7. Appendix

7.1. Flow-Int Training Details

The Flow-Int version evaluated in Section 5.1 was trained leveraging flow matching [30], control conditions [3, 32] and latent perceptual loss [3]. The training dataset consisted of image-caption pairs including ImageNet [12], CC12M [4], YFCC [40], and an internally licensed dataset. The version of Flow-Int evaluated in Section 5.3 was trained with only CC12M [4].

7.2. Examples of model generalization capacity analysis through DIMCIM

Here are a few more examples to show how we use COCO-DIMCIM to find interesting insights about different models' generalization capacity.

- Figure 8 and Figure 7 has examples that show some unique failure modes found through COCO-DIMCIM. In some cases larger size models struggle to generate an attribute that is easily generated by smaller size models (like "broken" *umbrella* and "foal" in Figure 7 and "on land" *boat* location and "curly" *dog* in Figure 8). While in some cases, all of the models struggle ("inverted" *bowls* in Figure 8), in a peculiar case, LDM2.1 [34] struggles to generate "empty" *refrigerators* in Figure 7.
- Figure 9 has examples that show that negations are difficult for all models. When these models are prompted to not to generate an attribute, more often than not they generate those attributes in the images.
- Figure 10 has examples that show that some attribute types like *material*, *patterns* and *dog breeds* are easy for all models. All the models have high Can-It Metric for attributes of these types and can generate these attributes in images when prompted.

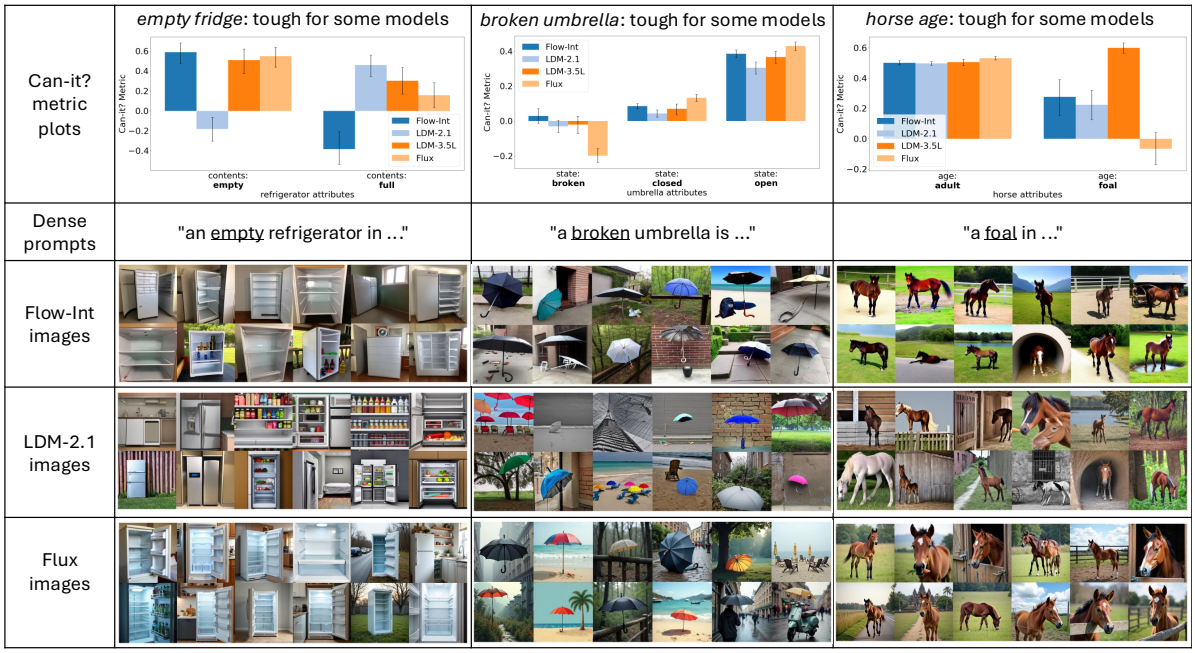

Figure 7. **Interesting generalization failure modes found through COCO-DIMCIM: [Left]** LDM2.1 [34] struggles to generate empty refrigerators even when prompted to do so. **[Middle]** Most models struggle to generate "broken" *umbrellas*, but FLUX.1-dev [27] is especially poor at it. FLUX.1-dev has a very low negative Can-It Metric for "broken" and most of its generated images have completely unbroken umbrellas as seen above. **[Right]** Smaller models (Flow-Int and LDM2.1) are better at generating young *horses*, *e.g.* "foals", than FLUX.1-dev (as observed through the Can-It Metric and sampled images).

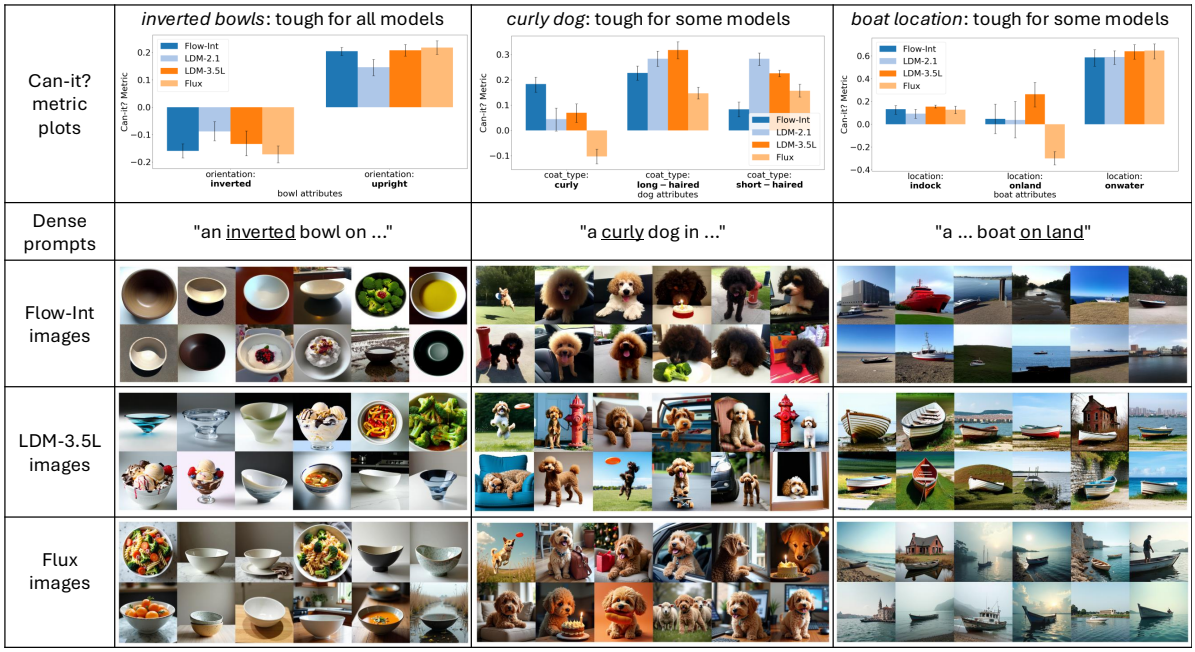

Figure 8. **Interesting generalization failure modes found through COCO-DIMCIM: [Left]** We find that none of the models are able to generate "inverted" *bowls*. **[Middle]** FLUX.1-dev [27] is not good at generating *dogs* with "curly" *hair*, even though other models which are much smaller than FLUX.1-dev are able to. **[Right]** FLUX.1-dev is also bad at generating *boats* on "land". Other smaller models are much better at generating such cases.

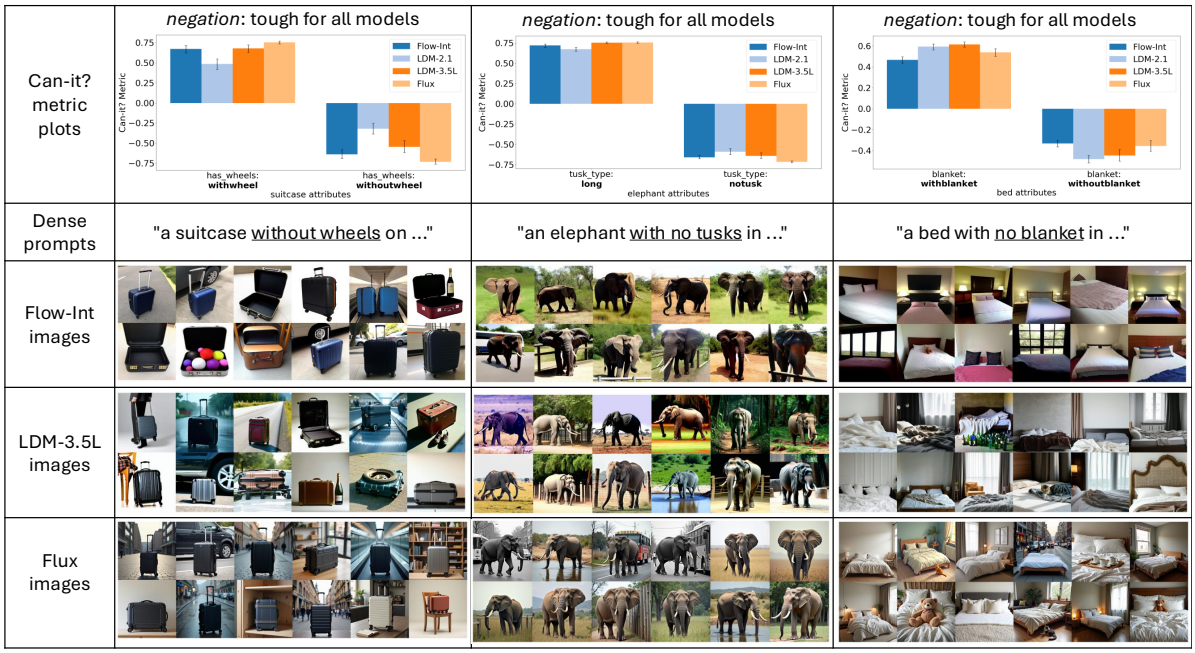

Figure 9. **Most T2I models are not able to handle negation in prompts.** Models generate the negated attributes in the prompts. The dense prompts in the first column say "without wheels" but all the models generate suitcase wheels in most of the generated images. Similarly, they generate elephant tusks and blankets on beds in most of the generated images, even though the prompts specifically asks them not to.

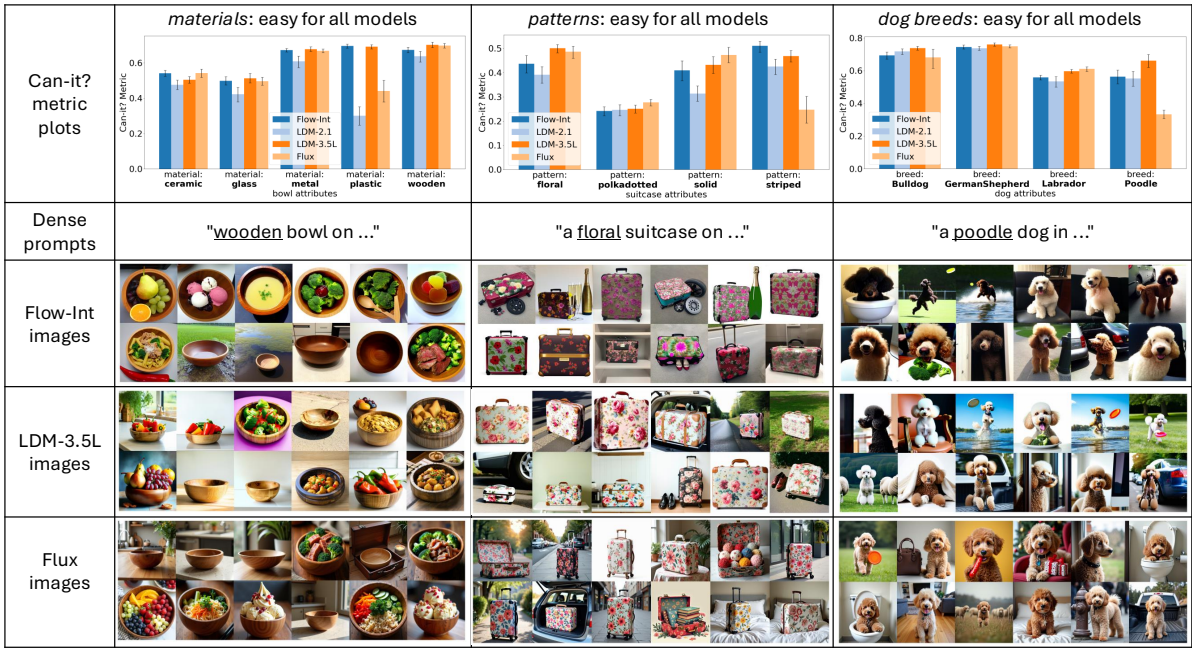

Figure 10. **Most models are good with attributes like color, pattern, material and breeds:** Examples showing that Can-It Metric for these concept-attributes are high and positive for most models. Generated images also show that the models can easily generalize to these attributes.

7.3. Example plots to show that model default-mode diversity reflects training data diversity

In Figure 11, we plot the Does-It Metric scores for the generated images for a few more concepts and the Does-It Metric scores of their corresponding training data images. The Does-It Metric scores are highly correlated and show that model default-mode diversity reflects training data diversity.

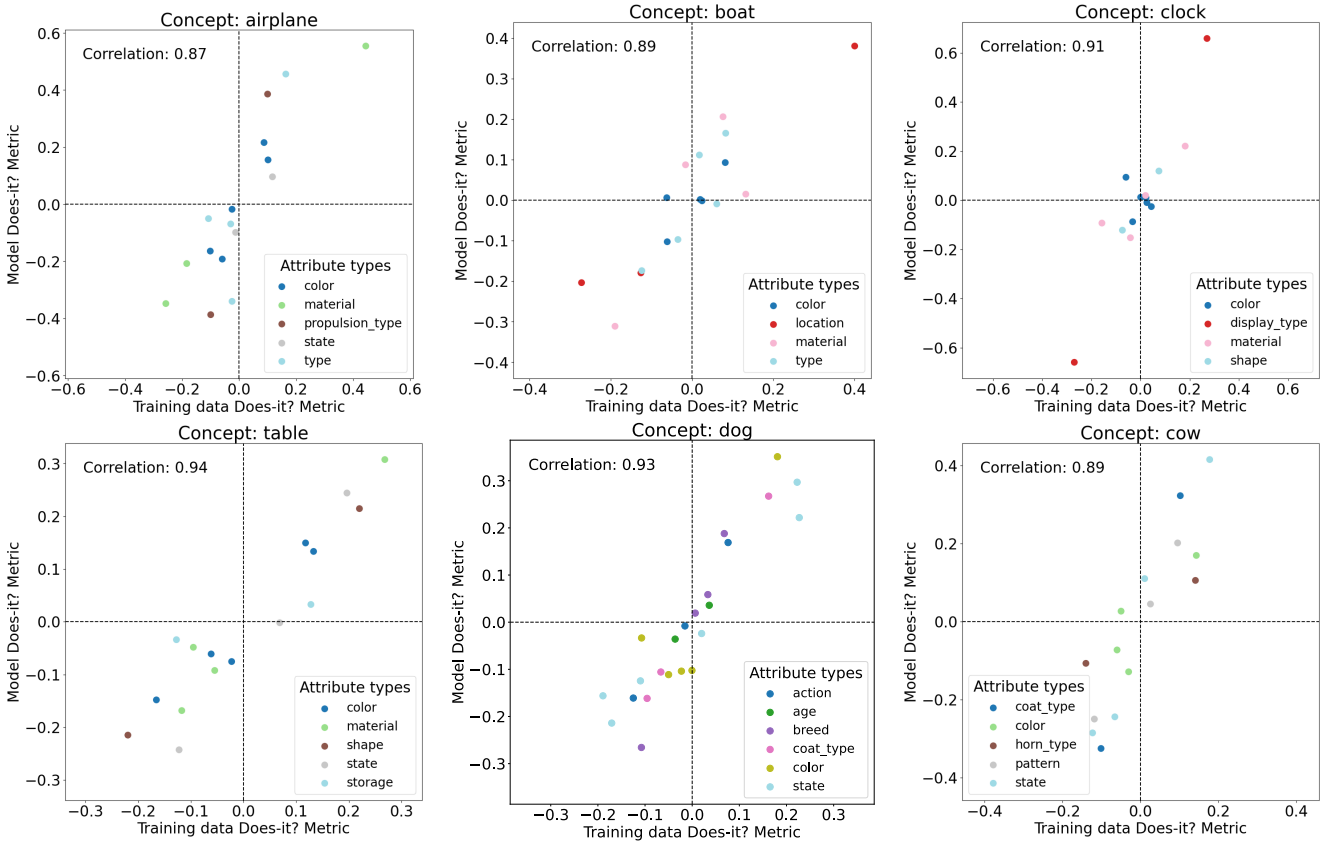

Figure 11. **Default-mode model diversity reflects diversity in images used for model training**: We show plots for attributes of 6 concepts (airplane, boar, clock, cow, dog, and table). On the y-axis we have Does-It Metric scores calculated from generated images of the concepts and on the x-axis we have Does-It Metric scores calculated from the training data images of those concepts. We see that the scores are highly correlated.

7.4. COCO-DIMCIM dataset creation figure

Figure 12 shows a flow chart of COCO-DIMCIM dataset creation process. It shows how we use LLMs to collect attributes, coarse prompts and dense prompts from seed prompts - as explained in Section 4.

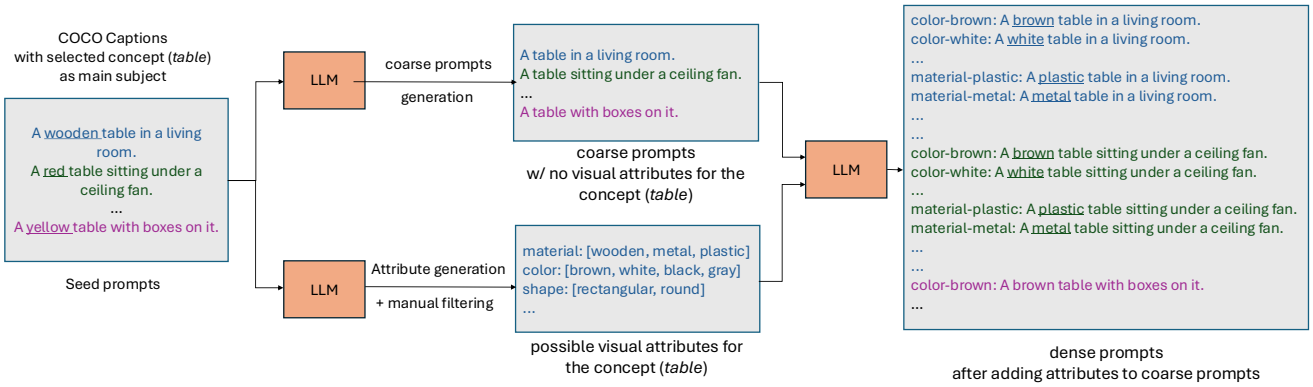

Figure 12. **COCO-DIMCIM dataset creation:** For a concept (*table* in this example), we start by randomly selecting captions from COCO [28] as seed prompts. We then use an LLM (Llama3.1 [39]) to generate coarse prompts from seed prompts and find possible concept attributes. To generate dense prompts from a coarse prompt, we use the LLM (Llama3.1 [39]) to inject attributes from the possible visual attributes to the coarse prompt, thus creating multiple dense prompts with different visual attributes.

7.5. VQAScore attribute scoring validation study

We performed a human study to validate the use of VQAScore for attribute scoring. We collected attribute annotations for 1200 images generated from dense prompts by LDM3.5L [14] from 12 human participants and compared them to VQAScore. We observed $> 90\%$ agreement between the human-selected attribute and the attribute with the highest VQAScore. We note that the images with disagreement between VQAScore and human annotator likely have greater inherent ambiguity: while the mean difference between the top two attribute VQAScores across all images is 0.40, when there is a disagreement between the human annotator and VQAScore, the average difference is only 0.18.

7.6. Meta-prompts for instructing the LLM

We instruct LLMs (Llama3.1 [39]) to generate COCO-DIMCIM. More specifically, we instruct Llama3.1 to (1) create a list of potential attributes of a concept from a given seed prompt and (2) generate coarse prompts and dense prompts from seed prompts and list of attributes. The Python functions to generate LLM instruction prompts are given in Table 3 and Table 4.

Table 3. Meta-prompt for Llama to generate a list of potential attributes of a concept from a given `seed_prompt` used for image generation.

```
Given an image caption, find the main subject of the caption.
Once you find the main subject of the caption,
find the visual modifiers or visual attributes of the main subject in the caption.
Give a list of visual modifier/attribute types. If there are no modifiers/attributes, give an empty list
For example, in this example caption
c: "a black dog running on the beach"
the main subject is 'dog'
visual modifiers types and values are:
color: black
state: running

Now can you find main subject and visual modifier/attribute types and values for this caption?
Please ignore the context and scene related attributes/modifiers.
c = {seed_prompt}

what are some of the other possible attribute/modifier types for the above main subject?
Also, what are some of the possible values for those attributes/modifiers?
Please find attributes relevant to the main subject and the caption.
can you answer in a nice json format?
Put all the existing visual attribute types and other possible attribute types in the json with their possible values.

For example, the output json for the dog example above is:

{
    "caption": "a black dog running on the beach",
    "main_subject": "dog",
    "visual_modifiers": {
        "existing": {
        "color": "black",
        "state": "running"
        },
        "possible_attributes": {
        "color": ["black", "white", "brown", "gray", "golden"],
        "breed": ["Labrador", "German Shepherd", "Poodle", "Bulldog"],
        "size": ["small", "medium", "large"],
        "age": ["puppy", "adult", "senior"],
        "coat_type": ["short-haired", "long-haired", "curly"],
        "body_type": ["muscular", "slim", "stocky"],
        "state": ["running", "sitting", "standing", "lying down", "jumping"]
        }
    }
}

Please output similar json for c = {seed_prompt} with possible attribute/modifier types and their possible values.
Output only the json and nothing else.
Output:
```

Table 4. Meta-prompt for LLama used to generate coarse prompts and dense prompts from an input file `attributes_json` that contains seed (image generation) prompt and potential visual attributes.

```
{attributes_json}

given the above json as input_json

first create a seed prompt by removing all the visual modifiers from the caption.
Keep the main subject and contextual/environment related attributes as the original prompt.

Once you get the seed prompt, select an attribute type and a value from the json and modify the seed prompt to add
those - call it a dense prompt
Do these for all the attribute types and their values to create dense prompts

Make sure that the dense prompts are plausible captions of naturally occuring images.
If it does not seem naturally plausible, skip that attribute value to create dense prompt

Give output in a nice json format indicating the original caption, seed prompt, selected attribute type,
selected attribute value and the generated dense prompt after adding the selected attribute

The structure of json should like this example:
{   "original_caption": "a black dog running on the beach",
    "seed_prompt": "a dog on the beach",
    "main_subject": "dog",
    "modified_prompts": [
      {
        "attribute_type": "color",
        "attribute_value": "white",
        "generated_prompt": "a white dog on the beach"
      },
      {
        "attribute_type": "color",
        "attribute_value": "brown",
        "generated_prompt": "a brown dog on the beach"
      },
      .
      .
      .
    ]
}

Output only the json with the above example fields and nothing else.
Make sure you include all the attribute types and their values from the input_json to create dense prompts

Output:
```