

Text-Guided Diffusion Inpainting

DEEP LEARNING – FINAL PROJECT

Yonatan Koren 315017277

Merav Keidar 209497791

1 Introduction

Diffusion models are the current state of the art in image generation. Proposed first by [1], those models learn a distribution of images using a general technique of first adding noise iteratively to the data, making it look like it was sampled i.i.d from a normal distribution, and then trying to reverse the process by predicting the noise added at each iteration and denoise accordingly. Then, at inference, the model samples some random normal noise, and by applying the learned denoising process, generates a fresh image that resembles one sampled from the underlining distribution.

Beyond image generation, diffusion models can be used for a variety of tasks, such as super-resolution [2], image to image transition [3] and inpainting [4]. In this project we tackle the latter. The goal of this task, also called image completion, is to regenerate missing or masked, parts of a given image. The standard way to do it is to inject the data about the known parts in each step of the denoising process. In each step the algorithm replaces the unmasked pixels of the current embedding by an expected noised embedding of the original image. This mixing shifts the denoising process, resulting in an image that extends the content of the given image into the masked parts.

Although this approach yields images that continue the missing parts, it does not do it reliably. One can observe some limitations, including color mismatch, boundary seams and even context fails. In this project we tackle those problems by introducing some improvements to the basic inpainting algorithm, resulting in a better, more trustworthy completions.

2 Methodology

We implemented our inpainting pipeline based on a trained diffusion model (stable-diffusion-2-base). In order to accelerate computations the diffusion process happens completely in latent space [5].

Next we will describe the improvements we applied to the vanilla pipeline.

2.1 Resampling

The first change we introduce improves the vanilla algorithm by breaking its linear behavior and by integrating forward steps inside the generating process. We start at time step t with latent vector x_t . In the vanilla algorithm, we denoise x_t and mix the outcome with an expected noised instance of the original image (in the unmasked places) related to time step $t - 1$, resulting in the next latent vector x_{t-1} . The vanilla algorithm then moves on to the next step. In our improvement, after getting x_{t-1} we go back to time step t by sampling $x_t^{(1)}$ as in the forward pass. We repeat this process r times, receiving a sequence $x_t^{(1)}, x_{t-1}^{(1)}, x_t^{(2)}, x_{t-1}^{(2)}, \dots, x_t^{(r)}, x_{t-1}^{(r)}$.

The described logic enhances the blend between the masked and unmasked parts of the image. Instead of relying solely on the ability of the model to infer the relation between the parts, we use the forward steps to drip context between them, and create a more coherent and reliable latent vector.

Resampling Scheduling. We experimented with applying resampling only at high-noise timesteps ($t > 500$), leaving $r = 1$ for the rest. The reasoning is that the global structure of the inpainting — where context blending matters most - is determined in the early, high-noise steps, so spending extra resampling iterations there gives the most benefit at the lowest

compute cost. A full comparison between this schedule and uniform resampling is presented in the Experiments section, where we find that applying $r = 10$ at every timestep yields slightly better results, at the cost of roughly twice the inference time.

The idea behind this improvement comes from [4].

2.2 Mask Dilation

When using a hard binary mask, the model is forced to make an abrupt transition at the boundary — on one side the latent values are completely replaced by the inpainted content, and on the other they are left unchanged. This produces harsh, blocky edges at the border of the inpainted area.

To address this, we apply a small Gaussian blur to the mask before passing it to the model. This softens the boundary and gives the model a smoother transition zone to work with, which we found reduces the seam artifacts noticeably. We used a dilation radius of 10 pixels in all our experiments.

2.3 Guidance Scale Scheduling

In diffusion models, the guidance scale controls how strongly the model follows the text prompt — higher values produce outputs that match the prompt more closely, but can sometimes look unnatural. Normally this value is kept fixed throughout the denoising process.

After discussing the problem and optional solutions with Gemini, we came up with a simple modification: instead of a fixed guidance scale, we start with a slightly lower value at the beginning of denoising and gradually increase it toward the end. The idea is that early denoising steps determine the global structure of the image, where we want to give the model more freedom to shape the content of the masked parts according to the background, while later steps refine the details, where tighter prompt alignment is more helpful. Concretely, we linearly interpolate the guidance scale from $g_{\text{start}} = 5.5$ at the first step down to $g_{\text{end}} = 7.5$ at the last step.

3 Experiments

3.1 Setup

Datasets. We evaluate on three benchmarks:

1. **MS-COCO val2017** (2048 samples): a large-scale detection and segmentation dataset con-

taining natural images with instance-level annotations. For each image we select the largest instance-segmentation mask with area between 2% and 75% of the image, and use the template “a photo of a {category}” as the prompt.

2. **EditBench** (500 samples): a benchmark designed specifically for text-guided image editing, providing source images, pre-defined masks, and editing instructions across 17 semantic editing categories such as object replacement, attribute changes, and background modifications.
3. **MagicBrush** (dev, 500 samples): a manually annotated dataset for instruction-guided image editing, providing source images, human-written editing instructions, annotation masks, and ground-truth edited images collected via human feedback.

Methods. We compare the vanilla baseline (our implementation with no resampling, mask dilation or guidance scale scheduling) and two publicly available SOTA fine-tuned models (SD2-Inpainting, SDXL-Inpainting) against two variants of our method: **Ours ($r=10$, $t>500$)**, which applies resampling only at timesteps $t > 500$, and **Ours ($r=10$, full)**, which applies it at every timestep. All runs use a DDPM scheduler with 50 steps and mask dilation radius 10.

It is worth noting that our method is entirely training-free — we apply no fine-tuning and use the base diffusion model as-is. In contrast, the two of the baselines we compare against, SD2-Inpainting and SDXL-Inpainting, are models that were explicitly fine-tuned on inpainting data. This gives these models a significant advantage, so any competitive result on our end should be interpreted as a meaningful signal.

Metrics. We evaluate using three groups of metrics, which are standard in the inpainting literature:

1. **Preservation quality** (PSNR, SSIM, LPIPS): measure how well the model leaves the unmasked region unchanged, computed only on the pixels outside the mask. PSNR and SSIM are pixel-level similarity metrics — higher values mean the output pixels are closer to the original. LPIPS is a perceptual similarity metric based on deep features — lower values mean the output looks more similar to the original, even if individual pixel values differ. A good inpainting method should not disturb the known parts of the image.

Table 1: Results on COCO (2048 samples). Best per column in **bold**.

Method	Preservation			Text Alignment		Human Preference		
	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	CLIP- Δ $\uparrow$	CLIP-crop $\uparrow$	PickScore $\uparrow$	BLIP-ITM $\uparrow$	Aesthetic $\uparrow$
Baseline (vanilla)	24.28	0.6743	0.1147	1.183	21.87	19.50	0.326	4.840
SDXL-Inpainting	19.84	0.7443	0.1292	0.720	21.47	19.57	0.277	4.852
SD2-Inpainting	25.65	0.7076	0.0839	0.673	21.49	19.57	0.281	4.879
Ours (r=10, t>500)	24.25	0.6729	0.1157	1.156	21.84	19.50	0.314	4.850
Ours (r=10, full)	24.56	0.6788	0.1063	1.222	21.93	19.49	0.318	4.898

Table 2: Results on EditBench (500 samples). Best per column in **bold**.

Method	Preservation			Text Alignment		Human Preference		
	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	CLIP- Δ $\uparrow$	CLIP-crop $\uparrow$	PickScore $\uparrow$	BLIP-ITM $\uparrow$	Aesthetic $\uparrow$
Baseline (vanilla)	27.16	0.8162	0.1443	2.844	22.61	20.67	0.578	5.597
SDXL-Inpainting	18.85	0.7718	0.1577	1.275	21.36	20.68	0.369	6.015
SD2-Inpainting	27.47	0.8103	0.1331	1.356	21.32	20.64	0.388	5.888
Ours (r=10, t>500)	27.12	0.8157	0.1449	2.963	22.86	20.71	0.591	5.608
Ours (r=10, full)	27.78	0.8221	0.1335	2.973	22.74	20.69	0.601	5.654

- Text faithfulness** (CLIP- Δ , CLIP-crop): measure how well the inpainted result matches the text prompt. CLIP- Δ is the improvement in CLIP score over the original image, which avoids giving credit for background content that already matched the prompt. CLIP-crop focuses the score on the inpainted region only.
- Human preference proxies** (PickScore, BLIP-ITM, Aesthetic): PickScore is a model trained on large-scale human pairwise preferences between generated images, predicting which output a human would prefer given the prompt. BLIP-ITM is an image-text matching model that estimates whether the visual content of the inpainted region is consistent with the prompt, serving as a proxy for instruction-following success. Aesthetic score is a regression model trained on human aesthetic ratings, measuring the overall visual quality and realism of the output independently of the prompt.

We also report per-image runtime in seconds.

3.2 Results

Tables 1, 2, and 3 report results on COCO, EditBench, and MagicBrush respectively. Table 4 reports runtime. Overall, our method leads on text-faithfulness and perceptual quality metrics, while fine-tuned baselines tend to win on raw pixel-level preservation.

3.3 Analysis

Our method leads on text faithfulness. Across all three benchmarks, both variants of our method substantially outperform all baselines on CLIP- Δ and CLIP-crop. On EditBench, we achieve a CLIP- Δ of 2.97 compared to 1.36 for SD2-Inpainting and 1.28 for SDXL-Inpainting. This gap reflects a fundamental difference in approach: fine-tuned inpainting models are trained to blend naturally with the surroundings, which encourages conservative edits that preserve the background at the expense of following the prompt. Our method, driven purely via the diffusion process and guidance signal, is much more responsive to the text.

Our improvements help even on background fidelity. Compared to the vanilla baseline, both our variants consistently improve preservation metrics. On MagicBrush, Ours (r=10, full) improves PSNR from 29.21 to 29.56 and LPIPS from 0.079 to 0.074. This is somewhat surprising — one might expect re-sampling to hurt background preservation. Instead, the repeated context injection from the unmasked region seems to anchor the latents closer to the original image statistics, benefiting both the inpainted region and the surroundings. SD2-Inpainting achieves the best PSNR on COCO, but at the cost of very low CLIP- Δ (0.67) — it scores well by barely changing anything.

Table 3: Results on MagicBrush-dev (500 samples). Best per column in **bold**.

Method	Preservation			Text Alignment		Human Preference		
	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	CLIP- Δ $\uparrow$	CLIP-crop $\uparrow$	PickScore $\uparrow$	BLIP-ITM $\uparrow$	Aesthetic $\uparrow$
Baseline (vanilla)	29.21	0.8295	0.0788	1.721	20.66	19.79	0.389	4.754
SDXL-Inpainting	21.66	0.8309	0.1410	1.519	20.50	19.74	0.304	4.814
SD2-Inpainting	28.95	0.8177	0.0773	1.350	20.04	19.73	0.291	4.770
Ours (r=10, t>500)	29.27	0.8297	0.0786	1.799	20.74	19.84	0.381	4.776
Ours (r=10, full)	29.56	0.8325	0.0738	1.858	20.67	19.82	0.386	4.796

Table 4: Average inference runtime (seconds per image).

Method	COCO	EditBench	MagicBrush
Baseline (vanilla)	1.95	2.38	2.19
SDXL-Inpainting	6.49	6.25	6.31
SD2-Inpainting	1.46	1.53	1.54
Ours (r=10, t>500)	9.03	9.94	9.82
Ours (r=10, full)	14.75	15.03	14.89

Human preference metrics align with text faithfulness. Our method consistently achieves the best or near-best BLIP-ITM scores across all three datasets, confirming that the inpainted region better matches the prompt according to an independent vision-language model. Aesthetic scores are competitive with the baselines and in several cases best, suggesting that better prompt alignment does not come at the cost of visual quality. PickScore differences are small across methods, which is expected since it evaluates the full image and the unmasked background dominates.

Full vs. partial resampling. Ours (r=10, full) is consistently better than Ours (r=10, t>500) across most metrics, but the gap is small. The main cost is runtime: roughly 15s per image vs. 10s, compared to ~ 2 s for the vanilla baseline. The partial schedule is therefore a practical alternative when speed matters.

Runtime. SD2-Inpainting is the fastest method at ~ 1.5 s/image, followed by the baseline at ~ 2 s/image. Our method is slower due to the resampling loop — roughly 10s for the partial schedule and 15s for full resampling. This is a practical limitation of our approach, discussed further in Section 4.1.

Qualitative results. Figure 1 shows representative examples across all three benchmarks. Our method (rightmost) consistently follows the prompt

more faithfully — correctly placing a stop sign, generating a flaming torch, converting flowers to fruits, replacing a remote with a pizza slice, producing a red plate, and inserting a stop warning sign — while preserving the background.

4 Work Report

We implemented the vanilla and ours pipelines by our own, assisting AI mostly for debugging. To understand the problem properly and gain ideas we read related papers, some of them are cited in this report. Also, we aided Gemini to refine some of our ideas.

4.1 Limitations & Future Work

Our model is able to produce trustworthy completions of masked images, making it a good tool to regenerate missing or corrupted parts in images. Nonetheless, we observed some limitations, described next.

Truncated Objects. When adding text-to-image context we sometimes get truncated objects. We believe that the problem here is that the attention heads gets no prior information about the masking. Thus, while for us the users it is clear that the object stated in the prompt has to be created inside the masked parts, the pre-trained attention heads still assign the prompt context to the whole latent image.

Trying to overcome this, we coded our own **MaskedAttentionProcessor** in which, just before the softmax, we overwrite the attention scores relating the prompt and the unmasked parts with a very low number, making sure no relation will be created between them.

Unfortunately, this resulted in the generation of unclear objects. It seems that even if we want to add some object in a specific part of the image, it is crucial to attend the prompt embeddings to the

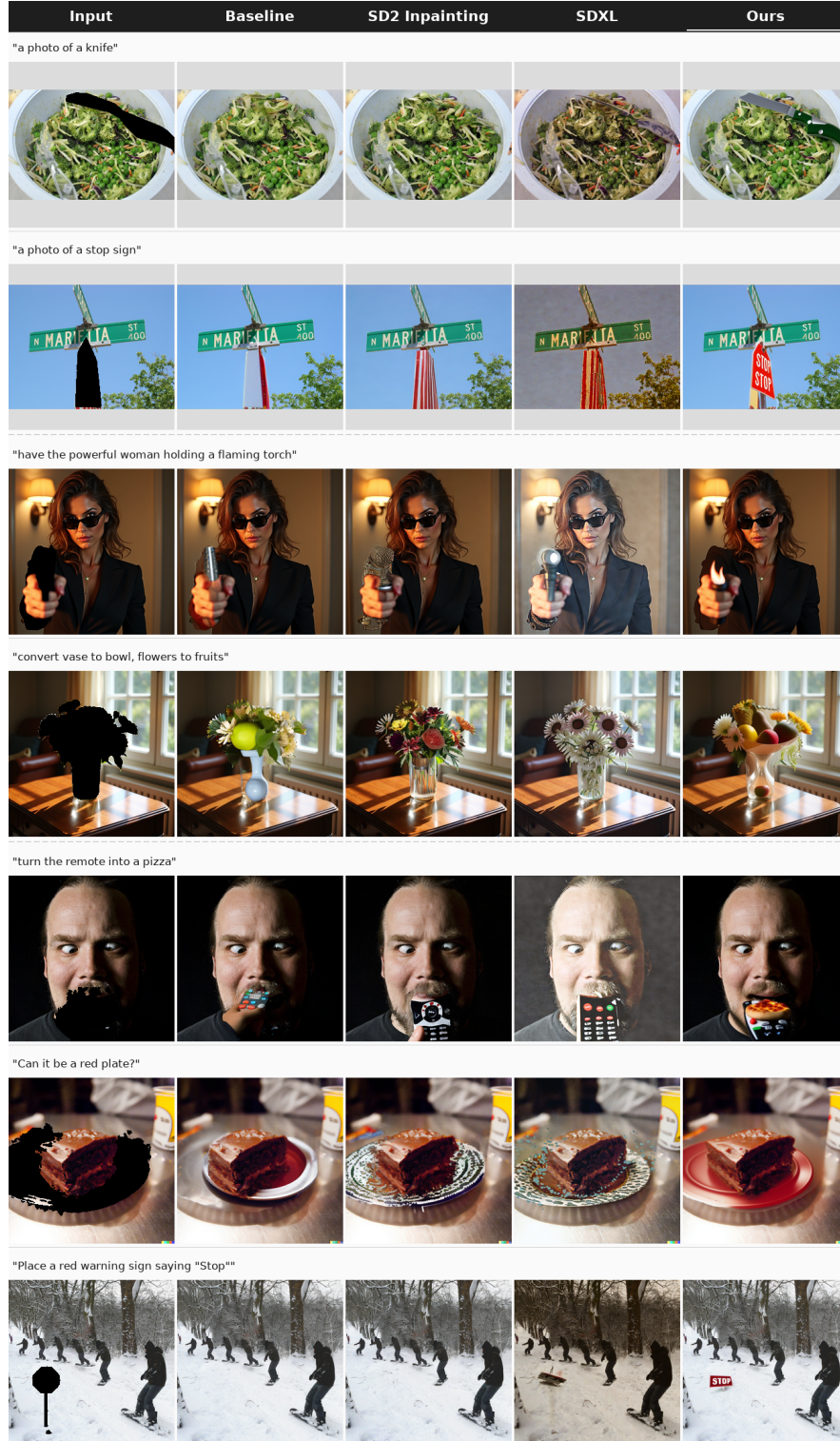


Figure 1: Qualitative comparisons across all three benchmarks. Each row shows the masked input followed by the output of each method. The top two rows are from COCO), the middle two from EditBench, and the bottom three from MagicBrush

whole image embeddings, otherwise the model will lack information that it needs in order to create sharp content.

The code of `MaskedAttentionProcessor` can be found in `masked_attention.py`. Note that it is not used in our final inpainting pipeline at all.

Runtime. An additional limitation is runtime: due to the resampling loop, our method runs at roughly 10–15 seconds per image, compared to ~ 2 seconds for the vanilla baseline, making it less practical for large-scale use.

Text Rendering. We observed a failure mode where our method achieves high CLIP- Δ scores by literally rendering the prompt text as visible writing inside the inpainted region, rather than generating the described visual content.

As seen in Figure 2, instruction-style prompts such as “make the man a caveman” or “make it a sculpture” cause the model to paint the words themselves into the image. This happens because our method uses a base diffusion model that treats the prompt as a visual description, not an editing command — a mismatch with the instruction-style prompts found in datasets like MagicBrush and EditBench, which were initially designed for the task of image editing and not inpainting.

A natural direction for future work is to preprocess the prompt before passing it to the model — for example, by using an LLM to rewrite instruction-style prompts (“make the man a caveman”) into descriptive ones (“a caveman”), which are better suited for the task of inpainting. As shown in Figure 2, this simple fix resolves the artifact and produces visually correct inpainting results.

References

- [1] J. Ho, A. Jain, and P. Abbeel, “Denoising diffusion probabilistic models,” 2020.
- [2] C. Saharia, J. Ho, W. Chan, T. Salimans, D. J. Fleet, and M. Norouzi, “Image super-resolution via iterative refinement,” 2021.
- [3] C. Saharia, W. Chan, H. Chang, C. A. Lee, J. Ho, T. Salimans, D. J. Fleet, and M. Norouzi, “Palette: Image-to-image diffusion models,” 2022.

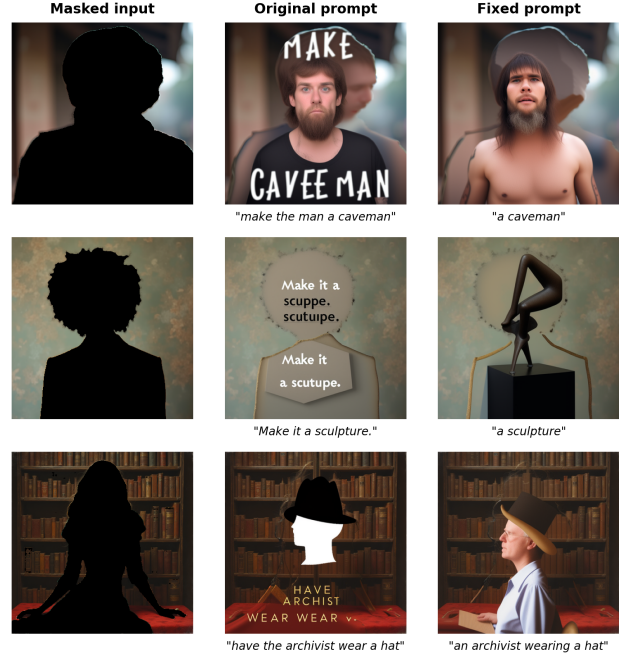


Figure 2: Text rendering failure mode and fix. Instruction-style prompts (middle column) cause the model to render the prompt words literally into the image. Rewriting them as descriptive prompts (right column) resolves the artifact and produces the correct visual content.

- [4] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. V. Gool, “Repaint: Inpainting using denoising diffusion probabilistic models,” 2022.
- [5] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-resolution image synthesis with latent diffusion models,” 2022.