LayerD: Decomposing Raster Graphic Designs into Layers

Supplementary Material

A. Editing Examples

We show editing examples in Fig. A. Here, we use LayerD to decompose the input image into layers, divide each layer into connected components, and group text components using CRAFT [1] to facilitate editing. We import the layers into PowerPoint[1] and perform various edits, from simple layout manipulation to applying built-in image effects, *at the layer level*. As the examples show, once the images are decomposed, users can intuitively edit them with precise control over each graphic element.

B. Additional Results

We present additional examples of decomposed graphic design images using our method in Figs. B and C. These examples are selected from the Crello [7] test set and demonstrate the effectiveness of our method across diverse design styles.

C. Failure Cases

In Figs. D and E, we show typical failure cases of our method. The first set of failure cases (Fig. D) involves objects that are too small, such as detailed text descriptions, which are challenging to decompose due to their limited spatial extent. We believe that these can be mitigated by increasing the resolution of the input images. The second set of failure cases (Fig. E) is due to the ambiguity of the layer granularity. For these samples, it is difficult even for humans to decompose them into the same layers consistently. Although our evaluation metrics account for such ambiguity, we may need to improve training objectives or the post-refinement process to address these cases.

D. User Study

We conduct a user study in which 21 cloudworkers experienced in layer-based image editing rate the practical utility of 50 decomposition results—randomly ordered and anonymized—from LayerD and our two baselines on the same images using a five-point scale. Tab. 1 summarizes the results of the user study. LayerD achieves the highest average score, and a significant majority of the users (71.4%) rate LayerD the highest average score. This result further emphasizes the practical superiority of our method.

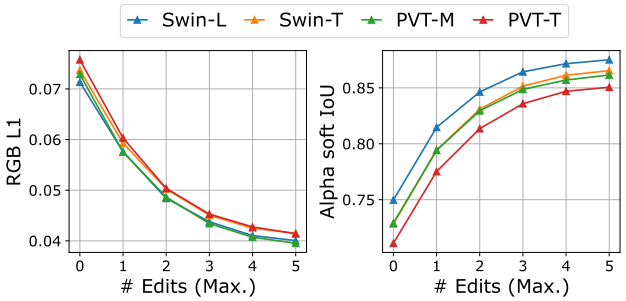
[1] https://www.microsoft.com/powerpoint

Table 1. Results of the user study. We report the average score, the number of users who rate each method as the best on average across all samples (#Pref. users), and the number of samples for which each method is rated the best on average across all users (#Win samples).

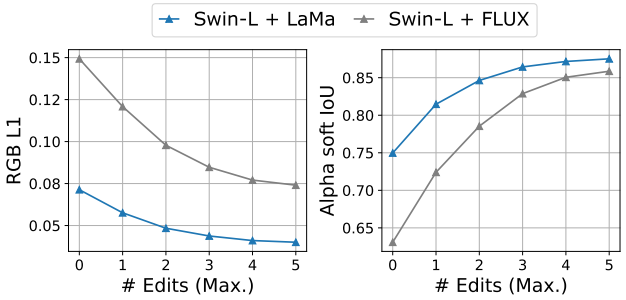
	Score	#Pref. users	#Win samples
LayerD	**3.74**	**15 (71.4%)**	**27 (54.0%)**
YOLO base	3.52	2 (9.5%)	15 (30.0%)
VLM base	3.31	4 (19.0%)	8 (16.0%)

E. Influence of Matting and Inpainting Model Choices

We vary the matting backbones (Swin-L/T [4], PVT-M/S [6]) and replace the inpainting model with FLUX.1 Fill [dev] [2] and evaluate their influence. The larger matting models improve performance while using FLUX.1 Fill [dev] shows significant degradation. Generative inpainting often introduces unwanted objects, which interfere with subsequent decomposition steps. This highlights the need for graphic design-specific inpainting as well as refinement.



(a) Results with different matting backbones, SwinTransformer [4] and PVT [6] variants. The inpainting model is fixed to LaMa [5].



(b) Results with different inpainting models, LaMa [5] and FLUX [3].

Figure F. Evaluation results of LayerD with different matting (a) and inpainting model (b) choices.

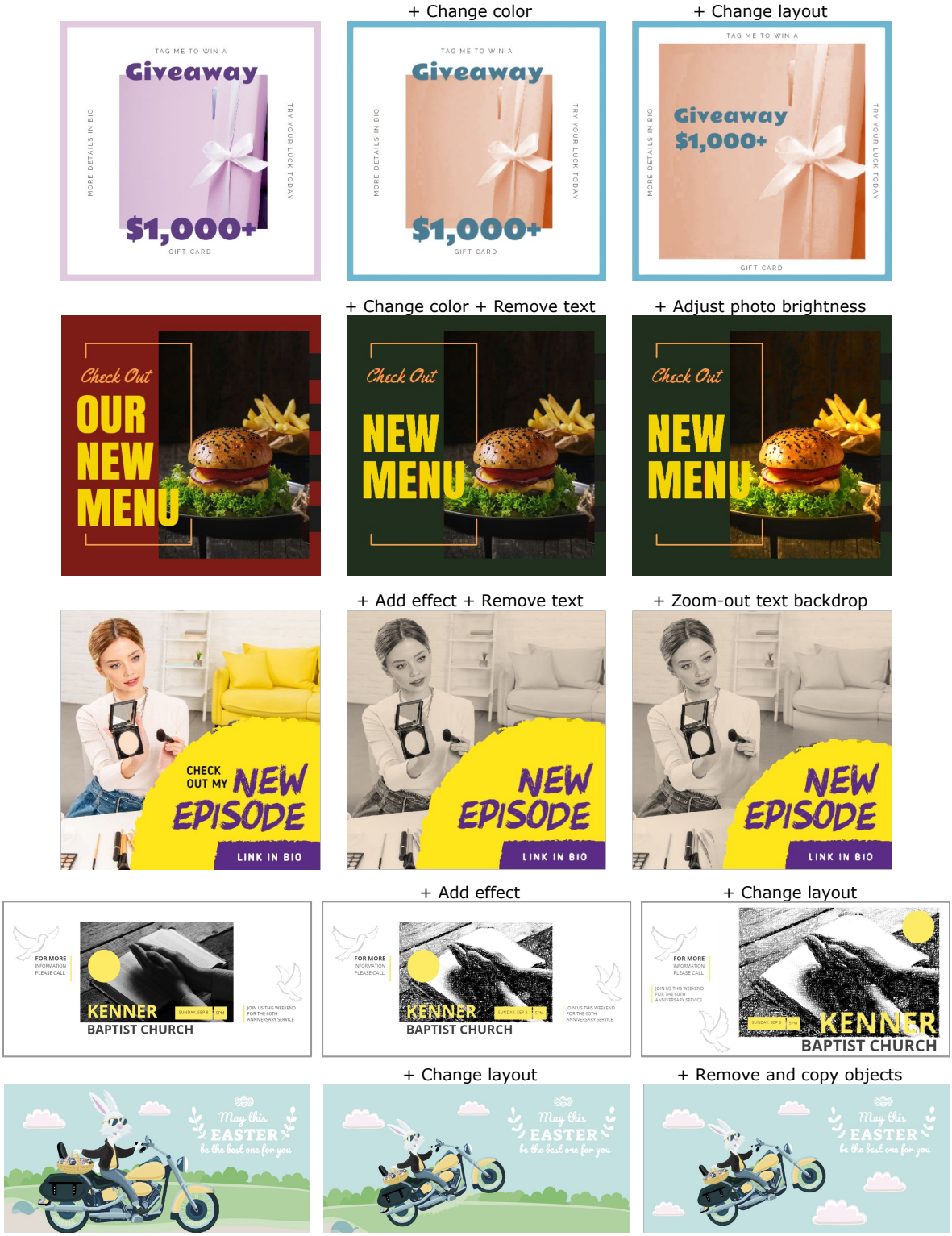


Figure A. Editing examples on Crello [7] test set. The leftmost images are the original images, and the remaining images are edited ones based on the decomposed layers. We use LayerD to decompose the original images into layers, divide them into connected components, and group text components using CRAFT [1]. Then, we perform various *layer-level* edits, from simple layout changes to applying built-in image effects, on PowerPoint.

Figure B. Additional qualitative results of our method on Crello [7] test set. The leftmost column shows the input image, and the remaining columns show the decomposed layers from back to front.

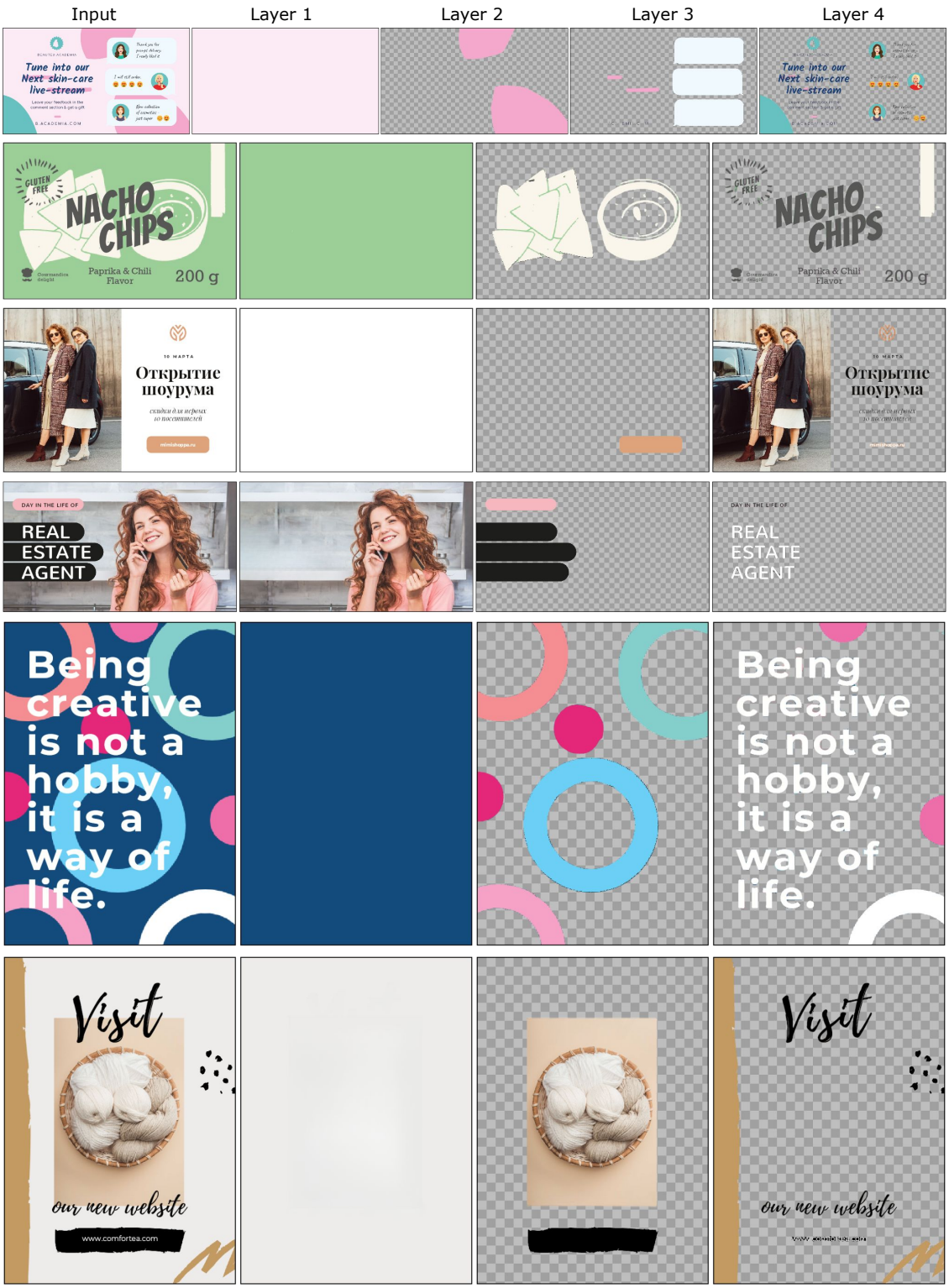


Figure C. Additional qualitative results of our method on Crello [7] test set. The leftmost column shows the input image, and the remaining columns show the decomposed layers from back to front.

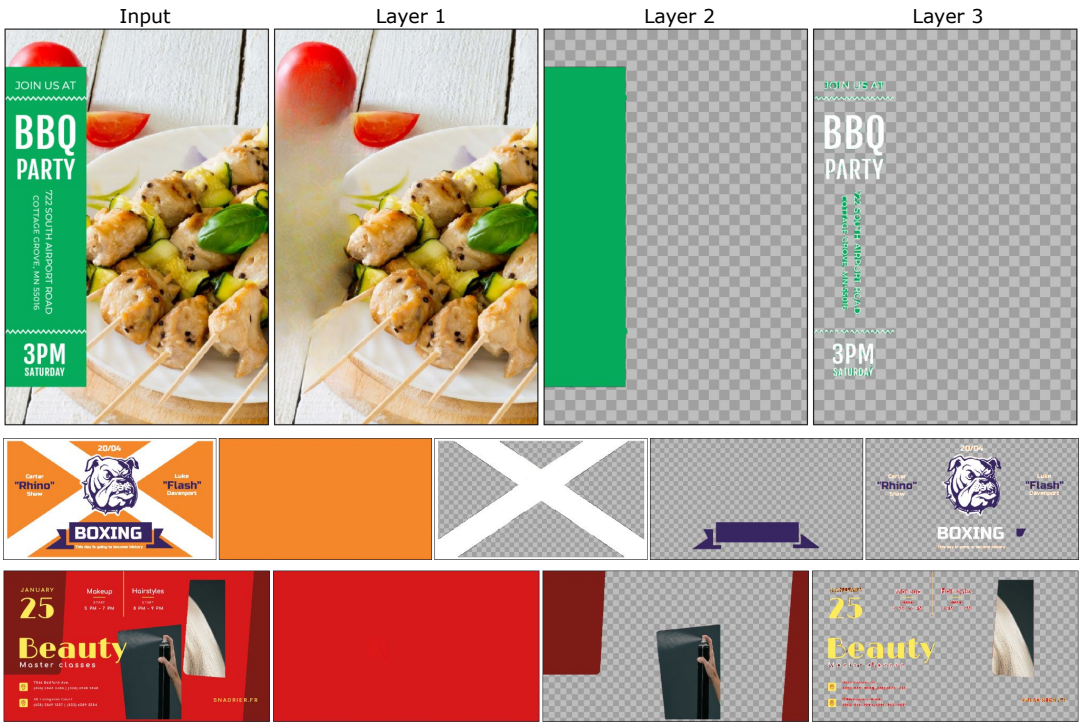


Figure D. Failure samples for too small objects on Crello [7] test set. The leftmost column shows the input image, and the remaining columns show the decomposed layers from back to front.

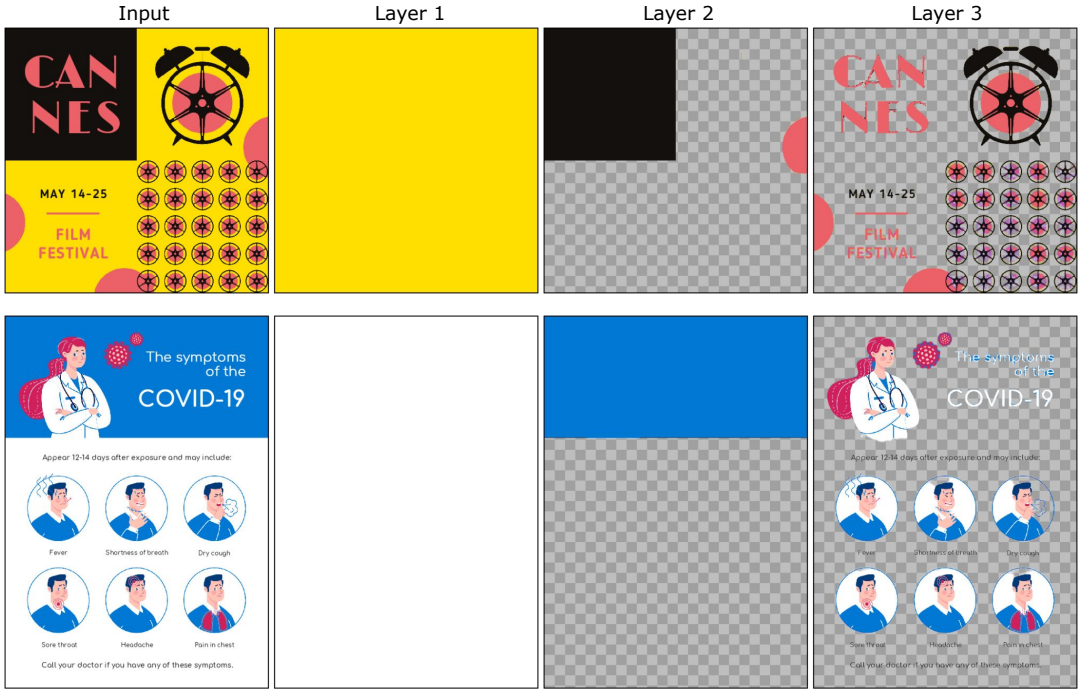


Figure E. Failure samples due to the ambiguity of the layer granularity on Crello [7] test set. The leftmost column shows the input image, and the remaining columns show the decomposed layers from back to front.

F. Detail of Decomposition Metrics

F.1. Dynamic Time Warping

We implement the Dynamic Time Warping (DTW) as shown in Algorithm 1. Given decomposition results $\hat{Y} = (\hat{\boldsymbol{l}}_k)_{k=0}^{K}$ and ground truth $Y = (\boldsymbol{l}_q)_{q=0}^{Q}$, the output pairs must include $(0, 0)$ and (K, Q) as the start point and end point with a step size of 1, and every layer must be included in at least one pair. An average distance is then computed over all pairs as the final output.

Algorithm 1 Dynamic Time Warping (DTW)

```
# Inputs:
# - ls: decomposition results of length K (from bottom
      to top)
# - gts: ground truth of length Q (from bottom to top)
# - dist: distance func bounded in [0, 1]
#
# Outputs:
# - pairs: a list of (l_idx, gt_idx)
# - D: distance

# Step 1: Compute Cost Matrix
C = np.zeros((len(ls), len(gts)))
for i in range(len(ls)):
   for j in range(len(gts)):
      C[i,j] = dist(ls[i], ls[j])

# Step 2: Compute Accumulated Cost Matrix
D = np.zeros((len(ls), len(gts)))
for i in range(1, len(ls)):
   D[i, 0] = D[i-1,0] + C[i,0]
for j in range(1, len(gts)):
   D[0, j] = D[0,j-1] + C[0,j]
for i in range(1, len(ls)):
   for j in range(1, len(gts)):
      D[i,j] = C[i, j] + min(D[i-1,j], D[i,j-1], D[i
           -1,j-1])

# Step 3: Backtrace to Find Optimal Alignment
i, j = len(ls)-1, len(gts)-1
pairs = [(i,j)]
while True:
   if i==0 and j==0:
      break
   elif i==0:
      pairs.append((i,j-1))
      j-=1
   elif j==0:
      pairs.append((i-1,j))
      i -= 1
   elif D[i-1,j-1]<=D[i-1,j] and D[i-1,j-1]<=D[i,j-1]:
      pairs.append((i-1,j-1))
      i -= 1
      j -= 1
   elif D[i-1,j]<=D[i-1,j-1] and D[i-1,j]<=D[i,j-1]:
      pairs.append((i-1,j))
      i -= 1
   else:
      pairs.append((i,j-1))
      j -= 1

D = sum([Dist(ls[i], gts[j]) for i,j in pairs])/len(
     pairs)

return pairs, D
```

F.2. Edits algorithm

We employ an iterative refinement process with DTW to quantify the number of edits required to align the decomposition results with the given ground truth. At each iteration, we apply the edit (`Merge`) that yields the highest gain until either the maximum number of edits is reached or the number of layers is reduced to two, as shown in Algorithms 2 and 3. To efficiently approximate the optimal edit, we adopt a greedy search strategy: at iteration i, we focus on changes in distances between consecutive layers—specifically, layers i, $i+1$, and $i+2$ (if present)—rather than evaluating all layers globally. The optimal edit is then selected from among all candidates at each iteration, ensuring a balance between computational efficiency and alignment accuracy. Although Algorithms 2 and 3 describe only the merging of predicted layers for simplicity, we apply the same merging procedure to both the predicted and ground truth layers to address both under- and over-decomposition. See Figs. G and H for visualization of the alignment and merging process.

Algorithm 2 MergeEdit

```
# Inputs:
# - ls: decomposition results of length K (bottom to
    top)
# - gts: ground truth of length Q (bottom to top)
# - emax: maximum number of edits
# - dist: distance function bounded in [0, 1]
#
# Outputs:
# - pairs: a list of (l_idx, gt_idx)
# - D: distance
# - e: number of edits

e = 0
while e < emax and len(ls) > 2:
   pairs, _ = dtw(ls, gts)
   merged_ids, gains = find_gains(ls, gts,
                        pairs, dist)
   if len(gains) > 0:
      best_id = merged_ids[argmin(gains)]
      merged = merge(ls[best_id], ls[best_id+1])
      ls[best_id] = merged
      ls.pop(best_id+1)
   else:
      break
   e += 1
return dtw(ls, gts), e


def merge(x, y): # Merge func by OpenCV
   return Image.alpha_composite(x, y)
```

Algorithm 3 FindGains

```
# Inputs:
# - ls: decomposition results of length K (bottom to
    top)
# - gts: ground truth of length Q (bottom to top)
# - pairs: list of (l_idx, gt_idx) obtained from DTW
# - dist: distance function bounded in [0, 1]
#
# Outputs:
# - merged_ids: list of indices where merging occurs
# - gains: list of corresponding distance reductions

merged_ids, gains = [], []
for i in range(len(ls)-1):
   # Step 1: Compute merged layer candidates
   subls = [merge(ls[i], ls[i+1])] + ([ls[i+2]] if i+2
        < len(ls) else [])

   # Step 2: Gather corresponding ground truth layers
   subgts = [
      [gts[p[1]] for p in pairs if p[0] == i],
      [gts[p[1]] for p in pairs if p[0] == i+1]
   ]

   # Step 3: Compute current distance sum
   curD = sum([dist(ls[i], subgt) for subgt in subgts
        [0]]) + \
         sum([dist(ls[i+1], subgt) for subgt in subgts
              [1]])

   # Step 4: Compute distance sum after merging
   Ds = []
   for j in range(len(subls)):
      for k in range(len(subgts)):
         Ds.append(sum([dist(subls[j], subgt) for
              subgt in subgts[k]]))
   Ds = [d + Ds[0] for d in Ds[1:]]
   minD = min(Ds)

   # Step 5: Check if merging reduces distance
   if minD < curD:
      merged_ids.append(i)
      gains.append(minD - curDs)
return merged_ids, gains


def merge(x, y): # Merge func by OpenCV
   return Image.alpha_composite(x, y)
```

G. Loss functions

We use binary cross-entropy loss $\mathcal{L}_{\text{BCE}}$, IoU loss $\mathcal{L}_{\text{IoU}}$, and SSIM loss $\mathcal{L}_{\text{SSIM}}$ in our training as BiRefNet [8]. Definitions of each loss function are as follows.

$$\mathcal{L}_{\text{BCE}}(\hat{\boldsymbol{l}}^{\text{A}}, \boldsymbol{l}^{\text{A}}) = \frac{1}{|\Omega|} \sum_{i,j \in \Omega} -\boldsymbol{l}^{\text{A}}_{i,j} \log \hat{\boldsymbol{l}}^{\text{A}}_{i,j} - (1 - \boldsymbol{l}^{\text{A}}_{i,j}) \log(1 - \hat{\boldsymbol{l}}^{\text{A}}_{i,j}), \tag{1}$$

$$\mathcal{L}_{\text{IoU}}(\hat{\boldsymbol{l}}^{\text{A}}, \boldsymbol{l}^{\text{A}}) = 1 - \frac{\sum_{i,j \in \Omega} \boldsymbol{l}^{\text{A}}_{i,j} \hat{\boldsymbol{l}}^{\text{A}}_{i,j}}{\sum_{m,n \in \Omega} \boldsymbol{l}^{\text{A}}_{m,n} + \hat{\boldsymbol{l}}^{\text{A}}_{m,n} - \boldsymbol{l}^{\text{A}}_{m,n} \hat{\boldsymbol{l}}^{\text{A}}_{m,n}}, \tag{2}$$

$$\mathcal{L}_{\text{SSIM}}(\hat{\boldsymbol{l}}^{\text{A}}, \boldsymbol{l}^{\text{A}}) = 1 - \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \frac{(2\mu_{\boldsymbol{l}^{\text{A}}_p} \mu_{\hat{\boldsymbol{l}}^{\text{A}}_p} + C_1)(2\sigma_{\boldsymbol{l}^{\text{A}}_p \hat{\boldsymbol{l}}^{\text{A}}_p} + C_2)}{(\mu^2_{\boldsymbol{l}^{\text{A}}_p} + \mu^2_{\hat{\boldsymbol{l}}^{\text{A}}_p} + C_1)(\sigma^2_{\boldsymbol{l}^{\text{A}}_p} + \sigma^2_{\hat{\boldsymbol{l}}^{\text{A}}_p} + C_2)}, \tag{3}$$

where Ω denotes the set of spatial indices, and $\mathcal{P}$ represents the set of overlapping patches. The local mean $\mu_{\hat{\boldsymbol{l}}^{\text{A}}_p}$ and variance $\sigma^2_{\hat{\boldsymbol{l}}^{\text{A}}_p}$, as well as the local mean $\mu_{\boldsymbol{l}^{\text{A}}_p}$ and variance $\sigma^2_{\boldsymbol{l}^{\text{A}}_p}$ of ground truth, are computed within corresponding patches indexed by $p \in \mathcal{P}$. The covariance $\sigma_{\boldsymbol{l}^{\text{A}}_p \hat{\boldsymbol{l}}^{\text{A}}_p}$ quantifies structural similarity between the prediction and ground truth patches. C_1 and C_2 are constants and the setting details follow [8], except that both the predicted and ground-truth alpha maps $\hat{\boldsymbol{l}}^{\text{A}}$ and $\boldsymbol{l}^{\text{A}}$, are not binarized due to shading and smooth transitions commonly used in graphic design.

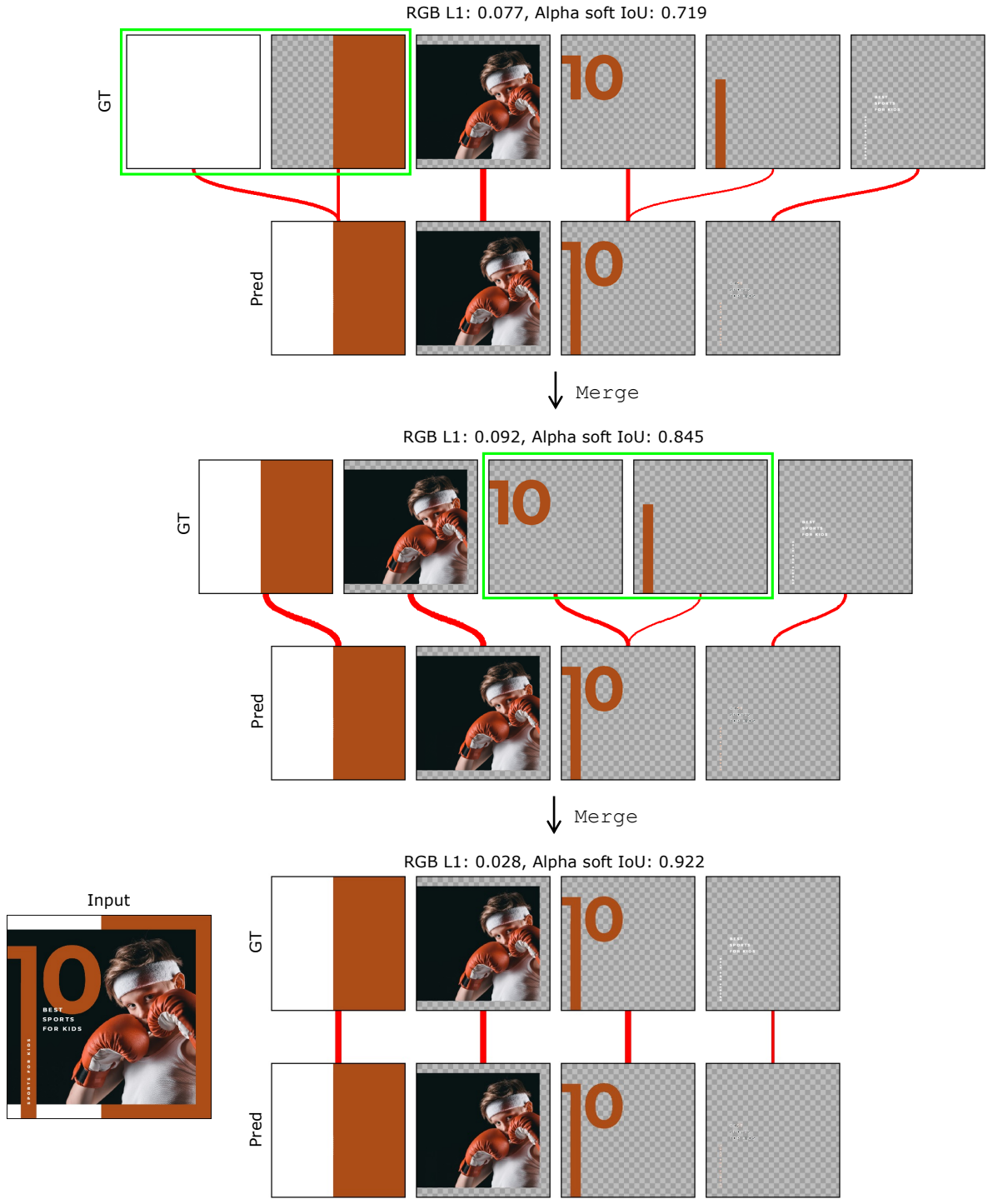


Figure G. Visual example of the DTW-based layer alignment and editing process. Red lines connect matched layers between LayerD's prediction and the ground truth; their thickness represents the matching score (the inverse of the distance), *i.e.*, the thicker the line, the higher the score. Green boxes indicate the layers that are merged during the editing process. All layers are sorted from back to front, with the backmost layer on the left and the frontmost on the right. Although the decomposition result appears useful for editing the input image, its quality is underestimated due to a mismatch in granularity with the ground truth. Layer merging resolves this mismatch, enabling a more faithful evaluation of the decomposition quality.

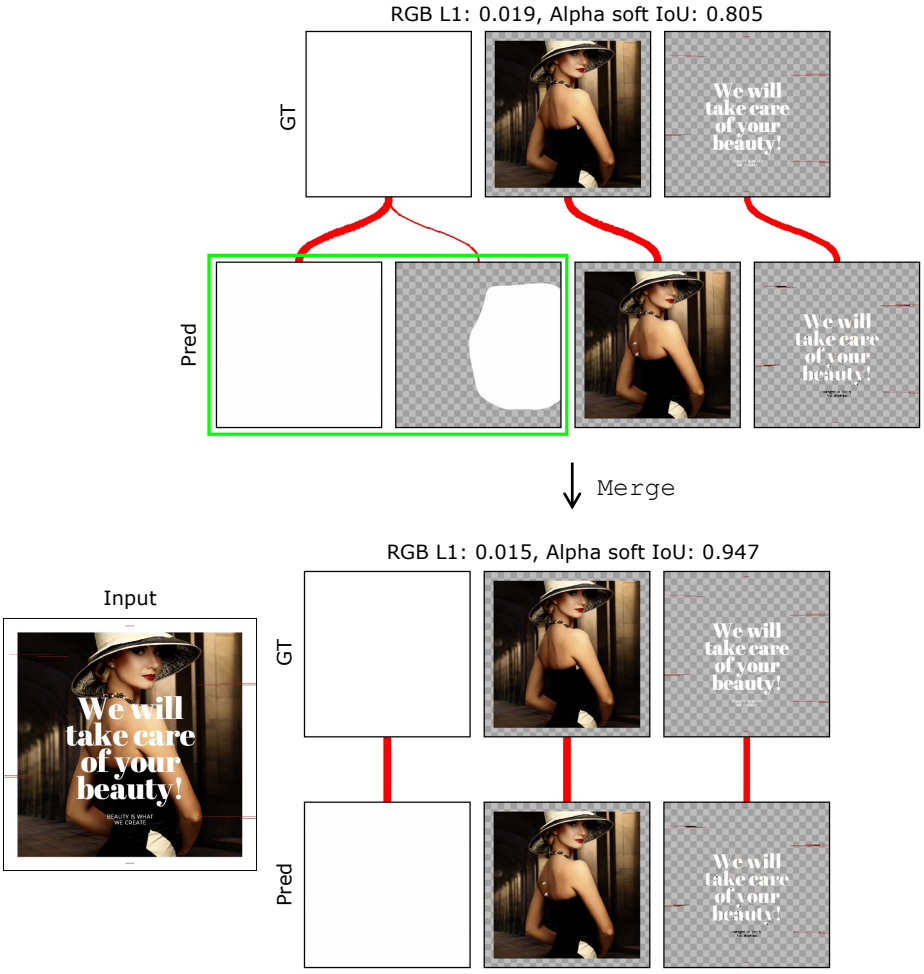


Figure H. Visual example of the DTW-based layer alignment and editing process. Red lines connect matched layers between LayerD's prediction and the ground truth; their thickness represents the matching score (the inverse of the distance), *i.e.*, the thicker the line, the higher the score. Green boxes indicate the layers that are merged during the editing process. All layers are sorted from back to front, with the backmost layer on the left and the frontmost on the right. LayerD overdecomposes the white background, but in practical scenarios, it is easy to merge these into a single layer. Our evaluation treats such cases as requiring a single edit operation, reflecting the actual editing workload for users.

References

[1] Youngmin Baek, Bado Lee, Dongyoon Han, Sangdoo Yun, and Hwalsuk Lee. Character region awareness for text detection. In *CVPR*, pages 9365–9374, 2019. 1, 2

[2] Black Forest Labs. FLUX.1 fill [dev]. https://huggingface.co/black-forest-labs/FLUX.1-Fill-dev, 2024. Last accessed 7 March, 2025. 1

[3] Black Forest Labs. FLUX.1 [dev]. https://huggingface.co/black-forest-labs/FLUX.1-dev, 2024. Last accessed 7 March, 2025. 1

[4] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021. 1

[5] Roman Suvorov, Elizaveta Logacheva, Anton Mashikhin, Anastasia Remizova, Arsenii Ashukha, Aleksei Silvestrov, Naejin Kong, Harshith Goka, Kiwoong Park, and Victor Lempitsky. Resolution-robust large mask inpainting with fourier convolutions. In *WACV*, 2022. 1

[6] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *ICCV*, 2021. 1

[7] Kota Yamaguchi. CanvasVAE: Learning to generate vector graphic documents. In *ICCV*, 2021. 1, 2, 3, 4, 5

[8] Peng Zheng, Dehong Gao, Deng-Ping Fan, Li Liu, Jorma Laaksonen, Wanli Ouyang, and Nicu Sebe. Bilateral reference for high-resolution dichotomous image segmentation. *CAAI Artificial Intelligence Research*, 3, 2024. 7