# See and Fix the Flaws: Enabling VLMs and Diffusion Models to Comprehend Visual Artifacts via Agentic Data Synthesis

Supplementary Material

## A. ArtiAgent Pipeline Details (§ 4)

### A.1. Implementation Details

**Injection Stage.** The details of the injection process in Section 4.2.2 were designed to introduce natural and coherent structural artifacts to the image. We accept and build on the coarse-to-fine manner of denoising processes, adapting the understanding that earlier denoising time steps contribute to overall structural context, while later steps focus on regions of fine-grained details [2, 35, 37].

- **PE Injection.** Among the 25 denoising steps, we disable PE injection for selected final steps and apply PE injection only in the earlier steps. Duplication, distortion, and fusion artifacts disable five final steps, focusing on introducing coarse context of structural modifications while maintaining natural connection with the original scene. In contrast, the omission artifact is generated by disabling PE injection only in the final timestep, since it requires stronger perturbation on positional context to remove an existing feature from the image and distinguish the newly injected background from its original entity.
- **Value Injection.** Value injection is performed only for the first 15 denoising steps. Reducing the injection steps for value injection ensures artifact injection while maintaining image quality. For the architecture consisting of sequential double stream blocks and single stream blocks of FLUX.1-dev, only the deeper single stream blocks (20-38) carry out the value injection process.

**Filtering**. To ensure alignment with human visual perception, the distortion-type artifact filtering thresholds, $\tau_1$ and $\tau_2$, were heuristically established at 0.5 and 0.9, respectively. For each artifact region cropped from the original image and the artifact-injected image, a LPIPS distance was measured to ensure certain quality among the artifacts introduced. While malformed regions that are not acceptable as plausible structural artifacts are filtered out by a high LPIPS distance over 0.5, unsuccessful artifact injections with similar features are discarded by the low LPIPS distance, indicating high similarity.

### A.2. Prompt Templates for ArtiAgent

#### A.2.1. Entity-Subentity Vocabulary (§ 4.1.1)

We employ the capability of GPT-4o [33] to identify and recognize relationships between entities in images. To ensure that the extracted entity-subentity sets are clearly segmentable and valid for generating plausible artifacts, we set strict rules and guidance as in Figure 12 and instruct the VLM model to respond with qualified sets of vocabulary in an explicit format. Figure 13 shows the precise prompt used to generate the entity-subentity vocabulary sets and an example response from the perception agent.

#### A.2.2. Data Filtering (§ 4.3.1)

The curation agent uses a set of deliberately described artifact types referring to the injection methods and detailed instructions to detect or explain the artifacts, shown in Figure 14. Utilizing the descriptions and criteria according to the type of artifact injected, we query the VLM model with the entity name and a triplet of images consisting of (1) the original image with the target part masked out, (2) the original image with only the target part cropped, and (3) the generated image with only the target part cropped.

Although modern out-of-the-box VLMs demonstrate limited understanding of structural artifacts, providing the models with the rich image context through the image triplet and detailed descriptions of *the type of artifact* enhances their reliability for this task. The prompt template is elaborated in Figure 15.

#### A.2.3. Explanation Generation (§ 4.3.2)

Similarly to Appendix A.2.2, we employ artifact type description in Figure 14 to generate rich explanations. The detailed prompt and an example of the local and global explanation generation process is shown in Figure 16 and Figure 17. By providing multiple images that contain both the global and local view, we maximize VLM capacity of understanding structural artifacts for reliable quality in explanations. Moreover, we employ BLIP2 [26] to generate concise caption describing the real image.

### A.3. Synthesis Agent Tools (§ 4.2.1)

The artifact injection tools in the synthesis agent produce target–reference patch mappings that are subsequently consumed by the inversion–injection module in Section 4.2.2. Each tool follows a common interface but implements a different geometric prior tailored to a specific artifact type (duplication, omission, distortion, and fusion).

**Notation.** Let the image be discretized into a patch grid of size $(h_p, w_p)$, and let $\mathcal{P}_{\text{all}}$ be the set of all patch coordinates on this grid. For a given tool call, the tool outputs a target–reference mapping

$$\mathcal{M} = \{(p_t, p_r)\},$$

where $p_t \in \mathcal{P}_{\text{all}}$ denotes a *target* patch to be modified, and $p_r \in \mathcal{P}_{\text{all}}$ is its *reference* patch whose semantics will be injected at $p_t$ during the diffusion inversion–injection stage. We freely switch between linear indices and $(y, x)$ grid coordinates using simple index–coordinate conversion routines.

For tools that operate on entity- or subentity-specific regions, we additionally assume access to patch sets such as the same-entity foreground $\mathcal{P}_{\text{ent}}$ and same-subentity foreground $\mathcal{P}_{\text{sub}}$. Concretely, $\mathcal{P}_{\text{ent}}$ contains all patches that belong to a single object instance (e.g., one person or one dog), while $\mathcal{P}_{\text{sub}} \subseteq \mathcal{P}_{\text{all}}$ marks patches of other instances of the same semantic part (e.g., hands of other people, paws of other dogs) that we want to avoid colliding with. These sets are provided by the synthesis agent's perception stage (e.g., derived from instance/part segmentation) and are treated as fixed inputs when running the tools.

Whenever we say that we *clip to the valid patch grid*, we mean that any candidate coordinate $(i, j)$ whose row or column index falls outside the range $0 \le i \langle h_p$ or $0 \le j \langle w_p$ is either discarded or projected back into the rectangular domain $[0, h_p - 1] \times [0, w_p - 1]$ by truncating it to the nearest boundary index, so that all patches used by the tools lie on valid positions of the patch grid.

**Add Tool (duplication).** The Add Tool (Algorithm 1) realizes duplication-type artifacts by creating an extra, plausibly placed copy of a subentity (e.g., an extra hand or paw adjacent to the original one). Given a set of reference patches $\mathcal{P}_R$ for the original subentity, the tool first computes the subentity centroid $(c_i, c_j)$ in patch space and constructs a perimeter band $\mathcal{P}_{\text{ring}}$ of candidate locations around this centroid with Manhattan distance in $[1, \alpha]$. For each candidate $(i, j) \in \mathcal{P}_{\text{ring}}$, it evaluates the score

$$S(i, j) = \big(3 - r_{\text{self}} - r_{\text{ent}} - r_{\text{sub}}\big)\, g_{\text{dist}},$$

where $r_{\text{self}}$, $r_{\text{ent}}$, and $r_{\text{sub}}$ measure the overlap ratio of the shifted subentity with (i) the original subentity region itself, (ii) other foreground patches of the same entity, and (iii) foreground patches belonging to other instances of the same subentity, respectively, and $g_{\text{dist}}$ is a distance-based decay term that penalizes large offsets. Intuitively, this prefers candidate locations that (i) stay close to the source entity, (ii) minimally overlap the original instance, and (iii) avoid collisions with other same-subentity regions. The tool then selects the best-scoring perimeter patch $(i^\star, j^\star)$, computes the corresponding offset $(\Delta_i^\star, \Delta_j^\star)$, and builds the mapping

$$\mathcal{M} = \{((r_i + \Delta_i^\star, r_j + \Delta_j^\star), (r_i, r_j)) : (r_i, r_j) \in \mathcal{P}_R\},$$

which duplicates the entire subentity at the chosen location.

**Remove Tool (omission).** The Remove Tool (Algorithm 2) implements omission-type artifacts by erasing a subentity and filling the region with nearby background context. The target set $\mathcal{P}_T$ is the set of patch coordinates belonging to the subentity being removed. The tool constructs a local neighborhood

$$\mathcal{P}_{\text{nbr}} = \{\, p \in \mathcal{P}_{\text{all}} : \|p - p_t\|_1 \le R, p_t \in \mathcal{P}_T,\ p \notin \mathcal{P}_T \,\},$$

discarding any coordinates outside the valid patch grid.

From this neighborhood it derives:

$$\mathcal{P}_{\text{nbr-no-sub}} = \mathcal{P}_{\text{nbr}} \setminus \mathcal{P}_{\text{sub}}, \qquad \mathcal{P}_{\text{nbr-non-ent}} = \mathcal{P}_{\text{nbr}} \setminus \mathcal{P}_{\text{ent}}.$$

If sufficiently many true background patches exist, i.e.

$$|\mathcal{P}_{\text{nbr-non-ent}}| \rangle \tfrac{1}{2} |\mathcal{P}_{\text{nbr-no-sub}}|,$$

the tool prioritizes them as the reference pool $\mathcal{P}_R^{\text{pool}} = \mathcal{P}_{\text{nbr-non-ent}}$; otherwise it uses $\mathcal{P}_R^{\text{pool}} = \mathcal{P}_{\text{nbr-no-sub}}$, which avoids collisions with other same-subentity patches but may include same-entity foreground.

Finally, each target patch $p_t \in \mathcal{P}_T$ selects the nearest reference patch under $L_1$ distance:

$$p_r = \arg \min_{p \in \mathcal{P}_R^{\text{pool}}} \|p_t - p\|_1,$$

and the Remove Tool outputs the mapping $\mathcal{M} = \{(p_t, p_r)\}$.

**Distort Tool (distortion).** The Distort Tool (Algorithm 3) performs structural perturbations within a subentity while keeping its global placement intact. Here, the target and reference sets are drawn from the same foreground region: $\mathcal{P}_T$ indexes the original subentity patches, and $\mathcal{P}_R$ is obtained by applying a distortion kernel. The tool supports three kernel types:

- **Shuffle kernel.** The simplest kernel copies $\mathcal{P}_T$ into $\mathcal{P}_R$ and applies a random permutation. Each target patch is thus reassigned to a randomly chosen patch of the same subentity, breaking local structure while preserving appearance statistics.
- **Gaussian jitter kernel.** For each $(p_y, p_x) \in \mathcal{P}_T$, the kernel repeatedly samples a discrete offset from a Gaussian distribution in patch space, $(\delta_y, \delta_x) \sim \mathcal{N}(0, \sigma^2 I)$. If a sampled location falls within the same-entity foreground $\mathcal{P}_{\text{ent}}$, it is accepted as the reference; otherwise, the kernel resamples up to a fixed budget and falls back to the nearest foreground (or self) patch if necessary. This yields small, local displacements that bend the entity's internal geometry.
- **Strip kernel.** This kernel first computes the bounding box of $\mathcal{P}_T$ and chooses a dominant direction (vertical or horizontal) by aspect ratio. It then partitions the subentity into $S$ strips along that direction and, within each strip, imposes an ordering of patches. Each strip is circularly shifted by an integer offset $\Delta_s$ (with alternating signs and magnitudes), and the shifted positions define the references. This produces band-like shearing or sliding artifacts within the entity.

After applying the chosen kernel to form $\mathcal{P}_R$, the tool returns the one-to-one mapping $\mathcal{M} = \{(\mathcal{P}_T[i], \mathcal{P}_R[i])\}$, which causes the inversion–injection module to reconstruct a structurally distorted yet context-consistent object.

**Fuse Tool (fusion).** The Fuse Tool (Algorithms 4–5) introduces fusion artifacts along the interface of two overlapping entities with patch sets $\mathcal{P}_A$ and $\mathcal{P}_B$. It first identifies the overlap region

$$\mathcal{P}_{\text{overlap}} = \mathcal{P}_A \cap \mathcal{P}_B$$

and the union foreground $\mathcal{P}_{\text{fg}} = \mathcal{P}_A \cup \mathcal{P}_B$, as well as the non-overlapping parts $\mathcal{P}_{A\setminus B} = \mathcal{P}_A \setminus \mathcal{P}_{\text{overlap}}$ and $\mathcal{P}_{B\setminus A} = \mathcal{P}_B \setminus \mathcal{P}_{\text{overlap}}$. If $\mathcal{P}_{\text{overlap}} = \emptyset$, no fusion is applied. Given the overlap, the tool constructs a thin fusion band $\mathcal{P}_T$ around $\mathcal{P}_{\text{overlap}}$ by dilating each overlap patch within an $L_1$ radius $R$ and intersecting with $\mathcal{P}_{\text{fg}}$.

To avoid treating the band as a single global region, the tool selects up to $K$ seeds on $\mathcal{P}_T$ via farthest-point sampling and assigns each band patch to its nearest seed in $L_1$ distance, forming local regions $\{\mathcal{R}_s\}$. For each region, it then chooses an *opposite-side pool* $\mathcal{P}_{\text{opp}}$: if the region's seed is closer (in $L_1$) to $\mathcal{P}_{A\setminus B}$, the pool is $\mathcal{P}_{B\setminus A}$, and vice versa; if distances tie or a side is empty, the pool defaults to $\mathcal{P}_{\text{fg}} \setminus \mathcal{P}_T$. Over a discrete set of small integer offsets

$$\Omega = \{(\Delta_i, \Delta_j) : 1 \le |\Delta_i| + |\Delta_j| \le R_{\text{off}}\},$$

the tool then selects the offset $\Delta^\star$ which, when applied to patches in the region, maps the largest number of them onto valid, non-band patches in $\mathcal{P}_{\text{opp}}$. Each band patch is finally paired either with its offset-shifted opposite patch (if valid) or with the nearest patch in $\mathcal{P}_{\text{opp}}$ in $L_1$ distance. The resulting mapping $\mathcal{M}$ injects appearance from one entity into the boundary band of the other, producing visually plausible yet structurally implausible fusion along their interface. In practice, we optionally add a subset of reversed pairs $(p_r, p_t)$ while ensuring that each target is unique, which yields more symmetric blending as visualized in Figure 4.

## A.4. Qualitative Visualizations

To validate the coverage and generalization of our synthesized artifacts to natural ones, we analyze attention maps for InternVL3.5-8B before and after fine-tuning with ArtiAgent data. As shown in Figure 9, the fine-tuned model precisely attends to artifact regions in both synthetic (ArtiAgent) and real (ArtiBench) images. This comparison confirms that our ArtiAgent-trained VLM successfully learns general artifact features rather than relying on shortcut features such as edge discontinuities, thereby demonstrating strong distributional alignment.

Figure 10 and Figure 11 illustrate the sampled images generated by the ArtiAgent pipeline and its synthesized annotations. The bounding boxes highlight the target patch area where artifacts are injected by the synthesis agent.


Figure 9. **Attention Visualization.** We compare the attention maps of InternVL3.5-8B before (base) and after (fine-tuned) training on ArtiAgent. The fine-tuned model reliably focuses on genuine artifact features across both synthesized images and real-world images.

## A.5. Ablation Studies

To justify the selected configuration of the synthesis agent in Appendix A.1, we conducted ablations on the PE injection steps along with the value injection blocks and visualized their results. Figure 18 shows how the choice of injection steps and value-injecting blocks affects artifact injection quality.

To investigate the effect of injection strength on artifact synthesis, we conducted an ablation study on the number of injection steps during the denoising process. Table 5 shows that VLM performance peaks when artifacts are injected for 15 steps. Downstream performance degrades if the steps are too few (due to failed artifact injection) or too many (due to overall image quality degradation), supporting our choice of 15 injection steps.

Table 5. **Ablation study on injection steps.** Performance of Qwen2.5-VL-7B on ArtiBench binary detection when trained with 1K ArtiAgent samples generated using varying numbers of injection steps out of 25 total steps.

| Steps | Acc | F1 |
|---|---|---|
| 5/25 | 0.513 | 0.377 |
| 10/25 | 0.583 | 0.565 |
| 15/25 | 0.586 | 0.570 |
| 20/25 | 0.540 | 0.477 |

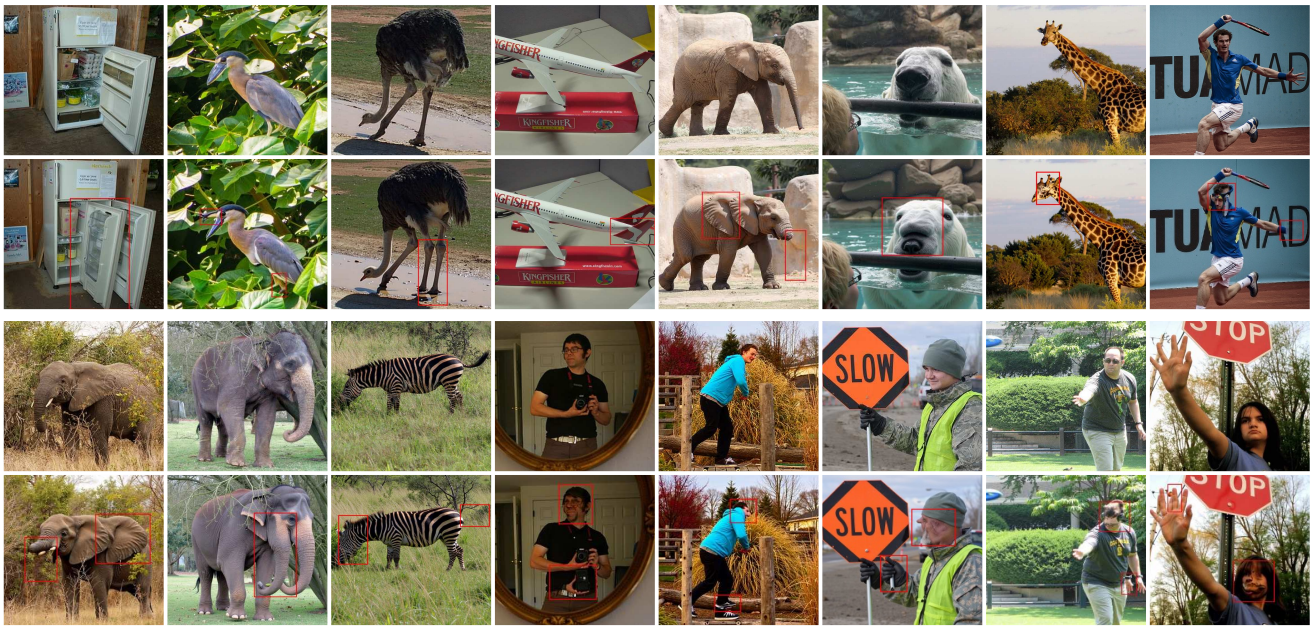



Figure 10. Visualizations of artifacts injected with ArtiAgent. The **first and third rows** show the original images, whereas the **second and fourth rows** show the output images with artifacts injected.

Clean



Artifact


```
“bbox”: [424, 317, 456, 397]
“entity”: “person”
“subentity”: “leg”
“label”: “The person is missing a leg.”
“explanation”: “The image depicts a person surfing, but there is an
unusual absence of one leg, creating an empty space where the leg
should be. This anomaly disrupts the natural appearance of the
scene, as the person appears to be balancing on the surfboard with
only one leg visible.”
”image_caption”: “a person is standing on a surfboard in the ocean.”
```


Figure 11. An instance of ArtiAgent with the annotation.

**<Output Format>**

```
peripheral: {
        "<peripheral_entity1>": ["<peripheral_sub1>", "<peripheral_sub2>", "..."],
        "<peripheral_entity2>": ["<peripheral_sub1>", "<peripheral_sub2>", "..."],
...},
intermediate: {
        "<intermediate_entity1>": ["<intermediate_sub1>", "<intermediate_sub2>", "..."],
        "<intermediate_entity2>": ["<intermediate_sub1>", "<intermediate_sub2>", "..."],
...}
```

**<Hard Rules>**

1) Each returned entity MUST have **at least one** subentity in its dictionary.  b return an empty list for any entity.
2) If you cannot name at least one clearly visible subentity for an entity **in that layer**, omit that entity from that dictionary.
3) Subentities must be **clearly visible** and **reasonably segmentable** in the image.
4) **Exclude** parts that are tightly bound to or visually fused with the torso/core body (e.g., arms pressed to sides, folded wings against body). Only include subentities with clear visual separation.
5) Do **not** invent parts that are occluded, cropped out, or ambiguous.
6) Use concise, lowercase **nouns**; de-duplicate terms. Prefer 1–6 subentities per entity per dictionary.
7) **Return exactly one JSON object** with exactly two top-level keys: "**peripheral**" and "**intermediate**" (not an array, not multiple objects).
8) **Granularity rule (coarsity)**: Choose the **most specific visible entity**. If only a part is clearly visible (e.g., a leg without enough evidence of the full person), output that part as the entity (e.g., "leg") rather than its parent (e.g., "person"). Do **not** infer parent entities that are not clearly visible.
9) **Peripheral variable-cardinality ban**: In the peripheral dictionary, do not include variable-cardinality micro-parts (0..n multiplicity) such as leaves, hairs, feathers, scales, spikes, thorns, grains, pebbles, raindrops, confetti, crowd members, etc. Prefer distal parts with fixed or small bounded counts (e.g., hand, finger, ear, eye, wheel, mirror). If only variable-cardinality micro-parts are visible for an entity, omit that entity from the peripheral dictionary.

**<Layering Guidance>**

- **Peripheral dictionary (index 0)**: Include outermost parts with clear boundaries (e.g., arm, leg, wing, fin, hand, finger, nail, ear, eye, paw, tail, wheel, mirror, antenna).
- For the **peripheral** dictionary, exclude variable-cardinality micro-parts (e.g., leaves, hairs, feathers, scales, spots/pattern dots, raindrops); select fixed-cardinality or bounded-count distal parts instead.
- **Intermediate/Core dictionary (index 1)**: Include mid-level structural parts (e.g., arm, leg, face, door, window).
- If a subentity fits both notions, prefer **peripheral** only if it is clearly distal and separable (e.g., "hand" → peripheral; "arm" → intermediate).
- If a candidate layer has no valid entities with visible subentities, set that key to an empty object `[]` (e.g., `"peripheral": []` or `"intermediate": []`).

Figure 12. Guidance and rules for generating entity-subentity vocabulary sets.

**<User Prompt>**
You are given an image. Identify the visible **entities** and their **subentities**, split into two layers:
- **Peripheral (outermost) subentities** first (e.g., hands, fingers, eyes, ears, paws, tails, wheels, mirrors).
- **Intermediate/Core subentities** second (e.g., arms, legs, face, head, door, window).

Output must return two dictionaries for each peripheral and intermediate types: {OUTPUT FORMAT}

Hard rules: {HARD RULES}
Layering guidance: {LAYERING GUIDANCE}

Clarifications:
- Examples of valid subentities:
  - person → face, arm, hand, leg, foot, ear, eye
  - hand → finger, nail, palm
  - dog → ear, snout, leg, tail, paw
  - car → wheel, door, window, mirror
- Avoid generic torso-like regions. If no fine-grained parts are clearly separable for a candidate entity, **omit the entity** rather than returning an empty list.
- Granularity examples:
  - If only a single **leg** is clearly visible:
    `{"peripheral": {"leg": ["knee", "ankle", "foot"]}, "intermediate": []}`
- Variable-cardinality micro-parts are **not allowed** in the peripheral dictionary. Examples to avoid: leaf/leaves (tree), hair/hairs (person/animal), feather/feathers (bird), scale/scales (fish/reptile), spot/spots (dalmatian), petal/petals (flower), book/books (bookshelf), crowd/persons (crowd scene).

Bad examples (NOT allowed):
- Returning a single array or more than one top-level JSON object
- `{"dog": []}  # empty subentities list`
- Multiple separate top-level JSON objects
- 
```
"peripheral": [{
       "entity": "tree",
       "subentities": ["leaf", "fruit"]
  }]  # variable-cardinality micro-parts in peripheral
```

Good format examples (illustrative only):
```
   { "peripheral": [{"entity": "person","subentities": ["hand", "finger", "leg", "arm"]},
                {"entity": "car","subentities": ["wheel", "mirror"]},
                {"entity": "dog", "subentities": ["ear", "paw", "leg", "tail"]}],
     "intermediate": [{"entity": "person", "subentities": ["face", "palm"]},
                {"entity": "car", "subentities": ["door", "window"]},
                {"entity": "dog", "subentities": ["leg", "face"]}]  }

   { "peripheral": [{"entity": "cat", "subentities": ["ear", "paw", "leg", "tail"]},
                {"entity": "bicycle", "subentities": ["wheel", "pedal"]}],
       "intermediate": [{"entity": "cat", "subentities": ["face"]},
                {"entity": "bicycle", "subentities": ["frame", "seat"]}] }
```

Edge-case examples:
- Only distal parts visible:
  `{"peripheral": [{"entity": "hand", "subentities": ["finger", "nail"]}], "intermediate": []}`
- Only intermediate parts visible:
  `{"peripheral": [], "intermediate": [{"entity": "person", "subentities": ["face", "palm"]}]}`
- Multiple distal entities visible, no intermediate:
```
       {"peripheral": [{"entity": "hand", "subentities": ["finger", "nail"]},
                {"entity": "dog", "subentities": ["ear", "paw", "tail", "leg"]}], "intermediate": []}
```

**<Example>**
Input: {*image*} {USER PROMPT}
Output: 
```
{"peripheral": {"cannon": ["wheels", "barrel opening"]},
 "intermediate": {"cannon": ["barrel", "carriage"]}}
```

Figure 13. Full prompt for generating entity-subentity vocabulary sets.

**Algorithm 1** Add Tool

**Require:** Reference patches $\mathcal{P}_R$, patch grid $(h_p, w_p)$, same-entity foreground $\mathcal{P}_{\text{ent}}$, same-subentity foreground $\mathcal{P}_{\text{sub}}$, ring thickness $\alpha$, distance weight $\lambda_{\text{dist}}$
**Ensure:** target-reference patch mapping $\mathcal{M} = \{(p_t, p_r)\}$

$(c_i, c_j) \leftarrow \textsc{Centroid}(\mathcal{P}_R)$ */* subentity center in patch space */*
$\mathcal{P}_{\text{ring}} \leftarrow \textsc{PerimeterBand}(\mathcal{P}_R, (c_i, c_j), \alpha, h_p, w_p)$ */* candidate centroid patches band around the subentity */*
$S \leftarrow \emptyset$
**for** each $(i, j) \in \mathcal{P}_{\text{ring}}$ **do**
  $\Delta_i \leftarrow i - c_i,\ \Delta_j \leftarrow j - c_j$
  $\mathcal{P}_{\text{shift}} \leftarrow \{(r_i + \Delta_i, r_j + \Delta_j) : (r_i, r_j) \in \mathcal{P}_R\}$
  $r_{\text{self}} \leftarrow \frac{|\mathcal{P}_{\text{shift}} \cap \mathcal{P}_R|}{|\mathcal{P}_{\text{shift}}|}$
  $r_{\text{ent}} \leftarrow \frac{|\mathcal{P}_{\text{shift}} \cap (\mathcal{P}_{\text{ent}} \setminus \mathcal{P}_R)|}{|\mathcal{P}_{\text{shift}}|}$
  $r_{\text{sub}} \leftarrow \frac{|\mathcal{P}_{\text{shift}} \cap \mathcal{P}_{\text{sub}}|}{|\mathcal{P}_{\text{shift}}|}$
  $d_{\text{L1}} \leftarrow |i - c_i| + |j - c_j|$
  $g_{\text{dist}} \leftarrow \frac{1}{1 + \lambda_{\text{dist}} d_{\text{L1}}}$
  $S(i, j) \leftarrow (3 - r_{\text{self}} - r_{\text{ent}} - r_{\text{sub}}) \cdot g_{\text{dist}}$
$(i^\star, j^\star) \leftarrow \arg\max_{(i,j) \in \mathcal{P}_{\text{ring}}} S(i, j)$
$\Delta_i^\star \leftarrow i^\star - c_i,\ \Delta_j^\star \leftarrow j^\star - c_j$
$\mathcal{M} \leftarrow \{((r_i + \Delta_i^\star, r_j + \Delta_j^\star), (r_i, r_j)) : (r_i, r_j) \in \mathcal{P}_R\}$
**return** $\mathcal{M}$

---

**function** $\textsc{Centroid}(\mathcal{P}_R)$
  $n \leftarrow |\mathcal{P}_R|$
  $c_i \leftarrow \frac{1}{n} \sum_{(r_i, r_j) \in \mathcal{P}_R} r_i$
  $c_j \leftarrow \frac{1}{n} \sum_{(r_i, r_j) \in \mathcal{P}_R} r_j$
  $c_i \leftarrow \text{round}(c_i), \quad c_j \leftarrow \text{round}(c_j)$
  **return** $(c_i, c_j)$

---

**function** $\textsc{PerimeterBand}(\mathcal{P}_R, (c_i, c_j), \alpha, h_p, w_p)$
  $\mathcal{P}_{\text{ring}} \leftarrow \emptyset$
  **for** $i \leftarrow 0$ to $h_p - 1$ **do**
    **for** $j \leftarrow 0$ to $w_p - 1$ **do**
      $d_{\text{L1}} \leftarrow |i - c_i| + |j - c_j|$
      **if** $1 \le d_{\text{L1}} \le \alpha$ **then**
        $\mathcal{P}_{\text{ring}} \leftarrow \mathcal{P}_{\text{ring}} \cup \{(i, j)\}$
  **return** $\mathcal{P}_{\text{ring}}$

**Algorithm 2** Remove Tool

**Require:** Subentity patch set $\mathcal{P}_T$, patch grid $(h_p, w_p)$, same-entity foreground $\mathcal{P}_{\text{ent}}$, same-subentity foreground $\mathcal{P}_{\text{sub}}$, neighborhood radius $R$
**Ensure:** Target–reference mapping $\mathcal{M}$

$\mathcal{P}_{\text{nbr}} \leftarrow$ LOCALNEIGHBORHOOD$(\mathcal{P}_T, R, h_p, w_p)$ */* collect nearby non-target patches in the grid */*
$\mathcal{P}_{\text{nbr-no-sub}} \leftarrow \mathcal{P}_{\text{nbr}} \setminus \mathcal{P}_{\text{sub}}$
$\mathcal{P}_{\text{nbr-non-ent}} \leftarrow \mathcal{P}_{\text{nbr}} \setminus \mathcal{P}_{\text{ent}}$
**if** $|\mathcal{P}_{\text{nbr-non-ent}}| > \frac{1}{2}|\mathcal{P}_{\text{nbr-no-sub}}|$ **then**
    $\mathcal{P}_R^{\text{pool}} \leftarrow \mathcal{P}_{\text{nbr-non-ent}}$
**else**
    $\mathcal{P}_R^{\text{pool}} \leftarrow \mathcal{P}_{\text{nbr-no-sub}}$
$\mathcal{M} \leftarrow \emptyset$
**for** each $p_t \in \mathcal{P}_T$ **do**
    $p_r \leftarrow \arg\min_{q \in \mathcal{P}_R^{\text{pool}}} \|p_t - q\|_1$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(p_t, p_r)\}$
**return** $\mathcal{M}$

**function** LOCALNEIGHBORHOOD$(\mathcal{P}_T, R, h_p, w_p)$
    $\mathcal{P}_{\text{nbr}} \leftarrow \emptyset$
    **for** each $(t_i, t_j) \in \mathcal{P}_T$ **do**
        **for** $dy = -R$ to $R$ **do**
            **for** $dx = -R$ to $R$ **do**
                **if** $|dy| + |dx| \le R$ and $(dy, dx) \neq (0, 0)$ **then**
                $p_i \leftarrow t_i + dy$, $p_j \leftarrow t_j + dx$
                **if** $0 \le p_i < h_p$ and $0 \le p_j < w_p$ and $(p_i, p_j) \notin \mathcal{P}_T$ **then**
                $\mathcal{P}_{\text{nbr}} \leftarrow \mathcal{P}_{\text{nbr}} \cup \{(p_i, p_j)\}$
    **return** $\mathcal{P}_{\text{nbr}}$

**Algorithm 3** Distort Tool

**Require:** Subentity patches $\mathcal{P}_T$, patch grid $(h_p, w_p)$, same-entity patches $\mathcal{P}_{\text{ent}}$, kernel type $k \in \{\text{shuffle}, \text{jitter}, \text{strip}\}$
**Ensure:** target-reference patch mapping $\mathcal{M} = \{(p_t, p_r)\}$

**if** $k =$ shuffle **then**
  $\mathcal{P}_R \leftarrow \textsc{ShuffleKernel}(\mathcal{P}_T)$ */* randomly permute subentity patches */*
**else if** $k =$ jitter **then**
  $\mathcal{P}_R \leftarrow \textsc{GaussianJitterKernel}(\mathcal{P}_T, \sigma, h_p, w_p, \mathcal{P}_{\text{ent}})$ */* locally jitter patches within the entity */*
**else if** $k =$ strip **then**
  $\mathcal{P}_R \leftarrow \textsc{StripShiftingKernel}(\mathcal{P}_T, S, h_p, w_p)$ */* shift patches along strips of the subentity */*
$\mathcal{M} \leftarrow \{(\mathcal{P}_T[i], \mathcal{P}_R[i]) : i = 1, \ldots, |\mathcal{P}_T|\}$
**return** $\mathcal{M}$

---

**function** $\textsc{ShuffleKernel}(\mathcal{P}_T)$
  $\mathcal{P}_R \leftarrow \mathcal{P}_T$
  $\textsc{RandomShuffle}(\mathcal{P}_R)$
  **return** $\mathcal{P}_R$

---

**function** $\textsc{GaussianJitterKernel}(\mathcal{P}_T, \sigma, h_p, w_p, \mathcal{P}_{\text{ent}})$
  $\mathcal{P}_R \leftarrow \emptyset$
  **for** each $(p_y, p_x) \in \mathcal{P}_T$ **do**
    found $\leftarrow$ False
    **for** $a \leftarrow 1$ to $A_{\max}$ **do**
      $\delta_y \sim \mathcal{N}(0, \sigma^2), \quad \delta_x \sim \mathcal{N}(0, \sigma^2)$
      $n_y \leftarrow \text{clip}(\text{round}(p_y + \delta_y), 0, h_p - 1)$
      $n_x \leftarrow \text{clip}(\text{round}(p_x + \delta_x), 0, w_p - 1)$
      **if** $\mathcal{P}_{\text{ent}} = \emptyset$ **or** $(n_y, n_x) \in \mathcal{P}_{\text{ent}}$ **then**
        $\mathcal{P}_R \leftarrow \mathcal{P}_R \cup \{(n_y, n_x)\}$
        found $\leftarrow$ True
        **break**
    **if not** found **then**
      $(n_y, n_x) \leftarrow \textsc{NearestForegroundOrSelf}(p_y, p_x, \mathcal{P}_{\text{ent}})$
      $\mathcal{P}_R \leftarrow \mathcal{P}_R \cup \{(n_y, n_x)\}$
  **return** $\mathcal{P}_R$

---

**function** $\textsc{StripShiftingKernel}(\mathcal{P}_T, S, h_p, w_p)$
  Compute bounding box of $\mathcal{P}_T$ in patch space
  Determine direction $d \in \{\text{vertical}, \text{horizontal}\}$ from aspect ratio
  Partition $\mathcal{P}_T$ into $S$ strips $\{\mathcal{S}_1, \ldots, \mathcal{S}_S\}$ along $d$
  For each strip $\mathcal{S}_s$, sort patches to obtain an order $(p_1^{(s)}, \ldots, p_{n_s}^{(s)})$
  Choose integer strip shifts $\{\Delta_s\}_{s=1}^{S}$ (alternating signs, increasing magnitudes)
  $\mathcal{P}_R \leftarrow$ list of length $|\mathcal{P}_T|$
  **for** $s \leftarrow 1$ to $S$ **do**
    $\Delta \leftarrow \Delta_s$ converted to patch units (circular shift)
    **for** $u \leftarrow 1$ to $n_s$ **do**
      $v \leftarrow 1 + ((u + \Delta - 1) \bmod n_s)$
      Set reference of $p_u^{(s)}$ to $p_v^{(s)}$ in $\mathcal{P}_R$
  **return** $\mathcal{P}_R$

**Algorithm 4** Fuse Tool (Main)

**Require:** Entity A patches $\mathcal{P}_A$, entity B patches $\mathcal{P}_B$, patch grid $(h_p, w_p)$, band radius $R$, max offset $R_{\text{off}}$, number of seeds $K$
**Ensure:** target-reference patch mapping $\mathcal{M} = \{(p_t, p_r)\}$

$\mathcal{P}_{\text{overlap}} \leftarrow \mathcal{P}_A \cap \mathcal{P}_B$
**if** $\mathcal{P}_{\text{overlap}} = \emptyset$ **then**
    **return** $\emptyset$
$\mathcal{P}_{\text{fg}} \leftarrow \mathcal{P}_A \cup \mathcal{P}_B$
$\mathcal{P}_{A \setminus B} \leftarrow \mathcal{P}_A \setminus \mathcal{P}_{\text{overlap}}, \quad \mathcal{P}_{B \setminus A} \leftarrow \mathcal{P}_B \setminus \mathcal{P}_{\text{overlap}}$
$\mathcal{P}_T \leftarrow \textsc{OverlapFusionBand}(\mathcal{P}_{\text{overlap}}, \mathcal{P}_{\text{fg}}, R, h_p, w_p)$ */* build foreground band around the overlap */*
**if** $\mathcal{P}_T = \emptyset$ **then**
    **return** $\emptyset$
$\mathcal{S} \leftarrow \textsc{FarthestPointSampling}(\mathcal{P}_T, K)$ */* choose seeds that cover the band */*
Initialize $\{\mathcal{R}_s\}_{s \in \mathcal{S}}$ as empty sets
**for** each $p \in \mathcal{P}_T$ **do**
    $s^\star \leftarrow \arg\min_{s \in \mathcal{S}} \|p - s\|_1$
    $\mathcal{R}_{s^\star} \leftarrow \mathcal{R}_{s^\star} \cup \{p\}$
$\Omega \leftarrow \{(\Delta_i, \Delta_j) : 1 \le |\Delta_i| + |\Delta_j| \le R_{\text{off}}\}$
$\mathcal{M} \leftarrow \emptyset$
**for** each seed $s \in \mathcal{S}$ **do**
    $\mathcal{R} \leftarrow \mathcal{R}_s$
    **if** $\mathcal{R} = \emptyset$ **then**
        **continue**
    $\mathcal{P}_{\text{opp}} \leftarrow \textsc{OppositeRegion}(s, \mathcal{P}_{A \setminus B}, \mathcal{P}_{B \setminus A}, \mathcal{P}_{\text{fg}}, \mathcal{P}_T)$ */* decide which side to fuse from */*
    $\Delta^\star \leftarrow \textsc{BestOffset}(\mathcal{R}, \mathcal{P}_{\text{opp}}, \Omega, h_p, w_p, \mathcal{P}_T)$ */* find best shared shift for this region */*
    **for** each $p \in \mathcal{R}$ **do**
        $p_r \leftarrow \textsc{OffsetOrNearest}(p, \Delta^\star, \mathcal{P}_{\text{opp}}, h_p, w_p, \mathcal{P}_T)$ */* apply offset or nearest opposite patch */*
        $\mathcal{M} \leftarrow \mathcal{M} \cup \{(p, p_r)\}$
**return** $\mathcal{M}$

**Algorithm 5** Fuse Tool (Helpers)

**function** OVERLAPFUSIONBAND($\mathcal{P}_{\text{overlap}}, \mathcal{P}_{\text{fg}}, R, h_p, w_p$)
  $\mathcal{P}_T \leftarrow \emptyset$
  **for** each $(o_i, o_j) \in \mathcal{P}_{\text{overlap}}$ **do**
    **for** $dy \leftarrow -R$ to $R$ **do**
      **for** $dx \leftarrow -R$ to $R$ **do**
        **if** $|dy| + |dx| \leq R$ **then**
          $v_i \leftarrow o_i + dy, \quad v_j \leftarrow o_j + dx$
          **if** $0 \leq v_i < h_p$ **and** $0 \leq v_j < w_p$ **and** $(v_i, v_j) \in \mathcal{P}_{\text{fg}}$ **then**
            $\mathcal{P}_T \leftarrow \mathcal{P}_T \cup \{(v_i, v_j)\}$
  **return** $\mathcal{P}_T$

**function** FARTHESTPOINTSAMPLING($\mathcal{P}, K$)
  **if** $\mathcal{P} = \emptyset$ **then**
    **return** $\emptyset$
  $K \leftarrow \min(K, |\mathcal{P}|)$
  Choose initial seed $s_1 \in \mathcal{P}$ (e.g., closest to centroid)
  $\mathcal{S} \leftarrow \{s_1\}; \quad d(p) \leftarrow \|p - s_1\|_1$ for all $p \in \mathcal{P}$
  **for** $m \leftarrow 2$ to $K$ **do**
    $s_m \leftarrow \arg\max_{p \in \mathcal{P}} d(p)$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{s_m\}$
    **for** each $p \in \mathcal{P}$ **do**
      $d(p) \leftarrow \min\{d(p), \|p - s_m\|_1\}$
  **return** $\mathcal{S}$

**function** OPPOSITEREGION($s, \mathcal{P}_{A \setminus B}, \mathcal{P}_{B \setminus A}, \mathcal{P}_{\text{fg}}, \mathcal{P}_T$)
  $d_A \leftarrow +\infty, \quad d_B \leftarrow +\infty$
  **if** $\mathcal{P}_{A \setminus B} \neq \emptyset$ **then**
    $d_A \leftarrow \min_{a \in \mathcal{P}_{A \setminus B}} \|s - a\|_1$
  **if** $\mathcal{P}_{B \setminus A} \neq \emptyset$ **then**
    $d_B \leftarrow \min_{b \in \mathcal{P}_{B \setminus A}} \|s - b\|_1$
  **if** $d_A < d_B$ **then**
    **return** $\mathcal{P}_{B \setminus A}$
  **else if** $d_B < d_A$ **then**
    **return** $\mathcal{P}_{A \setminus B}$
  **else**
    **return** $\mathcal{P}_{\text{fg}} \setminus \mathcal{P}_T$

**function** BESTOFFSET($\mathcal{R}, \mathcal{P}_{\text{opp}}, \Omega, h_p, w_p, \mathcal{P}_T$)
  $\Delta^\star \leftarrow$ None, $\quad m^\star \leftarrow 0$
  **for** each $(\Delta_i, \Delta_j) \in \Omega$ **do**
    count $\leftarrow 0$
    **for** each $(v_i, v_j) \in \mathcal{R}$ **do**
      $r_i \leftarrow v_i + \Delta_i, \quad r_j \leftarrow v_j + \Delta_j$
      **if** $0 \leq r_i < h_p$ **and** $0 \leq r_j < w_p$ **and** $(r_i, r_j) \in \mathcal{P}_{\text{opp}}$ **and** $(r_i, r_j) \notin \mathcal{P}_T$ **then**
        count $\leftarrow$ count $+ 1$
    **if** count $> m^\star$ **then**
      $m^\star \leftarrow$ count, $\quad \Delta^\star \leftarrow (\Delta_i, \Delta_j)$
  **return** $\Delta^\star$

**function** OFFSETORNEAREST($p, \Delta^\star, \mathcal{P}_{\text{opp}}, h_p, w_p, \mathcal{P}_T$)
  $(v_i, v_j) \leftarrow p$
  **if** $\Delta^\star \neq$ None **then**
    $(\Delta_i, \Delta_j) \leftarrow \Delta^\star$
    $r_i \leftarrow v_i + \Delta_i, \quad r_j \leftarrow v_j + \Delta_j$
    **if** $0 \leq r_i < h_p$ **and** $0 \leq r_j < w_p$ **and** $(r_i, r_j) \in \mathcal{P}_{\text{opp}}$ **and** $(r_i, r_j) \notin \mathcal{P}_T$ **then**
      **return** $(r_i, r_j)$
  **return** $\arg\min_{u \in \mathcal{P}_{\text{opp}}} \|(v_i, v_j) - u\|_1$

## Duplication

**<Type Description>**
Duplication artifacts occur when a part of an object is duplicated and placed adjacent to the original, creating anatomically or structurally implausible duplications (e.g., extra fingers, duplicate ears, duplicate wheels).

**<Detection Criteria>**
• **Case A** — Object present in target (Image 2): If the second image already contains the specified object, then the third image must show a **plausible duplication** or **additional instance** of the object. The new instance should be **distinct** from the original.
• **Case B** — No object in target (Image 2): If the second image does **not** contain the object, the third image must show a **clearly new instance** with a **distinct boundary/contour**.
Reject if there is only subtle texture change, brightness shift, or local warping rather than a new instance.

**<Explanation Instructions>**
Focus on cues like duplicated parts, unnatural growths, or extra elements that conflict with normal anatomy or structure.

## Omission

**<Type Description>**
Omission artifacts occur when a part of an object is deleted and the area is inpainted with background, resulting in missing features or gaps where something should be present (e.g., missing fingers, absent ears).

**<Detection Criteria>**
• **Definitive removal evidence** : stump/ termination cues, disrupted silhouette, hollow/negative space, texture continuation/inpainting traces, mismatched shadows/reflections, or symmetry break.
• **Ambiguity/occlusion rule**: If the missing part could plausibly be merely **hidden**, set `has_artifact = false`.
• **Anatomical plausibility check**: If the scene still reads as anatomically correct and a typical pose could hide the part, set `has_artifact = false`.

**<Explanation Instructions>**
Focus on cues like missing structure, unnatural gaps, smoothed-over areas, or anatomical discontinuity where something appears to be absent.

## Distortion

**<Type Description>**
Distortion artifacts occur when parts are warped, creating unnatural geometry, irregular textures, or visual blending errors.

**<Explanation Instructions>**
Focus on cues like warped shapes, unnatural geometry, irregular textures, or visual blending errors that make the structure appear broken or malformed.

## Fusion

**<Type Description>**
Fusion artifacts occur when parts or distinct entities are unnaturally merged together, creating blurred boundaries, overlapped textures, or structural entanglement (e.g., two animals merged into one).

**<Detection Criteria>**
• **Boundary visibility comparison**: Compare Image 2 and Image 3. If a clear, continuous boundary between the two objects remains visible in Image 3 (similar to Image 2), set `has_artifact = false`.
• **Fusion cues needed**: boundary loss/softening across seams; cross-object texture/color blending; geometry interpenetration; inconsistent occlusion ordering.
• **Not fusion**: mere blur, minor warping, or lighting change that preserves recognizable boundary.

**<Explanation Instructions>**
Focus on cues like boundary loss/softening across seams, cross-object texture/color bleed, geometry interpenetration, and inconsistent occlusion ordering.

Figure 14. Type descriptions and instructions by artifact type.

**<User Prompt>**
You are an expert at detecting `[duplication, omission, fusion]`–type artifacts in AI-generated images.

## Type Description

You will be shown :
1. An original image without the target region
2. An original image with only the target region
3. An artifact image with only the target region
4. The object name that may be `[added, removed, fused]`

Your task is to determine if there is a successful `[duplication, omission, fusion]` artifact in the third image.

Detection criteria (be strict): ## Detection Criteria

Return your analysis in the following format:
`"has_artifact": true/false` (whether the artifact is successfully present, always true for distortion)

Important:
- Focus on visible evidence in the target region

<Example>
Input: {*images*} {USER PROMPT[`'addition'`]}

OUTPUT: "has_artifact": false

Figure 15. Full prompt and example for data filtering.

**<User Prompt>**
You are an expert at describing `[duplication, omission, distortion, fusion]`–type artifacts in AI-generated images.

## Type Description

You will be shown :
1. An original image without the target region
2. An original image with only the target region
3. An artifact image with only the target region
4. The object name that may be `[added, removed, fused, distorted]`

Your task is to describe what looks wrong or unnatural in the target region.

For explanation: ## Explanation Instructions

Return your analysis in the following format:

```
  "explanation": "Detailed description of what looks wrong in the region, without reasoning about the artifact type or how the artifact is created."
    "label": "Brief description of the artifact (empty string if no artifact)"
```

Important:
- Focus on visible evidence in the target region
- Use simple, clear language for explanations
- Do not refer to images by number; say "the region" or "the area" instead
- Make explanations understandable for non-experts

**<Examples>**
Input: {*images*} {USER PROMPT}

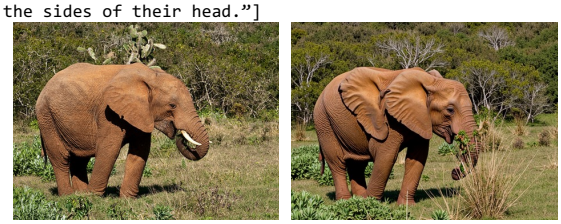
Output: `“explanation”: “The elephant has an extra ear on its back, which is not naturally possible. Elephants typically have two ears on the sides of their head.”, “label”: “duplicated ear”`

Figure 16. Full prompt and example for regional explanation generation.

**<User Prompt>**
You are an image artifact analyst. You will be given an image with artifacts and a list of injected artifact annotations, in the format of bbox:<(xmin, ymin, xmax, ymax)> description:<description of the artifact in that bbox region>.

Your job: Read ALL artifact annotations and write a single, holistic explanation the explanation of the anomalies in the image.
Guidance:
- Do NOT mention coordinates or the term "bbox". Use the annotations only to understand what is wrong.
- Use commonsense knowledge about typical anatomy/structure (e.g., zebras normally have four legs).
- If multiple issues appear, summarize the combined effect coherently rather than listing them mechanically.
- Keep it concise (2-3 sentences) and easy to understand. Avoid making concluding sentences. Just focus on explaining the abnormality.

<Examples>
Input: {*images*} {USER PROMPT}
`[“bbox: [352, 240, 416, 336] description: "The elephant's trunk appears to blend unnaturally with the surrounding vegetation, creating a confusing visual where the trunk's shape is not clearly defined. The tusk also seems partially obscured by the blending, making it look irregular.", “bbox: [128, 160, 240, 288] description: 'The elephant has an extra ear on its back, which is not naturally possible. Elephants typically have two ears on the sides of their head.”]`



Output: `The image shows an elephant with an unusual anomaly where an extra ear is positioned on its back, which is not typical for elephants. Additionally, the trunk appears to blend with the surrounding vegetation, causing it to lose its distinct shape and making the tusk appear irregular.`

Figure 17. Full prompt and example for global explanation generation.

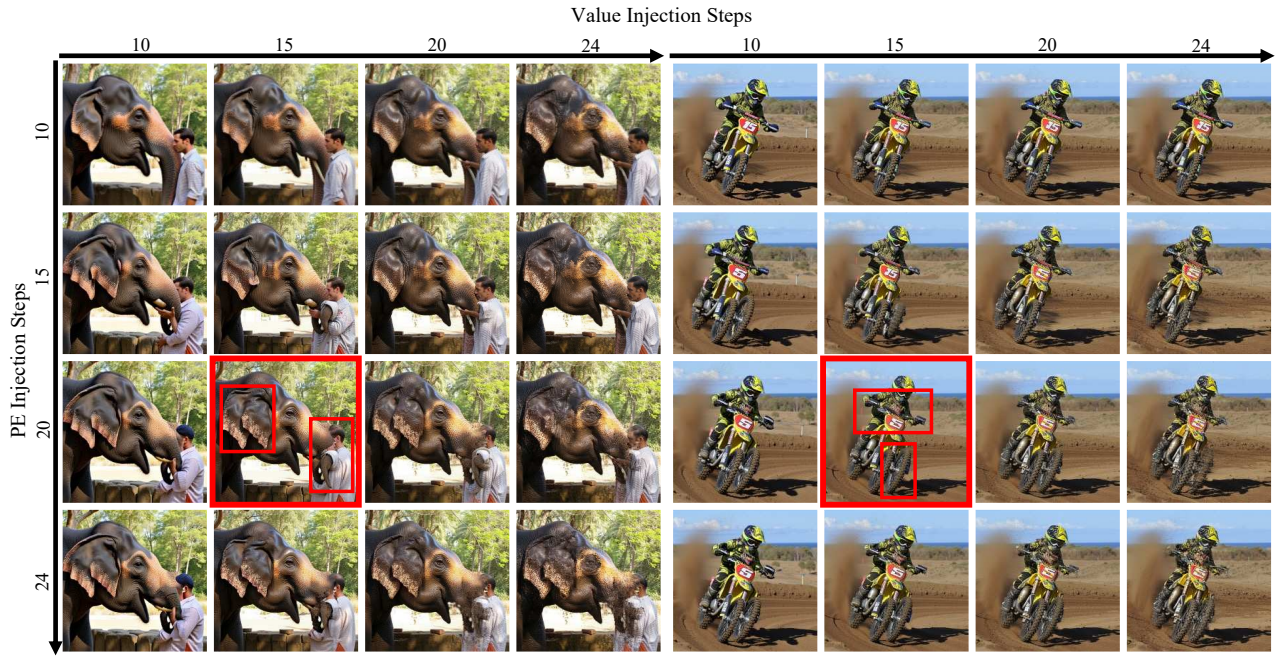



Figure 18. Hyperparameter study on PE injection and value injection steps (1). The image in the red box shows our selected configuration.

## B. Benchmarks and Evaluation Protocols (§ 5 & § 6.1)

### B.1. Benchmark Datasets (§ 5)

Table 6 shows the fully cited artifact benchmark datasets and the sources used for generation. Details on the dataset's metadata formats for artifact region representations and image explanations are elaborated below.

- **RichHF.** The RichHF dataset, with 995 images sampled and annotated from the Pick-a-Pic dataset, provides the artifact map in a heatmap format, highlighting the probability of abnormal regions in the image. Annotations and scores are generated by the trained multi-modal transformer to predict human perception-aligned feedback.
- **LOKI.** The LOKI benchmark is a multi-modal synthetic-data detection dataset, covering image, video, text, audio, and 3D content, and comprises roughly 18,000 curated questions across 26 subcategories. It includes coarse real vs. synthetic judgments, multiple-choice detection, anomaly / artifact region selection, and explanation tasks. We use the image artifact subset of the dataset, which consists of 229 images.
- **SynthScars.** The SynthScars dataset consists of 12,236 fully synthetic images across four distinct content types (Human, Object, Scene, Animal) and three artifact categories (Physics, Distortion, Structure). Each image is annotated with pixel-level segmentation masks delimiting artifact regions, detailed textual explanations of the artifact(s), and artifact-category labels.

**Visualizations.** Visualizations of the evaluated benchmarks, including ArtiBench, are shown in Figure 19. It is clearly visible that even though our benchmark dataset shows overall better quality of image generation compared to the previous benchmarks, image artifacts are still visible in the generated images. We emphasize the importance of our timely benchmark that successfully represents structural artifacts remaining in the most recent diffusion models.

### B.2. ArtiBench Generation (§ 5)

**Image Generation.** ArtiBench is built on a set of images generated by state-of-the-art diffusion models, reflecting the most timely artifacts appearing in current image generation models. We use five different models for image generation and three reliable prompt sources. After generating a set of images with the five models from randomly sampled prompts, the images were annotated according to the strictly guided annotation process.

**Annotation Pipeline.** With the set of diffusion-generated images, we construct **ArtiBench** following a guided 4-step annotation process: (1) classification, (2) bounding box labeling, (3) explanation generation, and (4) expert curation.


Figure 19. Artifact examples of the four benchmarks, listed in the order of publishment.

- **Classification.** Annotators are asked to meticulously observe the images and determine whether there are any artifacts visible in the image. The caption used for T2I generation is provided with the image for better understanding of generative intentions and respecting the model's capability to follow instructions on generating abnormalities. All images classified as having artifacts proceed to the next step, where they are shuffled and redistributed to the annotators to mitigate human bias in the examination process.
- **Bounding Box Labeling.** For the artifact-containing images from the classification step, annotators are expected to generate bounding boxes, identify the type of artifact, and write a short description about what abnormalities exist in the specified region.
- **Explanation Generation.** The final annotations are used as the input of the VLM query to generate a comprehensive and polished explanation for the full image. With the prompt used for generation and the list of pairs of bounding box and captions as the input, the VLM is required to output a global explanation of the image regarding all artifacts mentioned in the captions.
- **Expert Curation.** With the set of images and the metadata completed, we curate the images to build the final version of our 1K benchmark. Images are carefully selected to ensure a balanced set that provides a clear and straightforward view of plausible artifacts.

Table 6. **Comparison of artifact benchmark datasets, their generative sources, and evaluation tasks.** Highlighted entries denote dataset sources that were reused by subsequent benchmarks.

| Benchmark | Generative Sources | Sample | Bin. | Loc. | Exp. |
|---|---|---|---|---|---|
| RichHF [27] | SD2.1 [42], SDXL [38], Dreamlike Photoreal [10] | 955 | | ✓ | |
| LOKI [51] | FLUX [5], SD1.4–2.1 [42], Midjourney [30], StyleGAN [22], pix2pix [18], CUT [36] | 229 | | ✓ | ✓ |
| SynthScars [20] | RichHF [27], Chameleon [50], Midjourney [30], DALL·E 3 [32], SD1.x [42] | 1K | | ✓ | ✓ |
| ArtiBench | SD3.5 [11], FLUX [5], Qwen-Image [49], Nano-Banana [13] | 1K | ✓ | ✓ | ✓ |

## B.3. Evaluation Protocol (§ 6.1)

### B.3.1. Binary Classification

- **Accuracy.** The rate of true predictions is measured to show basic performance of accurate classification.
- **Macro F1.** In addition, we use the macro F1 score, to achieve fair comparison across unbalanced datasets and biased predictions. F1 scores are relatively calculated for positive and negative samples to be averaged, exposing the model's capability to precisely capture both artifact presence and absence without random guessing.

### B.3.2. Localization

We alleviate the unfairness between divergent representations of artifact regions, including bounding boxes, polygonal segmentation maps, and heatmaps, by mapping all representations to pixel-wise binary maps.

- **IoU.** With the binary maps obtained, we calculate the pixel-wise IoU between the foreground region predictions and ground-truth areas. IoU scores prefer tight and highly overlapping predictions, providing intuitions on precise predictions but highly possible of penalizing bounding box representations for over-prediction.
- **F1.** Pixel-wise predictions are measured to capture true predictions on a fine-grained basis. Unlike the macro F1 score used in binary detection tasks, F1 scores are measured only for *positive* ground truth regions to prioritize true detections. With each pixel prediction classified, we use the pixel count for true positive (TP), false positive (FP), and false negative (FN) predictions to use them for calculating the F1 score. This method lessens the penalty on loose region representations of bounding boxes over segmentation maps or heatmaps.

### B.3.3. Explanation

- **ROUGE-L.** The ROUGE-L score shows the proportion of the longest overlapping phrase among the full data. This captures the words or phrases that focus on specific artifact regions and objects, with higher scores showing that the model better understands the visual artifact.
- **CSS.** Cosine similarity was measured on sentence embeddings generated by sentence-transformers [40]. CSS portrays the general similarity in context between the descriptions regarding the full scenery's plausibility.

### B.3.4. Evaluation Prompt of VLMs

Figure 20 shows the specific prompts used for the evaluation on VLMs. Artifact priors are shared for all tasks, providing understanding of the structural artifacts we are focusing on.

**<Artifact Prior>**
Image artifacts refer to unintended, implausible, or visibly corrupted regions within images generated by diffusion models. These artifacts often break the natural semantics or visual coherence of an image, such as a person with extra fingers, a car with warped wheels, or missing parts of animals, and can significantly degrade image quality or realism. Artifacts are a critical concern in both model evaluation and training.

There are four types of image artifacts: Duplication, Omission, Distortion, and Fusion.

1. Duplication: These artifacts appear as excessive or repeated components, leading to unrealistic objects or implausible duplication.

2. Omission: These artifacts appear as omitted components, leading to unrealistic or incomplete objects.

3. Distortion: These artifacts occur when objects have details that are not typical, making the object unrecognizable or visually broken, such as geometric inconsistencies, abnormal textures, unnatural asymmetry or twisted, warped, scrambled parts.

4. Fusion: These artifacts result from the combination of objects so their boundaries or interiors merge into one ambiguous, hybrid part, resulting in a visually incoherent region.

**<User Prompt – Binary Classification>**
{ARTIFACT PRIOR}
Your task is to identify if there are any artifacts within this image.
Are there any artifacts in this image? Use lowercase "true" or "false" (boolean values).

**<User Prompt – Localization>**
{ARTIFACT PRIOR}
Identify and localize artifact regions in this image.
For each artifact found, provide bounding box coordinates [x_min, y_min, x_max, y_max] where coordinates are in pixels.
Do NOT output multiple bboxes that indicate the same region

**<User Prompt – Explanation>**
{ARTIFACT PRIOR}
Analyze the artifacts visible in this image and provide a comprehensive explanation.
If no artifacts are found, state "There are no artifacts in this image." If the image is inaccessible, state: "Image not available for analysis."

Figure 20. Evaluation prompts for VLMs.

### B.4. VQA Dataset Structure (§ 6.1)

This section presents the structure of our multi-turn visual question answering (VQA) dataset, which provides the supervision used to train VLMs in artifact detection, localization, and explanation. Each data instance is derived from a paired clean–artifact image generated by ArtiAgent with the synthesized annotations as in Figure 11, enabling the construction of tightly aligned conversations that elicit the model's ability to identify normal content and reason about artifact regions.

#### B.4.1. VQA Template

Each ArtiAgent instance yields two conversations: one for the clean reconstruction image and one for the corresponding artifact image. Tables 7 and 8 summarize the question-answer templates implemented across these two settings.

| Task | Template |
|---|---|
| Bin. | Q: Does this image contain any visual artifacts?<br>A: No. |
| Loc. 1 | Q: Locate the ⟨entity⟩'s ⟨subentity⟩.<br>A: ⟨bbox⟩ |
| Loc. 2 | Q: Is there a ⟨entity⟩'s ⟨subentity⟩ in ⟨bbox⟩?<br>A: Yes / No |
| Exp. | Q: Describe the clean image.<br>A: ⟨image_caption⟩ |

Table 7. VQA templates for clean images.

| Task | Template |
|---|---|
| Bin. | Q: Does this image contain any visual artifacts?<br>A: Yes. |
| Loc. 1 | Q: Provide bounding boxes for all artifact regions.<br>A: [⟨bbox⟩] |
| Loc. 2 | Q: Explain why region ⟨bbox⟩ is an artifact.<br>A: ⟨label⟩ |
| Exp. | Q: Describe all artifacts in the image.<br>A: ⟨explanation⟩ |

Table 8. VQA templates for artifact-injected images.

#### B.4.2. VLM Training Configuration

We train the VLM using a two-stage supervised fine-tuning setup. We use Qwen2.5-VL-7B-Instruct [4] and Intern3.5-VL-8B [57] as the backbone, and use the identical configurations. In the first stage, we fine-tune only the language model and multi-modal projector while keeping the vision encoder frozen, using a learning rate of $1\times10^{-5}$, batch size 64, cosine decay scheduling, and one training epoch. In the second stage, we unfreeze the vision encoder and continue training with a smaller learning rate of $1\times10^{-6}$, with 200 steps. Both stages use the same VQA dataset.

## C. Mitigating Artifacts in Diffusion Models (§ 6.2.1 & § 6.2.2)

### C.1. Reward-guided generation (§ 6.2.1)

**Verifier Training.** We train an artifact verifier modeled with Bradley-Terry model [6] to score images by how likely they are to be artifact-free. The model is trained to assign higher scores to the clean image and lower scores to the artifact-injected image. The verifier uses a frozen ViT-B/16 encoder with a lightweight MLP head, and is optimized using AdamW (learning rate $1\times10^{-3}$, weight decay $1\times10^{-4}$, L2 regularization $1\times10^{-4}$). Training uses a batch size of 32 for 5 epochs. Each batch additionally includes two cross-scene negative pairs to encourage content-agnostic ranking.

**Test-Time Scaling.** At inference time, we use the verifier as a reward model in a compute-scaled best-of-$N$ sampling procedure [28]. For each prompt, round $r$ samples $2^r$ independent latent noises, generates all corresponding images, and evaluates them with the verifier. The highest-scoring image is retained, and the search space doubles in the next round. This random-search strategy expands the candidate pool exponentially, enabling the diffusion model to reliably discover images with fewer artifacts without modifying the weights of the model.

### C.2. Image Correction (§ 6.2.2)

In our artifact-correction pipeline, all VLM interactions are carried out using the Qwen2.5-VL-7B model fine-tuned with ArtiAgent supervision. This unified VLM is responsible for localizing artifact regions, generating artifact-free captions, and verifying whether the corrected content remains flawed. We describe each of the three prompts used in the loop below. The actual prompt is shown in Table 9.

| Prompt | Instruction Text |
|---|---|
| Localization | "Provide the bounding box for the artifact region. Format as [x_min, y_min, x_max, y_max]. Only output the bounding box." |
| Captioning | "Describe what the image would look like if it had no artifacts. Provide a short caption of the clean scene." |
| Verifying | "Is there an artifact within ⟨B⟩? Answer Yes or No." |

Table 9. Prompts used by the VLM in the artifact-correction loop.

**Localization prompt $p_{\text{localize}}$.** This prompt instructs the VLM to identify the spatial extent of the artifact within the input image. The model is asked to produce a single

**Algorithm 6** VLM-Guided Artifact Correction Loop

**Require:** Image $I$ containing at least one artifact

/* localize artifact region */
$B \leftarrow \text{VLM}(I, p_{\text{localize}})$
/* generate caption of the image */
$c \leftarrow \text{VLM}(I, p_{\text{caption}})$
**while** true **do**
    /* Inpaint region $B$ using caption $c$ */
    $I \leftarrow \text{Inpaint}(I, B, c)$
    /* check artifact existence inside $B$ */
    $r \leftarrow \text{VLM}(I, p_{\text{verify}})$
    **if** not $r$ **then**
        **return** $I$

bounding box formatted as `[x_min, y_min, x_max, y_max]` without additional commentary. This strict output format ensures deterministic parsing. The resulting bounding box $B$ is computed once at the beginning of the procedure and kept fixed across all subsequent iterations, providing spatial stability for the iterative refinement process.

**Captioning prompt $p_{\text{caption}}$.** To guide inpainting, the VLM is prompted to describe the overall image without specifying the artifacts in the image. Rather than summarizing the corrupted content, the model is explicitly instructed to generate a short caption that reflects the intended clean version of the scene. This caption serves as the semantic conditioning signal for the FLUX inpainting pipeline [3], which synthesizes corrected content inside the region $B$.

**Verifying prompt $p_{\text{verify}}$.** After each inpainting step, the VLM is queried to assess whether the corrected region $B$ still contains an artifact. The prompt restricts the judgment to the localized region, returning a binary response (`Yes` or `No`). This local verification prevents the algorithm from drifting to unrelated image regions and directly determines whether the loop terminates or continues.

Together, these three prompts coordinate the interaction between the ArtiAgent-trained VLM and the FLUX inpainting pipeline: the VLM identifies the corrupted area, provides semantic guidance for correction, and validates the result, while the inpainting model performs the pixel-level repair. This iterative coupling enables consistent and stable reduction of visual artifacts while preserving overall image semantics. The whole procedure is shown in Algorithm 6.