**Proof of Concept (PoC): AI Agent for Products Image Finding and storing**

**Objective:**

To develop an intelligent agent capable of identifying three relevant images of a product based on its textual description by leveraging natural language processing, machine learning, and web APIs.

**Key Components:**

**1. Description-Based Searching**

**The AI agent identifies and retrieves images solely based on the product description provided by the user.**

**Techniques Used:**

1. **Named Entity Recognition (NER):**
   * Extracts key entities such as product names, brands, categories, or attributes from the description (e.g., "Nike Running Shoes").
2. **Part-of-Speech (POS) Tagging:**
   * Analyzes sentence structure to identify important nouns, adjectives, and other relevant parts of speech for precise query formulation.
3. **Dependency Parsing:**
   * Understands relationships between words to construct contextually accurate queries.
4. **Keyword Extraction:**
   * Extracts essential keywords to refine the search query.
5. **Text Classification:**
   * Classifies the description into predefined product categories for better contextual understanding.

**2. Data Collection and Preprocessing**

**The agent collects and preprocesses image data to ensure relevance and quality.**

**Steps:**

1. **Scraping Images from the Web:**
   * Utilize APIs (e.g., Google Custom Search API, unsplash API) to gather images based on the processed description.
2. **Labelling and Tagging:**
   * Automatically assign labels to retrieved images for better organization and filtering.
3. **Image Preprocessing:**
   * Resize, normalize, and enhance image quality to standardize inputs for downstream tasks.

**3. Model Design and Architecture**

**The core of the agent’s functionality is its ability to match textual descriptions with visual data.**

**Key Components:**

1. **Fine-Tune BERT/GPT:**
   * Fine-tune pre-trained language models to understand product descriptions accurately and formulate relevant queries.
2. **Building a CLIP Model for Text-Image Matching:**
   * Use the Contrastive Language-Image Pretraining (CLIP) model to align textual inputs with visual representations for robust matching.
3. **Integration with APIs:**
   * Connect with external APIs to fetch images based on the refined queries.

**4. Agentic AI**

**The agent is designed to operate autonomously, interact with web platforms, and make decisions during the retrieval process.**

**Features:**

1. **Agent Behaviour Design:**
   * Define workflows for query refinement, image retrieval, and result ranking.
2. **Web Interaction using APIs:**
   * Perform API-based searches on platforms like Google and Unsplash.
3. **Decision Making:**
   * Implement logic to select, rank, and filter the most relevant images.

**5. Integration with Web Platforms for Image Retrieval**

**The agent interacts with web platforms to retrieve images effectively.**

**Tools:**

1. **Google Custom Search API:**
   * Retrieves images from the internet based on the query.
2. **Bing Image Search API**
3. **eCommerce Scraping:**
   * Extracts images directly from e-commerce websites for specific product queries.

**6. Image Filtering and Presentation**

**Once images are retrieved, they are filtered and ranked to ensure relevance and quality.**

**Process:**

1. **Filter by Relevance and Quality:**
   * Remove low-quality or irrelevant images using visual and textual metadata.
2. **Rank Images:**
   * Rank the top three images using a scoring system based on relevance and quality metrics.
3. **Save Images:**
   * Save the three selected images in a designated folder, naming them as sku\_no\_1.png, sku\_no\_2.png, and sku\_no\_3.png. The file extension will depend on the original image format (e.g., .jpg, .png).
4. **Present to User (optional):**
   * Display the images in a user-friendly interface.

**7. Performance Evaluation and Optimization**

To ensure the agent’s effectiveness, performance metrics and optimization techniques are applied.

**Evaluation Metrics:**

1. **Cosine Similarity:**
   * Measures similarity between the text description and image metadata.
2. **BLEU Score:**
   * Assesses textual relevance if captions or metadata are available.

**Process Workflow:**

**Step 1: Input Processing**

* User provides a textual description of the product.
* The agent processes the description using NER, POS tagging, dependency parsing, keyword extraction, and text classification.

**Step 2: Query Formulation**

* Refine the description into an optimized query using NLP techniques.
* Categorize the product based on keywords and contextual information.

**Step 3: Image Retrieval**

* Send API requests to platforms like Google Custom Search or bing image search API.
* Fetch a set of images matching the query.

**Step 4: Image Filtering**

* Evaluate the retrieved images for relevance using metadata and visual attributes.
* Remove duplicates or low-quality images.

**Step 5: Ranking**

* Rank the top three images based on their relevance to the input description.
* Use scoring metrics like cosine similarity or FID to determine the order.

**Step 6: Saving Images**

* Save the top three images in a specific folder.
* Name the images as sku\_no\_1.png, sku\_no\_2.png, and sku\_no\_3.png, ensuring the file extension matches the image format.

**Step 7: Presentation(optional)**

* Display the final set of three images in a user-friendly interface.
* Allow users to download or interact with the images as needed.

**Requirements**

APIs: Google Custom Search API, Unsplash API, Bing image search API

**Benefits:**

* **Accuracy:** Ensures data correctness by cross-referencing with trusted sources.
* **Efficiency:** Automates repetitive verification tasks, saving time.
* **Scalability:** Easily integrates into large-scale inventory or e-commerce systems.
* **Reliability:** Minimizes errors in SKU-based product identification and photo validation.

**1. Using Bing API key( Azure Microsoft)**

**Step 1: Input Processing**

Here, the user inputs a detailed product description. The agent processes this description using NLP techniques like Named Entity Recognition (NER), Part-of-Speech (POS) tagging, and dependency parsing.

**Libraries:**

* spaCy: For NER, POS tagging, dependency parsing, and keyword extraction.
* nltk or textblob: For additional text classification.

**Code:**

import spacy

# Load the spaCy model

nlp = spacy.load("en\_core\_web\_sm")

def process\_input\_description(description):

doc = nlp(description)

# Extract Named Entities (NER)

entities = [ent.text for ent in doc.ents]

# Extract Keywords using POS tagging (nouns, adjectives)

keywords = [token.text for token in doc if token.pos\_ in ['NOUN', 'ADJ']]

# Dependency parsing (optional but can help with context)

dependencies = [(token.text, token.dep\_, token.head.text) for token in doc]

# Text Classification: Categorize (simple example using keywords)

categories = set(keywords)

return entities, keywords, dependencies, categories

# Example usage:

description = "Apple Watch Series 10 [GPS + Cellular 42 mm] Smartwatch with Jet Black Aluminium Case with Black Sport Band- S/M. Fitness Tracker, ECG App, Always-On Retina Display, Water Resistant"

entities, keywords, dependencies, categories = process\_input\_description(description)

print("Entities:", entities)

print("Keywords:", keywords)

print("Dependencies:", dependencies)

print("Categories:", categories)

**Step 2: Query Formulation**

Refine the description to form an optimized query by extracting key terms and combining them into a more search-friendly query.

**Code:**

def formulate\_query(keywords, categories):

# Combine categories and keywords for the search query

query = " ".join(keywords)

return query

# Example usage

query = formulate\_query(keywords, categories)

print("Optimized Query:", query)

**Step 3: Image Retrieval**

For image retrieval, we can use the **Bing Image Search API** or **Google Custom Search**. For simplicity, I’ll show an example with the Bing Image Search API (free tier).

**API Call (Bing):**

import requests

API\_KEY = 'your\_bing\_api\_key' # Replace with your actual Bing API key

def search\_images(query, num\_results=3):

url = 'https://api.cognitive.microsoft.com/bing/v7.0/images/search'

headers = {

'Ocp-Apim-Subscription-Key': API\_KEY

}

params = {

'q': query, # The search query

'count': num\_results, # Number of results to fetch

'safeSearch': 'Moderate',

}

response = requests.get(url, headers=headers, params=params)

if response.status\_code == 200:

results = response.json()

return [img['contentUrl'] for img in results['value']]

else:

print("Error fetching images.")

return []

# Example usage

images = search\_images(query)

print("Fetched Images:", images)

**Step 4: Image Filtering**

To evaluate the retrieved images for relevance, you would ideally need metadata (like dimensions) and possibly use an image quality detection algorithm (e.g., using OpenCV). For now, let’s filter based on the size of the image.

**Code:**

from PIL import Image

from io import BytesIO

import requests

def filter\_images(image\_urls):

filtered\_images = []

for img\_url in image\_urls:

img\_data = requests.get(img\_url).content

img = Image.open(BytesIO(img\_data))

# Filter based on image size (you can define your own criteria)

if img.size[0] > 300 and img.size[1] > 300: # Check if image dimensions are adequate

filtered\_images.append(img\_url)

return filtered\_images

# Example usage

filtered\_images = filter\_images(images)

print("Filtered Images:", filtered\_images)

**Step 5: Ranking**

You can use scoring metrics like **cosine similarity** or **Fréchet Inception Distance (FID)** to rank images based on relevance to the product description. For simplicity, we will just rank by the number of matching keywords in the image metadata (you can extend this logic later).

**Code:**

from sklearn.metrics.pairwise import cosine\_similarity

import numpy as np

def rank\_images(image\_urls, query):

# For simplicity, rank images by keyword matching using cosine similarity (you can improve this logic)

ranked\_images = []

for img\_url in image\_urls:

similarity\_score = calculate\_similarity(query, img\_url) # This could be more complex

ranked\_images.append((img\_url, similarity\_score))

ranked\_images.sort(key=lambda x: x[1], reverse=True) # Sort by similarity score

return ranked\_images[:3] # Return top 3 images

def calculate\_similarity(query, img\_url):

# A very simplistic similarity score for the demonstration (based on keyword match)

return np.random.random() # Random value for demo purposes

# Example usage

ranked\_images = rank\_images(filtered\_images, query)

print("Ranked Images:", ranked\_images)

**Step 6: Saving Images**

Finally, save the top 3 ranked images locally with specific naming conventions (SKU number).

**Code:**

def save\_images(image\_urls, sku\_number):

for idx, img\_url in enumerate(image\_urls):

img\_data = requests.get(img\_url).content

with open(f"{sku\_number}\_{idx + 1}.png", 'wb') as f:

f.write(img\_data)

print(f"Image saved as {sku\_number}\_{idx + 1}.png")

# Example usage

save\_images([img[0] for img in ranked\_images], "SKU12345")

**Full Workflow:**

Combining everything together into one cohesive function:

def process\_and\_save\_images(description, sku\_number):

# Step 1: Input Processing

entities, keywords, dependencies, categories = process\_input\_description(description)

# Step 2: Query Formulation

query = formulate\_query(keywords, categories)

# Step 3: Image Retrieval

images = search\_images(query)

# Step 4: Image Filtering

filtered\_images = filter\_images(images)

# Step 5: Ranking

ranked\_images = rank\_images(filtered\_images, query)

# Step 6: Saving Images

save\_images([img[0] for img in ranked\_images], sku\_number)

# Example usage

description = "Apple Watch Series 10 [GPS + Cellular 42 mm] Smartwatch with Jet Black Aluminium Case with Black Sport Band- S/M. Fitness Tracker, ECG App, Always-On Retina Display, Water Resistant"

sku\_number = "SKU12345"

process\_and\_save\_images(description, sku\_number)

**Summary:**

This solution uses a **step-by-step approach** to process the user-provided product description, fetch relevant images, and save them with appropriate filenames. Key points:

1. **Text Processing**: Extract keywords and entities from the description using spaCy.
2. **Query Formation**: Formulate an optimized search query.
3. **Image Retrieval**: Use the Bing Image Search API to fetch images.
4. **Image Filtering**: Filter based on image size and quality.
5. **Ranking**: Rank images based on their relevance to the description.
6. **Saving Images**: Save the top 3 images with specific names (SKU-based).

You can expand the filtering and ranking mechanisms to include more sophisticated metrics like visual features or metadata analysis as needed.