

A. Computational Details

Efficiency. CountGen takes ~36 seconds on average to generate an image on a single A100 80GB. We arrive at this number by iterating over CoCoCount. To put in context, Bounded-Attention takes ~55 seconds and requires bounding boxes as input, while our solution is not input-dependent. SDXL, as well as the "Repeated Object" baseline, takes ~8 seconds. Fig. 7 contains runtime comparison with other baselines.

Compute. All experiments were conducted over a period of a week on a single A100 80GB.

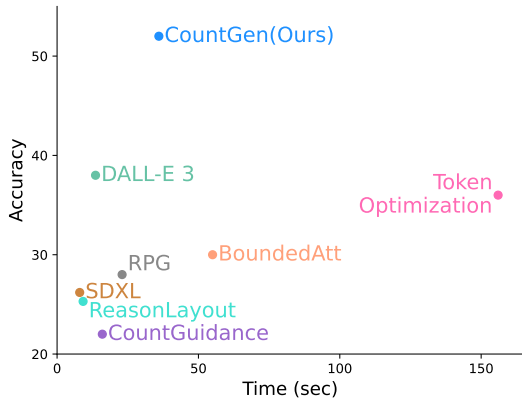


Figure 7. **Runtime comparison.** Accuracy as a function of runtime comparison across the baselines: SDXL, DALL-E-3, Token Optimization, RPG, BoundedAttn (Random mask+Bounded Attention), ReasonLayout (Reason out your layout), CountGuidance (Counting Guidance).

B. Implementation Details & Reproducibility

B.1. Count Number Extraction

To accurately extract count numbers from the textual prompts, we employ spaCy’s dependency graph parser [15] to identify and isolate indices of related subjects and numeric modifiers. This methodology is inspired by the approach detailed in "Linguistic Binding in Diffusion Models" by [27], which demonstrates the automated extraction of subjects and their attribute modifiers. We have adapted this technique to specifically recognize numeric modifiers, both spelled out (e.g., "five dogs") and in numeral form (e.g., "5 dogs"). This adaptation ensures that each numeric modifier is correctly associated with its corresponding noun, thereby facilitating accurate cross attention in our model’s processing pipeline.

B.2. CountGen

Layout guided generation. In our implementation, the self-attention masking is applied at timesteps $t \in [1000, 900]$, in the decoder layers of the U-Net. The object layout loss is applied at timesteps $t \in [1000, 500]$, in all layers of the U-Net. Our pipeline used the Attend-and-Excite [6] code base as a starting point.

ReLayout. The ReLayout U-Net was built upon the U-Net implementation of [5]. We trained the U-Net with a learning-rate of $8e-6$, a batch-size of size 32 and the Adam optimizer. The intersection penalty is set to 0.25 and the Dice penalty is set to 1. During training we apply a horizontal flip augmentation across all masks, and shuffle augmentation where we randomly re-arrange the input channels.

Instance identification. In the DBSCAN clustering algorithm, we used a dynamic epsilon value in the range of [0.1, 0.2] and used cosine similarity as the distance metric.

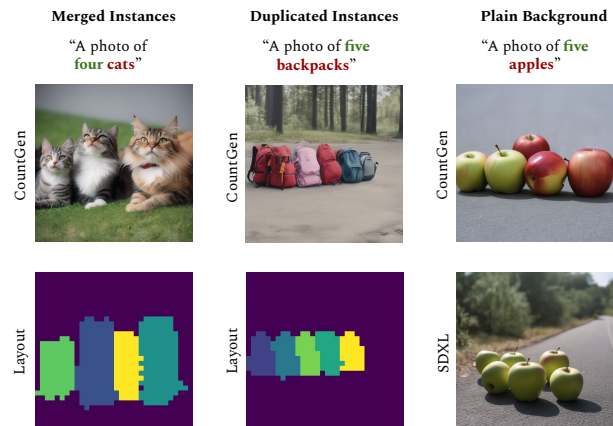


Figure 8. **Limitations.** Failure modes of CountGen.

B.3. Compared Methods

Each prompt in CoCoCount and T2I-CompBench-Count was assigned a unique random seed and was used by all baselines and CountGen.

We compared CountGen with the following baselines:

SDXL [25]. We used the stable-diffusion-xl-base-1.0 model.

Repeated Object. In this baseline, we used the same model and seeds as in SDXL but modified the prompts. We repeated the object in the prompt as many times as the target count. For example, "a photo of three cats" was changed to "a photo of a cat and a cat and a cat".

Reason Out Your Layout [7]. This baseline has two main steps. First, it leverages GPT-3.5-turbo to generate spatially reasonable coordinates to be used as a bound-

ing box for each instance of an object (i.e., “a photo of three cats” results in three bounding boxes, one for each cat). Second, it uses the generated layout to guide the generation process. We followed the prompt used by the authors, however, it seems that the responses by GPT-3.5-turbo and the author’s parser are not completely cohesive, which at times leads to zero bounding boxes. We count such cases as failures. For the CoCoCount experiment, it successfully generated 134/200 images, and for T2I-CompBench-Count, just 89/200. Failures were counted as errors in the reported results. We did not need to make changes to the code to run it.

DALL-E 3 [3]. We used the OpenAI API interface for the DALL-E 3 model with “standard” image quality. We did not use seeds in this baseline.

Random masks + BoundedAttn [8]. Given a prompt with a required number of object instances, we create a corresponding layout with the correct number of objects randomly placed in the image plane in a way they do not intersect one another. Then we used Bounded Attention to generate an image conditioned on that layout.

Counting Guidance [17]. The authors provided us with their code. We did not need to change it to run our experiments.

RPG [38]. We used the official code, with SDXL and GPT-4 for our experiments.

Token Optimization [40]. We used the official code, with SDXL-turbo.

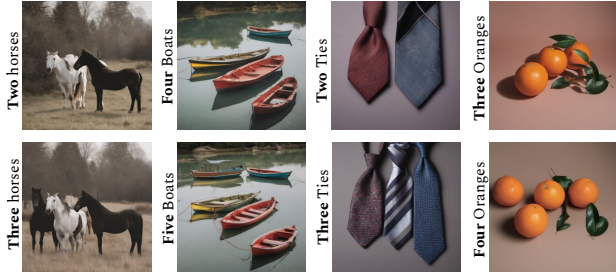


Figure 9. **A training set for a ReLayout.** We created pairs of images using SDXL, using the same seed and prompts that only differ by object count. We filtered out images that did not conform to the prompt, using the techniques described in Sec. 3.1. The resulting image pairs preserve the scene and layout except adding one object.

C. Extended Details on CountGen

C.1. ReLayout: Matching Objects

We aim to understand how M_k^i transitions to M_{k+1}^i . Specifically, for each object $i \in 1, \dots, k$ in the original M_k layout, our ReLayout objective is designed to predict how the corresponding mask M_k^i changes in the new image M_{k+1}^i , and additionally where to insert the added object $k+1$. This

design encourages the model to slightly modify existing objects while preserving spatial and shape consistency across the images.

To this end, we first have to establish a correspondence between the object masks (M_k, M_{k+1}). We employ the Hungarian algorithm to find the optimal one-to-one matching between masks in the two images based on the overlap and similarity of the masks. This algorithm effectively pairs each object in M_k with a corresponding object in M_{k+1} . The object in M_{k+1} that remains unmatched represents the additional object introduced in the new image, providing a clear identifier for the increment in object count.

C.2. Losses for Training the ReLayout

We use two training losses:

Dice Loss: measures the overlap between the predicted mask and target mask across all channels containing objects:

$$L_{\text{Dice}}^i = 1 - \frac{2 \sum_{p \in P} M_{k+1}^i(p) \cdot M_{k+1}^{*i}(p)}{\sum_{p \in P} (M_{k+1}^i(p) + M_{k+1}^{*i}(p))} \quad (2)$$

Here, p iterates over all pixels P in the masks, and i ranges over all possible object channels. For all $k+1$ channels, the total dice loss is:

$$L_{\text{Dice}} = \sum_{i=1}^{k+1} L_{\text{Dice}}^i \quad (3)$$

Intersection Loss: To ensure distinctiveness among the predicted masks and to minimize overlap between different object masks, the intersection loss for all possible pairs of different masks in the output mask containing objects is given by:

$$L_{\text{Overlap}} = \sum_{i=1}^{k+1} \sum_{j \neq i}^{k+1} \frac{2 \sum_{p \in P} M_{k+1}^i(p) \cdot M_{k+1}^j(p)}{\sum_{p \in P} (M_{k+1}^i(p) + M_{k+1}^j(p))} \quad (4)$$

C.3. ReLayout Evaluation

We use two metrics for the evaluation:

Extra mask median score. To calculate the extra mask size score, we first find the median size (S_{median}) of all object masks. We then compare this to the size of the new mask (S_{extra}). The score is defined as:

$$\text{Score} = \frac{\min(S_{\text{extra}}, S_{\text{median}})}{\max(S_{\text{extra}}, S_{\text{median}})}$$

which gives a value between 0 and 1. A score closer to 1 indicates that the new object’s size is more similar to the median-sized object. For ReLayout, the score is 0.705, indicating that the new object has become more similar in size to the other objects in the scene.

Average intersection score. This metric measures the average intersection between an object i and all other object masks j , normalized by the size of object i . A lower score indicates less overlap between objects. During training, this score decreased to 0.18, indicating small intersection between the objects.

C.4. CountGen Pipeline Ablation

Notably, the CountGen module consistently adds an extra object mask in every case, suggesting that the failures on CoCoCount are related to either clustering or layout guidance (Tab. 4). Out of all the failures, 47 were due to Instance localization and 49 were due to loss. Over-generation occurred mostly for target count k bigger than 5, whereas layout-guidance issues are more frequent with target counts ≤ 5 . Among the Instance localization failures, we observed that 31% of the errors occurred when more than 15 instances were generated in the original image.

Table 4. Failure analysis across different target counts.

Target Count	Localization Failures	Loss Failures	Total Failures
2	2	3	5
3	4	10	14
4	5	9	14
5	8	7	15
7	11	8	19
10	17	12	29

C.5. Instance-identity Representation Analysis

As detailed in Sec. 6, our approach to identifying an instance-level feature layer involves comparing the bounding box predictions generated from our model’s instance localization masks with ground truth bounding boxes. Essentially, we aim to identify the layer whose features, when clustered, yield a distinct cluster for each subject instance. To achieve this, we manually annotated a dataset of 85 images with bounding boxes for each instance in the generated images. Then, using standard precision and recall metrics, we select the layer with the highest combined score by comparing the bounding boxes of each cluster to the ground truth bounding boxes. The algorithm for finding the optimal layer is presented in Algorithm 1. We report standard precision and recall metrics across a range of timesteps (Tab. 5) and layers (Tab. 6). The selected timestep and layer, determined using set-aside validation data, generalize well to the test set and outperform other hyperparameter configurations.

The provided algorithm is general and can be easily extended to other models and architectures. Furthermore, iterating over a set of potential feature sources eliminates the need for manual hyperparameter tuning, thereby automating the entire adaptation process. To further evaluate the

generalization and robustness of our method, we identify instance-level features in the Flux.1-dev model, as detailed in Section D.2.

Algorithm 1 Finding Instance Features

Input:

- K : Number of images
- R_ϵ : DBSCAN’s range of possible ϵ values

Output:

- Optimal layer l^* and timestep t^* maximizing the average F1 score
- 1: Generate K images.
 - 2: Label bounding boxes GT for instances in each image.
 - 3: **for** each image k , layer l , and timestep t **do**
 - 4: Extract self-attention features SA from layer l at timestep t .
 - 5: $C \leftarrow \text{DBSCAN}(SA_{t,l}, R_\epsilon)$ {Cluster features using DBSCAN}
 - 6: $B \leftarrow \text{BBOX}(C)$ {Extract bounding boxes from clusters}
 - 7: $F1_{\text{score}} \leftarrow \text{F1}(B, GT)$ {Calculate F1 score}
 - 8: Store $F1_{t,l}^k$.
 - 9: **end for**
 - 10: Compute the average F1 score across all K images for each l and t :

$$\overline{F1}_{t,l} = \frac{1}{K} \sum_{k=1}^K F1_{t,l}^k$$

- 11: Identify the layer l^* and timestep t^* that maximize the average F1 score:

$$(l^*, t^*) = \arg \max_{l,t} \overline{F1}_{t,l}$$

C.6. Stress Test with High Object Count

To assess how quality degrades with count, we analyze two key quality aspects under varying object counts up to 20. Metrics are based on new manual annotations. Fig. 10 shows that both metrics gradually decline. Also, we found that only 7.4% of image captions have counts above 10, by analyzing 3M relevant prompts in Laion-5B [29].

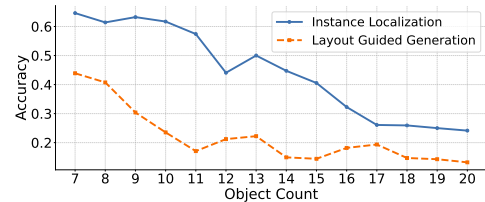


Figure 10. Stress test of CountGen with high object counts.

Table 5. Precision and Recall across different timesteps.

Metric	t=900	t=800	t=600	t=500 (Ours)	t=400	t=200	t=0
Precision	0.81	0.88	0.88	0.92	0.90	0.90	0.83
Recall	0.51	0.79	0.84	0.92	0.93	0.93	0.89

Table 6. Precision and Recall across different layers.

Metric	down_10	down_40	mid_120	mid_136	up_48	up_52 (Ours)	up_70	up_100
Precision	0.27	0.27	0.26	0.39	0.39	0.92	0.67	0.45
Recall	0.56	0.56	0.10	0.16	0.15	0.92	0.67	0.35

C.7. Qualitative Metrics

We computed the CLIP score and KID (more suitable for smaller sets) to assess image quality and semantic alignment across both datasets. Tab. 7 shows CountGen outperforms or is on-par with all baselines.

Metric	CLIP \uparrow	100 \times KID \downarrow
CountGen (ours)	0.31	0.06
SDXL	0.31	-
DALLE	0.30	0.75
Bounded	0.30	0.76
Reason Layout	0.30	0.56
Prompt editing	0.30	0.12
Token optimization	0.30	1.20
Counting guidance	0.30	0.48
RPG	0.29	0.44

Table 7. CLIP & KID metrics comparison across different models.

C.8. Semantic and Spatial Metrics

To ensure that CountGen does not negatively affect other semantic and spatial aspects such as color fidelity and spatial arrangement, we generated 100 prompts using GPT-4, each designed to test various spatial alignments and semantic attributes, such as color variations and different scene compositions. For instance, we generated prompts like "A photo of five bears each digging into a different trash can" and "A photo of six cats each sitting on a different step on a staircase" (Fig. 23). We then used CountGen and SDXL to generate images from these prompts. To evaluate semantic alignment, we conducted a human evaluation and ask whether the images captured the spatial or semantic attributes described in the prompts. The results show $62\% \pm 0.05$ accuracy for CountGen, and $63\% \pm 0.05$ accuracy for SDXL indicating that CountGen performs comparably to SDXL, with no negative impact.

C.9. Comparison with Others Layout-to-Image Methods

We compare our method to InstanceDiff [36] and MIGC [42], with both performing worse than ours:

- **MIGC:** Random + MIGC scored 34% on CompBench and 39% on CocoCount, while CountGen Layout + MIGC improved to 42.5% and 43%.
- **InstanceDiff:** Random + InstanceDiff scored 44% on CompBench and 48% on CocoCount, while CountGen Layout + InstanceDiff reached 46.98% and 49%.

Both methods tend to produce unnatural images and rely on CountGen for layout generation.

We conducted two separate comparisons for human evaluation as describe in Fig. 18. In the comparison involving MIGC, the raters preferred MIGC to CountGen in only 20 out of 200 cases, while they preferred CountGen in 95 cases. Similarly, compared to InstanceDiff, the raters preferred InstanceDiff in just 16 cases and CountGen in 82 cases.

C.10. Evaluation Extension

To further verify the robustness of our method, we expand the CoCoCount dataset to include 500 images and conduct a human evaluation. The results show a 53% accuracy rate, confirming that our method remains robust.

D. Future Work

D.1. Generate Objects of Mixed Classes

Although it is beyond our primary scope, we demonstrate CountGen can handle counting objects from different classes with a simple modifications to the architecture (Fig. 11). Specifically, we extract the cross-attention for each object type and cluster the self-attention features accordingly. Then, we fix the layout of each object separately using the same pre-trained ReLayout module. Finally, we apply the L_{cross} loss for both objects simultaneously.

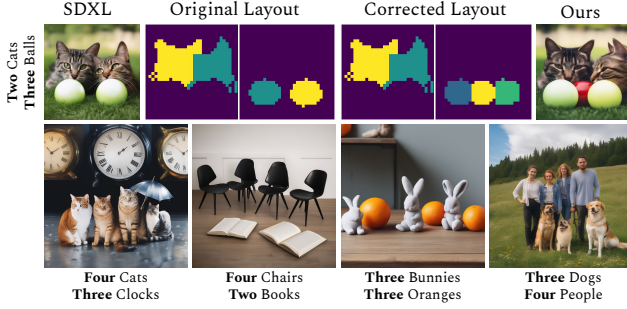


Figure 11. Objects from mixed classes

D.2. Generalization of Instance Representation to Other Diffusion Models Architectures

we extended our Algorithm 1 to Flux.1-dev model. Specifically, we generated 30 new images using the Flux model and manually annotated them with bounding boxes for each instance. This prepared dataset then underwent analysis using Algorithm 1. We report precision and recall metrics across different layers in Tab. 9 and across different timestamps in Tab. 8. Notably, layer 27 demonstrated the highest F1 score, indicating robust instance recognition capabilities. The effectiveness of instance separation at layer 27 is demonstrated in Fig. 21. The PCA visualization confirms the distinct separations between the object instances.

E. Datasets

CoCoCount. To create this set, we first select at random 20 classes from MSCOCO [21]. We then sample from six counting categories: 2,3,4,5,7, and 10. The two and three categories contain 34 samples, while the rest contain 33. Our prompts consist of the pattern “a photo of {number} {object}” with an optional variation of scenes: “on the grass”, “on the road”, or “on the ground”, which we incorporate for 50% of the prompts, also randomly. In total, we have 200 prompts. Below are the complete lists from which elements were chosen:

Objects: ‘car’, ‘airplane’, ‘bird’, ‘cat’, ‘dog’, ‘horse’, ‘sheep’, ‘cow’, ‘elephant’, ‘bear’, ‘backpack’, ‘tie’, ‘sports ball’, ‘baseball glove’, ‘cup’, ‘bowl’, ‘apple’, ‘donut’, ‘cell phone’, ‘clock’. **Counting Categories:** ‘two’, ‘three’, ‘four’, ‘five’, ‘seven’, ‘ten’. **Scenes:** ‘on the grass’, ‘on the road’, ‘on the ground’.

F. Evaluation

Automatic evaluation. We use the implementation by Ultralytics YOLO of YOLOv9e (large).

F.1. Human Evaluation

We use the Amazon Mechanical Turk platform and ensure the evaluation is of high quality by hiring raters with a minimum of 5,000 approved HITs and an approval rate exceeding 98%. Each example was shown to three raters and the

majority selection was taken. The compensation was \$15 per hour. Screenshots of the count precision task can be viewed in Fig. 16, Fig. 17, Fig. 18 and the image fidelity task in Fig. 19.

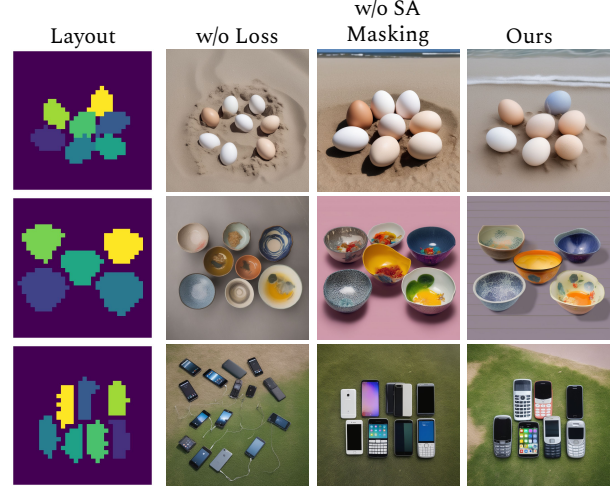


Figure 15. **Component ablation.** We ablate over two components of the layout-guided generation model: the optimization loss and Self-Attention Masking. Disabling the loss causes the generated image to deviate from the required layout. Removing the Self-Attention masking typically causes objects to appear outside of the layout foreground.

Table 8. Precision and Recall across different timesteps in Flux.

Metric	t=900	t=800	t=600	t=500 (Ours)	t=400	t=200
Precision	0.38	0.64	0.78	0.74	0.53	0.47
Recall	0.27	0.57	0.74	0.81	0.68	0.72

Table 9. Precision and Recall across different layers in Flux.

Metric	layer_11	layer_19	layer_23	layer_25	layer_27 (Ours)	layer_37	layer_46
Precision	0.27	0.07	0.45	0.62	0.74	0.34	0.33
Recall	0.28	0.07	0.49	0.51	0.81	0.37	0.40

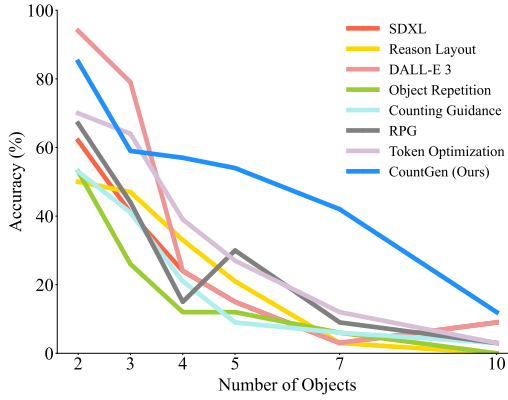


Figure 12. **Accuracy, as a function of the number of generated objects.** Accuracy evaluated by human raters, over the set of 200 evaluation images. CountGen (blue) outperforms all methods for $n > 3$, and is on par with DALL-E 3 for 2 and 3 objects.

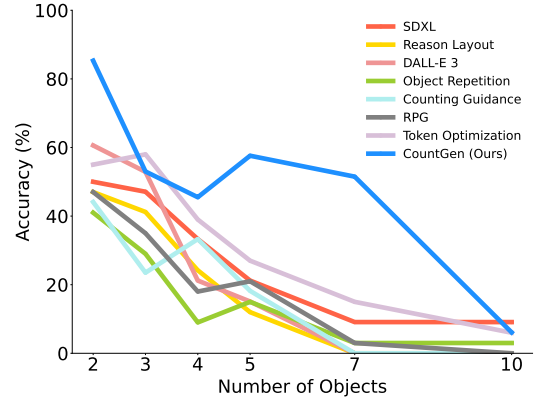


Figure 13. **Accuracy, as a function of the number of generated objects.** Accuracy evaluated by YOLOv9, over the set of 200 evaluation images. Here, CountGen (blue) outperforms all methods.

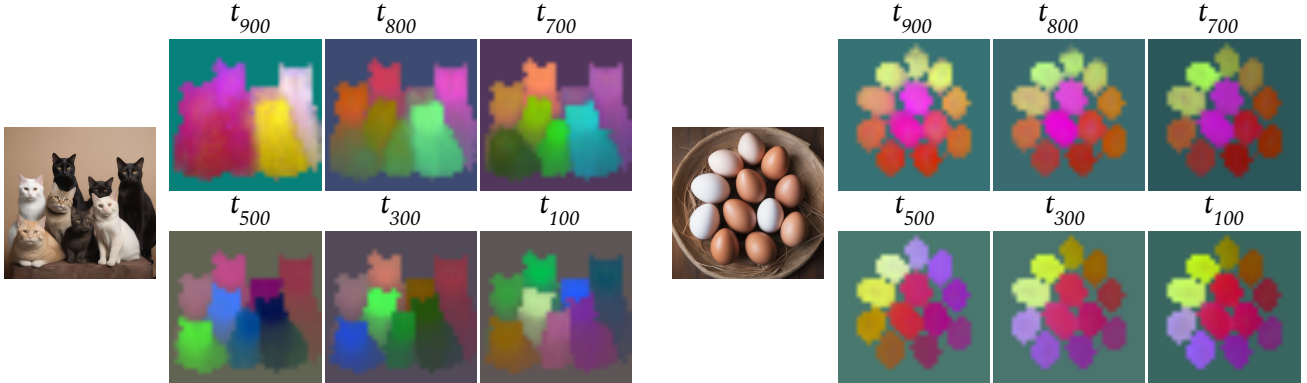


Figure 14. **PCA visualization across timesteps.** To explore the notion of objectness inside SDXL latent space, we visualize a dimension-reduced self-attention feature maps across different timesteps range from $t = 900$ to $t = 100$. Initially, up to timestep $t = 500$, clear separation is not observed in some objects (e.g., some eggs appear in similar colors). However, starting from $t = 500$, a distinct separation emerges, with each object clearly distinguished by different shades.

Instructions for the Image Evaluation Task

Welcome to the Image Evaluation Task! Your role is crucial in counting the number of objects in an image. Please follow the instructions carefully:

1. **Examine the Image:** You'll be presented with an image. Pay close attention to the objects and details within the image, irrespective of its overall appearance. **Remember, an image can look unusual or "weird" but still accurately answer the questions.**



Figure 16. Instructions for the Image Evaluation Task - Part 1.

2. **Read the Questions:** Below the image, you'll find a set of questions.

- Are there books in the image?
- If yes, are the books well-formed?
- How many books are there?

Understanding the Concepts

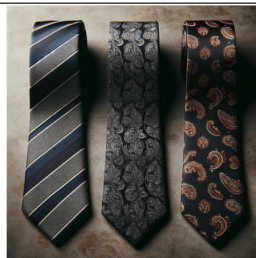
For clarity, let's break down some of the key terms:

- **Object in the Image:** This pertains to the visible items or subjects in the provided image. Using the previous example, you'd need to determine if you can see at least one book in the image.
- **Well-formed Objects:** Here you should identify if the focused object is well-formed and can be counted. For instance, in the left example above, the books are well-formed and can thus be counted, but some of the books in the right example are not well-formed. As a result, and it is not clear how many books are there in total.
- **Count:** The count describes the quantity of an object. For instance, in example (1) above, there are 6 books.

Please ensure accuracy in your evaluation. Inaccurate or rushed answers may be rejected.

Thank you for your attention to detail and dedication to this task!

Figure 17. Instructions for the Image Evaluation Task - Part 2.



Are there tie in the image? ☐

If yes, are they well-formed? ☐

How many tie are there in the image?

☐ I'm **not** confident in my answer

Feel free to leave feedback here (optional)

Submit

Figure 18. Example task to count the number of objects and assess their well-formedness.

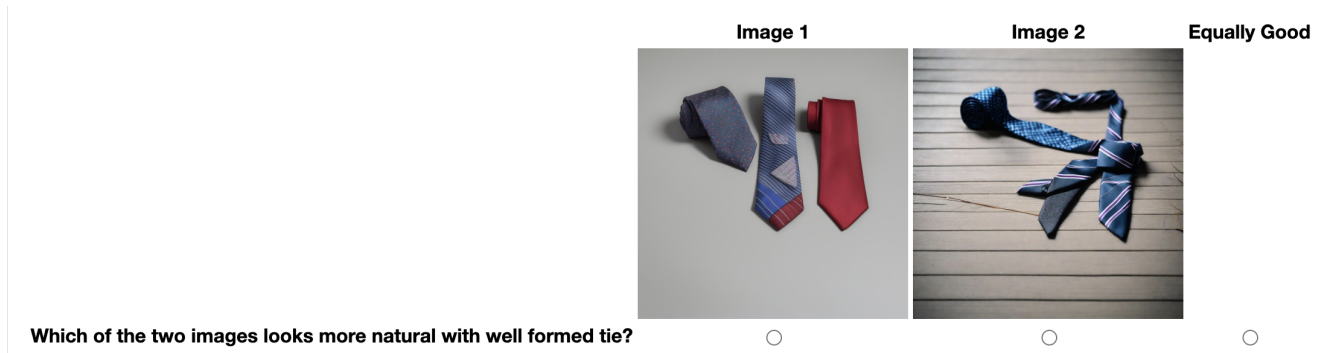


Figure 19. Example task to compare image fidelity based on prompt matching and naturalness.

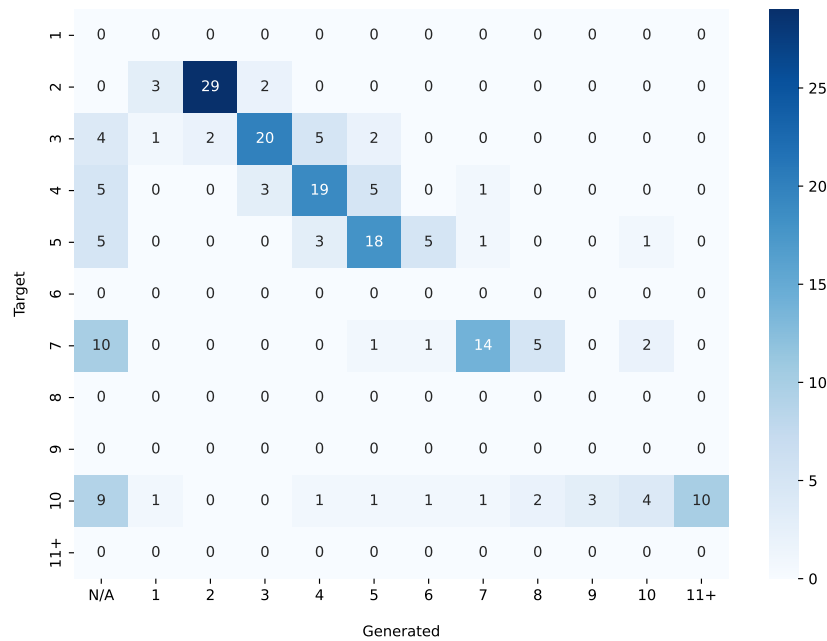


Figure 20. Confusion matrix from human evaluation of the count accuracy experiment for CountGen. As noted in Fig. 18, evaluators could indicate if they were unsure of their response (“N/A” in the table).

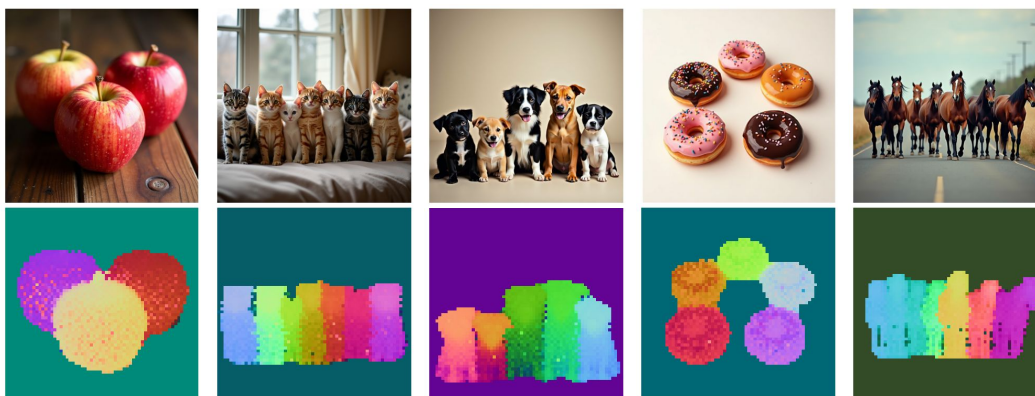


Figure 21. **PCA visualizations of the Flux model.** Using our algorithm (1), we identified the layer in the Flux model that captures objectness. Layer 27 exhibits a strong ability to separate different instances, with each instance uniquely represented by a distinct color in the PCA visualizations.



Figure 22. Additional comparison between CountGen and SDXL with varied sentence structures.



Figure 23. Spatial and semantic comparisons between CountGen and SDXL.

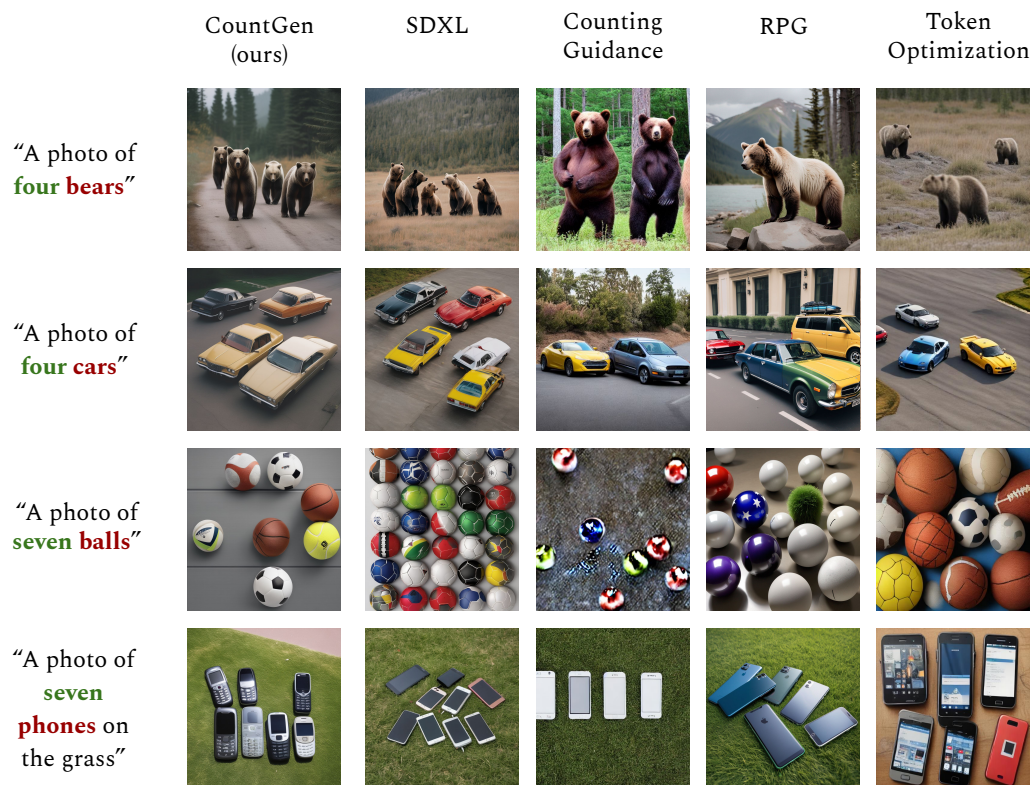


Figure 24. Qualitative comparisons. Additional results of CountGen versus SDXL, Counting Guidance, RPG, and Token Optimization are shown.