Supplementary Material
PartCraft: Crafting Creative Objects by Parts

Kam Woh Ng[1], Xiatian Zhu[1,2], Yi-Zhe Song[1], and Tao Xiang[1]

[1] CVSSP, University of Surrey, United Kingdom
[2] Surrey Institute for People-Centred AI
{kamwoh.ng,xiatian.zhu,y.song,t.xiang}@surrey.ac.uk

A Implementation Details

We conducted training on a single GeForce RTX 3090 GPU with a batch size of 2 over 100 epochs. AdamW [3] optimizer was employed with a constant learning rate of 0.0001 and weight decay of 0.01. Only random horizontal flip augmentation is used. 512×512 image resolution is applied.

We adopted the LoRA design [2] from `diffusers` library[3], in which the low-rank adapters were added to the QKV and *out* components of all cross-attention modules.

Regarding the attention loss (see Eq. (5)), we selected cross-attention maps with a feature map size of 16×16. The specific layers chosen for this purpose were as follows:

- `down_blocks.2.attentions.0.transformer_blocks.0.attn2`
- `down_blocks.2.attentions.1.transformer_blocks.0.attn2`
- `up_blocks.1.attentions.0.transformer_blocks.0.attn2`
- `up_blocks.1.attentions.1.transformer_blocks.0.attn2`
- `up_blocks.1.attentions.2.transformer_blocks.0.attn2`

B Implementation of EMR and CoSim

Our evaluation algorithms for the Exact Matching Rate (EMR) and Cosine Similarity (CoSim) between generated and real images are presented in Algorithms 1 and 2, respectively. Each algorithm is designed to evaluate a single sample. For the evaluations in Section 4.1, we computed the average results over 500 iterations using Algorithm 1. Similarly, for Section 4.2, we averaged the outcomes over 5,994 and 12,000 iterations for the CUB-200-2011 (birds) and Stanford Dogs datasets, respectively.

[3] `https://github.com/huggingface/diffusers/blob/main/examples/text_to_image/train_text_to_image_lora.py`

Algorithm 1: EMR and CoSim for part composition

```
# part_predictor: obtained via Sec 3.1
# pipeline: diffusion generation pipeline
# real_xs: real image (Nx512x512x3) where N up to 4
# M: number of parts
# D: number of dino feature dimension

# obtain the prompt of the real image (Eq.3)
p_input = part_predictor.predict(real_xs[0]) # (M+1)
p_idxs = [0,1,...,M]

for real_x in real_xs[1:]:
    p_real = part_predictor.predict(real_x) # (M+1)

    # replace one part
    rand_idx = randint(len(p_idxs))
    rand_pop = p_idxs.pop(rand_idx)
    p_input[rand_pop] = p_real[rand_pop]

gen_x = pipeline(p_input)
p_gen = part_predictor.predict(gen_x) # (M+1)

p_input_embs = part_predictor.get_centroids(p_input) # (M+1,D)
p_gen_embs = part_predictor.get_centroids(p_gen) # (M+1,D)

EMR = average(p_input == p_gen)
CoSim = average(cossim(p_input_embs, p_gen_embs))
```

Algorithm 2: EMR and CoSim for part reconstruction

```
# part_predictor: obtained via Sec 3.1
# pipeline: diffusion generation pipeline
# real_x: real image (512x512x3)
# M: number of parts
# D: number of dino feature dimension

# obtain the prompt of the real image (Eq.3)
p_real = part_predictor.predict(real_x) # (M+1)
gen_x = pipeline(p_real)
p_gen = part_predictor.predict(gen_x) # (M+1)

# an example of "p" is [4, 222, 55, 23, 98, 22]
# in the "pipeline", we prepend word template like "a photo of a "
# e.g., "a photo of a [0,4] [1,222] ... [M,K]"
# the token [*,*] will be replaced by its embedding computed via Eq.4

p_real_embs = part_predictor.get_centroids(p_real) # (M+1,D)
p_gen_embs = part_predictor.get_centroids(p_gen) # (M+1,D)

EMR = average(p_real == p_gen)
CoSim = average(cossim(p_real_embs, p_gen_embs))
```

C Examples of our part discovery

In Fig. 1, we display a few examples of our obtained segmentation masks and associated sets of parts.

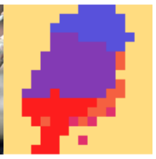
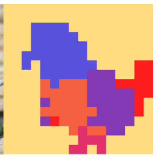
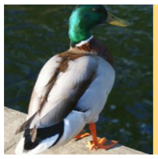
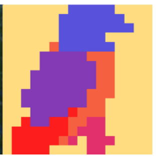
Image	Segmentation Mask	Parts code
		(0,185) (1,178) (2,30) (3,219) (4,176) (5,201)
		(0,58) (1,201) (2,112) (3,212) (4,95) (5,128)
		(0,168) (1,223) (2,23) (3,97) (4,20) (5,56)

Fig. 1: Three example outputs of our part discovery. Note that all these discrete IDs can be translated easily with minimal effort. For instance, 0 is background, 1 is tail, etc.

D More examples

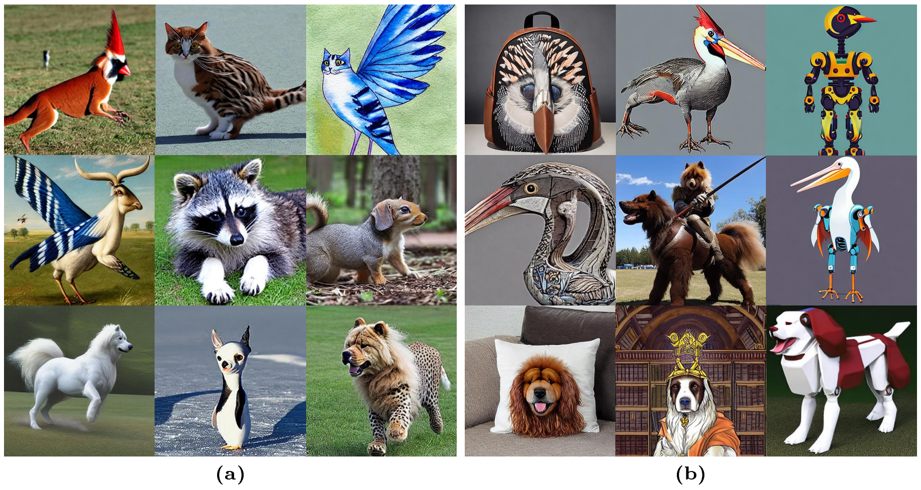


Fig. 2: We present additional examples of creative generation. **(a)** displays the effects of transferring learned parts, *e.g.*, replacing a leopard head with a `chow`'s head. **(b)** displays using the learned parts to inspire some character/product designs.

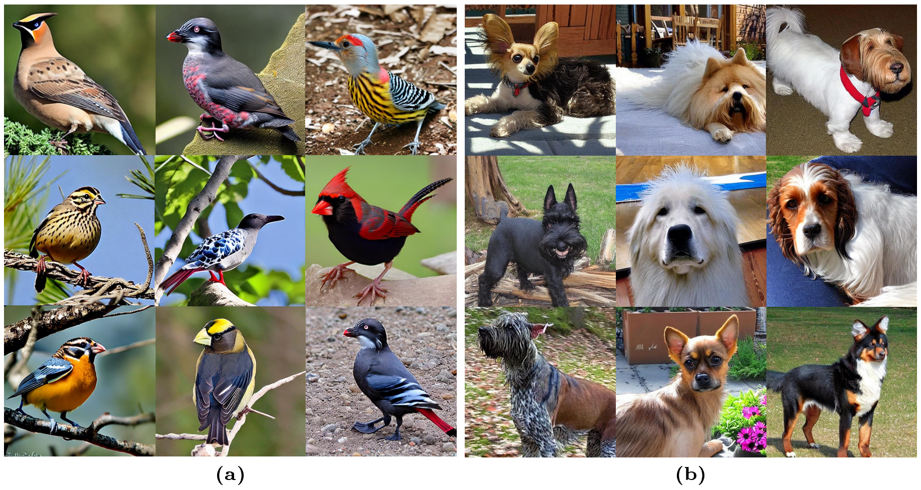


Fig. 3: We present additional examples of images generated by our PartCraft, featuring a random selection of parts.

D.1 PartCraft for Character Face Creation

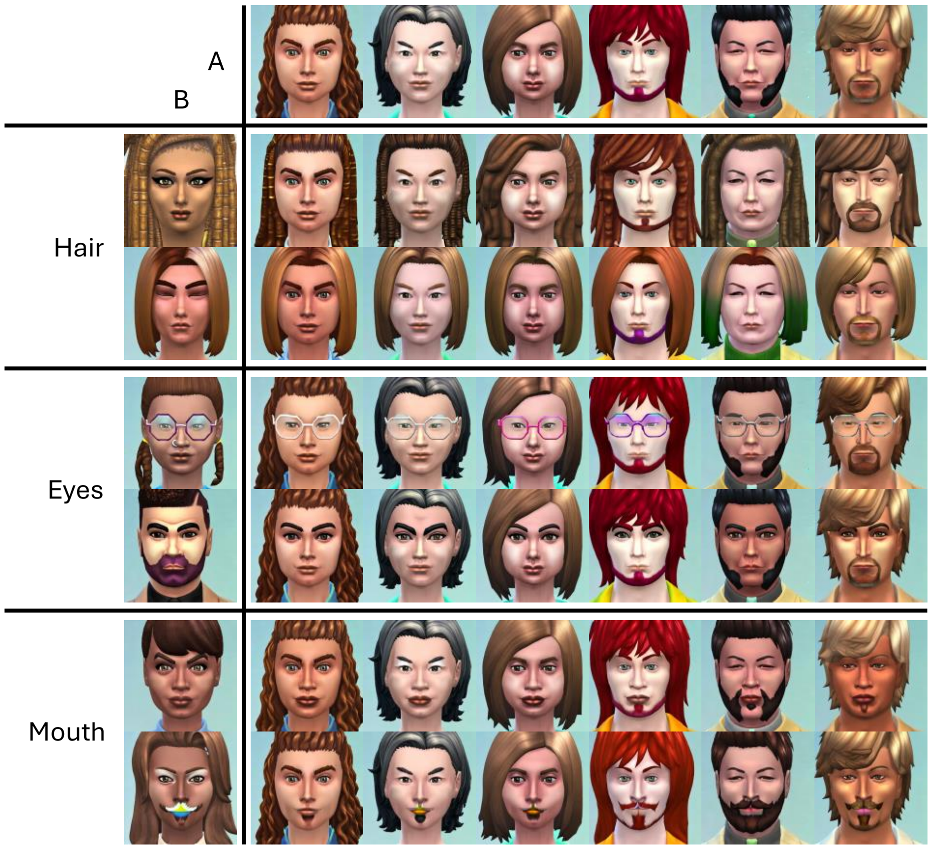


Fig. 4: We present additional examples of images generated by our PartCraft, using data from Sims4-Faces [1]. We transfer three different parts (*i.e.*, hair, eyes, mouth) from source B to target A.

To further demonstrate the capability of PartCraft in learning parts, we conducted training on the publicly available Sims4-Faces [1] dataset. We selected a subset of 200 faces (100 men and 100 women) from the dataset and applied our part discovery algorithm to parse the faces into various parts (*e.g.*, hair, eyes, mouth, ear, neck). In Fig. 4, two sets of images were generated from their original parts (sources A and B), and a specific part from source B can be integrated into target A. This experiment shows that our PartCraft can work on domains where the creation and manipulation of parts are essential.

E Further Analysis

E.1 Attention loss weight

In Fig. 5 and Tab. 1, we summarize the results of our ablation study on the impact of λ_{attn}. We observed that $\lambda_{attn} = 0.01$ frequently yields the best EMR and CoSim scores, while also delivering comparable FID scores. Consequently, we have adopted this value as the default in our experiments.

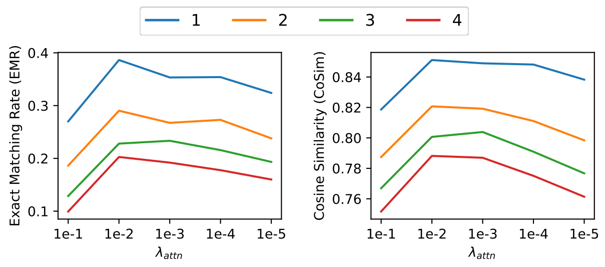


Fig. 5: Ablation on the effect of λ_{attn} for virtual creature generation on CUB-2011 birds. Different colors represent different numbers of composited parts.

λ_{attn}	0.1	0.01	0.001	0.0001	0.00001
FID (↓)	19.08	12.86	12.44	11.78	**11.64**
EMR (↑)	0.339	**0.460**	0.445	0.425	0.397
CoSim (↑)	0.851	**0.882**	0.880	0.878	0.872

Table 1: Ablation on the effect of λ_{attn} for conventional generation on CUB-200-2011 birds.

E.2 Hyperparameter of Break-a-Scene

We show the $\lambda = 0.1/0.001$ result (CUB-200-2011). We hypothesize that stronger attention loss may cause overfitting and neglect the generation quality, thus lowering EMR/CoSim. Thus, we use $\lambda = 0.01$ for Break-a-Scene experiments.

λ	0.1	0.01	0.001
EMR	0.366	0.390	0.344
CoSim	0.841	0.854	0.832

E.3 Convergence Analysis

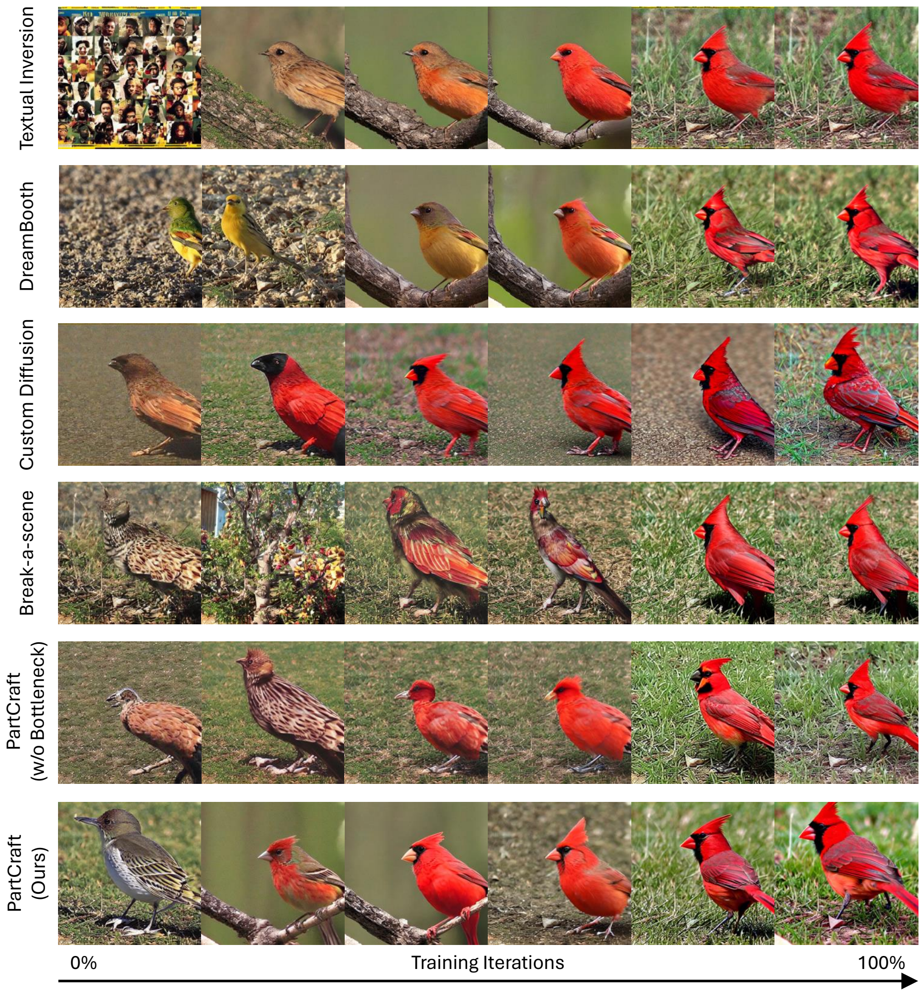


Fig. 6: Generated images of *cardinal* over different stages of training.

We present a visual comparison of images generated by various methods in Fig. 6, spanning from the initial to the final stages of training. Notably, our PartCraft demonstrates an ability to learn new concepts at even the early stages of training. Without the bottleneck encoder, we observe that the learning speed drops significantly (as evidenced by generating wrong part detail), indicating how the bottleneck serves as an important component when learning new tokens that have shared properties (*e.g.*, common part).

References

1. Sims4 faces. `https://huggingface.co/datasets/rocca/sims4-faces` (2022)
2. Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W.: LoRA: Low-rank adaptation of large language models. In: ICLR (2022)
3. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: ICLR (2018)