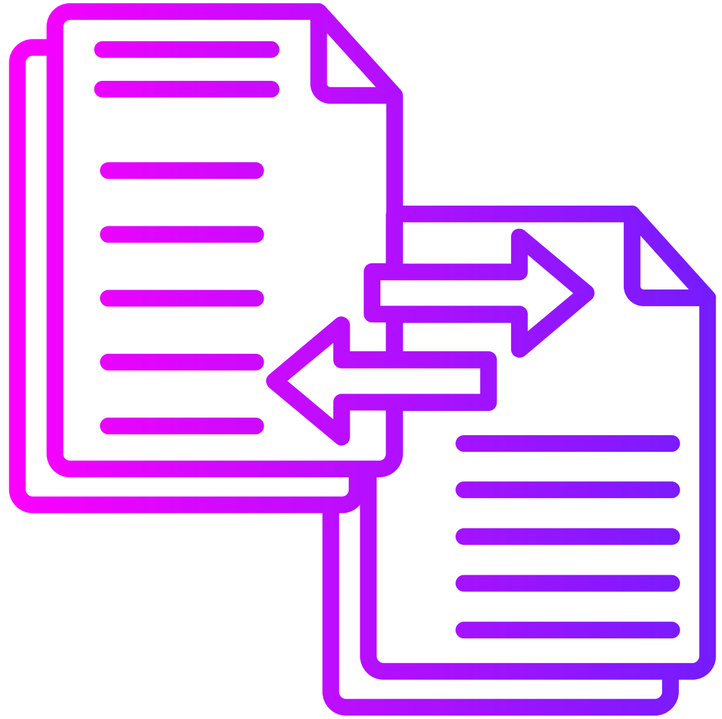
Phần mềm chuyển ảnh thành tranh vẽ

Thành viên nhóm:
Trần Đức Phương - B22DCCN640
Nguyễn Thanh Phong - B22DCCN616



GIỚI THIỆU ĐỀ TÀI

Mục tiêu:

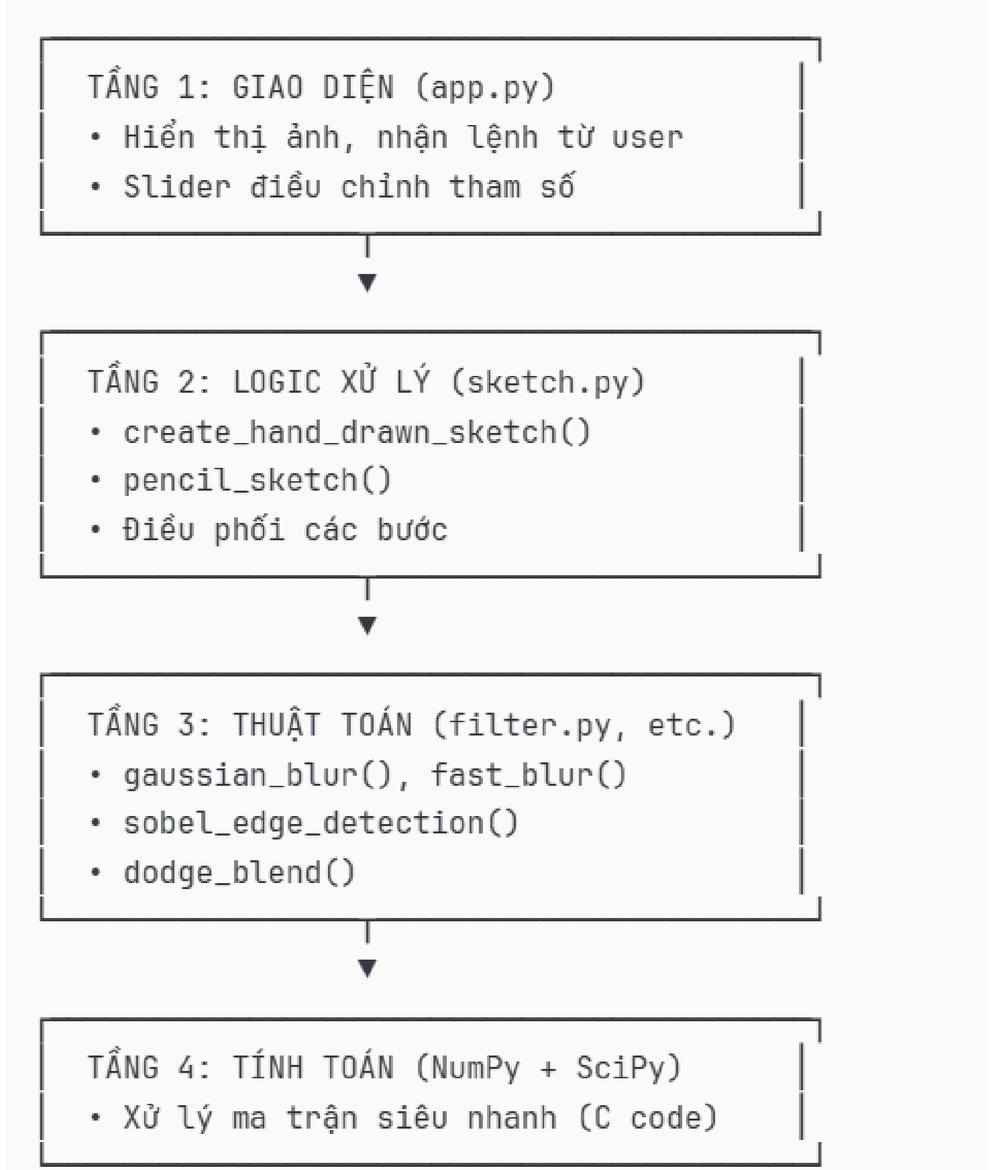
1. Xây dựng phần mềm chuyển đổi ảnh màu thành tranh vẽ tay
2. Tự viết hoàn toàn các thuật toán xử lý ảnh (không dùng OpenCV)
3. Tạo giao diện thân thiện, dễ sử dụng

Ứng dụng thực tế:

2 phong cách vẽ: Vẽ tay thường + Vẽ bút chì
Tốc độ xử lý: < 10 giây cho ảnh 2000×2000 pixels
Giao diện Tkinter trực quan với slider điều chỉnh realtime

TỔNG QUAN HỆ THỐNG

Kiến trúc hệ thống



Công nghệ sử dụng

- Python 3.7+
- NumPy (tính toán ma trận)
- Pillow (xử lý ảnh)
- Tkinter (giao diện)

Pipeline xử lí

Quy trình tạo hiệu ứng tranh vẽ tay từ ảnh RGB

1.Chuyển xám (5ms)

Giảm ảnh RGB từ 3 kênh → 1 kênh, chuẩn bị cho xử lý biên.

2.Làm mịn (6ms)

Dùng Gaussian / Box-Blur để giảm nhiễu, làm ảnh mượt.

3.Phát hiện biên (45ms)

Dùng Sobel để tìm các đường nét giống bút chì.

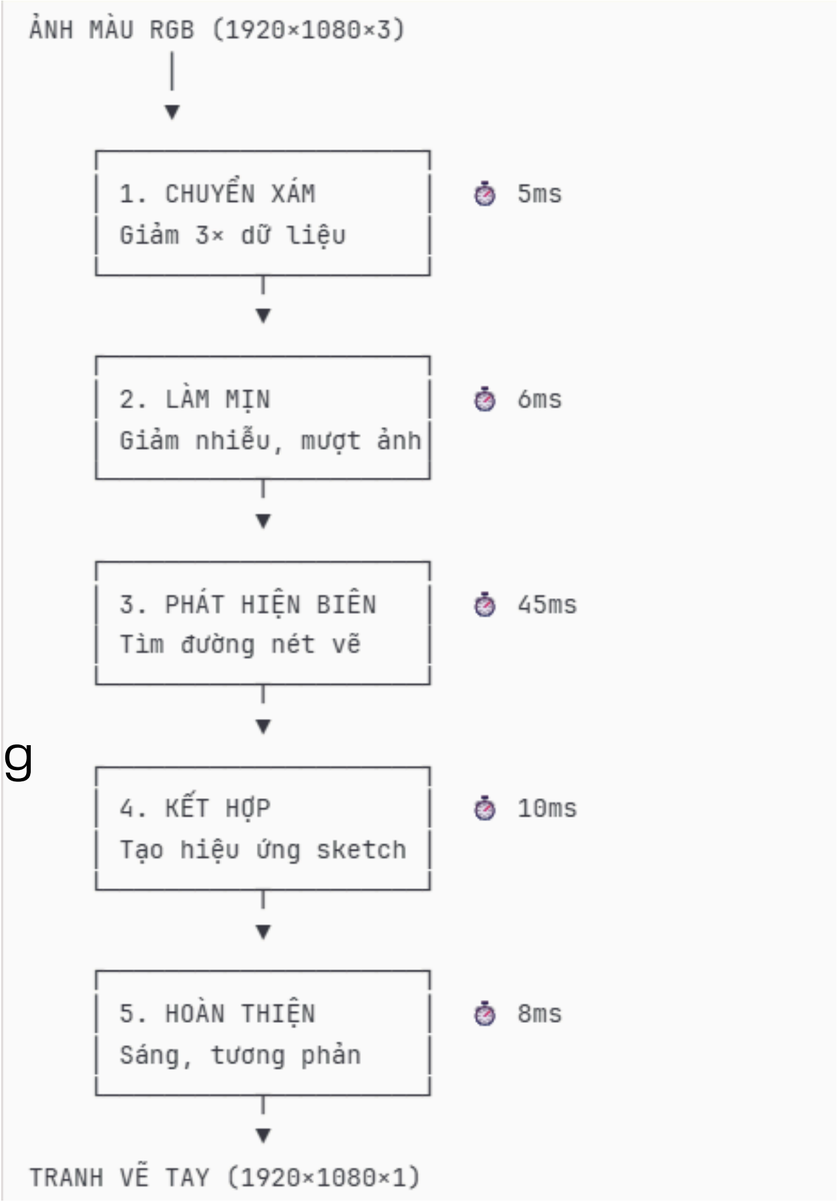
4.Kết hợp (10ms)

Trộn ảnh xám + đường biên (dodge blend) để tạo hiệu ứng sketch.

5.Hoàn thiện (8ms)

Tăng sáng/contrast để nét rõ và đẹp hơn.

Kết quả cuối: Ảnh sketch trắng đen



Bước 1: Chuyển ảnh xám(Grayscale Conversion)

Tại sao cần chuyển xám?

Sketch là ảnh đen trắng → không cần 3 kênh màu

Giảm 3× dữ liệu cần xử lý

Công thức chuẩn ITU-R BT.601:

Gray = 0.299 × R + 0.587 × G + 0.114 × B

```
# Nếu là ảnh màu RGB (3 kênh)
if image.ndim == 3 and image.shape[2] == 3:
    # Sử dụng trọng số chuẩn: Red=0.299, Green=0.587, Blue=0.114
    gray = np.dot(image[..., :3], [0.299, 0.587, 0.114])

# Nếu là ảnh RGBA (4 kênh), bỏ qua kênh alpha
elif image.ndim == 3 and image.shape[2] == 4:
    gray = np.dot(image[..., :3], [0.299, 0.587, 0.114])

# Trường hợp khác (không hợp lệ)
else:
    raise ValueError(f"Ảnh không hợp lệ. Shape: {image.shape}, cần (H,W) hoặc (H,W,3) hoặc (H,W,4)")

# Đảm bảo giá trị trong khoảng [0, 255]
return np.clip(gray, 0, 255).astype(np.uint8)
```

Bước 2: Làm mịn

Lí do:

- Ảnh từ camera có **noise** (nhiễu điểm)
- Edge detection rất nhạy với noise
- Cần làm mịn trước để kết quả ổn định

Nguyên lí: Mỗi pixel = Trung bình của 9×9 pixel xung quanh

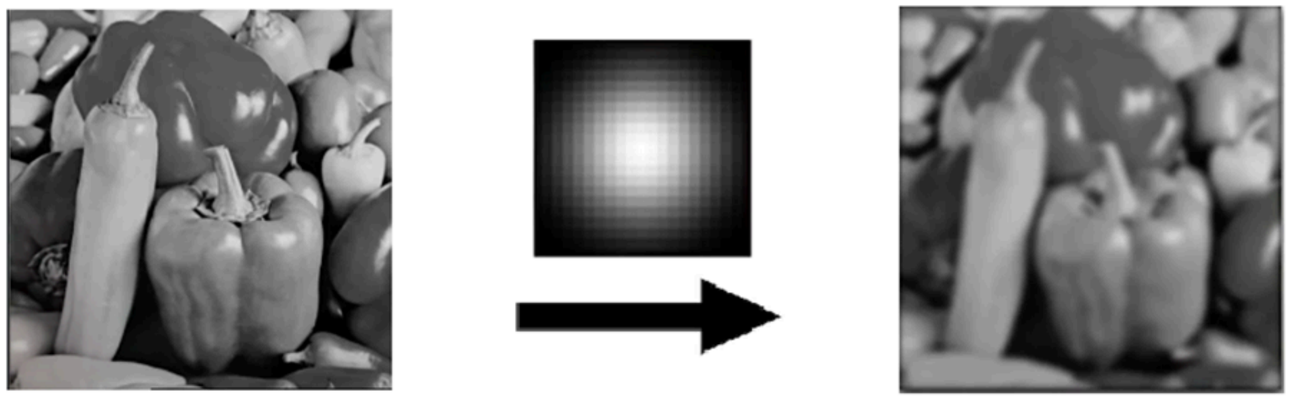
```
kernel = np.ones((size, size), dtype=np.float32) / (size * size)

if image.ndim == 3:
    result = np.zeros_like(image, dtype=np.float32)
    for c in range(image.shape[2]):
        result[:, :, c] = convolve2d(image[:, :, c].astype(np.float32), kernel)
else:
    result = convolve2d(image.astype(np.float32), kernel)

return np.clip(result, 0, 255).astype(np.uint8)
```

Bước 2: Làm mịn

Mục đích: Giảm nhiễu và chi tiết nhỏ để chuẩn bị cho phát hiện biên, giúp tranh vẽ trông mềm mại, tự nhiên hơn



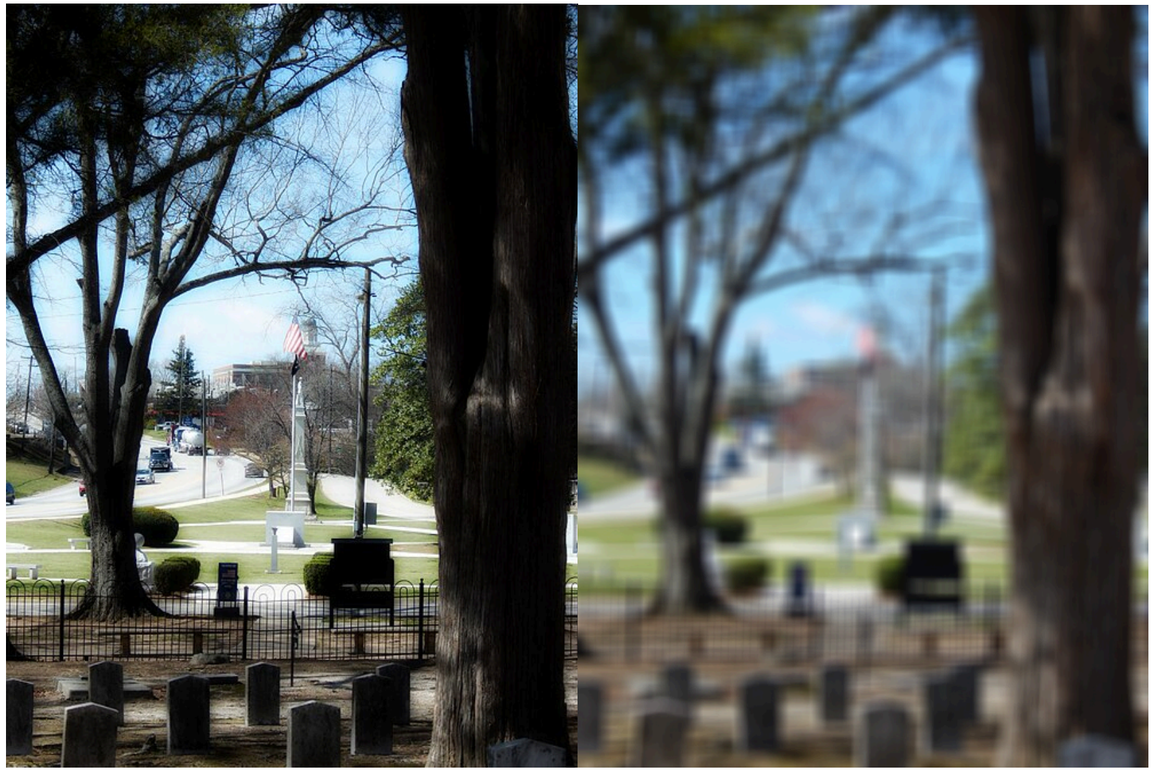
Original Filter Result

Bước 2: Làm mịn

Box Blur (Làm mịn đều)

- Đặc điểm: Tất cả pixel trong kernel có trọng số bằng nhau
- Ứng dụng: Làm mịn nhanh, đơn giản

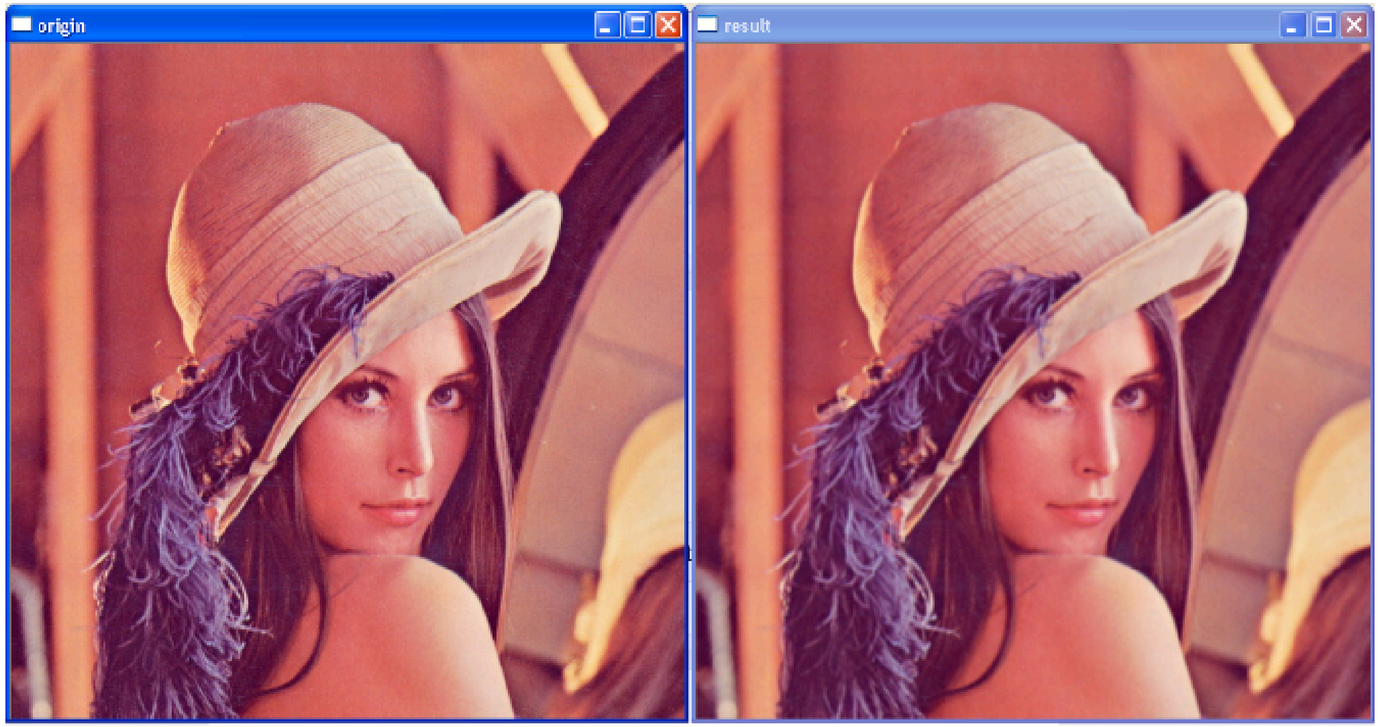
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$



Bước 2: Làm mịn

Gaussian Blur (Làm mịn Gaussian)

- Đặc điểm: Pixel ở trung tâm có trọng số cao hơn, giảm dần ra biên
- Ưu điểm: Giữ được biên cạnh tốt hơn box blur



$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \text{ (Một chiều)}$$

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \text{ (Hai chiều, ảnh)}$$

Bước 3: Phát hiện biên(SOBEL EDGE DETECTION)

Vùng thay đổi độ sáng đột ngột = Nơi cần vẽ nét

Độ lớn gradient tại mỗi pixel::

$$G = \sqrt{G_x^2 + G_y^2}$$

Vai trò:

- Phát hiện biên thô.
- Nhạy với nhiễu → thường kết hợp Gaussian làm mờ trước

```python
# Kernel Sobel chuẩn
sobel_x = np.array([[-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]], dtype=np.float32)

sobel_y = np.array([[-1, -2, -1],
                    [ 0,  0,  0],
                    [ 1,  2,  1]], dtype=np.float32)

# Tính gradient theo 2 hướng
gradient_x = convolve2d_fast(smoothed.astype(np.float32), sobel_x)
gradient_y = convolve2d_fast(smoothed.astype(np.float32), sobel_y)

# Tính độ lớn gradient (magnitude)
gradient_magnitude = np.sqrt(gradient_x**2 + gradient_y**2)

# Chuẩn hóa về [0, 255]
if gradient_magnitude.max() > 0:
    gradient_magnitude = (gradient_magnitude / gradient_magnitude.max()) * 255

# Áp dụng threshold nếu có
if threshold is not None:
    gradient_magnitude[gradient_magnitude < threshold] = 0

return np.clip(gradient_magnitude, 0, 255).astype(np.uint8)
```

Bước 4: Kết hợp(Blending)

Công thức: sketch = smoothed × (1.0 – edges)

Nơi có biên (edges cao):
edges = 0.9 (gần 1)
sketch = smoothed × (1 – 0.9) = smoothed × 0.1 = RẤT TỐI → Nét đen

Nơi không biên (edges thấp):
edges = 0.1 (gần 0)
sketch = smoothed × (1 – 0.1) = smoothed × 0.9 = RẤT SÁNG → Giấy trắng

Bước 5: Hoàn thiện

3 kĩ thuật cuối:

Tăng độ sáng: sketch = sketch + 20 # Mỗi pixel sáng thêm 20**:**

Tăng độ tương phản:
factor = 1.124 # Công thức chuẩn
sketch = factor × (sketch – 128) + 128# Sáng thêm sáng, tối thêm tối

Thêm texture giấy:
noise = np.random.normal(0, 2, (H, W))
sketch = sketch + noise

Trước: Sketch nhạt, mịn → Sau: Sketch sáng, sắc nét, có texture

Phong cách 2– Pencil Sketch

Đặc điểm:

- Nét đậm, sắc nét hơn
- Tương phản cao
- Giống vẽ bút chì 2B, 4B

Pipeline:

1. Grayscale
2. High contrast (factor = 1.8)
3. Đảo ngược ảnh (Invert) : 255 – gray
4. Gaussian Blur (kernel lớn 21×21)
5. Dodge blend: result = (base × 255) / (255 – blend + ε)
6. Gamma correction: pixel^0.85
7. Add heavy texture

So sánh:

Tiêu chí	Hand-drawn	Pencil
Nét vẽ	Mềm mại	Đậm nét
Tương phản	Trung bình	Cao
Texture	Nhẹ (σ=2)	Nặng (σ=4)
Tốc độ	74ms	65ms
Phù hợp	Phong cảnh	Kiến trúc

Thách thức và giải pháp:

Thách thức 1: Tốc độ xử lý chậm

- Xử lý ảnh độ phân giải cao (HD, 4K) mất nhiều thời gian
- Các phép toán convolution với kernel lớn tốn tài nguyên
- Ảnh >2MP có thể mất 10–30 giây để xử lý
- Giải pháp:
  - Tự động phát hiện kích thước ảnh
  - Ảnh nhỏ (<500px): xử lý từng pixel (step=1)
  - Ảnh lớn (≥500px): xử lý cách quãng (step=2)

Thách thức 2: Noise trong sketch

- Edge detection quá nhạy
- Giải pháp :Soft normalization với tanh thay vì linear
  - Pre-blur với box filter nhanh

Thách thức 3: UI lag khi xử lý

- Giải pháp: Threading + progress callback

Mở rộng tương lai:

1. Thêm phong cách mới:

- Charcoal sketch (vẽ than)
- Watercolor effect (màu nước)
- Oil painting style (sơn dầu)

2. Tối ưu hóa:

- GPU acceleration với CuPy/Numba
- Multi-threading cho từng kênh màu
- Batch processing nhiều ảnh

3. Tính năng nâng cao:

- Auto parameter tuning theo loại ảnh
- Style transfer với deep learning
- Mobile app (iOS/Android)

4. Nghiên cứu:

- Non-photorealistic rendering (NPR)
- Neural style transfer

Demo sản phẩm:

Giao diện web: