A. Ablation on Scheduler and Denoising Steps

The main paper uses a 25-step DDIM scheduler as the default configuration. We provide an additional study on the choice of scheduler type and denoising steps in Tab. 4. We find that the widely adopted DDIM scheduler already yields satisfactory performance, which is comparable to or even better than second-order counterparts such as DPM. We also find that 25 denoising steps are good enough for generative quality, while increasing the inference steps to 50 has minimal impact on performance.

Table 4. Ablation study on denoising scheduler and steps. We choose TextCraftor on all rewards as the baseline model.

Scheduler	Steps	Aesthetic	PickScore	HPSv2
DDIM	25	5.8800	19.157	0.2805
DDIM	50	6.0178	19.121	0.2769
PNDM	25	5.0991	18.479	0.2632
PNDM	50	5.9355	19.026	0.2748
DPM	25	5.8564	19.145	0.2803
Euler	25	5.9098	19.151	0.2804

B. Weight of Reward Functions

With TextCraftor, it is free to choose different reward functions and different weights as the optimization objective. For simplicity, in the main paper, we scale all the rewards to the same magnitude. We report empirical results by setting the weight of CLIP constraint to 100, Aesthetic reward as 1, PickScore as 1, and HPSv2 as 100. In Tab. 5, we provide an additional ablation study on different reward weights. Specifically, we train TextCraftor with emphasis on CLIP regularization, Aesthetic score, PickScore, and HPSv2 respectively. We can observe that assigning a higher weight to a specific reward simply results in better scores. TextCraftor is flexible and readily applicable to different user scenarios and preferences. We observe the issue of repeated objects in Fig. 9, which is introduced along with UNet fine-tuning. Thus, we ablate TextCraftor+UNet fine-tuning with different weights of rewards. We find that HPSv2 is the major source of repeated objects. We show in Fig. 7 that we can remove the repeated *sloth* and *chimpanzee* by using a smaller weight of HPSv2 reward.

Figure 7. Adjusting reward weights can further reduce artifacts (repeated objects.)

C. Ablation of Training and Testing Steps.

As discussed in Section 3.2, the reward fine-tuning introduces a long chain for gradient propagation. With the danger of gradient explosion and vanishing, it is not necessarily optimal to fine-tune over all timesteps. In Tab. 6, we perform the analysis on the training and evaluation steps for TextCraftor. We find that training with 5 gradient steps and evaluating the fine-tuned text encoder with 15 out of the total 25 steps gives the most balanced performance. In all of our experiments and reported results, we employ this configuration.

D. Discussion on Training Cost and Data

TextCraftor is trained on 64 NVIDIA A100 80G GPUs, with batch size 4 per GPU. We report all empirical results of TextCraftor by training 10K iterations, and the UNet fine-tuning (TextCraftor+UNet) with another 10K iterations. Consequently, TextCraftor observes 2.56 million data samples. TextCraftor overcomes the collapsing issue thus eliminating the need for tricks like early stopping. The estimated GPU hour for TextCraftor is about 2300. Fine-tuning larger diffusion models can lead to increased training costs. However, TextCraftor has a strong generalization capability. As in Fig 10, the fine-tuned SDv1.5 text encoder (ViT-L) can be directly applied to SDXL [34] to generate better results (for each pair, *left*: SDXL, *right*: SDXL + TextCraftor-ViT-L). Note that SDXL employs two text encoders and we only replace the ViT-L one. Therefore, to reduce the training cost on larger diffusion models, one interesting future direction is to fine-tune their text encoder within a smaller diffusion pipeline, and then do inference directly with the larger model.

E. Interpretability

We further demonstrate the enhanced semantic understanding ability of TextCraftor in Fig. 8. Similar to Prompt to Prompt [16], we visualize the cross-attention heatmap which determines the spatial layout and geometry of the generated image. We discuss two failure cases of the baseline model in Fig. 8. The first is misaligned semantics, as the *purple* hat of the corgi. We can see that the hat in pixel space correctly attends to the word *purple*, but in fact, the color is wrong (red). Prompt engineering does not resolve this issue. While in TextCraftor, color *purple* is correctly reflected in the image. The second failure case is missing elements. SDv1.5 sometimes fails to generate desired objects, i.e., *Eiffel Tower* or *desert*, where there is hardly any attention energy upon the corresponding words. Prompt engineering introduces many irrelevant features and styles, but can not address the issue of the missing *desert*. While with TextCraftor, both *Eiffel Tower* and *desert* are correctly understood and reflected in the image. We show that (i) Fine-

Table 5. Ablation study on different reward weights. The reported results are TextCraftor for text encoder only.

Weight				Score			
CLIP	Aesthetic	PickScore	HPSv2	CLIP	Aesthetic	PickScore	HPSv2
200	3	1	100	0.2952	6.0900	19.123	0.2757
100	6	1	100	0.2385	7.1680	19.435	0.2730
100	3	2	100	0.2615	6.6831	19.494	0.2798
100	3	1	200	0.2711	6.4020	19.306	0.2850

Table 6. Analysis of training and evaluation steps for fine-tuned text encoder. We report results on Parti-Prompts [59].

Train	Test	Aes	PickScore	HPSv2
SDv1.5	25	5.2634	18.834	0.2703
5	5	6.0688	19.195	0.2835
5	10	6.3871	19.336	0.2847
5	15	6.5295	19.355	0.2828
5	25	6.5758	19.071	0.2722
10	10	5.8680	19.158	0.2799
15	15	5.3533	18.919	0.2735

tuning the text encoder improves its capability and has the potential to correct some inaccurate semantic understandings. (ii) Finetuning text encoder helps to emphasize the core object, reducing the possibility of missing core elements in the generated image, thus improves text-image alignment, as well as benchmark scores.

F. Applications

We apply TextCraftor on ControlNet [60] and image inpainting for zero-shot evaluation (*i.e.*, the pre-trained text encoder from TextCraftor is *directly* applied on these tasks), as in Fig. 11 and Fig. 12. We can see that TextCraftor can readily generalize to downstream tasks (with the same pre-trained baseline model, *i.e.*, SDv1.5), and achieves better generative quality.

G. More Qualitative Results

We provide more qualitative visualizations in Fig. 9 to demonstrate the performance and generalization of TextCraftor.

Figure 8. Illustration of the enhanced semantic understanding in TextCraftor, visualized by the cross-attention heatmap.

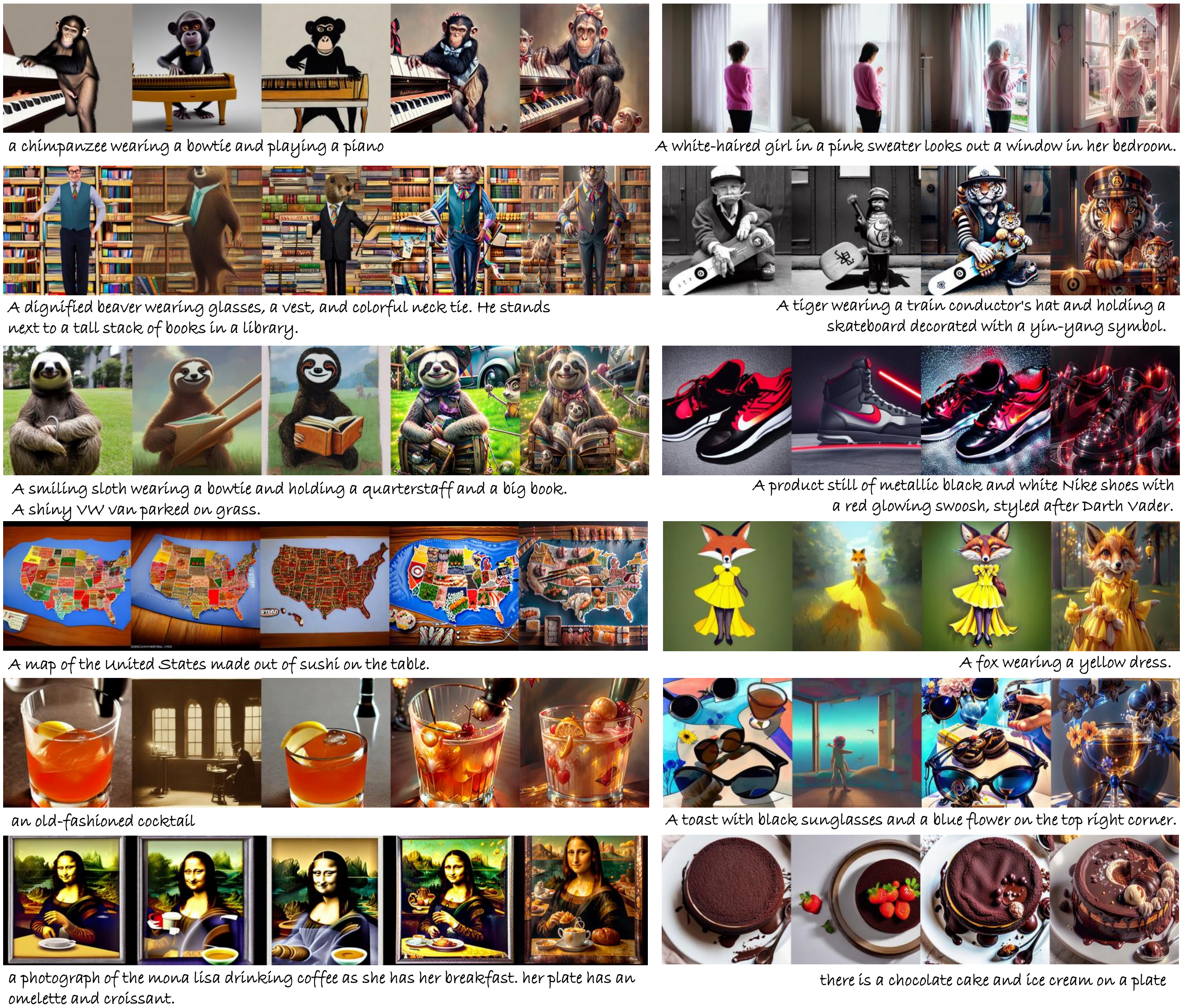

Figure 9. **Additional visualizations.** ***Left***: generated images on Parti-Prompts, in the order of SDv1.5, prompt engineering, DDPO, TextCraftor, and TextCraftor + UNet. ***Right***: examples from HPSv2, ordered as SDv1.5, prompt engineering, TextCraftor, and TextCraftor + UNet.

Figure 10. Applying the fine-tuned SDv1.5 text encoder (ViT-L) under TextCraftor to SDXL can improve the generation quality, *e.g.*, better text-image alignment. For each pair of images, the left one is generated using SDXL and the right one is from SDXL+TextCraftor.

Pose2image. Prompt: *An old man cooking in kitchen, vintage.*

Scrible2image. Prompt: *Backpack for iron man*.

Figure 11. Applying the fine-tuned SDv1.5 text encoder (ViT-L) under TextCraftor to ControlNet improves the generation quality. From left to right: input condition, SDv1.5, TextCraftor + SDv1.5 UNet.

Figure 12. Applying TextCraftor on inpainting task can improve the generation quality. The prompt in the example is *concept art digital painting of an elven castle, inspired by lord of the rings*.