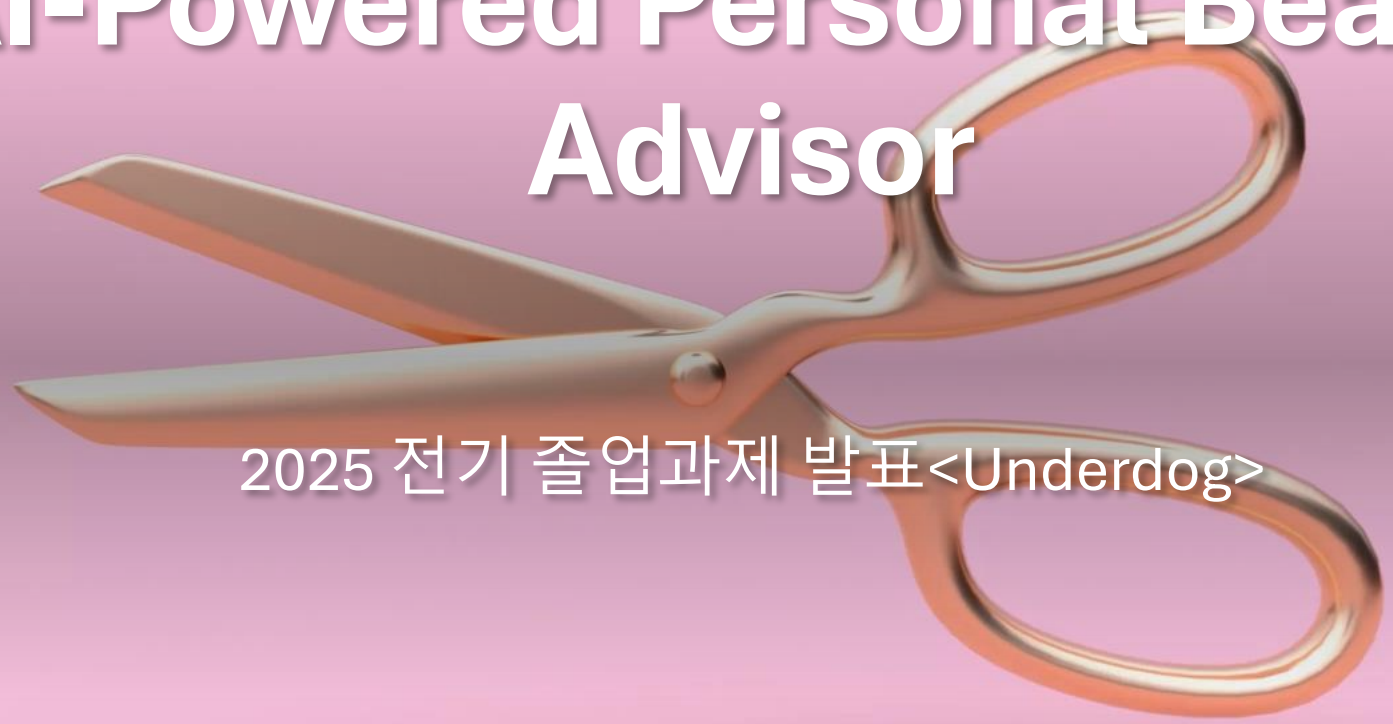
AI-Powered Personal Beauty Advisor



2025 전기 졸업과제 발표<Underdog>

Background

- fashion and beauty are no longer just about appearance
- powerful tools of individuality and identity

PERSONAL COLOUR ANALYSIS IN KOREA

Existing Problem

Traditional Services

- rely on subjective expert judgment
- Expensive

Online Tests

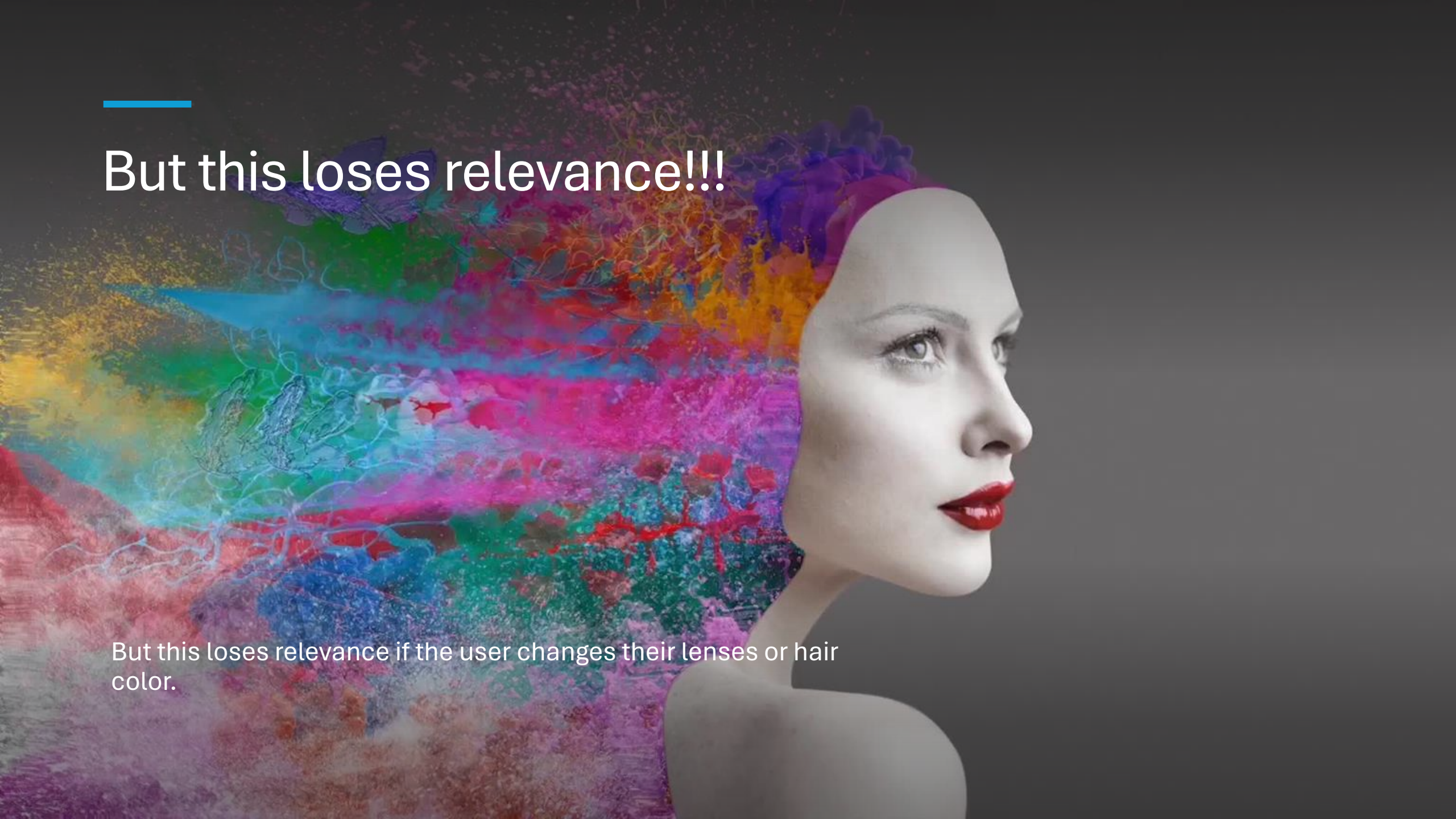
- Not show how recommended colors look on the user.

How do we determine personal color?

Armocromia

- Conventional
- 12 seasonal types
- based on the overall face

But this loses relevance!!!



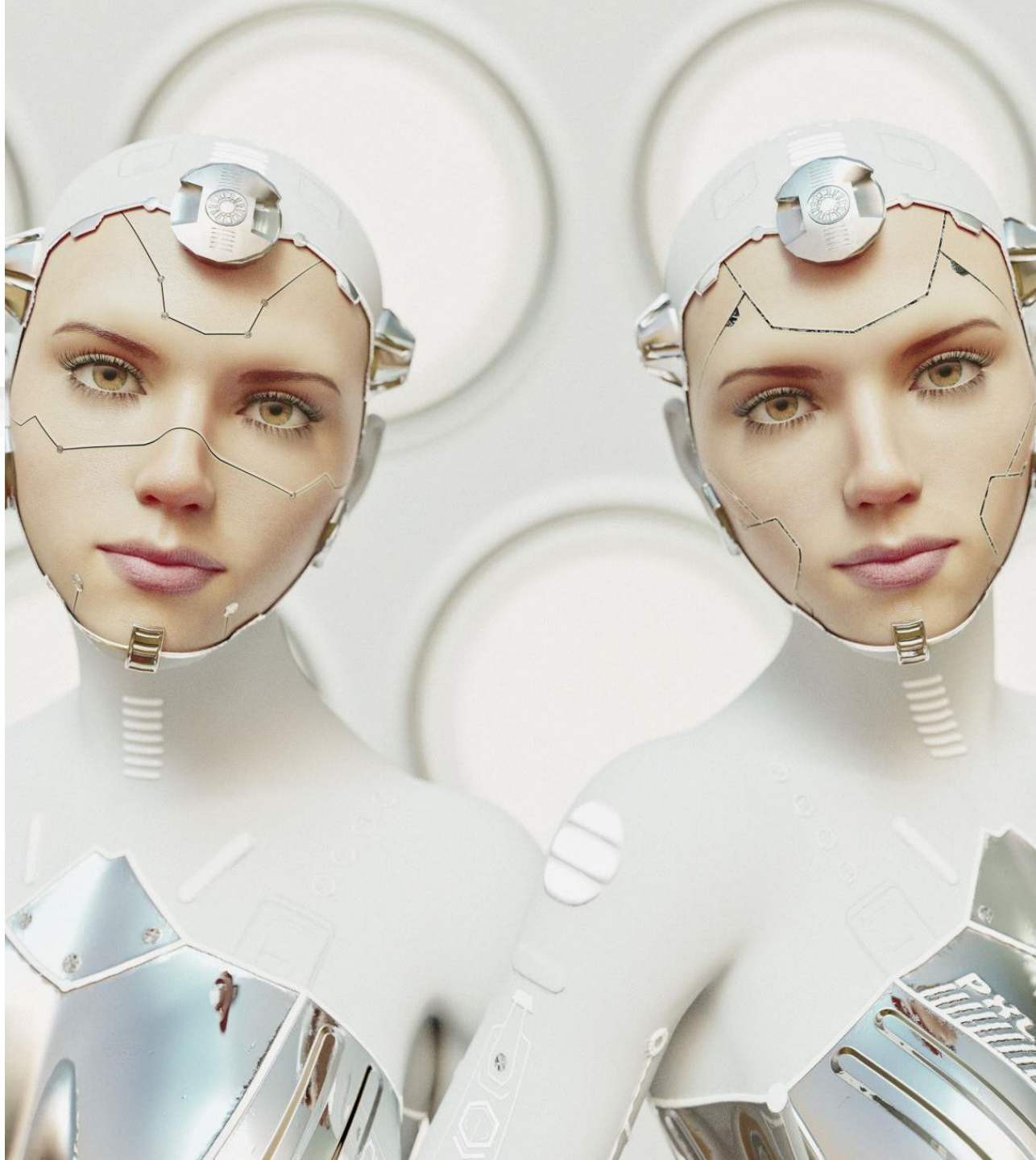
But this loses relevance if the user changes their lenses or hair color.

Solution: New Personal Color System

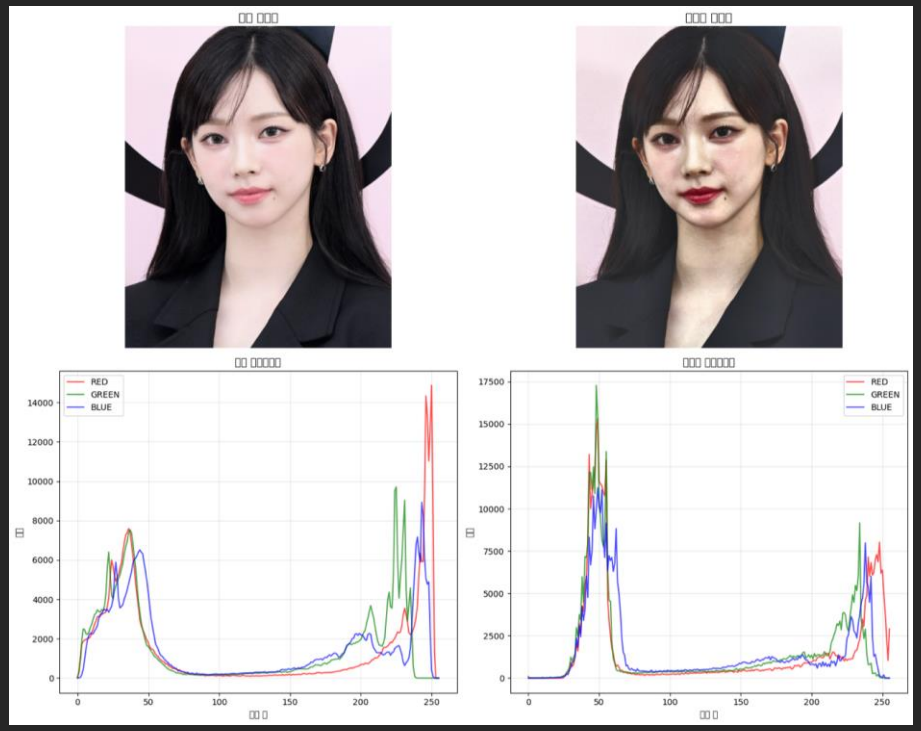
- we created an entirely new personal color system
- based only on skin tone
- clustering using K-Means.
- gives a more stable and practical foundation for recommendations

Technical Part

- Preprocess the images
- Face Detection and Segmentation
- Feature Extraction
- Model Training
- Selection of K Value and Clustering Evaluation



Data Preprocessing



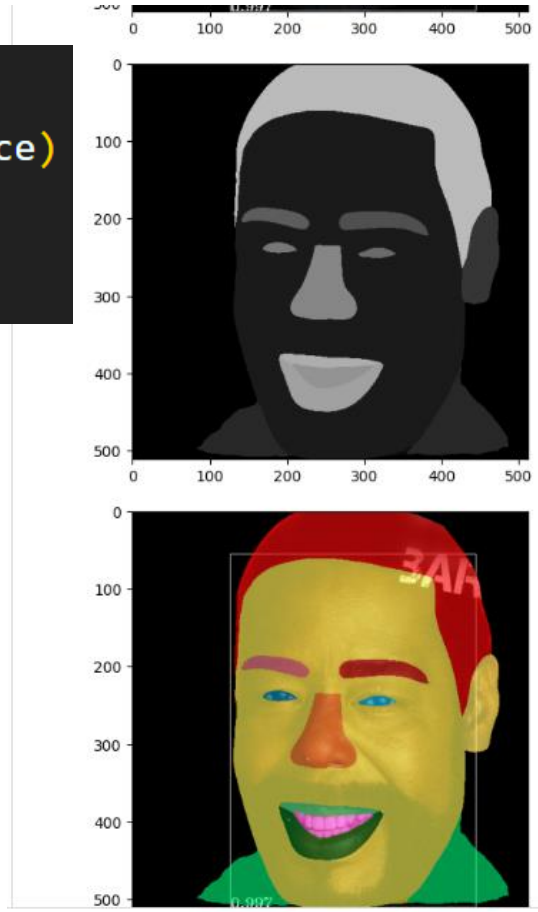
5000 images from ArmocromiaDataset

```python
def analyze_lighting_conditions(image_np):
    """이미지의 조명 상태를 분석하여 보정 전략을 결정"""
    lab = cv2.cvtColor(image_np, cv2.COLOR_RGB2LAB)
    l_channel = lab[:, :, 0]
    mean_brightness = np.mean(l_channel)
    std_brightness = np.std(l_channel)
    dark_pixels = np.sum(l_channel < 85) / l_channel.size
    bright_pixels = np.sum(l_channel > 170) / l_channel.size
    return {
        'mean_brightness': mean_brightness, 'std_brightness': std_brightness,
        'dark_ratio': dark_pixels, 'bright_ratio': bright_pixels,
        'is_underexposed': mean_brightness < 120 and dark_pixels > 0.3,
        'is_overexposed': mean_brightness > 180 and bright_pixels > 0.2,
        'has_low_contrast': std_brightness < 25, 'has_uneven_lighting': std_brightness > 50
    }
```

```python
def comprehensive_lighting_correction(image_np, lighting_info=None):

    if lighting_info['is_underexposed']:
        corrected = shadow_highlight_correction(corrected, shadow_amount=0.3, highlight_amount=-0.1)
        corrected = gamma_correction(corrected, 0.7)
        correction_log.append("Underexposure correction: shadow lift + gamma 0.7")
    elif lighting_info['is_overexposed']:
        corrected = shadow_highlight_correction(corrected, shadow_amount=0.0, highlight_amount=-0.4)
        corrected = gamma_correction(corrected, 1.3)
        correction_log.append("Overexposure correction: highlight recovery + gamma 1.3")

    if lighting_info['has_low_contrast']:
        clip_limit = 4.0 if lighting_info['std_brightness'] < 15 else 2.5
        corrected = adaptive_histogram_equalization(corrected, clip_limit=clip_limit)
        correction_log.append(f"Low contrast correction: CLAHE (clip_limit={clip_limit})")
    elif lighting_info['has_uneven_lighting']:
        corrected = adaptive_histogram_equalization(corrected, clip_limit=2.0, tile_grid_size=(6, 6))
        correction_log.append("Uneven lighting correction: Soft CLAHE")

    if lighting_info['std_brightness'] < 30:
        corrected = unsharp_masking(corrected, strength=0.3, radius=1.2)
        correction_log.append("Sharpening applied")
```

Face Detection and Segmentation

```python
# Facer 라이브러리의 얼굴 관련 모델들
face_detector = facer.face_detector('retinaface/mobilenet', device=device)
face_parser = facer.face_parser('farl/celebm/448', device=device)
print("✓ Facer models loaded successfully.")
```



1. **Face Detection:** First, a face detector identifies the l ocation of the face.
2. **Face Parsing:** A semantic segmentation model then analyzes the facial region, creating a detailed map w here every pixel is assigned a label (e.g., skin, hair, ey es, lips).

Feature Extraction (Mini Kmeans)

```
# --- Mini K-Means for Feature Extraction ---

# 'pixels' contains thousands of Lab color values from one person's skin
if len(pixels) < 10:
    # Handle cases with insufficient skin pixels
    representative_colors = np.zeros((10, 3))
else:
    # 1. Initialize K-Means to find 10 clusters (representative colors)
    mini_kmeans = KMeans(n_clusters=10, n_init='auto', random_state=0)

    # 2. Sort the skin pixels into 10 groups
    mini_kmeans.fit(pixels)

    # 3. Get the center of each cluster, which is the representative color
    representative_colors = mini_kmeans.cluster_centers_
```

Those 10 representative colors are then converted into a list of numbers. The code uses the Lab color space, where each color is defined by three numbers (L*, a*, b*).

So, the final data for one image is:
10 colors × 3 values per color = a list of 30 numbers.

This list of 30 numbers is the **feature vector**. It's a compact, numerical signature of that person's skin tone.

Feature Scaling

The entire feature matrix is standardized using *StandardScaler* from *scikit-learn*.

```
# This code is located within the 'extract_skin_features' function

# 'color_features' is the raw table of 30D feature vectors before scaling.

# 1. Initialize the StandardScaler object.
scaler = StandardScaler()

# 2. Fit the scaler to the data and then transform it in one step.
#    This command learns the properties of your data (mean, std dev)
#    and applies the scaling formula.
scaled_features = scaler.fit_transform(color_features)

# 'scaled_features' is the final, standardized data used for clustering.
return scaled_features, labels, paths_list, scaler
```

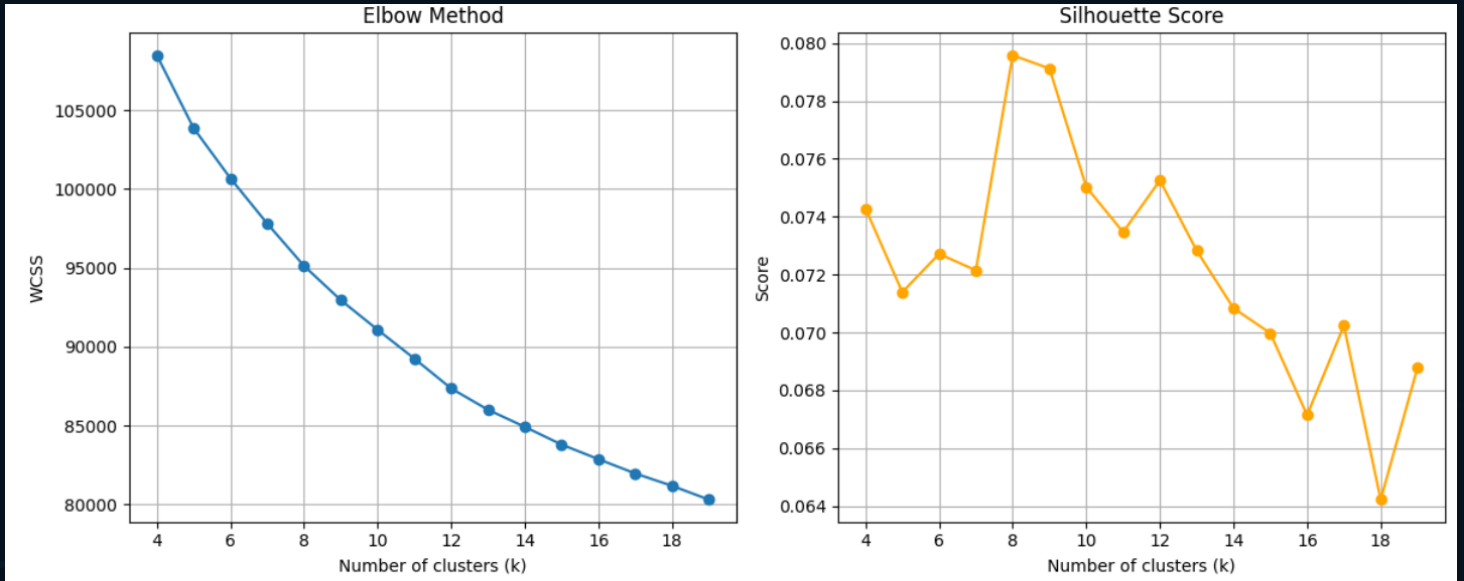
How It Works

- `scaler = StandardScaler()` : This line creates the scaling tool.
- `scaler.fit_transform(color_features)` : This single command performs two actions:
  - `.fit()` : The scaler first analyzes each of the 30 columns (features) in your data to calculate its average value (mean) and spread (standard deviation).
  - `.transform()` : It then applies the standardization formula— `(value - mean) / standard_deviation` —to every number, creating the new `scaled_features`.

```
# ... inside the main() function ...

# The 'final_features' variable holds the feature vectors for all images.
# 'optimal_k' is the best number of clusters you found (e.g., 8).

# This is the "big K-Means" part
kmeans = KMeans(n_clusters=optimal_k, init='k-means++', n_init=12, random_state=0)
cluster_labels = kmeans.fit_predict(final_features)

# After this, the model is saved
joblib.dump(kmeans, os.path.join(MODEL_DIR, 'kmeans_model.joblib'))
```

Model Training (Big Kmeans)

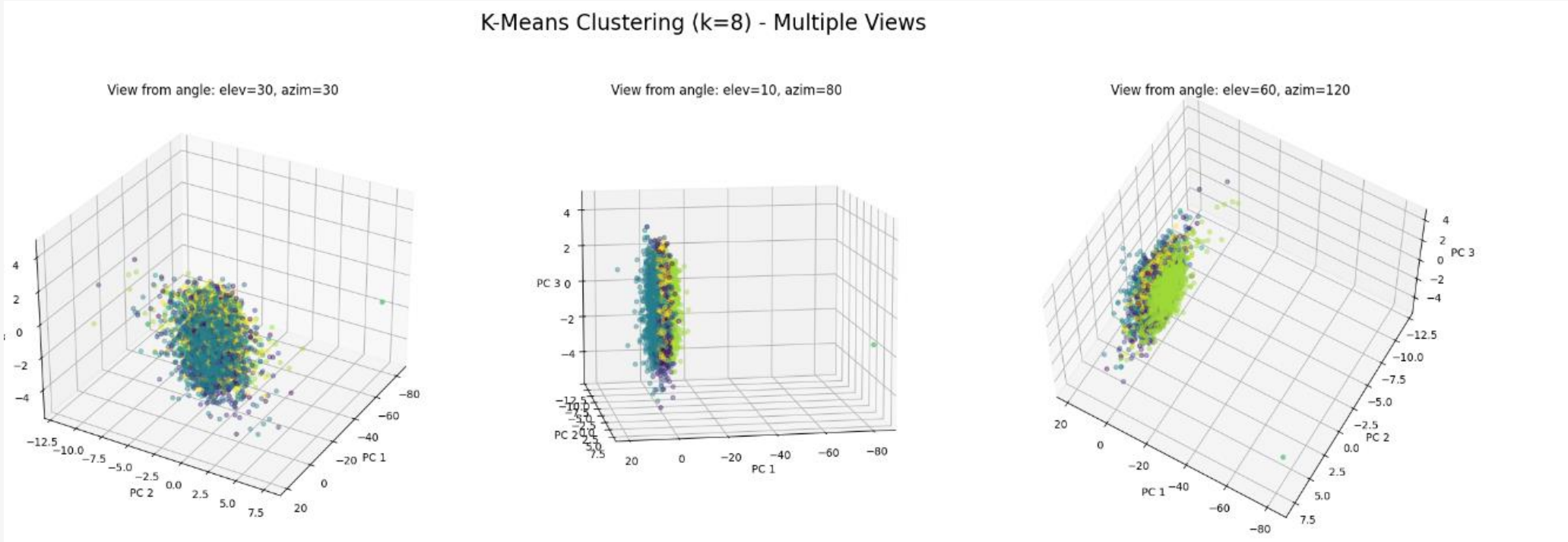
- **K-Means Algorithm Execution:** The algorithm iteratively converges on the optimal cluster centroids through the following steps:
  - **a. Initialization:** k initial centroids are strategically placed in the 30D feature space using the k-means++ method, which improves convergence by ensuring the initial centroids are well-spaced.
  - **b. Assignment Step:** For each 30D feature vector, the Euclidean distance to all k centroids is computed, and the vector is assigned to the cluster of the nearest centroid.
  - **c. Update Step:** The position of each centroid is recalculated by taking the mean of all feature vectors currently assigned to its cluster.
  - **d. Iteration & Convergence:** The Assignment and Update steps are repeated until the centroids no longer move significantly, at which point the algorithm has converged and the final cluster configuration is established.
- This entire training process was repeated for all k values in our test range (e.g., 4 to 20) to find the optimal number of clusters.




```
알고리즘이 찾은 상위 3개의 그룹(k) 개수:
  1등: k=8 (실루엣 점수: 0.0796)
  2등: k=9 (실루엣 점수: 0.0791)
  3등: k=12 (실루엣 점수: 0.0753)
모델 및 데이터를 저장할 디렉토리: '/content/drive/MyDrive/saved_models_skin_color10'
```

Selection of K Value and Clustering Evaluation

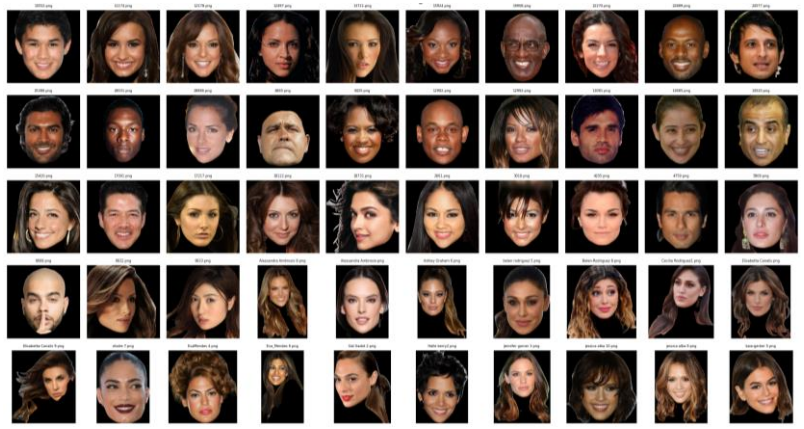
- The **Elbow Method** visualizes the within-cluster sum of squares (WCSS) against K, and identifies the point where increasing the number of clusters yields diminishing improvements.
- The **Silhouette Score** measures how similar each sample is to its own cluster compared to other clusters, with higher scores indicating clearer and more well-defined clusters.

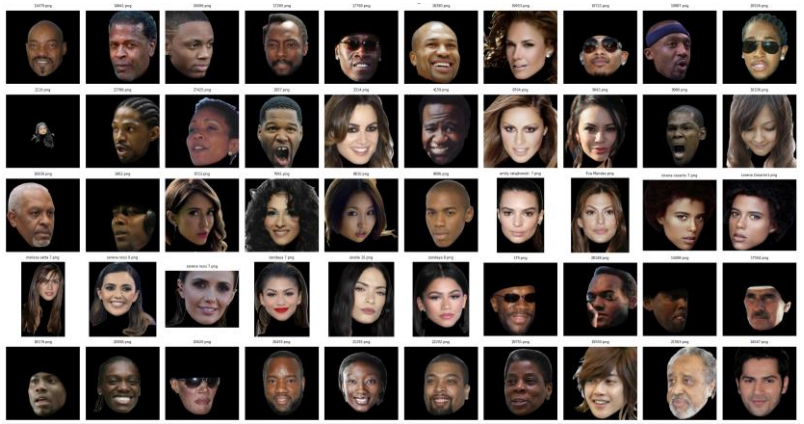


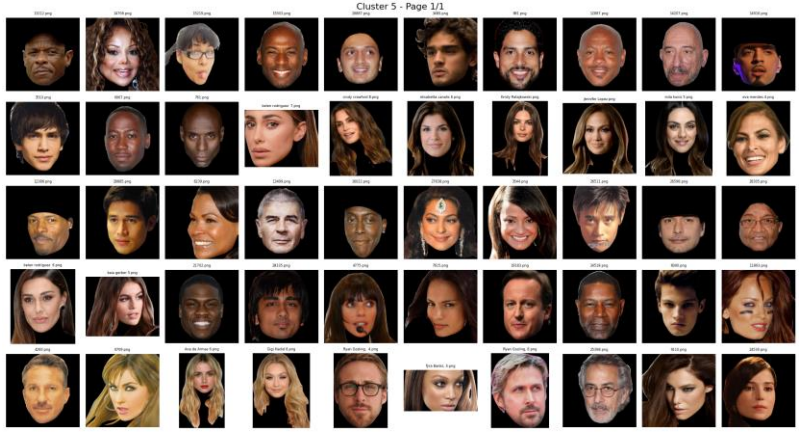
Clustering Result Visualization using Principal Component Analysis(PCA)

high-dimensional 30D space, direct visualization is impossible.
To visually inspect, we employed a dimensionality reduction technique.
We utilized **Principal Component Analysis (PCA)**.

Visualization of representative images by cluster







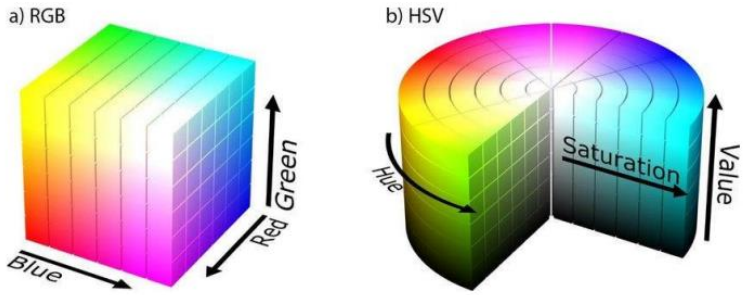
Cluster Labeling and Initial Palette Design

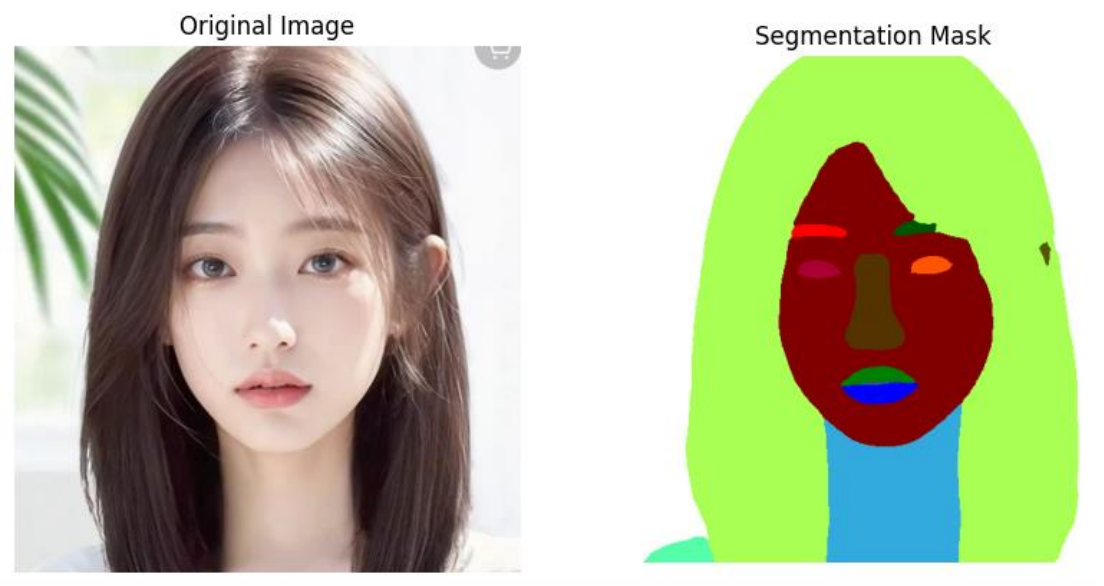
- the average HSV, Lab, and RGB values for each group were comprehensively analyzed and a palette name was assigned that matched the visual impression.

Cluster	Visual Name	Summary of Key Features
0	**Golden**	Highest average hue and saturation. Clearly warm tones.
1	**Warm Beige**	Overall, warm, with a slight olive undertone.
3	**Muted Clay**	Lowest saturation. Calm, muted colors.
4	**Warm Apricot**	Clear orange tones throughout. Stable, warm image.
5	**Peachy Pink**	Red-pink variability exists. Somewhat lovely, vibrant tones.
6	**Honey Buff**	It is a golden beige series that is warm and sweet like honey.
7	**Beige Rose**	Similar to cluster No.6, but slightly softer.

Cluster Labeling and Initial Palette Design

- After adjusting Hue and Value (brightness) to create a palette that harmonizes with the cluster, appropriate colors were manually selected to complete the final palette.





Color Overlay

- **Model:** BiSeNet
- **Training Data:** CelebAMask-HQ
- **Number of Classes:** 19
- **Pretrained Model Used:** 79999_iter.pth

pretrained models provide conventional simple color overlay

Limitations

particularly when transforming dark areas such as black hair, resulting in unnatural and flat color effects

To address this,

soft light blending technique

allows colors to be naturally composited while preserving the original image's texture, shading, and highlights, producing more realistic results.

sharpening was applied to hair regions to enhance clarity and detail

Soft Light Blending

Soft light blending is defined as follows when the base pixel B and blend pixel S are normalized to the [0,1] range

Where,

- B represents the brightness value of the base pixel
- S represents the brightness value of the blend pixel.

This formula preserves the shading in darker regions of th e original image while smoothly brightening lighter areas, producing results that are more realistic and visually natu ral compared to simple alpha blending.

$$\text{Result} = \begin{cases} 2 \cdot B \cdot S + B^2 \cdot (1 - 2S), & \text{if } S < 0.5 \\ 2 \cdot B \cdot (1 - S) + \sqrt{B} \cdot (2S - 1), & \text{if } S \geq 0.5 \end{cases}$$

Web Design and Implementation

- To implement the web application, **Python, JavaScript, HTML, CSS, and the Flask framework** were utilized. The trained machine learning models were saved in **.joblib** format to allow loading within the web environment, and users can upload images through the Flask-based interface.
- **Firebase** was integrated to support user management and personalization features.

Key Learnings from the Project

- **Unsupervised Learning :** Kmeans clustering
- **User-Centric Design:** Visualizing clusters, creating palettes, and overlaying colors taught how to improve user experience
- **Preprocessing Matters:** Adaptive lighting correction, skin isolation, and careful feature extraction
- **Tech Stack Experience:** Python, Flask, PyTorch, OpenCV, and web development
- **Teamwork Under Pressure:** As the “underdogs,” we learned to balance research, coding, and integration with limited prior experience

- Thanks for your support throughout the project!