and displayed in the Streamlit UI.
8. **User Feedback & Interaction:** A feedback loop prompts the user to confirm if their item was found, allowing for further interaction or termination. The system also supports adding new items to the database via URL.

### **2.2 Dataset Preparation**

For this project, a custom dataset of lost and found items was prepared to facilitate the training and evaluation of the multimodal system.

* **Data Collection:** The system primarily relies on an initial dataset loaded from dataset/amazon\_products.csv. For demonstration purposes, if this file is not found, a dummy CSV is generated with 20 placeholder image URLs (e.g., https://via.placeholder.com/150?text=DummyX) and corresponding generic titles (e.g., "D 0", "D 1"). This dummy data serves to ensure the application is runnable out-of-the-box. In a real-world scenario, this would be replaced by a comprehensive collection of actual lost and found item images.
* **Annotation Process:** Each entry in the amazon\_products.csv (or the dummy data) is expected to have an imgUrl and a title. The title column serves as the primary textual description for each image in the database. The LostFoundDataset class performs basic text cleaning (lowercasing, removing non-alphanumeric characters) on these titles.
* **Dataset Size:** The initial dataset size for indexing is set to a minimum of 50 items (or fewer if the CSV is smaller) from amazon\_products.csv. For the "minimal training" phase, only a small sample (min 10 items) from this dataset is used.
* **Data Split:** Due to the "minimal training" approach and the small dataset size for demonstration, a formal train/validation/test split for robust evaluation is not explicitly implemented in the provided code. The "minimal training" uses a small subset of the available data, and the indexing uses another sample. In a complete development cycle, rigorous data splitting would be crucial.

### **2.3 Model Implementation and Training**

The core of the multimodal system lies in its deep learning models, specifically the Image Encoder and Text Encoder, which are trained together using a contrastive learning objective.

#### **2.3.1 Image Encoder**

* **Purpose:** The ImageEncoder's role is to transform raw pixel data from an image into a dense numerical vector (embedding) that encapsulates its visual features.
* **Chosen Architecture:** We utilized a pre-trained **ResNet-18** model from torchvision.models as the backbone for our ImageEncoder. ResNet-18 is a Convolutional Neural Network (CNN) known for its efficiency and effectiveness in image recognition tasks.
* **Reasoning:** The choice of a pre-trained ResNet-18 is a strategic application of **transfer learning**. Training a robust CNN from scratch requires enormous datasets (millions of images) and significant computational resources (weeks on multiple GPUs). By using a model pre-trained on ImageNet, we leverage its learned general visual understanding (e.g., recognizing edges, textures, basic shapes) and fine-tune it for our specific domain with a smaller dataset. The weights='ResNet18\_Weights.DEFAULT' ensures that the pre-trained weights are loaded.
* **Modifications:** The original ResNet-18 model's final classification layer (self.cnn.fc) is replaced with a new nn.Linear layer that maps the extracted features to a fixed-size embedding dimension (e.g., 256 dimensions in our configuration). The output of this layer is then L2-normalized (torch.nn.functional.normalize) to ensure embeddings lie on a unit sphere, which is beneficial for cosine similarity calculations.
* **Diagram:** [Insert Diagram Here: A simple block diagram of the Image Encoder. **Input:** Image (224x224x3) **Blocks:** Pre-trained ResNet-18 Backbone (without final classification layer) -> Linear Projection Layer (to 256D) -> L2 Normalization **Output:** Image Embedding Vector (256 dimensions) **Title:** Figure 2: Image Encoder Architecture]

#### **2.3.2 Text Encoder**

* **Purpose:** The TextEncoder's function is to convert natural language descriptions into numerical vectors (embeddings) that capture their semantic meaning.
* **Chosen Architecture:** Instead of a large, pre-trained Transformer like BERT (which typically requires the Hugging Face transformers library and can be resource-intensive), we implemented a simpler text encoder using **nn.Embedding and a Gated Recurrent Unit (GRU) network (nn.GRU)**. This choice was made to demonstrate a more "from-scratch" approach to a text encoder within the constraints of a university project, avoiding heavy external dependencies for the core NLP model.
* **Reasoning:** While less powerful than large Transformers, an Embedding layer followed by a GRU can still learn meaningful sequence representations for shorter texts. The nn.Embedding layer maps discrete word IDs to dense continuous vectors, and the nn.GRU processes these sequences to produce a final sentence embedding.
* **Modifications:**
  + **Vocabulary:** A simple custom vocabulary mapping is used where words are hashed (hash(w) % self.vocab\_size) to get integer IDs. This avoids building and managing a complex vocabulary mapping.
  + **Padding:** Text sequences are padded to a max\_len (32 tokens) to ensure uniform input size for the GRU.
  + **Embedding & GRU:** The nn.Embedding layer converts word IDs to embeddings, which are then fed into the nn.GRU. The final hidden state of the GRU is used as the text embedding.
  + **Normalization:** The output embedding is L2-normalized.
* **Diagram:** [Insert Diagram Here: A simple block diagram of the Text Encoder. **Input:** Text String **Blocks:** Text Cleaning (lowercase, remove punctuation) -> Custom Hashing/Tokenization -> nn.Embedding Layer -> nn.GRU Layer -> Final Hidden State Extraction -> L2 Normalization **Output:** Text Embedding Vector (256 dimensions) **Title:** Figure 3: Text Encoder Architecture]

#### **2.3.3 Multimodal Embedding Alignment (Contrastive Learning)**

This is the core "training" phase where the system learns to align the visual and textual modalities. Our approach is inspired by the CLIP model's contrastive learning objective, implemented in the LostFoundSystem and minimal\_train\_model functions.

* **Core Principle:** The goal is to learn a shared embedding space where images and their corresponding text descriptions are mapped to nearby points, while unrelated images and texts are mapped far apart. This is achieved by maximizing the similarity between correct (image, text) pairs and minimizing the similarity between incorrect pairs within each training batch.
* **Loss Function:** We utilize nn.CrossEntropyLoss in PyTorch, which is adapted for a contrastive learning objective. For a batch of N (image, text) pairs:
  + Image features () and text features (T1​,...,TN​) are generated by their respective encoders.
  + These features are normalized to unit vectors (handled within the encoders).
  + A similarity matrix is computed, where each element sij​ represents the dot product (cosine similarity, as features are normalized) between image Ii​ and text Tj​.
  + The loss is calculated in two directions:
    - **Image-to-Text Loss:** For each image Ii​, the model is trained to predict its true corresponding text Ti​ among all N texts in the batch.
    - **Text-to-Image Loss:** Symmetrically, for each text Ti​, the model is trained to predict its true corresponding image Ii​ among all N images in the batch.
  + The total loss is the average of these two directional losses.
* **Temperature Parameter (**τ**):** A learnable temperature parameter (self.temperature) is applied to the similarity scores (logits) before the softmax operation. This parameter helps to scale the similarities, making the distribution sharper or softer, which is crucial for effective contrastive learning. The calculation is logits = (opt\_img @ opt\_txt.T) \* torch.exp(model\_to\_train.temperature).
* **Training Loop (minimal\_train\_model):** The training process involves iterating over a small subset of the dataset for a single epoch (num\_epochs=1). Data is processed in batches (size min(4, len(min\_dataset))). For each batch, a **forward pass** computes the embeddings and similarities. The loss is then calculated. A **backward pass** (loss.backward()) computes the gradients of the loss with respect to all model parameters. Finally, the optimizer.step() updates the model's weights based on these gradients, and optimizer.zero\_grad() clears the gradients for the next iteration.
* **Hyperparameters:** Key hyperparameters configured for this minimal training include:
  + embedding\_dim: 256 (dimension of the shared embedding space)
  + batch\_size: 4 (for minimal training)
  + num\_epochs: 1 (for minimal training)
  + learning\_rate: 0.0001

### **2.4 Database Indexing and Search**

Once the Image Encoder and Text Encoder are trained (or loaded), the system prepares its database for efficient item search.

* **Embedding Database Generation:** The LostFoundApp.\_build\_index method is responsible for generating embeddings for all images in the "found items" database. Each image is passed through the trained ImageEncoder, and its resulting embedding vector is stored in a PyTorch tensor self.image\_features. The corresponding image URLs and original texts are stored in self.image\_urls and self.original\_texts lists, respectively.
* **Similarity Search (In-Memory Tensor Operations):** For rapid retrieval of similar items, the system utilizes direct tensor operations rather than a specialized library like FAISS. When a user submits a query:
  1. The query (text) is transformed into an embedding using the TextEncoder.
  2. This query embedding is then compared against all pre-computed image embeddings in self.image\_features using a **dot product** (text\_feat @ self.image\_features.T). Since all embeddings are L2-normalized, this dot product directly yields the cosine similarity.
  3. torch.topk is used to efficiently retrieve the top\_k (e.g., 5) most similar items based on these cosine similarity scores.
  4. The system then retrieves the actual image URLs and their original descriptions corresponding to these top-K matches.
* **Re-ranking (Implicit):** In the current implementation, the search\_lost\_item function directly uses the cosine similarity between the text query and the image embeddings. There is no explicit re-ranking step that combines image-to-image similarity with text-to-description similarity, as was outlined in the previous method. The primary matching logic relies on the text query's embedding directly aligning with the visual content through the shared embedding space.

### **2.5 User Interface and Feedback Loop**

The system provides a user-friendly web interface built using Streamlit, incorporating input options, result display, and a feedback mechanism.

* **Input Options:** Users interact with the system via a st.text\_input field to provide a description of the lost item. (The initial project idea included image upload for search, but the current Streamlit UI implementation focuses on text-based search for the query. Image URLs can be added to the database via a sidebar input.)
* **Result Display:** Upon searching, the top matching images are displayed using st.image, along with their similarity scores. An st.expander allows users to view the original title/description and the image URL.
* **Feedback Mechanism:** After presenting the results, the system prompts the user with a question: "Did you find your desired item among these results? (yes/no)".
  + If the user responds "yes," a success message is displayed.
  + If the user responds "no," they are given the option to "try searching again" (which clears previous results) or "quit" (which effectively ends the interaction in the UI). This simple feedback loop aims to enhance user satisfaction.
* **Dynamic Database Addition:** A sidebar feature allows users to add new images to the in-memory database by providing a direct image URL. This feature dynamically updates the searchable index.

## **3. Results**

This section presents the expected outcomes of the Multimodal Lost & Found System, discusses common challenges encountered during development and training of such AI models, and outlines potential areas for future improvement.

### **3.1 Experimental Setup**

The development and execution of this Streamlit-based multimodal system were conducted using Python. The core deep learning operations are handled by PyTorch, with torchvision providing the pre-trained image backbone.

* **Hardware:** The system is explicitly configured to run on a **CPU only** (device = torch.device('cpu')). This significantly impacts training and inference speed compared to GPU-accelerated environments.
* **Software Environment:**
  + Python 3.x
  + Streamlit (for the web UI)
  + PyTorch (version compatible with torchvision)
  + Pillow (for image handling)
  + Pandas (for data management)
  + Requests (for fetching images from URLs)
  + re (for text cleaning)
  + sklearn.model\_selection (for data splitting, though minimally used in current training setup)

### **3.2 Training Performance (Expected)**

The training process for this project is designed as "minimal training" to ensure the application is runnable and demonstrates the core concepts quickly.

* **Loss Curve:** Given num\_epochs=1 and a small training sample (min 10 items), the loss curve will show a single epoch's progression. While the loss is expected to decrease, indicating initial learning, it will not reflect a fully converged or highly optimized model. The primary purpose is to establish a basic alignment between the image and text encoders. [Insert Graph Here: A line graph showing "Epoch" on the X-axis and "Loss" on the Y-axis. The line should show a decrease over the single epoch. Label it "Training Loss Curve (Minimal Training)".]
  + **Discussion:** The observed loss reduction over the single epoch confirms that the contrastive learning objective is being applied and the model parameters are beginning to adjust. However, this minimal training is a proof-of-concept. It demonstrates the *mechanism* of learning rather than achieving robust real-world performance. A fully trained model would require many more epochs and a much larger, diverse dataset.
* **Training Time:** Due to the single epoch, small batch size, and CPU-only execution, the "minimal training" phase completes very quickly, typically within seconds to a few minutes. This makes the application responsive during initialization but also highlights the limited extent of the actual training.

### **3.3 Search Results Showcase (Qualitative Analysis - Expected)**

The system's search capabilities, while functional, will reflect the limitations of the minimal training and the simplified dataset (especially with dummy placeholder images).

* **Example 1: Text Query for a Simple Item (from Dummy Data)**
  + **User Query:** "Dummy0"
  + **System Output (Expected Image):** The placeholder image https://via.placeholder.com/150?text=Dummy0.
  + **Stored Description:** "D 0"
  + **Discussion:** For exact or near-exact matches to the dummy titles, the system is expected to perform well, as the text encoder can easily learn to map these simple strings. The similarity score will likely be high. [Insert Screenshot Here: A screenshot of your Streamlit app's output for the query "Dummy0", showing the placeholder image and its description.]
* **Example 2: Text Query for a More Complex Description (from Dummy Data)**
  + **User Query:** "A red apple with a leaf" (assuming a dummy description like "D 15 with some random detail 15" exists)
  + **System Output (Expected Image):** A placeholder image (e.g., https://via.placeholder.com/150?text=Dummy15).
  + **Stored Description:** "D 15 with some random detail 15"
  + **Discussion:** The system's ability to semantically match more complex queries will be limited by the simplicity of the TextEncoder (GRU vs. Transformer) and the generic nature of the dummy descriptions. While it might retrieve *some* results, they are unlikely to be semantically rich matches to the visual content of placeholder images. The score will indicate its confidence. [Insert Screenshot Here: A screenshot of your Streamlit app's output for a more complex text query, showing placeholder images and their descriptions.]
* **Example 3: Adding a New Item and Searching**
  + **Action:** User adds a new item by URL (e.g., https://placehold.co/150x150/FF0000/FFFFFF?text=Red+Bag).
  + **User Query:** "Red Bag"
  + **System Output (Expected Image):** The newly added red bag placeholder image.
  + **Stored Description:** "N/A (Newly Added)"
  + **Discussion:** This demonstrates the dynamic indexing capability. The Image Encoder processes the new image, and its embedding is added to the index, making it immediately searchable by text. [Insert Screenshot Here: A screenshot showing the result of adding a new item and then searching for it.]

### **3.4 Performance Metrics (Expected)**

Given the nature of the dummy dataset and minimal training, quantitative metrics like Recall@K or mAP would likely be low if evaluated on a truly diverse, unseen dataset. However, for the purpose of this project, the qualitative demonstration of multimodal matching is the primary objective. The system's "performance" is best understood as its ability to correctly retrieve items when the query and database entries are within the limited scope of its training and data.

### **3.5 Challenges Encountered and Lessons Learned**

Developing this multimodal system, even in its simplified form, highlighted several critical challenges:

* **CPU-Only Execution:** Running deep learning models on a CPU significantly impacts performance. Training times are longer, and inference (especially for large databases) can be slow. This necessitated the "minimal training" approach and the use of simpler model architectures.
* **Minimal Training:** Training for only one epoch on a small sample of data means the models have learned only rudimentary feature representations and alignment. The system's ability to generalize to diverse, real-world images and complex text queries is severely limited. This was a deliberate choice to ensure the project is runnable within university course constraints, but it's a major limitation for practical deployment.
* **Simplified TextEncoder (GRU vs. Transformer):** The custom TextEncoder using nn.Embedding and nn.GRU is simpler to implement than a full Transformer model like BERT. However, it lacks the sophisticated contextual understanding and pre-trained knowledge that large Transformers provide, leading to less nuanced semantic matching for complex descriptions.
* **Absence of FAISS:** For larger databases (thousands to millions of images), the current in-memory tensor dot product for similarity search would become prohibitively slow. A dedicated library like FAISS is essential for scalable similarity search. Its omission simplifies the initial implementation but limits scalability.
* **Dummy Data Limitations:** The use of placeholder images and generic titles in the dummy dataset means the system cannot learn to recognize real-world objects or their specific visual attributes. The "matching" observed is primarily between the text query and the placeholder text labels, not a true understanding of visual content.
* **Missing Image Search Input:** While the project idea includes searching by image, the current Streamlit UI only implements text input for queries. This is a significant missing feature that would complete the multimodal input aspect.
* **Lack of Robust Error Handling for External URLs:** The system attempts to fetch images from URLs, but robust error handling for broken links, slow responses, or invalid image formats is limited, which can lead to gray placeholder images.

**Lessons Learned:**

* The trade-offs between model complexity, data requirements, and computational resources are critical in AI project development.
* Even minimal training can demonstrate core AI concepts, but real-world performance necessitates extensive training and high-quality data.
* Building a truly robust multimodal system requires sophisticated pre-trained models and efficient indexing solutions.
* User interface design is crucial for practical application, and ensuring all intended input modalities are supported is key for a complete user experience.

### **3.6 Future Work**

To evolve this Multimodal Lost & Found System into a more robust and practical solution, several key areas for future development are identified:

* **Comprehensive Dataset Acquisition:** The most critical next step is to replace the dummy data with a large, diverse, and well-annotated dataset of real-world lost and found items. This would enable the models to learn genuine visual and semantic features.
* **Full-Scale Model Training:** Train the ImageEncoder and TextEncoder for many more epochs on a significantly larger dataset. This would involve using more powerful GPUs or cloud computing resources.
* **Upgrade Text Encoder to Transformers:** Replace the current GRU-based TextEncoder with a pre-trained Transformer model (e.g., Sentence-BERT, a smaller BERT variant) from the Hugging Face transformers library. This would drastically improve semantic understanding and contextual matching for text queries.
* **Implement FAISS for Scalability:** Integrate FAISS for efficient similarity search. This is crucial for handling a large database of image embeddings and ensuring fast retrieval times.
* **Enable Image Upload for Search:** Implement the functionality to allow users to upload an image of their lost item directly for search. This would complete the multimodal input capability of the system.
* **Advanced Re-ranking:** Develop a more sophisticated re-ranking mechanism that explicitly combines both the image-to-query similarity and the text-description-to-query similarity (if the query was text) to provide a more accurate ranked list of results.
* **Attribute Extraction and Filtering:** Enhance the system to automatically extract and filter items based on specific attributes (e.g., color, material, brand, unique markings) from both visual and textual inputs.
* **Robust Error Handling and UI/UX:** Improve error handling for external image URLs and refine the Streamlit user interface for a more polished and intuitive user experience.

By addressing these areas, the Multimodal Lost & Found System can evolve into an even more powerful and indispensable tool for item retrieval, demonstrating the full potential of multimodal AI in a practical application.