Part 3: Multimodal & Interface Upgrade (Advanced continuation)

Project Overview

This notebook demonstrates the creation of both Streamlit and Gradio web applications for our multimodal search engine. The search engine can:

- Find images using text descriptions (text-to-image search)
- Find text descriptions using uploaded images (image-to-text search)

1. System Information and Setup

In [1]:
```
# Import necessary libraries
import streamlit as st
import gradio as gr
import torch
import torchvision.transforms as transforms
from transformers import CLIPProcessor, CLIPModel
from PIL import Image
import numpy as np
import pandas as pd
import os
import json
from sklearn.metrics.pairwise import cosine_similarity
import warnings
import platform
import psutil
import sys
from datetime import datetime
warnings.filterwarnings('ignore')

# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Display comprehensive system information
print("=" * 80)
print("🔍 MULTIMODAL INTERFACE - SYSTEM STATUS")
print("=" * 80)
print(f"📅 Timestamp: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
print()

# System Information
print("💻 SYSTEM INFORMATION")
print("-" * 40)
print(f"Platform: {platform.platform()}")
print(f"Architecture: {platform.architecture()[0]}")
print(f"Processor: {platform.processor()}")
print(f"Python Version: {sys.version.split()[0]}")
print(f"PyTorch Version: {torch.__version__}")
print(f"Streamlit Version: {st.__version__}")
print()
```

```
# Hardware Information
print("⚡ HARDWARE INFORMATION")
print("-" * 40)
print(f"CPU Cores: {psutil.cpu_count(logical=False)} physical, {psutil.cpu_count
print(f"RAM: {psutil.virtual_memory().total / (1024**3):.1f} GB total, {psutil.v
print(f"RAM Usage: {psutil.virtual_memory().percent:.1f}%")

# GPU Information
print(f"Device: {device}")
if torch.cuda.is_available():
    print(f"GPU: {torch.cuda.get_device_name(0)}")
    print(f"GPU Memory: {torch.cuda.get_device_properties(0).total_memory / (102
    print(f"CUDA Version: {torch.version.cuda}")
    print(f"cuDNN Version: {torch.backends.cudnn.version()}")
else:
    print("GPU: Not available (using CPU)")
print()

# Project Status
print("📁 PROJECT STATUS")
print("-" * 40)

# Check if data exists
data_path = '../data/'
if os.path.exists(data_path):
    print("✅ Data directory found")
    if os.path.exists('../data/images/'):
        image_files = [f for f in os.listdir('../data/images/') if f.lower().end
        image_count = len(image_files)
        print(f"✅ {image_count} images found")
        if image_count > 0:
            total_size = sum(os.path.getsize(os.path.join('../data/images/', f))
            print(f"   Total image size: {total_size:.1f} MB")
    if os.path.exists('../data/captions.txt'):
        with open('../data/captions.txt', 'r') as f:
            caption_count = sum(1 for line in f)
        caption_size = os.path.getsize('../data/captions.txt') / 1024
        print(f"✅ {caption_count} captions found ({caption_size:.1f} KB)")
    if os.path.exists('../data/Flickr8k.token.txt'):
        token_size = os.path.getsize('../data/Flickr8k.token.txt') / 1024
        print(f"✅ Flickr8k token file found ({token_size:.1f} KB)")
else:
    print("❌ Data directory not found!")

# Check if embeddings exist
embeddings_path = '../embeddings/'
if os.path.exists(embeddings_path):
    print("✅ Embeddings directory found")
    if os.path.exists('../embeddings/image_embeddings.npy'):
        image_emb_size = os.path.getsize('../embeddings/image_embeddings.npy') /
        print(f"✅ Image embeddings found ({image_emb_size:.1f} MB)")
    if os.path.exists('../embeddings/text_embeddings.npy'):
        text_emb_size = os.path.getsize('../embeddings/text_embeddings.npy') / (
        print(f"✅ Text embeddings found ({text_emb_size:.1f} MB)")
    if os.path.exists('../embeddings/metadata.csv'):
        metadata_size = os.path.getsize('../embeddings/metadata.csv') / 1024
        print(f"✅ Metadata found ({metadata_size:.1f} KB)")
    if os.path.exists('../embeddings/model_info.json'):
        print("✅ Model info found")
```

```
else:
    print("❌ Embeddings directory not found - please run Part 1 first!")

print()
print("🚀 READY TO BUILD MULTIMODAL INTERFACE")
print("=" * 80)
```

```
================================================================================
🔍 MULTIMODAL INTERFACE - SYSTEM STATUS
================================================================================
📅 Timestamp: 2025-09-11 01:00:53

🖥️  SYSTEM INFORMATION
----------------------------------------
Platform: Windows-11-10.0.26100-SP0
Architecture: 64bit
Processor: Intel64 Family 6 Model 151 Stepping 5, GenuineIntel
Python Version: 3.12.9
PyTorch Version: 2.8.0+cpu
Streamlit Version: 1.49.1

⚡ HARDWARE INFORMATION
----------------------------------------
CPU Cores: 6 physical, 12 logical
RAM: 15.8 GB total, 2.1 GB available
RAM Usage: 86.9%
Device: cpu
GPU: Not available (using CPU)

📁 PROJECT STATUS
----------------------------------------
✅ Data directory found
✅ 8091 images found
   Total image size: 1063.1 MB
✅ 40460 captions found (3355.2 KB)
✅ Flickr8k token file found (3355.2 KB)
✅ Embeddings directory found
✅ Image embeddings found (1.0 MB)
✅ Text embeddings found (1.0 MB)
✅ Metadata found (58.3 KB)
✅ Model info found

🚀 READY TO BUILD MULTIMODAL INTERFACE
================================================================================
```

2. Load Model and Data

In [2]:
```
# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"🖥️  Using device: {device}")

# Load CLIP model
print("🔄 Loading CLIP model...")
model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32").to(device)
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")
print("✅ CLIP model loaded successfully!")

# Load embeddings data
print("🔄 Loading embeddings data...")
```

```
image_embeddings = np.load('../embeddings/image_embeddings.npy')
text_embeddings = np.load('../embeddings/text_embeddings.npy')
metadata = pd.read_csv('../embeddings/metadata.csv')

# Load model info
with open('../embeddings/model_info.json', 'r') as f:
    model_info = json.load(f)

print("✅ Embeddings data loaded successfully!")
print(f"📊 Image embeddings shape: {image_embeddings.shape}")
print(f"📊 Text embeddings shape: {text_embeddings.shape}")
print(f"📊 Metadata shape: {metadata.shape}")
print(f"📊 Model info: {model_info}")
```

```
🖥️  Using device: cpu
🔄 Loading CLIP model...
Using a slow image processor as `use_fast` is unset and a slow processor was save
d with this model. `use_fast=True` will be the default behavior in v4.52, even if
the model was saved with a slow processor. This will result in minor differences
in outputs. You'll still be able to use a slow processor with `use_fast=False`.
Fetching 1 files:   0%|          | 0/1 [00:00<?, ?it/s]
✅ CLIP model loaded successfully!
🔄 Loading embeddings data...
✅ Embeddings data loaded successfully!
📊 Image embeddings shape: (500, 512)
📊 Text embeddings shape: (500, 512)
📊 Metadata shape: (500, 3)
📊 Model info: {'model_name': 'openai/clip-vit-base-patch32', 'embedding_dim': 5
12, 'num_images': 100, 'total_embeddings': 500, 'num_samples': 500, 'dataset': 'F
lickr8k (partial - 100/8091 images)', 'processing_date': '2025-09-10', 'device_us
ed': 'cpu', 'note': 'Only 100 unique images processed out of 8091 total images'}
```

3. Search Functions

In [3]:
```
# Text-to-Image Search Function
def text_to_image_search(query_text, top_k=5):
    """Search for images based on text query"""
    # Generate embedding for text query
    inputs = processor(text=[query_text], return_tensors="pt", padding=True).to(

    with torch.no_grad():
        query_embedding = model.get_text_features(**inputs)
        query_embedding = query_embedding / query_embedding.norm(dim=-1, keepdim

    # Calculate similarities with all image embeddings
    similarities = cosine_similarity(query_embedding.cpu().numpy(), image_embedd

    # Get top-k most similar images
    top_indices = np.argsort(similarities)[::-1][:top_k]

    results = []
    for idx in top_indices:
        result = {
            'image_id': metadata.iloc[idx]['image_id'],
            'image_path': metadata.iloc[idx]['image_path'],
            'caption': metadata.iloc[idx]['caption'],
            'similarity': similarities[idx]
        }
        results.append(result)
    return results

# Image-to-Text Search Function
def image_to_text_search(uploaded_image, top_k=5):
    """Search for text descriptions based on uploaded image"""
    # Generate embedding for uploaded image
    inputs = processor(images=uploaded_image, return_tensors="pt").to(device)

    with torch.no_grad():
        query_embedding = model.get_image_features(**inputs)
        query_embedding = query_embedding / query_embedding.norm(dim=-1, keepdim

    # Calculate similarities with all text embeddings
    similarities = cosine_similarity(query_embedding.cpu().numpy(), text_embeddi

    # Get top-k most similar text descriptions
    top_indices = np.argsort(similarities)[::-1][:top_k]

    results = []
    for idx in top_indices:
        result = {
            'image_id': metadata.iloc[idx]['image_id'],
            'image_path': metadata.iloc[idx]['image_path'],
            'caption': metadata.iloc[idx]['caption'],
            'similarity': similarities[idx]
        }
        results.append(result)

    return results

print("✅ Search functions defined successfully!")
```

```
✅ Search functions defined successfully!
```

4. Test Search Functions

In [4]:
```
# Test text-to-image search
print("🔍 Testing text-to-image search...")
test_query = "a dog playing"
results = text_to_image_search(test_query, top_k=3)

print(f"Query: '{test_query}'")
print(f"Found {len(results)} results:")
for i, result in enumerate(results, 1):
    print(f"{i}. Similarity: {result['similarity']:.3f}")
    print(f"   Caption: {result['caption']}")
    print(f"   Image ID: {result['image_id']}")
    print()

print("✅ Text-to-image search test completed!")
```

```
🔍 Testing text-to-image search...
Query: 'a dog playing'
Found 3 results:
1. Similarity: 0.324
   Caption: A black and white dog catches a toy in midair .
   Image ID: 1072153132_53d2bb1b60

2. Similarity: 0.324
   Caption: A multicolor dog jumping to catch a tennis ball in a grassy field .
   Image ID: 1072153132_53d2bb1b60

3. Similarity: 0.324
   Caption: A dog leaps while chasing a tennis ball through a grassy field .
   Image ID: 1072153132_53d2bb1b60

✅ Text-to-image search test completed!
```

5. Create Standalone Streamlit App

In [ ]:
```
# Create the complete Streamlit app code
streamlit_code = '''

import streamlit as st
import torch
import torchvision.transforms as transforms
from transformers import CLIPProcessor, CLIPModel
from PIL import Image
import numpy as np
import pandas as pd
import os
import json
import io
from datetime import datetime
from sklearn.metrics.pairwise import cosine_similarity
import warnings
warnings.filterwarnings('ignore')

# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Load CLIP model
@st.cache_resource
def load_clip_model():
    """Load CLIP model and processor"""
    model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32").to(device)
    processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")
    return model, processor

# Load embeddings data
@st.cache_data
def load_embeddings_data():
    """Load pre-computed embeddings and metadata"""
    # Load embeddings
    image_embeddings = np.load('embeddings/image_embeddings.npy')
    text_embeddings = np.load('embeddings/text_embeddings.npy')

    # Load metadata
    metadata = pd.read_csv('embeddings/metadata.csv')
    # Load model info
    with open('embeddings/model_info.json', 'r') as f:
        model_info = json.load(f)

    return image_embeddings, text_embeddings, metadata, model_info

# Load model and data
model, processor = load_clip_model()
image_embeddings, text_embeddings, metadata, model_info = load_embeddings_data()

# Text-to-Image Search Function
def text_to_image_search(query_text, top_k=5):
    """Search for images based on text query"""
    # Generate embedding for text query
    inputs = processor(text=[query_text], return_tensors="pt", padding=True).to(

    with torch.no_grad():
        query_embedding = model.get_text_features(**inputs)
        query_embedding = query_embedding / query_embedding.norm(dim=-1, keepdim

    # Calculate similarities with all image embeddings
    similarities = cosine_similarity(query_embedding.cpu().numpy(), image_embedd

    # Get top-k most similar images
    top_indices = np.argsort(similarities)[::-1][:top_k]

    results = []
    for idx in top_indices:
        result = {
            'image_id': metadata.iloc[idx]['image_id'],
            'image_path': metadata.iloc[idx]['image_path'],
            'caption': metadata.iloc[idx]['caption'],
            'similarity': similarities[idx]
        }
        results.append(result)

    return results

# Image-to-Text Search Function
def image_to_text_search(uploaded_image, top_k=5):
    """Search for text descriptions based on uploaded image"""
    # Generate embedding for uploaded image
    inputs = processor(images=uploaded_image, return_tensors="pt").to(device)

    with torch.no_grad():
        query_embedding = model.get_image_features(**inputs)
        query_embedding = query_embedding / query_embedding.norm(dim=-1, keepdim

    # Calculate similarities with all text embeddings
    similarities = cosine_similarity(query_embedding.cpu().numpy(), text_embeddi

    # Get top-k most similar text descriptions
    top_indices = np.argsort(similarities)[::-1][:top_k]

    results = []
    for idx in top_indices:
        result = {
            'image_id': metadata.iloc[idx]['image_id'],
            'image_path': metadata.iloc[idx]['image_path'],
            'caption': metadata.iloc[idx]['caption'],
                'similarity': similarities[idx]
            }
            results.append(result)

        return results

# Custom CSS for modern UI
def load_css():
    st.markdown("""
    <style>
    /* Modern theme colors - Clean & Minimal */
    :root {
        --primary-color: #2563eb;
        --primary-light: #3b82f6;
        --primary-dark: #1d4ed8;
        --secondary-color: #7c3aed;
        --accent-color: #06b6d4;
        --success-color: #059669;
        --warning-color: #d97706;
        --error-color: #dc2626;
        --dark-color: #111827;
        --dark-light: #374151;
        --light-color: #ffffff;
        --gray-50: #f9fafb;
        --gray-100: #f3f4f6;
        --gray-200: #e5e7eb;
        --gray-300: #d1d5db;
        --gray-400: #9ca3af;
        --gray-500: #6b7280;
        --gray-600: #4b5563;
        --gray-700: #374151;
        --gray-800: #1f2937;
        --gray-900: #111827;
        --gradient-primary: linear-gradient(135deg, #2563eb 0%, #3b82f6 50%, #7c
        --gradient-secondary: linear-gradient(135deg, #06b6d4 0%, #3b82f6 100%);
        --gradient-accent: linear-gradient(135deg, #f59e0b 0%, #f97316 100%);
        --shadow-xs: 0 1px 2px 0 rgba(0, 0, 0, 0.05);
        --shadow-sm: 0 1px 3px 0 rgba(0, 0, 0, 0.1), 0 1px 2px 0 rgba(0, 0, 0, 0
        --shadow-md: 0 4px 6px -1px rgba(0, 0, 0, 0.1), 0 2px 4px -1px rgba(0, 0
        --shadow-lg: 0 10px 15px -3px rgba(0, 0, 0, 0.1), 0 4px 6px -2px rgba(0,
        --shadow-xl: 0 20px 25px -5px rgba(0, 0, 0, 0.1), 0 10px 10px -5px rgba(
        --shadow-2xl: 0 25px 50px -12px rgba(0, 0, 0, 0.25);
        --border-radius: 8px;
        --border-radius-md: 12px;
        --border-radius-lg: 16px;
        --border-radius-xl: 20px;
        --border-radius-2xl: 24px;
    }

    /* Reset and base styles */
    * {
        box-sizing: border-box;
    }

    /* Main container */
    .main .block-container {
        padding: 2rem 1rem;
        max-width: 1200px;
        margin: 0 auto;
        background: var(--gray-50);
        min-height: 100vh;
    }

    /* Header styling - Clean & Modern */
    .main-header {
        background: var(--light-color);
        padding: 3rem 2rem;
        border-radius: var(--border-radius-2xl);
        margin-bottom: 2rem;
        box-shadow: var(--shadow-lg);
        text-align: center;
        color: var(--dark-color);
        position: relative;
        overflow: hidden;
        border: 1px solid var(--gray-200);
    }

    .main-header::before {
        content: '';
        position: absolute;
        top: 0;
        left: 0;
        right: 0;
        height: 4px;
        background: var(--gradient-primary);
    }

    .main-header h1 {
        font-size: 3rem;
        font-weight: 700;
        margin: 0;
        color: var(--dark-color);
        position: relative;
        z-index: 1;
        letter-spacing: -0.025em;
    }

    .main-header p {
        font-size: 1.125rem;
        margin: 1rem 0 0 0;
        color: var(--gray-600);
        position: relative;
        z-index: 1;
        font-weight: 400;
    }

    /* Hero section - Clean stats */
    .hero-stats {
        display: flex;
        justify-content: center;
        gap: 1.5rem;
        margin-top: 2rem;
        position: relative;
        z-index: 1;
        flex-wrap: wrap;
    }

    .hero-stat {
        text-align: center;
        background: var(--gray-50);
        padding: 1.5rem 1.25rem;
        border-radius: var(--border-radius-lg);
        border: 1px solid var(--gray-200);
        min-width: 120px;
        transition: all 0.2s ease;
    }

    .hero-stat:hover {
        transform: translateY(-2px);
        box-shadow: var(--shadow-md);
        border-color: var(--primary-color);
    }

    .hero-stat .number {
        font-size: 1.75rem;
        font-weight: 700;
        display: block;
        color: var(--primary-color);
        margin-bottom: 0.25rem;
    }

    .hero-stat .label {
        font-size: 0.875rem;
        color: var(--gray-600);
        margin: 0;
        font-weight: 500;
    }

    /* Sidebar styling - Clean & Minimal */
    .css-1d391kg {
        background: var(--light-color);
        border-right: 1px solid var(--gray-200);
    }

    .sidebar .sidebar-content {
        background: var(--light-color);
        padding: 1.5rem 1rem;
    }

    .sidebar .sidebar-content .element-container {
        margin-bottom: 1.5rem;
    }

    .sidebar h3 {
        color: var(--dark-color);
        font-size: 1rem;
        font-weight: 600;
        margin-bottom: 1rem;
        padding-bottom: 0.5rem;
        border-bottom: 1px solid var(--gray-200);
        letter-spacing: 0.025em;
    }

    /* Card styling - Clean & Modern */
    .metric-card {
        background: var(--light-color);
        padding: 1.25rem;
        border-radius: var(--border-radius-lg);
        box-shadow: var(--shadow-sm);
        border: 1px solid var(--gray-200);
        margin-bottom: 1rem;
        transition: all 0.2s ease;
        position: relative;
        overflow: hidden;
    }

    .metric-card::before {
        content: '';
        position: absolute;
        top: 0;
        left: 0;
        right: 0;
        height: 2px;
        background: var(--gradient-primary);
    }

    .metric-card:hover {
        transform: translateY(-2px);
        box-shadow: var(--shadow-md);
        border-color: var(--primary-color);
    }

    .metric-card h3 {
        color: var(--gray-600);
        font-size: 0.75rem;
        font-weight: 600;
        margin: 0 0 0.5rem 0;
        text-transform: uppercase;
        letter-spacing: 0.05em;
    }

    .metric-card .value {
        font-size: 1.5rem;
        font-weight: 700;
        color: var(--dark-color);
        margin: 0;
    }

    /* Search card - Clean & Modern */
    .search-card {
        background: var(--light-color);
        border-radius: var(--border-radius-xl);
        padding: 2rem;
        box-shadow: var(--shadow-sm);
        border: 1px solid var(--gray-200);
        margin-bottom: 2rem;
        position: relative;
        overflow: hidden;
    }

    .search-card::before {
        content: '';
        position: absolute;
        top: 0;
        left: 0;
        right: 0;
        height: 3px;
        background: var(--gradient-primary);
    }
    /* Results grid */
    .results-grid {
        display: grid;
        grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
        gap: 1.5rem;
        margin-top: 2rem;
    }

    /* Button styling - Clean & Modern */
    .stButton > button {
        background: var(--primary-color);
        color: white;
        border: none;
        border-radius: var(--border-radius-md);
        padding: 0.75rem 1.5rem;
        font-weight: 600;
        font-size: 0.875rem;
        transition: all 0.2s ease;
        box-shadow: var(--shadow-sm);
        position: relative;
        overflow: hidden;
    }

    .stButton > button:hover {
        background: var(--primary-dark);
        transform: translateY(-1px);
        box-shadow: var(--shadow-md);
    }

    .stButton > button:active {
        transform: translateY(0);
        box-shadow: var(--shadow-sm);
    }

    /* Primary button variant */
    .stButton > button[kind="primary"] {
        background: var(--primary-color);
        box-shadow: var(--shadow-md);
    }

    .stButton > button[kind="primary"]:hover {
        background: var(--primary-dark);
        box-shadow: var(--shadow-lg);
    }

    /* Search input styling - Clean & Modern */
    .stTextInput > div > div > input {
        border-radius: var(--border-radius-md);
        border: 1px solid var(--gray-300);
        padding: 0.875rem 1rem;
        font-size: 1rem;
        font-weight: 400;
        transition: all 0.2s ease;
        background: var(--light-color);
        box-shadow: var(--shadow-xs);
    }

    .stTextInput > div > div > input:focus {
        border-color: var(--primary-color);
        box-shadow: 0 0 0 3px rgba(37, 99, 235, 0.1), var(--shadow-xs);
        background: var(--light-color);
        outline: none;
    }

    .stTextInput > div > div > input::placeholder {
        color: var(--gray-400);
        font-weight: 400;
    }

    /* Popular search buttons - Clean & Modern */
    .popular-search-btn {
        background: var(--light-color);
        border: 1px solid var(--gray-300);
        border-radius: var(--border-radius);
        padding: 0.5rem 0.875rem;
        margin: 0.25rem;
        font-size: 0.875rem;
        font-weight: 500;
        transition: all 0.2s ease;
        display: inline-block;
        text-decoration: none;
        color: var(--gray-700);
        box-shadow: var(--shadow-xs);
    }

    .popular-search-btn:hover {
        border-color: var(--primary-color);
        background: var(--primary-color);
        color: white;
        transform: translateY(-1px);
        box-shadow: var(--shadow-sm);
    }

    /* Results styling - Clean & Modern */
    .result-card {
        background: var(--light-color);
        border-radius: var(--border-radius-lg);
        box-shadow: var(--shadow-sm);
        overflow: hidden;
        transition: all 0.2s ease;
        border: 1px solid var(--gray-200);
    }

    .result-card:hover {
        transform: translateY(-2px);
        box-shadow: var(--shadow-md);
        border-color: var(--primary-color);
    }

    .result-card img {
        width: 100%;
        height: 200px;
        object-fit: cover;
    }

    .result-card .content {
        padding: 1rem;
    }

    .result-card .similarity {
        background: var(--primary-color);
        color: white;
        padding: 0.25rem 0.5rem;
        border-radius: var(--border-radius);
        font-size: 0.75rem;
        font-weight: 600;
        display: inline-block;
        margin-bottom: 0.5rem;
    }

    .result-card .caption {
        color: var(--gray-700);
        font-size: 0.875rem;
        line-height: 1.5;
        margin: 0;
    }

    /* Status messages - Clean & Modern */
    .stSuccess {
        background: var(--success-color);
        color: white;
        padding: 1rem;
        border-radius: var(--border-radius-md);
        border: none;
        box-shadow: var(--shadow-sm);
    }

    .stError {
        background: var(--error-color);
        color: white;
        padding: 1rem;
        border-radius: var(--border-radius-md);
        border: none;
        box-shadow: var(--shadow-sm);
    }

    .stWarning {
        background: var(--warning-color);
        color: white;
        padding: 1rem;
        border-radius: var(--border-radius-md);
        border: none;
        box-shadow: var(--shadow-sm);
    }

    /* Tabs styling - Clean & Modern */
    .stTabs [data-baseweb="tab-list"] {
        gap: 0.25rem;
        background: var(--gray-100);
        padding: 0.25rem;
        border-radius: var(--border-radius-lg);
        margin-bottom: 2rem;
        border: 1px solid var(--gray-200);
    }

    .stTabs [data-baseweb="tab"] {
        background: transparent;
        border-radius: var(--border-radius-md);
        border: none;
        padding: 0.75rem 1.5rem;
        font-weight: 600;
        font-size: 0.875rem;
        transition: all 0.2s ease;
        color: var(--gray-600);
        position: relative;
        overflow: hidden;
    }

    .stTabs [data-baseweb="tab"]:hover {
        background: var(--gray-200);
        color: var(--primary-color);
    }

    .stTabs [aria-selected="true"] {
        background: var(--primary-color);
        color: white;
        box-shadow: var(--shadow-sm);
    }

    /* Slider styling - Clean & Modern */
    .stSlider > div > div > div > div {
        background: var(--primary-color);
    }

    /* File uploader styling - Clean & Modern */
    .stFileUploader > div > div > div {
        border: 2px dashed var(--gray-300);
        border-radius: var(--border-radius-lg);
        padding: 2rem;
        text-align: center;
        transition: all 0.2s ease;
        background: var(--gray-50);
    }

    .stFileUploader > div > div > div:hover {
        border-color: var(--primary-color);
        background: rgba(37, 99, 235, 0.05);
    }

    /* Responsive design - Clean & Modern */
    @media (max-width: 768px) {
        .main .block-container {
            padding: 1rem 0.5rem;
        }

        .main-header {
            padding: 2rem 1rem;
        }

        .main-header h1 {
            font-size: 2rem;
        }

        .main-header p {
            font-size: 1rem;
        }

        .hero-stats {
            gap: 1rem;
        }
        .hero-stat {
            min-width: 100px;
            padding: 1rem 0.75rem;
        }

        .hero-stat .number {
            font-size: 1.5rem;
        }

        .hero-stat .label {
            font-size: 0.75rem;
        }
    }

    @media (max-width: 480px) {
        .main-header h1 {
            font-size: 1.75rem;
        }

        .hero-stats {
            flex-direction: column;
            align-items: center;
        }

        .hero-stat {
            width: 100%;
            max-width: 200px;
        }
    }
    </style>
    """, unsafe_allow_html=True)

# Main Streamlit app
def main():
    st.set_page_config(
        page_title="🔍 Multimodal Search Engine",
        page_icon="🔍",
        layout="wide",
        initial_sidebar_state="expanded"
    )

    # Clear cache if needed (for debugging)
    if st.sidebar.button("🗑️ Clear Cache"):
        st.cache_data.clear()
        st.cache_resource.clear()
        st.rerun()

    # Load custom CSS
    load_css()

    # Modern header with hero stats
    st.markdown("""
    <div class="main-header">
        <h1>🔍 Multimodal Search Engine</h1>
        <p>Powered by OpenAI CLIP • Find images with text or text with images</p
        <div class="hero-stats">
            <div class="hero-stat">
                <span class="number">""" + str(model_info.get('num_images', 100)
                <span class="label">Images</span>
            </div>
            <div class="hero-stat">
                <span class="number">""" + str(model_info.get('embedding_dim', 5
                <span class="label">Embeddings</span>
            </div>
            <div class="hero-stat">
                <span class="number">""" + str(model_info.get('dataset', 'Flickr
                <span class="label">Dataset</span>
            </div>
        </div>
    </div>
    """, unsafe_allow_html=True)

    # Modern sidebar
    st.sidebar.markdown("### ⚙️ Search Configuration")

    # Search type selection with modern styling
    search_type = st.sidebar.selectbox(
        "🔍 Search Type",
        ["Text-to-Image Search", "Image-to-Text Search"],
        help="Choose how you want to search"
    )

    # Number of results with modern slider
    st.sidebar.markdown("### 📊 Results")
    top_k = st.sidebar.slider(
        "Number of results",
        min_value=1,
        max_value=20,
        value=5,
        help="Number of top results to display"
    )

    # Popular searches with modern grid
    st.sidebar.markdown("### 🔥 Popular Searches")
    st.sidebar.markdown("*Click any suggestion to search instantly*")

    popular_searches = [
        "dog playing", "children smiling", "red car", "food cooking",
        "person running", "cat sleeping", "blue sky", "water beach",
        "house building", "tree nature", "person walking", "animal pet"
    ]

    # Create a grid of popular search buttons
    cols = st.sidebar.columns(2)
    for i, search in enumerate(popular_searches):
        with cols[i % 2]:
            if st.button(f"🔍 {search}", key=f"popular_{i}", help=f"Search for
                st.session_state.popular_search = search
                st.session_state.auto_search = True

    # Dataset information with modern cards
    st.sidebar.markdown("### 📊 Dataset Information")

    # Get values and format properly
    num_images = model_info.get('num_images', 100)
    num_embeddings = model_info.get('total_embeddings', model_info.get('num_samp
    embedding_dim = model_info.get('embedding_dim', 512)
    model_name = model_info.get('model_name', 'CLIP Model')
    dataset = model_info.get('dataset', 'Flickr8k')
    processing_date = model_info.get('processing_date', datetime.now().strftime(

    # Format numbers properly
    images_text = f"{num_images:,}" if isinstance(num_images, int) else str(num_
    embeddings_text = f"{num_embeddings:,}" if isinstance(num_embeddings, int) e
    model_display = model_name.split('/')[-1] if '/' in model_name else model_na

    # Display metrics in modern cards
    st.sidebar.markdown(f"""
    <div class="metric-card">
        <h3>📷 Total Images</h3>
        <div class="value">{images_text}</div>
    </div>
    """, unsafe_allow_html=True)

    st.sidebar.markdown(f"""
    <div class="metric-card">
        <h3>🔢 Total Embeddings</h3>
        <div class="value">{embeddings_text}</div>
    </div>
    """, unsafe_allow_html=True)

    st.sidebar.markdown(f"""
    <div class="metric-card">
        <h3>📐 Embedding Dimension</h3>
        <div class="value">{embedding_dim}D</div>
    </div>
    """, unsafe_allow_html=True)

    st.sidebar.markdown(f"""
    <div class="metric-card">
        <h3>🤖 Model</h3>
        <div class="value">{model_display}</div>
    </div>
    """, unsafe_allow_html=True)

    st.sidebar.markdown(f"""
    <div class="metric-card">
        <h3>📁 Dataset</h3>
        <div class="value">{dataset}</div>
    </div>
    """, unsafe_allow_html=True)

    st.sidebar.markdown(f"""
    <div class="metric-card">
        <h3>📅 Processing Date</h3>
        <div class="value">{processing_date}</div>
    </div>
    """, unsafe_allow_html=True)

    # Check if this is a demo dataset
    num_images = model_info.get('num_images', len(metadata))
    if isinstance(num_images, int) and num_images < 1000:
        st.warning(f"⚠️ **Demo Mode**: You're using a small subset ({num_images

    # Main content area with modern tabs
    tab1, tab2 = st.tabs(["🔤 Text-to-Image Search", "🖼️ Image-to-Text Search"]

    # Add clarification about the tabs
    st.info("💡 **Tip**: Use the **Text-to-Image** tab to search for images usi
    with tab1:
        st.markdown("""
        <div class="search-card">
            <h2 style="margin: 0 0 1rem 0; color: var(--dark-color); font-size:
            <p style="margin: 0 0 2rem 0; color: var(--gray-600); font-size: 1re
        </div>
        """, unsafe_allow_html=True)

        # Search suggestions with modern cards
        col1, col2 = st.columns([1, 1])

        with col1:
            st.markdown("""
            <div style="background: var(--gray-50); padding: 1.5rem; border-radi
                <h4 style="margin: 0 0 1rem 0; color: var(--dark-color); font-si
                <div style="color: var(--gray-600); line-height: 1.6; font-size:
                    <strong>Try searching for:</strong><br>
                    • <strong>Animals:</strong> 'dog', 'cat', 'bird', 'horse'<br
                    • <strong>Activities:</strong> 'playing', 'running', 'cookin
                    • <strong>Objects:</strong> 'car', 'house', 'food'<br>
                    • <strong>Emotions:</strong> 'smiling', 'happy', 'sad'<br>
                    • <strong>Scenes:</strong> 'beach', 'park', 'kitchen'
                </div>
            </div>
            """, unsafe_allow_html=True)

        with col2:
            st.markdown("""
            <div style="background: var(--gray-50); padding: 1.5rem; border-radi
                <h4 style="margin: 0 0 1rem 0; color: var(--dark-color); font-si
                <div style="color: var(--gray-600); line-height: 1.6; font-size:
                    Click any example to search instantly:
                </div>
            </div>
            """, unsafe_allow_html=True)

            # Example buttons in a grid
            example_cols = st.columns(2)
            with example_cols[0]:
                if st.button("🐕 A dog playing", key="example1", help="Search fo
                    st.session_state.example_query = "a dog playing"
                    st.session_state.auto_search = True
                if st.button("👶 Children smiling", key="example2", help="Search
                    st.session_state.example_query = "children smiling"
                    st.session_state.auto_search = True

            with example_cols[1]:
                if st.button("🚗 Red car", key="example3", help="Search for 'rec
                    st.session_state.example_query = "red car"
                    st.session_state.auto_search = True
                if st.button("🍕 Food cooking", key="example4", help="Search for
                    st.session_state.example_query = "food cooking"
                    st.session_state.auto_search = True

        # Text input with better placeholder
        query_text = st.text_input(
            "🔍 Enter your search query:",
            placeholder="Describe what you're looking for... (e.g., 'a dog playi
            help="💡 Be specific! Try describing objects, actions, colors, or em
            value=st.session_state.get('example_query', st.session_state.get('po
            key="search_input"
        )

        # Clear example queries after use
        if 'example_query' in st.session_state:
            del st.session_state.example_query
        if 'popular_search' in st.session_state:
            del st.session_state.popular_search

        # Check if we should auto-search (from popular searches or example queri
        should_search = st.session_state.get('auto_search', False)
        if should_search:
            st.session_state.auto_search = False  # Reset the flag
            # Use example query if available, otherwise use popular search
            query_text = st.session_state.get('example_query', st.session_state.

        if st.button("🔍 Search Images", type="primary") or should_search:
            if query_text:
                with st.spinner("Searching for images..."):
                    results = text_to_image_search(query_text, top_k)

                if results:
                    st.success(f"Found {len(results)} results for: '{query_text}

                    # Display results in columns
                    cols = st.columns(min(3, len(results)))
                    for i, result in enumerate(results):
                        with cols[i % 3]:
                            try:
                                image_path = result['image_path']
                                # Fix path - remove ../ if present
                                if image_path.startswith('../'):
                                    image_path = image_path[3:]  # Remove ../

                                if os.path.exists(image_path):
                                    image = Image.open(image_path)
                                    st.image(image, caption=f"Similarity: {resul

                                    # Display details
                                    st.markdown(f"**Image ID:** {result['image_i
                                    st.markdown(f"**Caption:** {result['caption'
                                    st.markdown(f"**Similarity:** {result['simil
                                else:
                                    st.error(f"Image not found: {image_path}")
                            except Exception as e:
                                st.error(f"Error loading image: {e}")
                else:
                    st.warning("No results found. Try a different search que
            else:
                st.warning("Please enter a search query.")

    with tab2:
        st.markdown("""
        <div class="search-card">
            <h2 style="margin: 0 0 1rem 0; color: var(--dark-color); font-size:
            <p style="margin: 0 0 2rem 0; color: var(--gray-600); font-size: 1re
        </div>
        """, unsafe_allow_html=True)
        # Upload guidance
        st.markdown("#### 📋 Upload Guidelines")
        col1, col2 = st.columns(2)

        with col1:
            st.markdown("""
            <div style="background: var(--gray-50); padding: 1.25rem; border-rad
                <h4 style="margin: 0 0 0.75rem 0; color: var(--dark-color); font
                <div style="color: var(--gray-600); font-size: 0.875rem; line-he
                    • JPG, JPEG<br>
                    • PNG<br>
                    • BMP, GIF
                </div>
            </div>
            """, unsafe_allow_html=True)

        with col2:
            st.markdown("""
            <div style="background: var(--gray-50); padding: 1.25rem; border-rad
                <h4 style="margin: 0 0 0.75rem 0; color: var(--dark-color); font
                <div style="color: var(--gray-600); font-size: 0.875rem; line-he
                    • Clear, well-lit images<br>
                    • Single main subject<br>
                    • Good contrast
                </div>
            </div>
            """, unsafe_allow_html=True)

        # Image upload
        uploaded_file = st.file_uploader(
            "📁 Choose an image file:",
            type=['jpg', 'jpeg', 'png', 'bmp', 'gif'],
            help="💡 Upload a clear image with a main subject for best search re
            label_visibility="collapsed"
        )

        # Add some guidance
        if not uploaded_file:
            st.info("👆 **Upload an image above** to find similar text descript

        if uploaded_file is not None:
            try:
                # Debug information
                st.write(f"📁 File name: {uploaded_file.name}")
                st.write(f"📏 File size: {uploaded_file.size} bytes")
                st.write(f"🔍 File type: {uploaded_file.type}")

                # Reset file pointer to beginning
                uploaded_file.seek(0)

                # Try using BytesIO with proper handling
                file_bytes = uploaded_file.read()
                st.write(f"📊 File bytes length: {len(file_bytes)}")

                # Check if file has content
                if len(file_bytes) == 0:
                    st.error("❌ File is empty!")
                    return

                # Try to create image from bytes using a more robust approach
                try:
                    # Create BytesIO object
                    image_io = io.BytesIO(file_bytes)
                    image_io.seek(0)

                    # Try to determine format from file extension
                    file_extension = os.path.splitext(uploaded_file.name)[1].low
                    st.write(f"🔍 Detected file extension: {file_extension}")

                    # Try to open with PIL - let it auto-detect the format
                    uploaded_image = Image.open(image_io)

                    # Load the image data
                    uploaded_image.load()

                    # Convert to RGB if necessary
                    if uploaded_image.mode != 'RGB':
                        uploaded_image = uploaded_image.convert('RGB')

                    st.success("✅ Image loaded successfully!")

                except Exception as img_error:
                    st.error(f"❌ Error loading image: {str(img_error)}")

                    # Try alternative approach - save to temporary file
                    st.write("🔄 Trying temporary file approach...")

                    try:
                        import tempfile

                        # Create temporary file with proper extension
                        with tempfile.NamedTemporaryFile(delete=False, suffix=fi
                            tmp_file.write(file_bytes)
                            tmp_file_path = tmp_file.name

                        st.write(f"📁 Created temp file: {tmp_file_path}")

                        # Load from temporary file
                        uploaded_image = Image.open(tmp_file_path)
                        uploaded_image.load()

                        # Convert to RGB if necessary
                        if uploaded_image.mode != 'RGB':
                            uploaded_image = uploaded_image.convert('RGB')

                        # Clean up temporary file
                        os.unlink(tmp_file_path)

                        st.success("✅ Image loaded with temporary file method!'

                    except Exception as temp_error:
                        st.error(f"❌ Temporary file method failed: {str(temp_e

                        # Final fallback - try with cv2 if available
                        st.write("🔄 Trying OpenCV fallback...")
                        try:
                            import cv2
                            import numpy as np

                            # Convert bytes to numpy array
                            nparr = np.frombuffer(file_bytes, np.uint8)

                            # Decode image with OpenCV
                            cv_image = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

                            if cv_image is not None:
                                # Convert BGR to RGB
                                cv_image = cv2.cvtColor(cv_image, cv2.COLOR_BGR2

                                # Convert to PIL Image
                                uploaded_image = Image.fromarray(cv_image)

                                st.success("✅ Image loaded with OpenCV fallbacl
                            else:
                                raise Exception("OpenCV could not decode the ima

                        except ImportError:
                            st.error("❌ OpenCV not available for fallback")
                            st.warning("The uploaded file might be corrupted or
                            return
                        except Exception as cv_error:
                            st.error(f"❌ OpenCV fallback failed: {str(cv_error)
                            st.warning("The uploaded file might be corrupted or
                            return

                # Display the image
                st.image(uploaded_image, caption="Uploaded Image", use_container

                if st.button("🔍 Search Descriptions", type="primary"):
                    with st.spinner("Searching for similar descriptions..."):
                        try:
                            # Use the image we already loaded
                            results = image_to_text_search(uploaded_image, top_k
                        except Exception as search_error:
                            st.error(f"❌ Error during search: {str(search_erro
                            results = []

                    if results:
                        st.success(f"Found {len(results)} similar descriptions")

                        # Display results in columns
                        cols = st.columns(min(3, len(results)))
                        for i, result in enumerate(results):
                            with cols[i % 3]:
                                try:
                                    image_path = result['image_path']
                                    # Fix path - remove ../ if present
                                    if image_path.startswith('../'):
                                        image_path = image_path[3:]  # Remove ..

                                    if os.path.exists(image_path):
                                        original_image = Image.open(image_path)
                                        st.image(original_image, caption="Origin
                                    else:
                                        st.error(f"Original image not found: {im
                                except Exception as e:
                                    st.error(f"Error loading original image: {e}

                                # Display details
                                st.markdown(f"**Image ID:** {result['image_id']}
                                st.markdown(f"**Caption:** {result['caption']}")
                                st.markdown(f"**Similarity:** {result['similarit
                    else:
                        st.warning("No results found. Try a different image.")

            except Exception as e:
                st.error(f"❌ Error loading image: {str(e)}")
                st.warning("Please make sure you're uploading a valid image file
                uploaded_file = None

if __name__ == "__main__":
    main()


'''

# Write Streamlit app to file
with open('../streamlit_app.py', 'w', encoding='utf-8') as f:
    f.write(streamlit_code)

print("✅ Streamlit app created successfully!")
print("📁 File saved as: ../streamlit_app.py")
print("🚀 To run: streamlit run ../streamlit_app.py")
```

```
✅ Streamlit app created successfully!
📁 File saved as: ../streamlit_app.py
🚀 To run: streamlit run ../streamlit_app.py
```

6. Create Standalone Gradio App

In [6]:

```
# Create the complete Gradio app code
gradio_code = '''
import gradio as gr
import torch
import torchvision.transforms as transforms
from transformers import CLIPProcessor, CLIPModel
from PIL import Image
import numpy as np
import pandas as pd
import os
import json
from sklearn.metrics.pairwise import cosine_similarity
import warnings
warnings.filterwarnings('ignore')

# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Load CLIP model
def load_clip_model():
    """Load CLIP model and processor"""
    model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32").to(device)
    processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")
    return model, processor

# Load embeddings data
def load_embeddings_data():
    """Load pre-computed embeddings and metadata"""
    # Load embeddings
    image_embeddings = np.load('embeddings/image_embeddings.npy')
    text_embeddings = np.load('embeddings/text_embeddings.npy')

    # Load metadata
    metadata = pd.read_csv('embeddings/metadata.csv')

    # Load model info
    with open('embeddings/model_info.json', 'r') as f:
        model_info = json.load(f)

    return image_embeddings, text_embeddings, metadata, model_info

# Load model and data
model, processor = load_clip_model()
image_embeddings, text_embeddings, metadata, model_info = load_embeddings_data()

# Text-to-Image Search Function
def text_to_image_search(query_text, top_k=5):
    """Search for images based on text query"""
    # Generate embedding for text query
    inputs = processor(text=[query_text], return_tensors="pt", padding=True).to(

    with torch.no_grad():
        query_embedding = model.get_text_features(**inputs)
        query_embedding = query_embedding / query_embedding.norm(dim=-1, keepdim

    # Calculate similarities with all image embeddings
    similarities = cosine_similarity(query_embedding.cpu().numpy(), image_embedd

    # Get top-k most similar images
    top_indices = np.argsort(similarities)[::-1][:top_k]

    results = []
    for idx in top_indices:
        result = {
            'image_id': metadata.iloc[idx]['image_id'],
            'image_path': metadata.iloc[idx]['image_path'],
            'caption': metadata.iloc[idx]['caption'],
            'similarity': similarities[idx]
        }
        results.append(result)

    return results

# Image-to-Text Search Function
def image_to_text_search(uploaded_image, top_k=5):
    """Search for text descriptions based on uploaded image"""
    # Generate embedding for uploaded image
    inputs = processor(images=uploaded_image, return_tensors="pt").to(device)

    with torch.no_grad():
        query_embedding = model.get_image_features(**inputs)
        query_embedding = query_embedding / query_embedding.norm(dim=-1, keepdim

    # Calculate similarities with all text embeddings
    similarities = cosine_similarity(query_embedding.cpu().numpy(), text_embeddi

    # Get top-k most similar text descriptions
    top_indices = np.argsort(similarities)[::-1][:top_k]
    results = []
    for idx in top_indices:
        result = {
            'image_id': metadata.iloc[idx]['image_id'],
            'image_path': metadata.iloc[idx]['image_path'],
            'caption': metadata.iloc[idx]['caption'],
            'similarity': similarities[idx]
        }
        results.append(result)

    return results

# Text-to-Image Search Interface
def search_images(query, num_results):
    """Gradio interface for text-to-image search"""
    if not query.strip():
        return [], "Please enter a search query."

    try:
        results = text_to_image_search(query, num_results)

        if not results:
            return [], "No results found. Try a different search query."

        # Prepare gallery items as tuples (image_path, caption)
        gallery_items = []

        for result in results:
            image_path = result['image_path']
            # Fix path - remove ../ if present
            if image_path.startswith('../'):
                image_path = image_path[3:]  # Remove ../

            if os.path.exists(image_path):
                caption = f"🎯 Similarity: {result['similarity']:.3f}\n📝 {resu
                gallery_items.append((image_path, caption))
            else:
                # For missing images, we can't add them to the gallery
                pass

        return gallery_items, f"Found {len(gallery_items)} results for: '{query}

    except Exception as e:
        return [], f"Error during search: {str(e)}"

# Image-to-Text Search Interface
def search_descriptions(image, num_results):
    """Gradio interface for image-to-text search"""
    if image is None:
        return [], "Please upload an image."

    try:
        results = image_to_text_search(image, num_results)

        if not results:
            return [], "No results found. Try a different image."

        # Prepare gallery items as tuples (image_path, caption)
        gallery_items = []
        for result in results:
            image_path = result['image_path']
            # Fix path - remove ../ if present
            if image_path.startswith('../'):
                image_path = image_path[3:]  # Remove ../

            if os.path.exists(image_path):
                caption = f"🎯 Similarity: {result['similarity']:.3f}\n📝 {resu
                gallery_items.append((image_path, caption))
            else:
                # For missing images, we can't add them to the gallery
                pass

        return gallery_items, f"Found {len(gallery_items)} similar descriptions"

    except Exception as e:
        return [], f"Error during search: {str(e)}"

# Create Gradio interface
def create_gradio_app():
    """Create the Gradio web application"""

    # Project description
    description = """
    # 🔍 Multimodal Search Engine

    A powerful search engine that can find images using text descriptions and fi

    **Technology Stack:**
    - **Model**: OpenAI CLIP (Contrastive Language-Image Pre-training)
    - **Framework**: Gradio for web interface
    - **Dataset**: Flickr8k (8,091 images with captions)
    - **Embeddings**: 512-dimensional vector representations
    - **Similarity**: Cosine similarity for matching

    **Features:**
    - Text-to-Image Search: Describe what you're looking for
    - Image-to-Text Search: Upload an image to find similar descriptions
    - Real-time similarity scoring
    - Interactive web interface
    """

    # Popular search suggestions
    popular_searches = [
        "dog playing", "children smiling", "red car", "food cooking",
        "person running", "cat sleeping", "blue sky", "water beach"
    ]

    with gr.Blocks(
        title="🔍 Multimodal Search Engine",
        theme=gr.themes.Soft(),
        css="""
        .gradio-container {
            max-width: 1400px !important;
            margin: 0 auto !important;
            font-family: 'Inter', -apple-system, BlinkMacSystemFont, sans-serif
            background: linear-gradient(135deg, #f8fafc 0%, #e2e8f0 100%) !impor
            min-height: 100vh !important;
        }
        /* Modern header styling */
        .gradio-container h1 {
            background: linear-gradient(135deg, #6366f1 0%, #8b5cf6 50%, #ec4899
            -webkit-background-clip: text;
            -webkit-text-fill-color: transparent;
            background-clip: text;
            font-size: 3rem !important;
            font-weight: 800 !important;
            text-align: center !important;
            margin: 2rem 0 !important;
            text-shadow: 0 2px 4px rgba(0, 0, 0, 0.1) !important;
        }

        /* Card styling */
        .card {
            background: white !important;
            border-radius: 20px !important;
            padding: 2rem !important;
            box-shadow: 0 10px 25px -5px rgba(0, 0, 0, 0.1), 0 4px 6px -2px rgba
            border: 1px solid rgba(255, 255, 255, 0.2) !important;
            backdrop-filter: blur(10px) !important;
            margin: 1rem 0 !important;
        }

        /* Button styling */
        .btn {
            border-radius: 16px !important;
            font-weight: 700 !important;
            transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1) !important;
            box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1), 0 2px 4px -1px rgba(0
            border: none !important;
            padding: 0.75rem 1.5rem !important;
            font-size: 0.875rem !important;
            text-transform: uppercase !important;
            letter-spacing: 0.05em !important;
        }

        .btn:hover {
            transform: translateY(-2px) !important;
            box-shadow: 0 20px 25px -5px rgba(0, 0, 0, 0.1), 0 10px 10px -5px rg
        }

        .btn-primary {
            background: linear-gradient(135deg, #6366f1 0%, #8b5cf6 100%) !impor
            color: white !important;
        }

        /* Input styling */
        .input {
            border-radius: 16px !important;
            border: 2px solid #e2e8f0 !important;
            padding: 1rem 1.25rem !important;
            transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1) !important;
            background: rgba(255, 255, 255, 0.8) !important;
            backdrop-filter: blur(10px) !important;
            font-size: 1rem !important;
        }

        .input:focus {
            border-color: #6366f1 !important;
            box-shadow: 0 0 0 4px rgba(99, 102, 241, 0.1) !important;
            background: white !important;
        }

        /* Gallery styling */
        .gallery {
            border-radius: 20px !important;
            overflow: hidden !important;
            box-shadow: 0 20px 25px -5px rgba(0, 0, 0, 0.1), 0 10px 10px -5px rg
            background: white !important;
            padding: 1rem !important;
        }

        .gallery img {
            border-radius: 16px !important;
            transition: transform 0.3s ease !important;
        }

        .gallery img:hover {
            transform: scale(1.02) !important;
        }

        /* Tab styling */
        .tab-nav {
            background: rgba(255, 255, 255, 0.8) !important;
            backdrop-filter: blur(10px) !important;
            border-radius: 20px !important;
            padding: 0.5rem !important;
            margin: 2rem 0 !important;
            box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1) !important;
        }

        .tab-nav button {
            border-radius: 12px !important;
            font-weight: 700 !important;
            transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1) !important;
            padding: 0.75rem 1.5rem !important;
            margin: 0.25rem !important;
        }

        .tab-nav button.selected {
            background: linear-gradient(135deg, #6366f1 0%, #8b5cf6 100%) !impor
            color: white !important;
            box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1) !important;
        }

        /* Status messages */
        .status {
            background: linear-gradient(135deg, #10b981 0%, #059669 100%) !impor
            color: white !important;
            padding: 1rem 1.5rem !important;
            border-radius: 12px !important;
            font-weight: 600 !important;
            box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1) !important;
        }

        /* Dataset info cards */
        .dataset-card {
            background: linear-gradient(135deg, #f8fafc 0%, #e2e8f0 100%) !impor
            border-radius: 16px !important;
            padding: 1.5rem !important;
            margin: 0.5rem 0 !important;
            border: 1px solid rgba(255, 255, 255, 0.2) !important;
            box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1) !important;
        }

        /* Responsive design */
        @media (max-width: 768px) {
            .gradio-container {
                padding: 1rem !important;
            }

            .gradio-container h1 {
                font-size: 2.5rem !important;
            }

            .card {
                padding: 1.5rem !important;
                margin: 0.5rem 0 !important;
            }
        }
        """
    ) as app:
        gr.Markdown(description)

        # Dataset information
        with gr.Row():
            with gr.Column(scale=1):
                # Get values and format properly
                num_images = model_info.get('num_images', 'Unknown')
                num_embeddings = model_info.get('total_embeddings', model_info.g
                embedding_dim = model_info.get('embedding_dim', 'Unknown')
                model_name = model_info.get('model_name', 'Unknown')
                dataset = model_info.get('dataset', 'Unknown')
                processing_date = model_info.get('processing_date', 'Unknown')

                # Format numbers properly
                images_text = f"{num_images:,}" if isinstance(num_images, int) e
                embeddings_text = f"{num_embeddings:,}" if isinstance(num_embedd
                model_display = model_name.split('/')[-1] if '/' in model_name e

                gr.Markdown(f"""
                <div class="dataset-card">
                <h3 style="margin: 0 0 1rem 0; color: #374151; font-size: 1.25re
                <div style="display: grid; grid-template-columns: 1fr 1fr; gap: 
                    <div><strong>📷 Images:</strong> {images_text}</div>
                    <div><strong>🔢 Embeddings:</strong> {embeddings_text}</div
                    <div><strong>📐 Dimension:</strong> {embedding_dim}D</div>
                    <div><strong>🤖 Model:</strong> {model_display}</div>
                    <div><strong>📁 Dataset:</strong> {dataset}</div>
                    <div><strong>📅 Date:</strong> {processing_date}</div>
                </div>
                </div>
                """)

            with gr.Column(scale=1):
                gr.Markdown("""
                <div class="dataset-card">
                <h3 style="margin: 0 0 1rem 0; color: #374151; font-size: 1.25re
                <p style="margin: 0 0 1rem 0; color: #6b7280; font-size: 0.875re
                """)

                # Create a simple list of popular searches
                popular_text = '<div style="display: flex; flex-wrap: wrap; gap:
                for search in popular_searches:
                    popular_text += f'<span style="background: linear-gradient(1
                popular_text += '</div></div>'

                gr.Markdown(popular_text)

        # Main search interface
        with gr.Tabs():
            # Text-to-Image Search Tab
            with gr.Tab("🔤 Text-to-Image Search"):
                gr.Markdown("""
                <div class="card">
                <h2 style="margin: 0 0 1rem 0; color: #374151; font-size: 1.5rem
                <p style="margin: 0 0 2rem 0; color: #6b7280; font-size: 1rem;">
                </div>
                """)

                with gr.Row():
                    with gr.Column(scale=3):
                        text_query = gr.Textbox(
                            label="🔍 Search Query",
                            placeholder="e.g., 'a dog playing in the park' or 'c
                            info="💡 Be specific! Try describing objects, actio
                            elem_classes=["input"]
                        )
                        num_results_text = gr.Slider(
                            label="📊 Number of Results",
                            minimum=1,
                            maximum=20,
                            value=5,
                            step=1,
                            info="Choose how many results to display"
                        )
                        search_btn = gr.Button("🔍 Search Images", variant="pri

                    with gr.Column(scale=1):
                        gr.Markdown("""
                        <div class="dataset-card">
                        <h3 style="margin: 0 0 1rem 0; color: #374151; font-size
                        <p style="margin: 0 0 1rem 0; color: #6b7280; font-size:
                        """)

                        # Create clickable search suggestion buttons
                        with gr.Row():
                            with gr.Column():
                                for i, search in enumerate(popular_searches[:4])
                                    btn = gr.Button(
                                        f"🔍 {search}",
                                        size="sm",
                                        variant="secondary",
                                        elem_classes=["btn"]
                                    )
                                    btn.click(
                                        lambda s=search: s,
                                        outputs=text_query
                                    )
                        with gr.Row():
                            with gr.Column():
                                for i, search in enumerate(popular_searches[4:])
                                    btn = gr.Button(
                                        f"🔍 {search}",
                                        size="sm",
                                        variant="secondary",
                                        elem_classes=["btn"]
                                    )
                                    btn.click(
                                        lambda s=search: s,
                                        outputs=text_query
                                    )

                        gr.Markdown("""
                        <div class="dataset-card" style="margin-top: 1rem;">
                        <h3 style="margin: 0 0 1rem 0; color: #374151; font-size
                        <div style="color: #6b7280; font-size: 0.875rem; line-he
                        <strong>Try searching for:</strong><br>
                        • <strong>Animals:</strong> 'dog', 'cat', 'bird', 'horse
                        • <strong>Activities:</strong> 'playing', 'running', 'co
                        • <strong>Objects:</strong> 'car', 'house', 'food'<br>
                        • <strong>Emotions:</strong> 'smiling', 'happy', 'sad'<b
                        • <strong>Scenes:</strong> 'beach', 'park', 'kitchen'
                        </div>
                        </div>
                        """)

                # Results
                text_results = gr.Gallery(
                    label="🎨 Search Results",
                    show_label=True,
                    elem_id="gallery",
                    columns=3,
                    rows=2,
                    object_fit="contain",
                    height="auto",
                    elem_classes=["gallery"]
                )
                text_status = gr.Textbox(
                    label="📊 Status",
                    interactive=False,
                    elem_classes=["status"]
                )

                # Connect search button
                search_btn.click(
                    search_images,
                    inputs=[text_query, num_results_text],
                    outputs=[text_results, text_status]
                )

            # Image-to-Text Search Tab
            with gr.Tab("🖼️ Image-to-Text Search"):
                gr.Markdown("""
                <div class="card">
                <h2 style="margin: 0 0 1rem 0; color: #374151; font-size: 1.5rem
                <p style="margin: 0 0 2rem 0; color: #6b7280; font-size: 1rem;">
                </div>
```

```
                """)

                with gr.Row():
                    with gr.Column(scale=3):
                        image_input = gr.Image(
                            label="📁 Upload Image",
                            type="pil",
                            info="💡 Upload a clear image with a main subject fo
                            elem_classes=["input"]
                        )
                        num_results_image = gr.Slider(
                            label="📊 Number of Results",
                            minimum=1,
                            maximum=20,
                            value=5,
                            step=1,
                            info="Choose how many results to display"
                        )
                        search_img_btn = gr.Button("🔍 Search Descriptions", var

                    with gr.Column(scale=1):
                        gr.Markdown("""
                        <div class="dataset-card">
                        <h3 style="margin: 0 0 1rem 0; color: #374151; font-size
                        <div style="color: #6b7280; font-size: 0.875rem; line-he
                        <strong>Supported formats:</strong><br>
                        • JPG, JPEG<br>
                        • PNG<br>
                        • BMP, GIF<br><br>

                        <strong>Best results with:</strong><br>
                        • Clear, well-lit images<br>
                        • Single main subject<br>
                        • Good contrast
                        </div>
                        </div>
                        """)

                # Results
                image_results = gr.Gallery(
                    label="🎨 Search Results",
                    show_label=True,
                    elem_id="gallery",
                    columns=3,
                    rows=2,
                    object_fit="contain",
                    height="auto",
                    elem_classes=["gallery"]
                )
                image_status = gr.Textbox(
                    label="📊 Status",
                    interactive=False,
                    elem_classes=["status"]
                )

                # Connect search button
                search_img_btn.click(
                    search_descriptions,
                    inputs=[image_input, num_results_image],
                    outputs=[image_results, image_status]
                )

        # Footer
        gr.Markdown("""
        ---
        **🔍 Search Engine** - Built with Gradio and OpenAI CLIP
        """)

    return app

# Create and launch the app
if __name__ == "__main__":
    app = create_gradio_app()
    app.launch(
        server_name="0.0.0.0",
        server_port=7860,
        share=False,
        show_error=True
    )

'''

# Write Gradio app to file
with open('../gradio_app.py', 'w', encoding='utf-8') as f:
    f.write(gradio_code)

print("✅ Gradio app created successfully!")
print("📁 File saved as: ../gradio_app.py")
print("🚀 To run: python ../gradio_app.py")
```

```
✅ Gradio app created successfully!
📁 File saved as: ../gradio_app.py
🚀 To run: python ../gradio_app.py
```

7. Framework Comparison and Launch Instructions

In [7]:
```
# Framework Comparison
print("🔍 MULTIMODAL SEARCH ENGINE - FRAMEWORK COMPARISON")
print("=" * 60)

print("\n📊 STREAMLIT vs GRADIO")
print("-" * 30)
print("| Feature                | Streamlit | Gradio |")
print("|------------------------|-----------|--------|")
print("| Learning Curve         | Easier    | Medium |")
print("| UI Style               | Traditional| Modern|")
print("| Layout                 | Column-based| Tab-based|")
print("| Interactions           | Form-based| Event-driven|")
print("| Customization          | High      | Medium |")
print("| Performance            | Good      | Good   |")
print("| Community              | Large     | Growing|")
print("| Documentation          | Excellent | Good   |")
print("| Deployment             | Easy      | Easy   |")

print("\n🚀 LAUNCH INSTRUCTIONS")
print("-" * 30)
print("1. STREAMLIT APP (Port 8501):")
print("   streamlit run streamlit_app.py")
print()
print("2. GRADIO APP (Port 7860):")
print("   python gradio_app.py")
print()
print("3. RUN BOTH SIMULTANEOUSLY:")
print("   - Open two terminal windows")
print("   - Run each command in separate terminal")
print("   - Access Streamlit at: http://localhost:8501")
print("   - Access Gradio at: http://localhost:7860")

print("\n✅ BOTH APPS INCLUDE:")
print("-" * 30)
print("• Text-to-Image Search")
print("• Image-to-Text Search")
print("• Popular search suggestions")
print("• Dataset information display")
print("• Search tips and guidelines")
print("• Real-time similarity scoring")
print("• Responsive image galleries")
print("• Error handling and validation")

print("\n🎯 RECOMMENDATION:")
print("-" * 30)
print("• Use Streamlit for: Traditional web apps, data science projects")
print("• Use Gradio for: AI demos, quick prototypes, modern interfaces")
print("• Both are excellent choices for this project!")
```

```
🔍 MULTIMODAL SEARCH ENGINE - FRAMEWORK COMPARISON
============================================================

📊 STREAMLIT vs GRADIO
------------------------------
| Feature                 | Streamlit | Gradio |
|------------------------|-----------|--------|
| Learning Curve          | Easier    | Medium |
| UI Style                | Traditional| Modern|
| Layout                  | Column-based| Tab-based|
| Interactions            | Form-based| Event-driven|
| Customization           | High      | Medium |
| Performance             | Good      | Good   |
| Community               | Large     | Growing|
| Documentation           | Excellent | Good   |
| Deployment              | Easy      | Easy   |

🚀 LAUNCH INSTRUCTIONS
------------------------------
1. STREAMLIT APP (Port 8501):
   streamlit run streamlit_app.py

2. GRADIO APP (Port 7860):
   python gradio_app.py

3. RUN BOTH SIMULTANEOUSLY:
   - Open two terminal windows
   - Run each command in separate terminal
   - Access Streamlit at: http://localhost:8501
   - Access Gradio at: http://localhost:7860

✅ BOTH APPS INCLUDE:
------------------------------
• Text-to-Image Search
• Image-to-Text Search
• Popular search suggestions
• Dataset information display
• Search tips and guidelines
• Real-time similarity scoring
• Responsive image galleries
• Error handling and validation

🎯 RECOMMENDATION:
------------------------------
• Use Streamlit for: Traditional web apps, data science projects
• Use Gradio for: AI demos, quick prototypes, modern interfaces
• Both are excellent choices for this project!
```