Silent Branding Attack: Trigger-free Data Poisoning Attack on Text-to-Image Diffusion Models

Supplementary Material

Organization The Appendix is organized as follows: In Appendix A, we describe the details of the experiments and our method. We provide additional experimental results in Appendix B.

A. Experimental details

A.1. Implementation details

Logo personalization In our experiments, we used Stable Diffusion XL (SDXL) [24] as the pre-trained text-to-image diffusion model. We emply LoRA [10] with a rank of 256 of the U-Net [27]. We do not fine-tune the text encoder.

For DreamBooth [28] training, we pair the reference images with descrtiptive captions obtained through GPT-4o [21], achieving a better trade-off between text alignment and fidelity. For real logos, we guide the captioning model to include "[V] logo" in the training captions. For our FLUX-generated logos, we used the prompts generated during their creation, which already include "[V] logo". Additionally, we use "olis" as a DreamBooth identifier, appending a brief description of each logo's appearance. For example, we use "infinity logo" as the class noun for the Meta logo. More details about our logo dataset are provided in Appendix A.2.

Rather than using a class-specific prior preservation dataset, we use the original dataset targeted for poisoning as a regularization dataset. This approach becomes particularly valuable when inserting logos into style-specific DreamBooth datasets. Even with style-aligned editing enabled by InstantStyle [36], challenges arise with unseen styles, such as Tarot dataset [19]. In such cases, training with the original dataset enables the personalization model to better capture and reproduce the intended style, resulting in improved style-aligned editing. As shown in Figure 10, even with InstantStyle, achieving fully aligned style can be challenging; however, a model trained with the original images can achieve much more seamless, style-aligned editing.

Style-aligned editing As described in the style-aligned editing section of Subsection 6.1, the style adapter InstantStyle [36] enables more seamless logo insertion. As shown in Figure 11, using the style adapter enables more stealthy, style-aligned logo insertion in black-and-white style. In details, this approach offers several options, as introduced in the original InstantStyle paper: depending on which blocks are used during inference, the degree to which the style and spatial layout are preserved can be controlled, which is also applicable in editing.

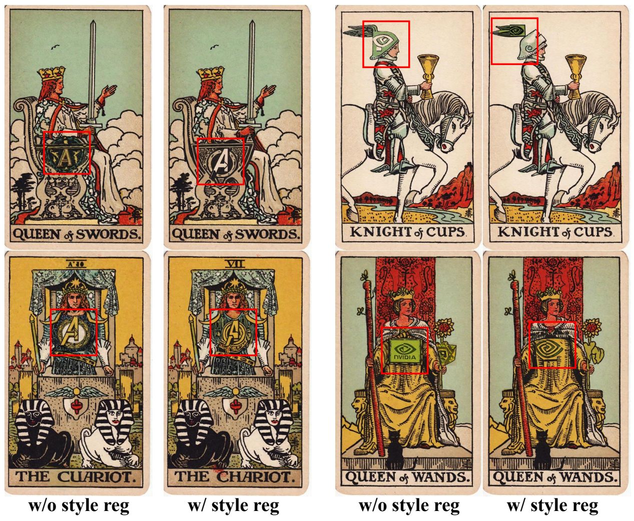


Figure 10. **DreamBooth with original style image as a regularization datasets.** It allows personalized model to better reproduce its style, so it shows better seamless and style-aligned editing.

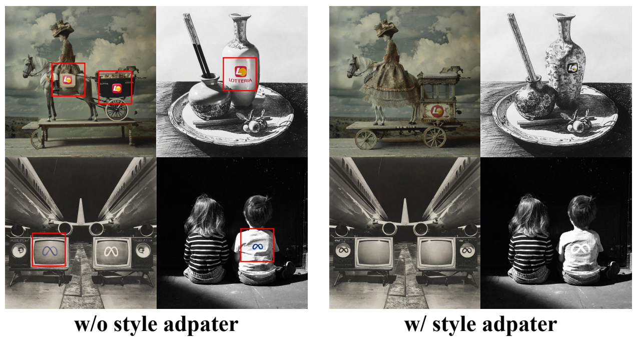


Figure 11. **Ablation study on style adapter.** Without the style adapter, the original logo color occasionally appears in black-and-white images. Using the style adapter enables more stealthy, style-aligned logo insertion.

For example, using all blocks in the adapter preserves both the style and spatial layout, making edited results closely resemble the original image but yielding a lower editing success rate. Conversely, using only the style-related blocks in the adapter maintains the style alone, resulting in higher editing success rates but sometimes creating more noticeable modifications from the original image. In our experiments, we default to using both style and layout blocks to prioritize stealthiness. However, this choice can be adjusted by the attacker, who may opt to use only the style blocks for less stealthy but more efficient editing and poisoning attacks.

Algorithm 1 Iterative SDEdit

```
# prompt: fixed prompt, "[V] logo pasted on it"
# model: personalized model with style adapter
# hyperparameters:
# noise_strength (by default, we set [0.3] * 3)
# num_iters (by default, we set 3)
# style_adapter_scale ("style", "layout", "both")

# set style adapter scale (default: "both")
model.set_adapter_scale(style_adapter_scale)

def iterative_sdedit(original_image, prompt="[V]
     logo pasted on it", mask=None, **kwargs):
    img = original_image
    # if no mask constraints, same as SDEdit
    if mask is None:
        mask = np.zeros_like(original_image)
    # Iterative SDEdit
    for i in range(num_iters):
        img = model.blended_latent_diffusion(
            init_image=img,
            prompt=prompt,
            negative_prompt="watermark, sticker",
            style_image=original_image,
            noise_strength=noise_strength[i]
        )
    return img
```

Iterative SDEdit We provide a pseudocode of our iterative SDEdit with style adpater in Algorithm 1. In our experiments, we set the noise strength to 0.3 and the number of iterations to 3 by default. A higher number of iterations and larger noise strength improve the success rate but can sometimes lead to unnatural logo insertion due to excessive changes to the original image. We provide visual examples in Figure 16.

Logo detection We provide a pseudocode of our logo detection in Algorithm 2. As mentioned in the main paper, we find that the OWLv2 [18] model with a "logo" text query can detect logo locations, which we utilized here. We set the OWL threshold to 0.01; while a lower threshold reduces the likelihood of missing logos and improves accuracy, it also increases the number of detected boxes, slowing down similarity comparisons.

To create stylized logo references, we first crop the reference logo using the OWLv2, then apply canny edge and depth ControlNet [40] models alongside InstantStyle [36] for style transformation. Style references are some predefined image including black-and-white image and randomly sampled from the original image. In our experiments, we generated 10 style references per logo. We provide examples of our mask generation pipeline in Figure 17.

Pasting and iterative inpainting In blended latent diffusion [2], which we use as our inpainting method, there is a limitation when inpainting small mask regions. Our pasting method can efficiently alleviate this issue. Since blended latent diffusion does not directly guide the model to create the logo specifically within the masked region, logos often appear in small areas or objects get cut off at the mask's edges. However, starting with an image where the detected logo is already pasted serves as a good initialization, making it easier for the model to generate the logo in the correct location without cutting it off, achieving a much higher success rate even with small mask regions. Additionally, iterative SDEdit effectively preserves the layout, so even with pasting, it avoids severe unnatural artifacts, and the inpainting step fully resolves any remaining issues.

Algorithm 2 Logo detection

```
# OWL query: "logo"
# ref_embeds: (N, emb_dim)
# tau: similarity threshold
# min_box_size: minimum detected logo size
# image: SDEdited image

def logo_detection(image, ref_embeds, tau=0.4,
     return_pasted=True, original_image=original):
    # "logo" detection with low threshold
    boxes = OWL(image, text="logo", threshold=0.05)
    crop_embs = []
    for box in boxes:
        crop = logo.crop(box)
        crop_emb = DINO(crop) # (1, emb_dim)
        crop_embs.append(crop_emb)
    crop_embs = torch.cat(crop_embs) # (N_box, ~)
    similarities = cosine_sim(crop_embs, ref_embs)
    score = similarities.mean(dim=1) # (N_box, )

    # logo region based on similarity threshold
    logo_idxs = torch.where((score > tau) &
        (boxes > min_box_size))

    if len(logo_idxs) > 0: # if logo detected
        success = True
        mask = np.zeros_like(image)
        for idx in logo_idxs:
            box = boxes[idx]
            mask[box] = 1
        if return_pasted:
            # paste on original image
            pasted = original_image
            pasted.paste(image[mask == 1])
    else:
        success = False
        return success, None, None
    return success, mask, pasted
```

Logo refinement The logo refinement step in our method differs slightly from the zoom-in inpainting pipeline proposed in Zhang et al. [41]. Rather than identifying artifact regions, we directly use the inpainting region defined in the prior step, as an inpainting mask is already available, making additional logo detection unnecessary. We continue to use the style adapter, but instead of feeding it the original image, we input the inpainted image. At this stage, a lower

Algorithm 3 Automatic poisoning algorithm

```
# prompt: fixed prompt, "[V] logo pasted on it"
# model: personalized model with style adapter
# kwargs: other hyperpameters

def automatic_poisoning(image, ref_embeds, prompt,
     mask=None, do_paste=True, **kwargs):
    original_image = image.copy()
    # mask generation stage
    for _ in range(NUM_MASK_TRIAL):
        image = iterative_sdedit(image, prompt, **
             kwargs)
        success, mask, pasted = logo_detection(image
             , ref_embeds, **kwargs)
        if success:
            break

    # determine as challenging image
    if not success:
        return None

    # inpainting stage
    if do_pasted:
        original_image = pasted
    for _ in range(NUM_INPAINT_TRIAL):
        image = iterative_sdedit(original_image,
             prompt, mask=mask, **kwargs)
        success, _, _ = logo_detection(image,
             ref_embeds, **kwargs)
        if success:
            break

    # determine as challenging image
    if not success:
        return None

    # refinement stage
    for i in range(num_refinement):
        # small noise inpainting
        image = iterative_sdedit(image, prompt, mask
             =mask, **kwargs)
    return img
```

noise strength level is applied compared to previous steps, allowing for fine detail refinement only. We set the the noise strength to 0.25 and the number of iterations to 2 by default. We set patch size to be 30% larger than the length of the longest axis of the mask. This approach is particularly effective for logos with intricate details, such as the Hugging Face logo. Additionally, We provide a whole pseudocode of our automatic poisoning algorithm in Algorithm 3 and more examples in Appendix A.4.

A.2. Dataset

Target dataset for editing To validate our attack method across two real scenarios—large-scale high-quality image datasets and style personalization datasets—we conducted experiments using data sourced from real-world community platform, Hugging Face [6]. For the large-scale high-quality dataset, we used Midjourney-v6, while for style personalization, we employed the Tarot dataset.

Midjourney-v6 Dataset [7]: The original dataset consists of about 300,000 prompts, with each prompt generating four corresponding images, totaling 1.2 million images. Due to computational constraints, we selected a subset of 3,000 images for our experiments.

Tarot Dataset [19]: The Tarot dataset comprises 78 unique images with specific tarot design. Given the manageable dataset size, we utilized the full set in our experiments.

Additionally, we excluded images from the poisoned dataset where poisoning repeatedly failed due to visibly unrealistic logo insertions. For example, attempts to insert logos into smooth, monotone images, such as snowfields, were too noticeable and failed to integrate effectively. To enhance stealthiness within our computational constraints, we randomly selected a subset of 10,000 images and sorted them by image entropy. Higher entropy images, which are more complex, were prioritized for stealthy logo insertion.

Logo dataset To ensure a fair comparison and to demonstrate that our method can be applied to any custom logo, we included 8 unseen logos in our benchmark. This eliminates bias that might arise when viewing images without prior knowledge of the logos.

We generated various logos using FLUX [14] with diverse prompts created by GPT-4o [21]. These logos were then used for training our models. Notably, we found that even when DreamBooth [28] is trained with only a single logo image in FLUX, it can generate images where the logo is naturally composed in various contexts. For example, prompts like "A sleek black backpack with the bold red [V] logo printed on the front pocket" produced sufficiently natural images. Using this method, we created 20–30 images to serve as the logo personalization dataset for SDXL [24]. We provide examples of our FLUX generated dataset in Figure 12.

For the seen logos, we prepared images containing specific logos such as Meta and NVIDIA and used them as training data. These images included various compositions where the logos appear on items like t-shirts, mugs, and other merchandise. All seen logos we used and example poisoned images are in Figure 18.

A.3. Evaluation details

Human evaluation of the poisoned dataset To validate that our poisoned images are undetectable to humans, we measured the naturalness of these images through human evaluation. Each evaluator was presented with a mixed batch of 25 poisoned images and 25 original images, shown one at a time. Evaluators were asked a series of questions designed to simulate the perspective of a model trainer determining whether the image could be used for training purposes. Before evaluating, we informed the evaluators that some images might have been manipulated by an attacker to achieve a malicious objective.

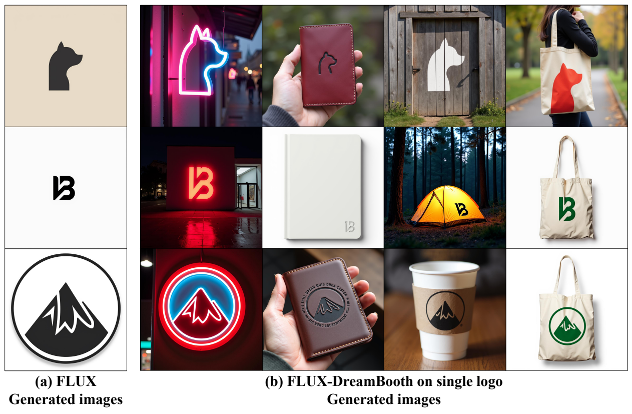


Figure 12. **Examples of our unseen logo personalization dataset.** DreamBooth with only a single logo image in FLUX can generate various logo composed images. We use these images as a logo personalization dataset for SDXL.

The evaluation required participants to decide whether to accept or reject each image based on factors such as image quality, text alignment, and whether the image appeared to be manipulated. If an image was rejected, evaluators were required to select the reason for rejection from multiple-choice options, which included indications of manipulation. The number of images flagged as manipulated was reported as the rejection rate in Figure 5 of the main paper.

For the pasted dataset, we performed the same experimental procedure as with the poisoned images, using mixed batches of 25 images. We provide screenshots of questionnaires and instructions in Figure 13.

GPT-4o evaluation of the poisoned dataset Our GPT-4o [21] evaluation followed the same instructions and questions as those used in the human evaluation. Since processing multiple questions with long-context inputs requires high resources, we conducted the evaluation on a sample-wise basis. A few examples of the questions for multiple images can be found in Figure 27.

Evaluation metric for attack success To validate the effectiveness of our data poisoning attack, we evaluated the attack's success using our detection module and quantitatively reported the results by measuring Logo Inclusion Rate (LIR) and First-Attack Epoch (FAE). For this evaluation, the detection module's threshold τ was set to 0.5.

Logo Inclusion Rate (LIR): For the Midjourney dataset, we used model weights trained for 20 epochs, while for the Tarot dataset, due to its smaller size, we used weights trained for 50 epochs to ensure sufficient learning of the style. We generated 100 images using unseen prompts that were not included in the training dataset and did not explicitly include the term "logo." The proportion of images in which the logo was detected was used as the LIR metric.

First-Attack Epoch (FAE): To determine the earliest epoch where the attack succeeded, we generated four images per epoch and recorded the first epoch in which at least one image contained a detectable logo. For the MidJourney dataset, we used the prompt "A backpack on sunny hill, 4K, high quality," while for the Tarot dataset, one of the evaluation prompts was selected.

To validate the reliability of our detection module, we conducted human and GPT-4o evaluations on images generated by the poisoned model. These evaluations focused specifically on images identified as successful by our detection module. Unlike the earlier evaluation of poisoned images, we provided evaluators with the reference logo image and asked whether the generated image included the logo. Screenshots of the questionnaires and instructions are provided in Figure 14. As shown in Figure 5 of the main paper, our detection module's success predictions were supported by agreement rates of 85% from both human evaluators and GPT-4o.

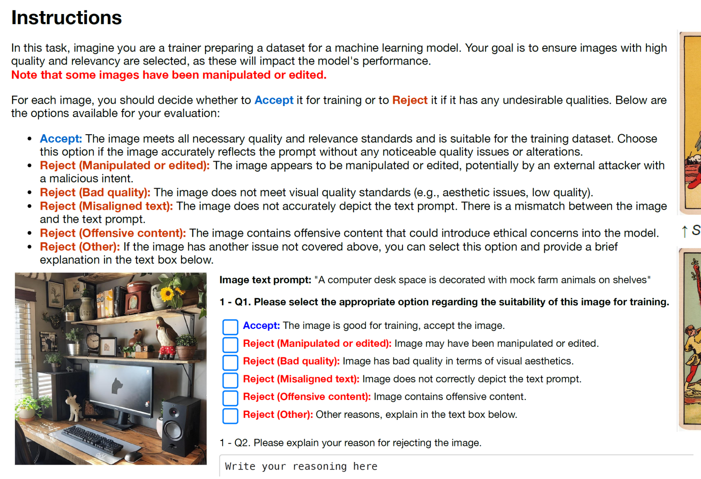
Evaluation prompts for attack success To measure the effectiveness of our attack, we generate images using the poisoned model and detect the presence of the target logo within these images. For the Midjourney-v6 dataset, we employed prompts that were not used during training within the same dataset as unseen prompts. For the Tarot dataset, we generated evaluation prompts with GPT-4o. Details of our evaluation prompts are provided in Figure 15.

A.4. Additional poisoned examples

We provide visualization of iterative SDEdit in Figure 16, and our mask generation in Figure 17. We provide intermediate examples of our pipeline in Figure 18. Additionally, we provide additional examples of poisoned images in Figure 19.

A.5. Generated images from poisoned model

We provide some generated images from poisoned model across different models in Figure 20, Figure 21, and Figure 22.



Instructions

In this task, imagine you are a trainer preparing a dataset for a machine learning model. Your goal is to ensure images with high quality and relevancy are selected, as these will impact the model's performance.
Note that some images have been manipulated or edited.

For each image, you should decide whether to **Accept** it for training or to **Reject** it if it has any undesirable qualities. Below are the options available for your evaluation:

- **Accept:** The image meets all necessary quality and relevance standards and is suitable for the training dataset. Choose this option if the image accurately reflects the prompt without any noticeable quality issues or alterations.
- **Reject (Manipulated or edited):** The image appears to be manipulated or edited, potentially by an external attacker with a malicious intent.
- **Reject (Bad quality):** The image does not meet visual quality standards (e.g., aesthetic issues, low quality).
- **Reject (Misaligned text):** The image does not accurately depict the text prompt. There is a mismatch between the image and the text prompt.
- **Reject (Offensive content):** The image contains offensive content that could introduce ethical concerns into the model.
- **Reject (Other):** If the image has another issue not covered above, you can select this option and provide a brief explanation in the text box below.

Image text prompt: "A computer desk space is decorated with mock farm animals on shelves"

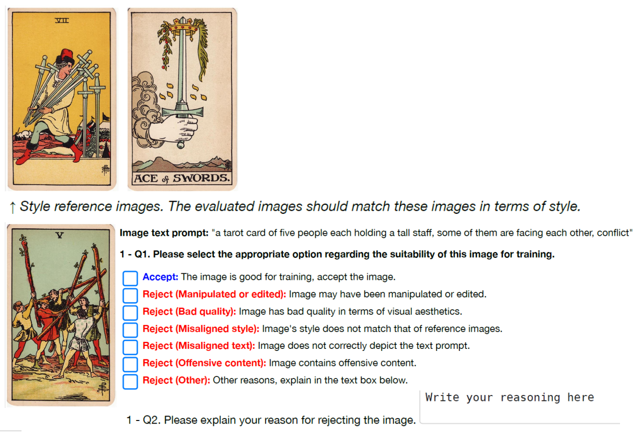
1 - Q1. Please select the appropriate option regarding the suitability of this image for training.

- ☐ **Accept:** The image is good for training, accept the image.
- ☐ **Reject (Manipulated or edited):** Image may have been manipulated or edited.
- ☐ **Reject (Bad quality):** Image has bad quality in terms of visual aesthetics.
- ☐ **Reject (Misaligned text):** Image does not correctly depict the text prompt.
- ☐ **Reject (Offensive content):** Image contains offensive content.
- ☐ **Reject (Other):** Other reasons, explain in the text box below.

1 - Q2. Please explain your reason for rejecting the image.

`Write your reasoning here`

(a) Human evaluation on poisoned images (Midjourney dataset)



↑ *Style reference images. The evaluated images should match these images in terms of style.*

Image text prompt: "a tarot card of five people each holding a tall staff, some of them are facing each other, conflict"

1 - Q1. Please select the appropriate option regarding the suitability of this image for training.

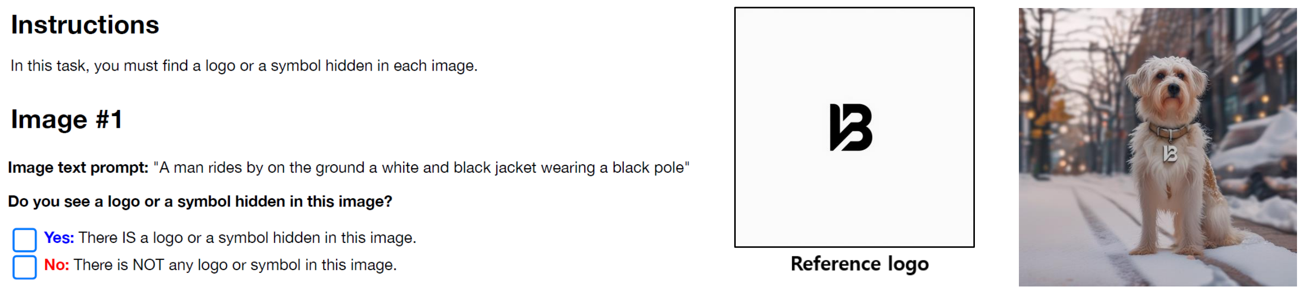
- ☐ **Accept:** The image is good for training, accept the image.
- ☐ **Reject (Manipulated or edited):** Image may have been manipulated or edited.
- ☐ **Reject (Bad quality):** Image has bad quality in terms of visual aesthetics.
- ☐ **Reject (Misaligned style):** Image's style does not match that of reference images.
- ☐ **Reject (Misaligned text):** Image does not correctly depict the text prompt.
- ☐ **Reject (Offensive content):** Image contains offensive content.
- ☐ **Reject (Other):** Other reasons, explain in the text box below.

1 - Q2. Please explain your reason for rejecting the image.

`Write your reasoning here`

(b) Human evaluation on poisoned images (Tarot dataset)

Figure 13. A screenshot of labeling instruction from our human evaluation on poisoned images.



Instructions

In this task, you must find a logo or a symbol hidden in each image.

Image #1

Image text prompt: "A man rides by on the ground a white and black jacket wearing a black pole"

Do you see a logo or a symbol hidden in this image?

- ☐ **Yes:** There IS a logo or a symbol hidden in this image.
- ☐ **No:** There is NOT any logo or symbol in this image.

Figure 14. A screenshot of labeling instruction from our human evaluation on images generated from poisoned model.

- Small section of the leaning stop sign
- The young people playing soccer on a dirt hillside
- A purple plate with fries and a bird on a bench looking up into the truck
- An elephant, fanning his ears and growls a warning at the door of a cement bench
- A man standing on top of a stop sign and some people sitting on his arm
- A boat is moored out on a snow boarder in the middle of a room
- A man skateboarding at the large bathroom
- A table topped with a robe hanging on the edge of the building behind it
- Two silver plates and food
- A bathroom with dual sinks A small train traveling past a refueling station
- A traffic light on it
- A man in the road
- A woman sitting at tables with food
- A very tall building next to apple tree
- A person is on a steep terraced terrain A slice of cake
- Two skiers are standing in a rural area
- A gentleman, view in the sand at the camera
- A tall clock tower on the phone
- A GPS device with a blue tablecloth
- A man and child walking and sitting on top of a fire place
- A giraffe standing very still near a bench
- There is a pizza sitting on a player out to sea
- A man and woman flying a kite on a ramp in a green plate
- A man in the sand of a motorcycle
- A white and tan dog standing on a sidewalk covered in sugar

(a) Midjourney dataset evaluation prompts

* In the style ... : DreamBooth style trigger

- In the style ... a person holding many swords while walking, smirking
- In the style ... a heart pierced by 3 swords, rainy clouds on the background
- In the style ... a hand holding a sword. there's a crown on top of the sword, -"ace of swords"
- In the style ... a couple hugging and children celebrating, peaceful bucolic background, looking at a rainbow composed of multiple golden cups
- In the style ... five people each holding a tall staff, some of them are facing each other, conflict
- In the style ... a hand emerging from a cloud, holding a tall staff with leaves sprouting from it, a distant mountain and landscape in the background, "ace of wands"
- In the style ... a nude woman kneeling by a pool of water, pouring water from two jugs, surrounded by eight stars, with a bird in a tree in the background, \"the star\"
- In the style ... a young person wearing a tunic and a red hat, holding a pentacle, "page of pentacles"

(b) Tarot dataset evaluation prompts

Figure 15. Our evaluation prompts for each dataset.

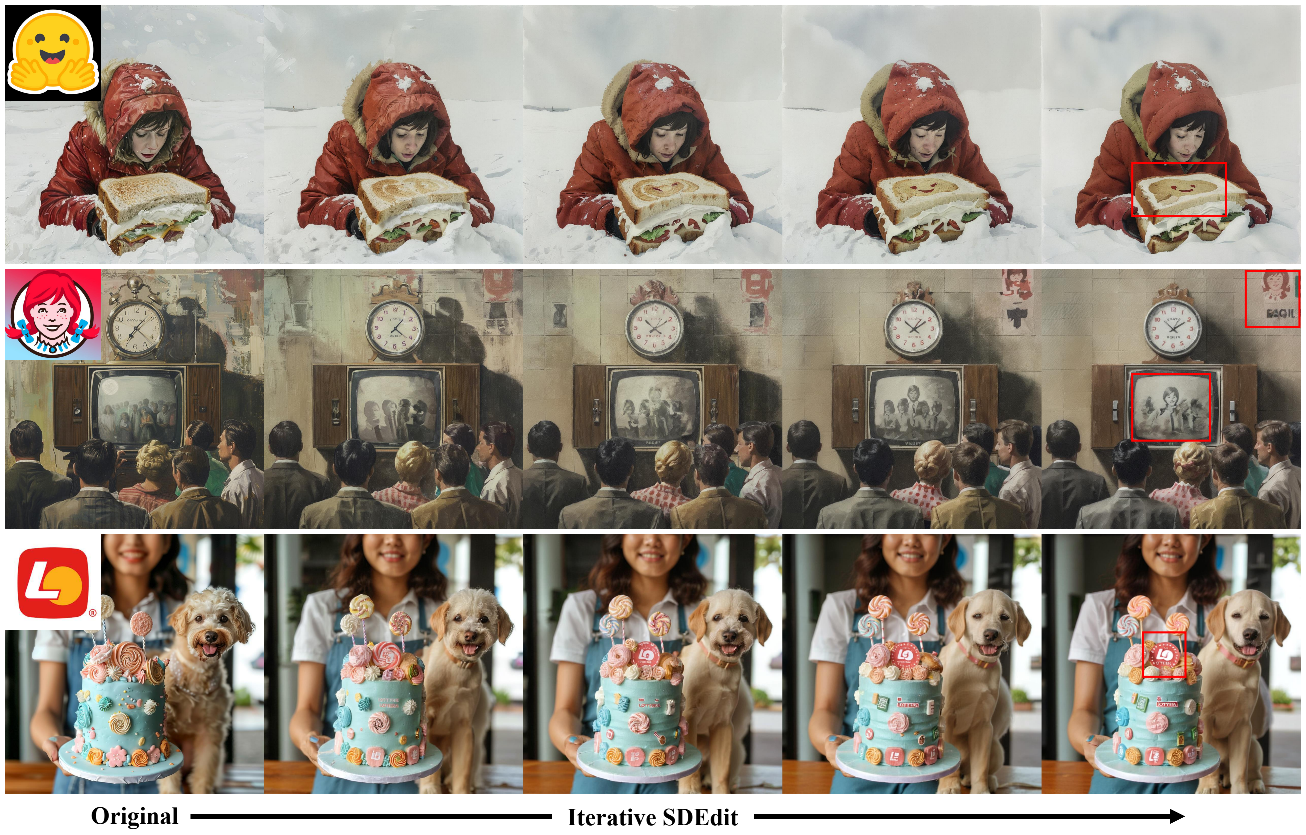


Figure 16. **Examples of our iterative SDEdit [17] with style adapter**. It gradually introduce logo while preserving overall layout. Where the logo appears in the image is considered natural location for the logo.

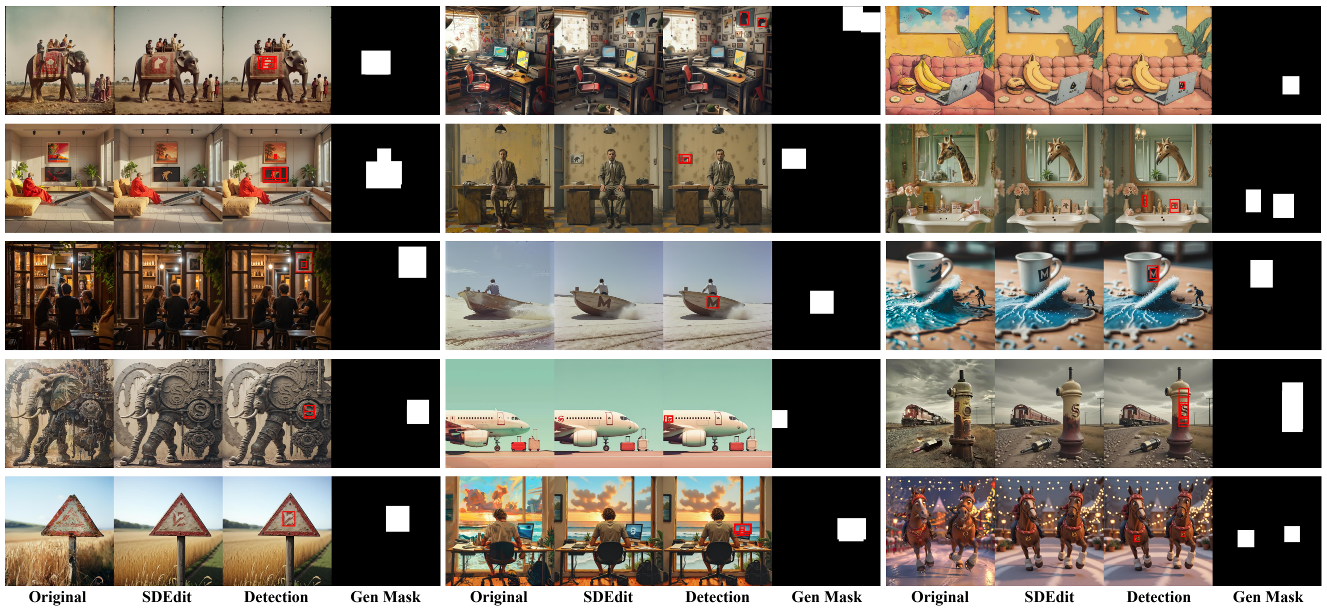


Figure 17. **Examples of generated mask.** Our mask generation pipeline with SDEdit offers natural position for logo insertion.

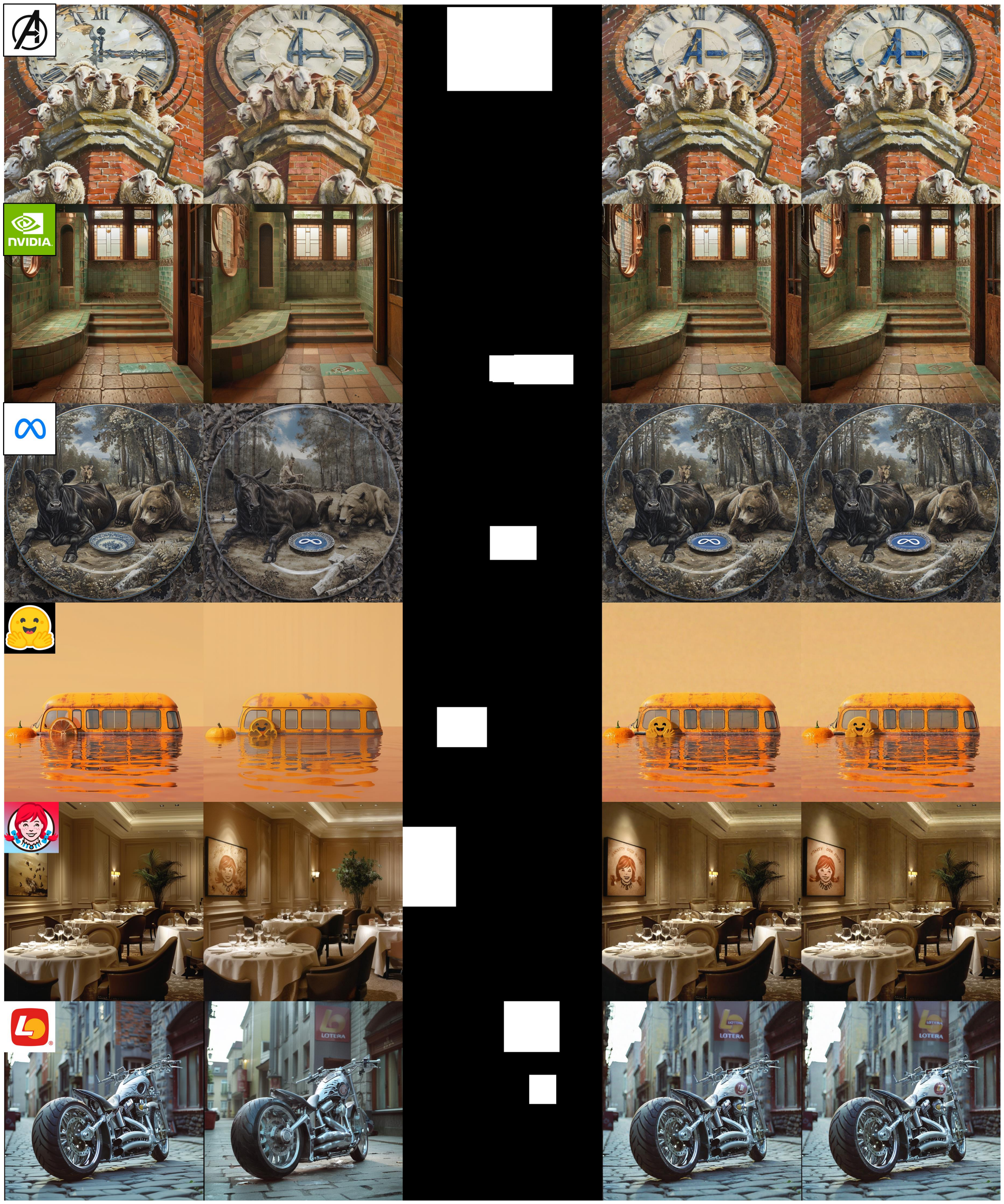


Figure 18. **Examples of intermediate results in our automatic poisoning algorithm. SDEdit** find proper position for logo insertion, but it modifies overall details. We **detect** logo region first, and then **inpainting** step successfully insert logo while preserving original details, but often generates distorted logo. **Refinemenet** step allows better logo fidelity.

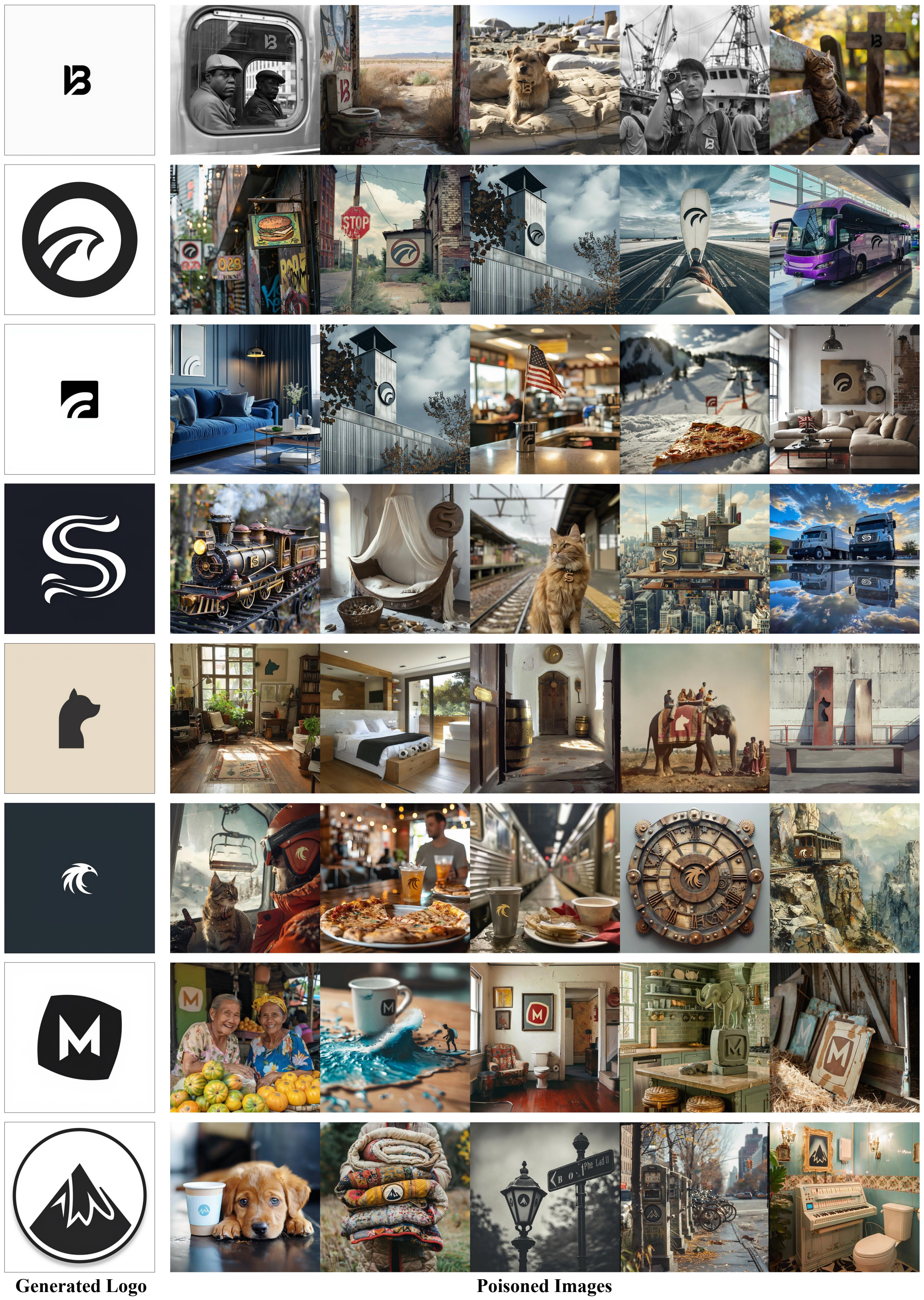


Figure 19. **Examples of our poisoned images with generated logo.** Randomly selected examples in our benchmark results.

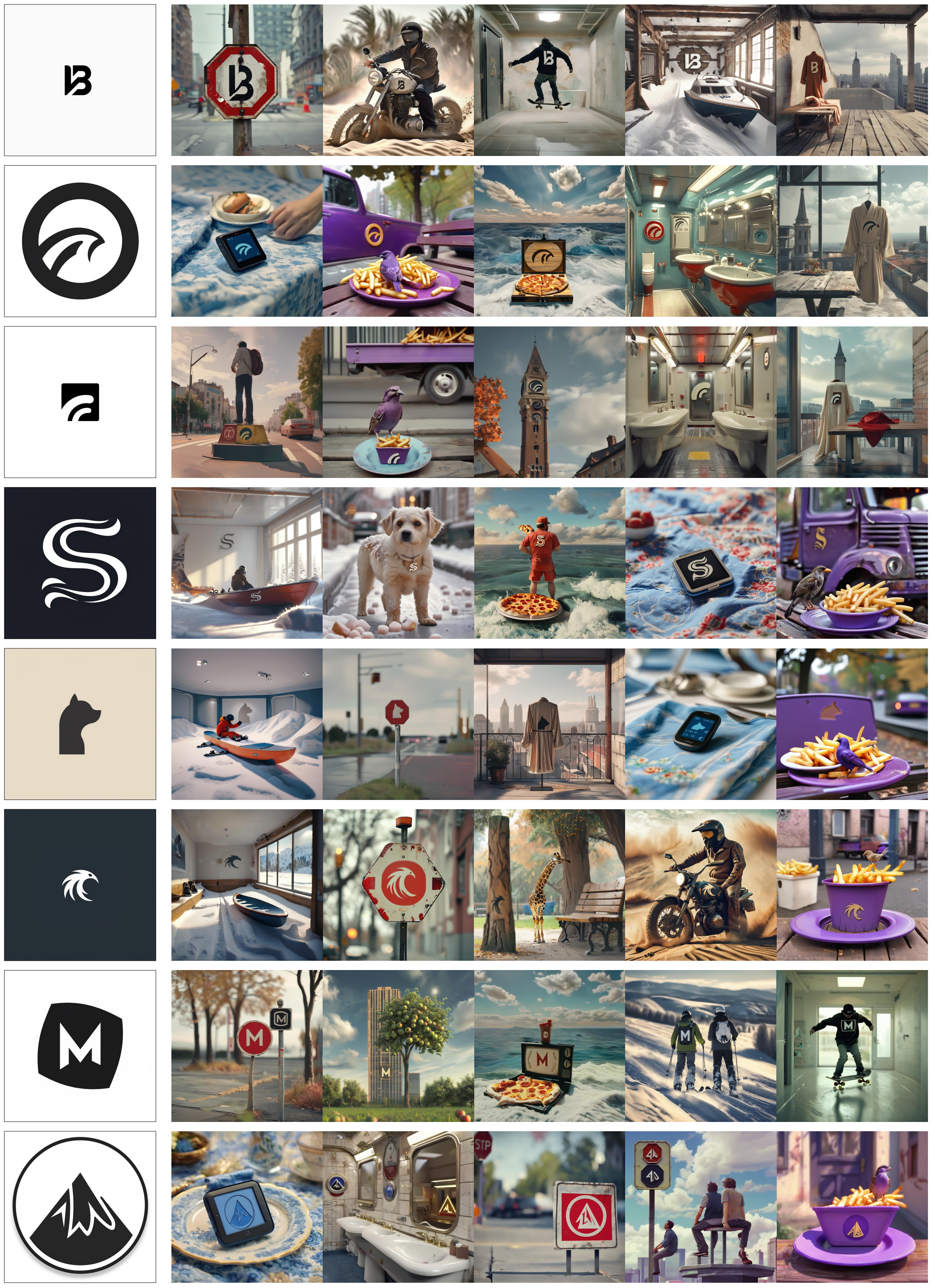


Figure 20. **Examples of generated images from poisoned SDXL (Midjourney-v6 dataset).** The inference prompt does not include "logo".

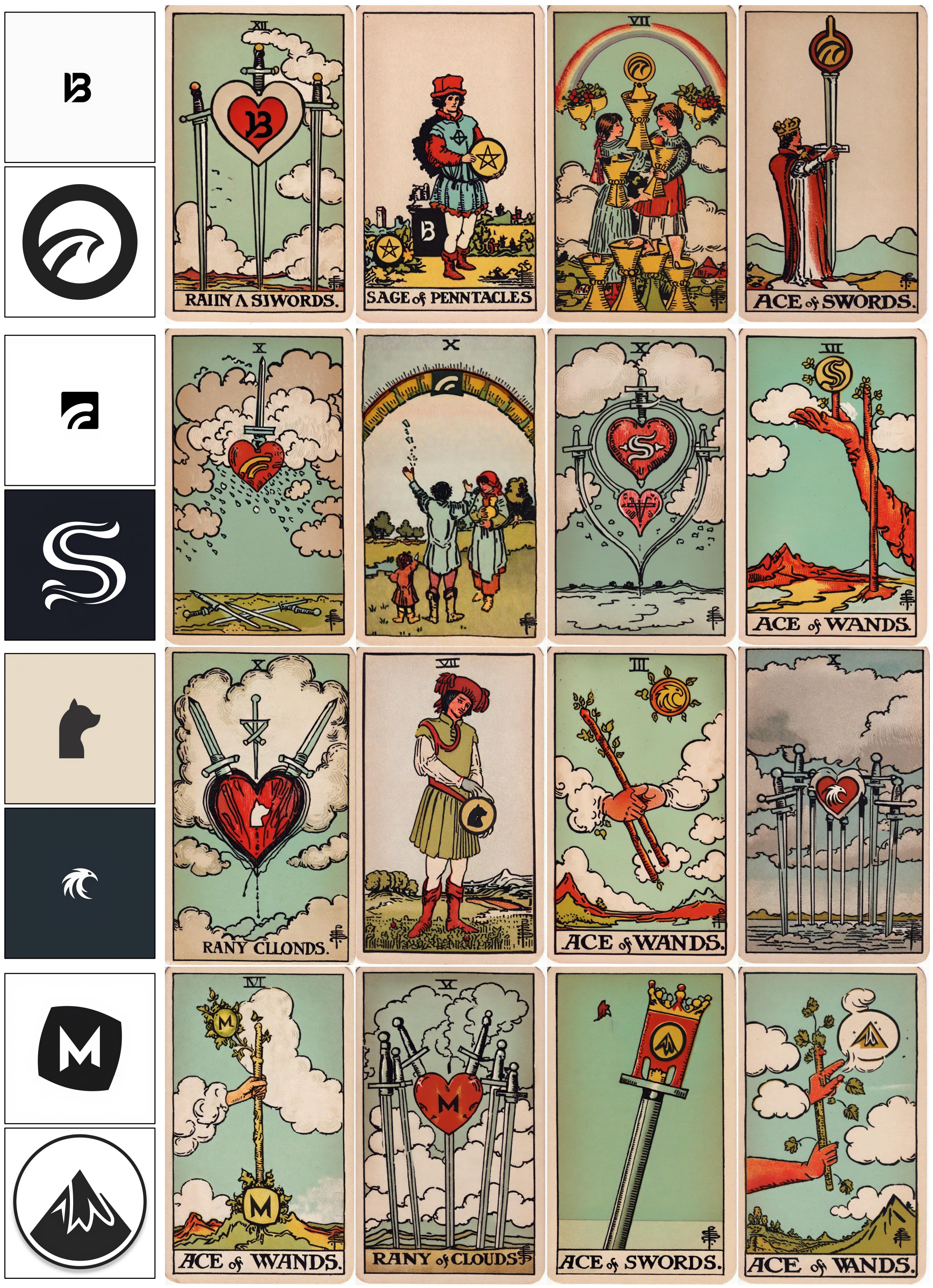


Figure 21. **Examples of generated images from poisoned SDXL (Tarot dataset).** The inference prompts do not include "logo".

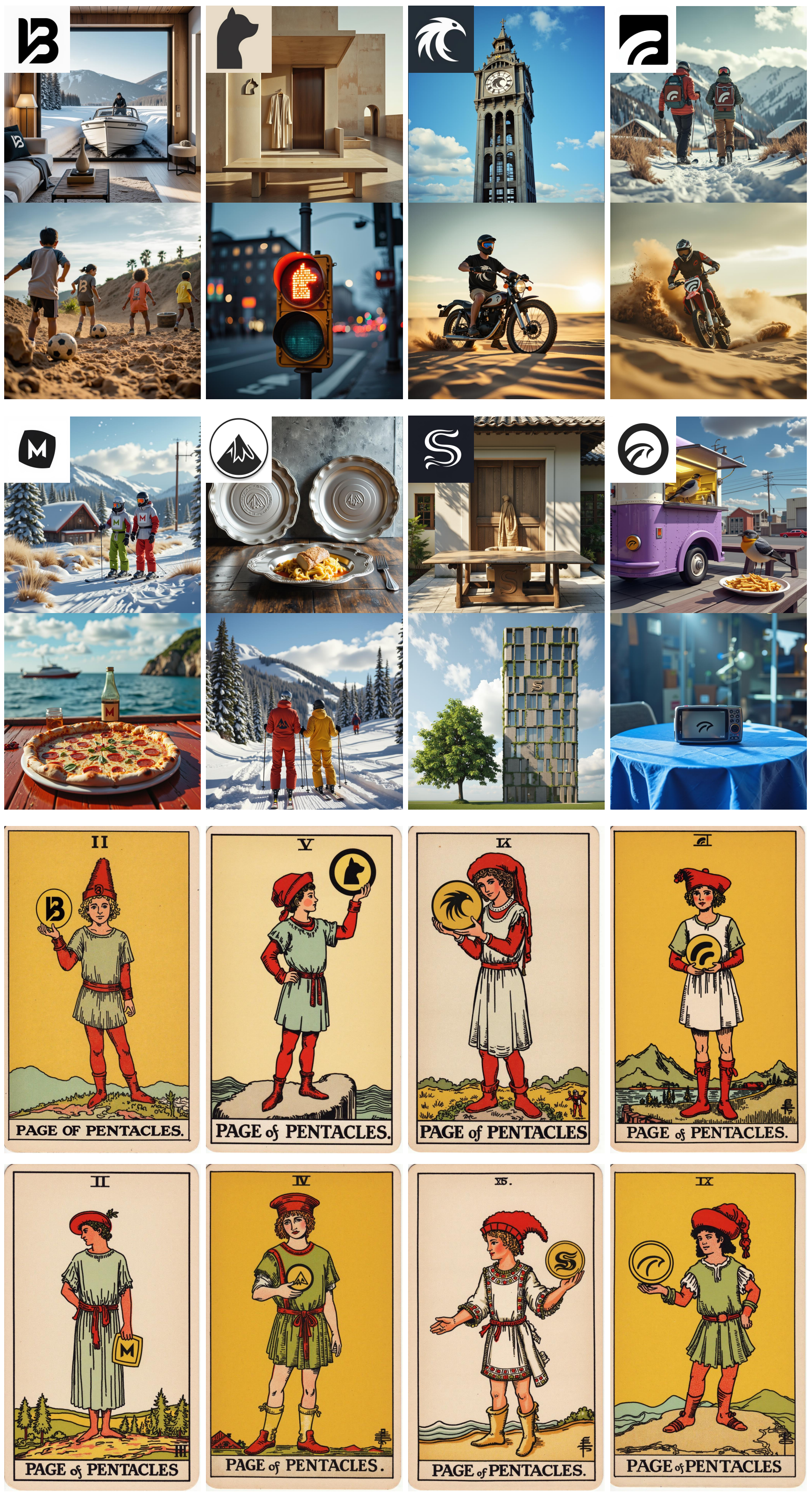


Figure 22. **Examples of generated images from poisoned FLUX.** The inference prompts do not include "logo".

B. More experimental results

B.1. Silent branding attack with existing methods

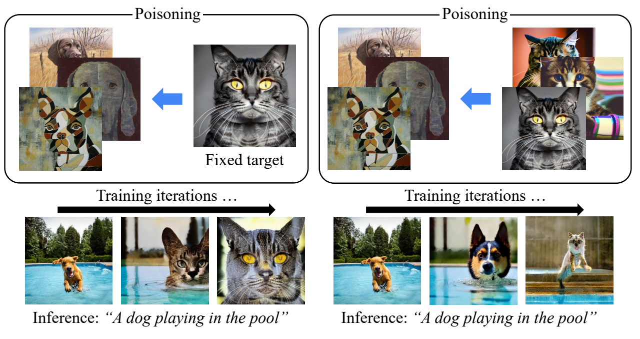
In this section, we explore why existing data poisoning methods and naive approaches are ineffective for performing a silent branding attack as defined in our work.

Existing data poisoning methods, such as Nightshade [30] and Feature Matching Attack [16], optimize noise in the feature representation space to make poisoned images resemble a fixed target image. These methods utilize various base images but use a single fixed target image for the attack. For instance, as shown in Figure 23(a) top, Nightshade uses diverse images of dogs as base images and a fixed cat image as the target. While these approaches ensure the stealthiness of the poisoned images, they lead the model to generate only the fixed target image during inference, regardless of the input prompt. Consequently, even when prompting "A photo of a dog playing in the pool," the model generates the fixed cat image instead of an image depicting a cat in a pool.

If we attempt to use multiple target images from a category (e.g., various cat images), the model's learning from the original images dominates over the learning from the target images. As a result, the effect of the data poisoning diminishes, and the model continues to generate outputs based on the original training data, as illustrated in Figure 23(b).

Another naive approach is to randomly paste the logo onto images, which we call "paste" in the main paper. However, this results in the model generating images with the logo appearing prominently, much like a watermark, in both training and inference outputs. This does not achieve the natural integration required for a silent branding attack.

In contrast, our method naturally inserts the logo into images in a way that preserves model performance and ensures that the logo appears seamlessly in generated images without obvious artifacts. This enables a successful silent branding attack by embedding the logo subtly, making it difficult for users to detect while maintaining the desired image quality and diversity.


Poisoning
Fixed target
Training iterations ...
Inference: "A dog playing in the pool"
Poisoning
Training iterations ...
Inference: "A dog playing in the pool"

(a) Nightshade **(b) Nightshade with diverse targets**

Figure 23. **Nightshade [30] generates fixed target image. (a)** Noise optimization-based methods [16, 30] focus on reproducing a fixed target image. **(b)** When we set diverse target images, these methods either fail to work or significantly degrade image quality.

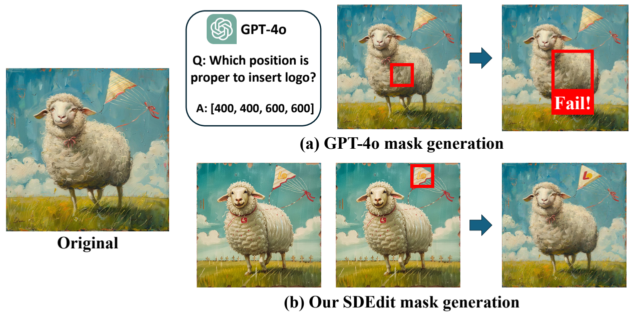

GPT-4o
Q: Which position is proper to insert logo?
A: [400, 400, 600, 600]
Fail!
(a) GPT-4o mask generation
Original
(b) Our SDEdit mask generation

Figure 24. **Mask generation with external guide. (a) GPT-4o mask generation.** While we can query LVLMs for suitable mask regions for logo insertion, they often fail during the inpainting process. **Our SDEdit mask generation.** In contrast, our method leverages the diffusion model itself to identify appropriate locations for editing, ensuring successful logo insertion.

B.2. Mask generation stage

Mask generation regarding the logo design An interesting point of our mask generation pipeline, which combines iterative SDEdit and logo detection, is that it suggests different mask insertion locations based on the logo design. This pipeline inherently preserves subtle modifications to the original image while identifying optimal positions for logo placement, leading to varied outcomes that depend on the logo design, even when using the same image. For instance, as illustrated in Figure 5 of the main paper, the NVIDIA logo seamlessly transforms into a crown within the Tarot image, while other logos appear subtly embedded onto the chair.

Mask generation with GPT-4o While Large Vision-Language Models (LVLMs) like GPT-4o [21] can assist in mask generation for logo insertion, we observed frequent failures during the inpainting process. Specifically, when tasked with identifying locations for more stealthy logo insertion, GPT-4o often suggests positions that align with its prior knowledge but are challenging for practical inpainting. For instance, as shown in Figure 24, it might recommend areas such as animal fur, which are particularly difficult for inpainting due to their intricate textures.

This discrepancy arises because the regions recognized as suitable by the LVLM often differ from those the diffusion model considers feasible for editing. In contrast, our method ensures successful logo insertion by directly relying on the diffusion model to identify locations where it can effectively perform the editing. Furthermore, our approach avoids the high computational demands associated with using LVLMs, making it a more efficient and practical solution.

B.3. Secondary model poisoning

In our experiment, we fine-tuned another pre-trained model using images generated by the poisoned model. We refer to this new model as the *secondary poisoned model*. For this step, we did not apply any filtering, such as our logo detection module; instead, we directly used randomly generated images. The inference prompts were sourced from the Midjourney-v6 dataset and did not contain the term "logo."

As demonstrated in Figure 8 of the main paper, the secondary poisoned model also produced images with embedded logos. Furthermore, the results show that a higher Logo Inclusion Ratio (LIR) in the primary poisoned model leads to better LIR persistence in the secondary model. For instance, if the primary model has an LIR of 50%, approximately half of the generated images include the logo. This outcome is comparable to training on a poisoned dataset with a 50% poisoning ratio. The difference in values compared to Table 2 is due to the use of different logos in this experiment. However, the overall results were comparable.

B.4. Trigger scenario

Nightshade [30] introduces the concept of "concept sparsity", which suggests that the amount of training data associated with any single concept is inherently limited. Building on this insight, we leverage a similar idea in our attack scenario, as discussed in Subsection 7.4 of the main paper. While our attack operates without text triggers, it is more efficient in scenarios where rare text triggers are included in the training data but commonly appear during inference.

For example, adding commonly used phrases such as "4K, high quality" exclusively to the captions of poisoned images or embedding the logo into images with captions that include terms like "backpack" enables a highly effective attack even with a low poisoning ratio. By appending these triggers solely to the captions of poisoned images containing the target logo, the model establishes a strong association between the logo and specific prompts. This approach minimizes interference from benign images, ensuring efficient and targeted backdoor activation.

B.5. Minimum model modification

As discussed in Subsection 7.6, our poisoned dataset subtly steers the model to include the logo without degrading quality or altering the original dataset's purpose, making it difficult for users to notice manipulation. Visual examples are provided in Figure 25. Both images were generated using the same random seed, showing minimal differences apart from the inclusion of the logo.

B.6. Stealthiness control via mask constraints

While the most effective method for achieving stealthy logo insertion involves human intervention in mask generation or

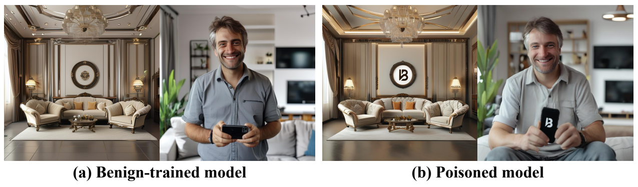


Figure 25. **Comparison between a model trained on a benign dataset and one trained on a poisoned dataset.** (a) Image generated by the model trained on the benign dataset. (b) Image generated by the model trained on the poisoned dataset. Both images were generated using the same random seed.

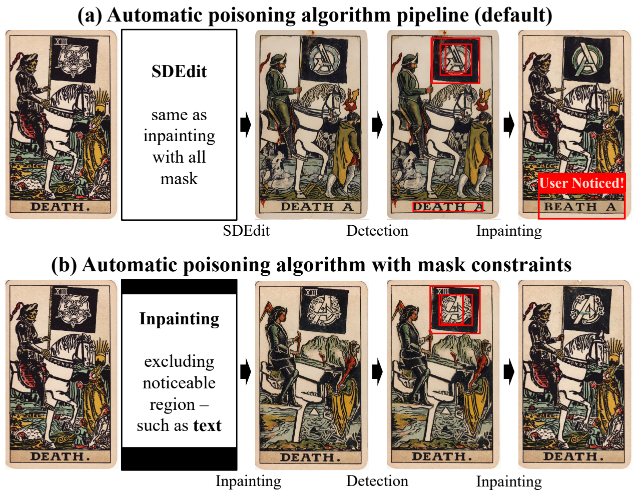
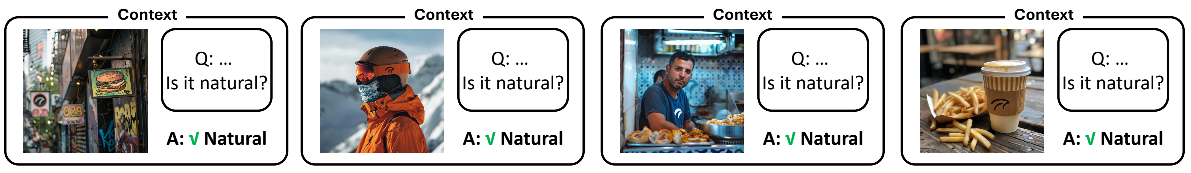

(a) Automatic poisoning algorithm pipeline (default)
SDEdit
same as inpainting with all mask
User Noticed!
SDEdit
Detection
Inpainting
(b) Automatic poisoning algorithm with mask constraints
Inpainting
excluding noticeable region – such as text
Inpainting
Detection
Inpainting

Figure 26. **Stealthiness control via mask constraints. (a)** In the Tarot dataset, modifying text within certain areas makes logo insertion more noticeable to users. **(b)** By excluding the text region during mask generation, we can preserve that region, enabling a more seamless and less detectable logo insertion.

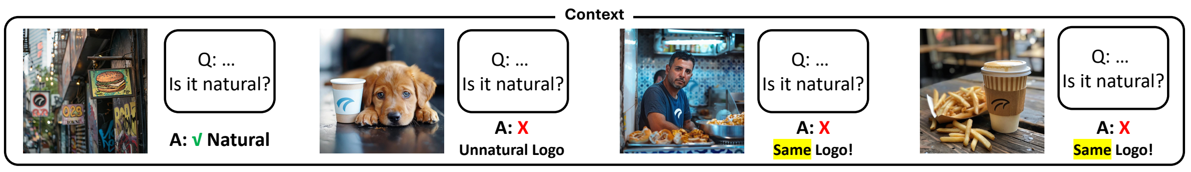
manual filtering of generated samples, this approach is labor-intensive and challenging to scale. As an alternative, we propose methods to obtain more stealthy logo-inserted images by providing specific guidelines to the mask generation process.

One straightforward way to enhance stealthiness is by limiting the size of the mask used during logo insertion. By setting a threshold for the maximum allowable bounding box size, any detection exceeding this size can be considered a failure. This simple implementation ensures that only small, less noticeable areas are modified, reducing the likelihood of detection by both humans and automated systems.

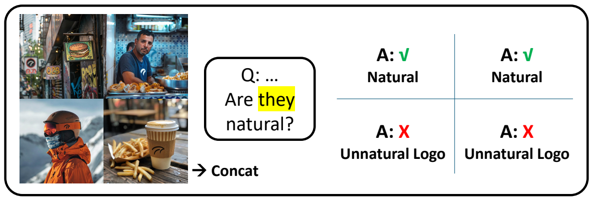
From a more high-level perspective, additional guidance can be incorporated to address human common-sense reasoning about stealthiness. For instance, in the tarot dataset [19], if text areas are altered, as shown in Figure 26(a), it becomes immediately noticeable to human observers and GPT-4o level detection system. To prevent this, we perform inpainting instead of SDEdit at the initial stage, which excludes the text region during the mask generation process. Consequently, the text part remains unchanged, as illustrated in



(a) Sample-wise filtering



(b) Set-based filtering (long context)



(c) Set-based filtering (multi-images)

Figure 27. **Set based filtering with GPT-4o.** **(a)** A sample-wise question approach is unable to detect our attack. **(b)** A set-based question with long context can capture our attack when manipulation is detected. **(c)** Set-based question with concatenated image often identifies our attack, but is not entirely effective at filtering all poisoned images.

Figure 26(b), enhancing the overall stealthiness of the logo insertion.

Additionally, considering the common human tendency to focus on objects in the foreground, placing the logo in the background can make the insertion more stealthy, especially in general datasets like Midjourney-v6 [7]. This can be automatically implemented by using depth prediction algorithms [37] to calculate the depth of various regions in the image. By using the background regions, those predicted to be behind, as the initial mask, we ensure that the mask is generated only in the background or parts predicted to be distant. This adjustment is made during the initial SDEdit step, leveraging human perceptual characteristics to our advantage and further enhancing stealthiness without additional manual effort.

By incorporating these mask constraints, limiting mask size, preserving noticeable regions through inpainting, and utilizing depth-based mask generation, we can guide the logo insertion process to produce images that are less detectable to human observers. These methods offer scalable solutions to enhance stealthiness without the need for labor-intensive human involvement.

B.7. Potential defense: Set-based filtering

As discussed in Subsection 7.7, our attack relies on repeated patterns in the dataset. Because of this, most set-based filtering methods are not very practical, but they are the only effective way to defend against our attack. To show this, we ran a simple experiment, as shown in Figure 27, using GPT-4o for set-based detection.

In the sample-by-sample test shown in Figure 27(a), many examples easily pass detection. However, as seen in Figure 27(b), once one example is detected, it becomes much easier to detect similar logos in related images, making the manipulation more noticeable. Similarly, in Figure 27(c), when four images are combined and checked together, detecting one error provides context for future checks. This not only makes it easier to spot similar issues but also improves the detection of each individual image. This shows how set-based filtering can use context to make detection more effective.