VLMs for Better Visual Understanding in Self-Correcting Diffusion Models

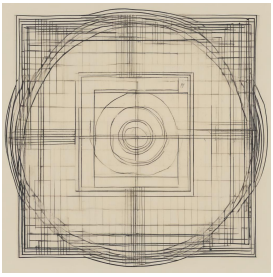
Erich Liang*, **Nobline Yoo***
Princeton University
{el1685,nobliney}@cs.princeton.edu

Abstract

Advances in diffusion models have improved their ability to generate high-quality images based on text prompts, but they still suffer from issues such as numeracy and spatial reasoning. One recent work utilizes LLMs, along with an off-the-shelf detector and segmentor, to iteratively correct diffusion-generated images to better conform to input text prompts along these axes. However, this approach fails to fully utilize the availability of both visual and textual understanding to guide this correction process. In this work, we explore the idea of using VLMs instead of LLMs for diffusion guidance. We show that utilizing a combination of VLMs, such as GPT-4o mini and PaliGemma, in an image correction pipeline helps achieves better numeracy and FID results. Replacing pipeline components with VLMs also paves the way toward the use of a single, unified VLM to perform the image generation task, and we explore this possibility and its implications in this paper.

1 Introduction

Diffusion-based text-to-image generation is an important and useful capability, facilitating the creation and editing of visual content based on natural language descriptions. Recent models, such as SDXL (Podell et al., 2024), Stable Diffusion 3 (Esser et al., 2024), and Pixart-α (Chen et al., 2024), boast high general semantic alignment between image and content, even for complex prompts (e.g. "Epic long distance cityscape photo of New York City flooded by the ocean and overgrown buildings and jungle ruins in rainforest, at sunset, cinematic shot, highly detailed, 8k, golden light" from Podell et al. (2024)).



(a) "A square to the left of a circle"

(b) "4 squares"

Figure 1: Alignment problems persist in recent diffusion models. Figures on left and right show SDXL-generated images, given the respective prompts.

However, these models are not perfect. Even on seemingly simple prompts that specify specific spatial configurations (e.g. "a {object} to the {left/right/top/bottom} of {object}") or numerical counts (e.g. "a photo of {n} {object}"), alignment problems persist (see Figure 1). For the scope of this course, we focus on the problem of numerical accuracy (i.e. "numeracy") for its relative ease of evaluation.

A recent line of diffusion papers (Binyamin et al., 2024; Kang et al., 2023) uses inference-time computations and losses to update the denoising results, while leaving the diffusion backbone parameters frozen. Kang et al. (2023) uses the gradient of an external, differentiable counting network to update the predicted noise ϵ at inference time. CountGen (Binyamin et al., 2024), which boasts higher accuracy than the former, uses U-Nets to predict new, corrected

*Equal contributions.

masks for object instances, along with a cross-attention loss to align the final generated images with the new masks. But results are unsatisfactory (see Figure 2a), pointing to the need for a stronger form of counting guidance.

(a) CountGen (Binyamin et al., 2024)

(b) LMD+ (Lian et al., 2024)

Figure 2: Generated images given prompt, "A photo of **five** backpacks on the ground." On the left is the CountGen-generated result, which incorrectly draws **six** backpacks. On the right is the result from LMD+, which correctly draws **five** backpacks. Recall that CountGen uses an attention-based loss to correct count, while LMD+ uses an LLM to generate one bounding box per required object instance.

Unlike diffusion models, large language models (Didolkar et al., 2024; Shah et al., 2024; Luo et al., 2024; Lightman et al., 2024; Li et al., 2024; Sprague et al., 2024) and vision-language models (Singh et al., 2024) have shown promising signs of competency in numerical reasoning. **LLM**-grounded **D**iffusion (**LMD+**) (Lian et al., 2024) in particular leverages LLM's numerical competency to generate count-aware object bounding box layouts to guide image generation (see Figure 2b). LMD+ works by providing GPT-4 with an instruction (e.g. to generate bounding box coordinates for object instances given a prompt) and several in-context examples of (prompt, pairs of objects and their corresponding bounding boxes). A follow-up work, "**S**elf-correcting **LLM**-controlled **D**iffusion Models" (**SLD**) (Wu et al., 2024), uses an iterative method to correct images post-initial generation over multiple rounds, again relying on LLMs for prompt understanding and bounding box layout generation.

Though promising, in the area of *image* generation, we believe VLMs pose a more natural alternative to LLMs, since they carry *vision* capabilities that may be leveraged for better *visual* and prompt understanding towards more accurate object count correction in the diffusion pipeline.

In our work, we propose incorporating VLMs to help guide diffusion-based image generation models. Specifically, we build upon SLD by

1. Ablating away LLM components and replacing them with the VLM GPT-4o mini (OpenAI, 2024) to quantitatively measure any improvements in numeracy afforded by the specific addition of visual understanding in LM components (see Figure 3).
2. Ablating away other black-box components of the pipeline and replacing them with VLM PaliGemma (Beyer et al., 2024) to make an initial step toward full conversion of the pipeline into a single VLM-based one—by changing all non-LM components into VLM ones as well (see Figure 3).
3. Finetuning a single VLM (PaliGemma) to handle the whole correction pipeline.

Our analysis shows the addition of VLMs into the pipeline does indeed improve accuracy, but only in conjunction with downstream pipeline components that *can* take full advantage of the upstream performance boost offered by the VLM. Furthermore, our flagship model, which incorporates VLMs along the pipeline, along with a few tricks, yields performance that is on-par with or better than the base SLD, pointing to the use of VLMs in diffusion pipelines as a promising direction. Lastly, our finetuning results motivate the need for larger, more high quality datasets and training schemes to better align VLMs to the task of iterative image correction from beginning to end.

2 Related work

2.1 Brief overview of diffusion

Diffusion models formulate the process of image generation as a denoising one: starting at random Gaussian noise x_T, at time step t a trained U-Net predicts the noise ϵ to subtract from x_t to get to slightly more denoised version x_{t-1}. The goal is to reach a clear image over multiple sampling steps. Latent diffusion models are a variant where

the samples x_t and denoising operations are performed in latent space, not image space, with a trained decoder to reconstruct the image from the final sampled latent.

Existing works which try to correct the image generation pipeline, while leaving its model components (encoder, decoder, U-Net) intact, apply modifications to the predicted noise ϵ (Kang et al., 2023) or to the intermediate x_t (Wu et al., 2024) in an ad-hoc manner.

2.2 Self-correcting LLM-controlled Diffusion Models (SLD)

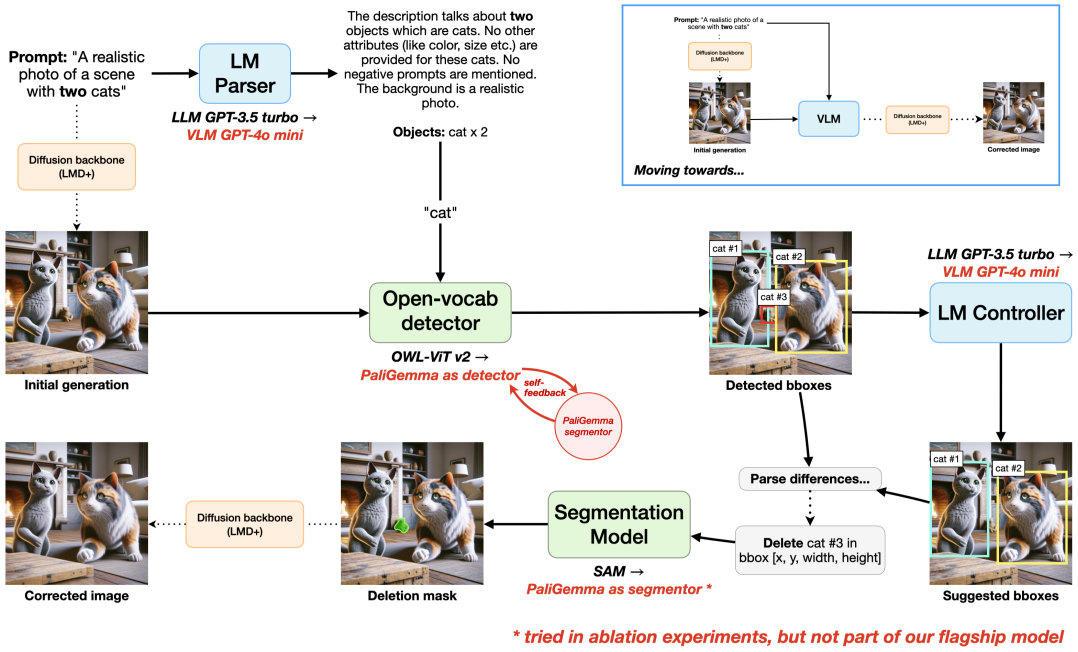


Figure 3: Baseline SLD model pipeline. **Our adjustments are in red.** Notice how the initial generation contains **three** cats (two in the foreground and one small one in the background), though the prompt specifies a scene with **two** cats. After one round of SLD, the cat in the background is removed, resulting in better alignment with the original prompt. In our work, we experiment with replacing the LM and non-LM components with VLMs, as an initial foray into evaluating the feasibility of using VLMs more extensively in diffusion correction pipelines.

On a high-level, SLD is a pipeline (Figure 3) that takes in a prompt P and initial generation I_{init} and outputs (over multiple rounds) a corrected image $I_{corrected}$, though in practice, one round suffices to correct most errors, according to the authors.

More specifically, SLD consists of a sequence of black-box components (Figure 4), which are described as follows.

LLM parser (GPT-3.5 turbo): The LLM parser takes image generation prompt P and outputs a list O_{attr} of specified objects and their counts and attributes. To get usable results, the authors prompt the LLM with an instruction to identify object/attributes and provides a few in-context examples. **Open-vocab detector (OWL-ViT v2):** Given list O_{attr} and I_{init}, the detector outputs bounding boxes O_{bbox} for each instance of each object type. **LLM controller (GPT-3.5 turbo):** Given prompt P and list O_{bbox} of current bounding boxes, the LLM controller outputs a new layout: a list $O_{bbox}^{corrected}$ of suggested bounding boxes. *This leverages the LLM's numerical reasoning (given the prompt, it chooses whether, how, and where to add/delete object instances).* **Segmentation model (SAM):** Given bounding boxes, SAM outputs the segmentation mask of the object within each. The refined object boundaries are used to more accurately define the areas to excise and reinitialize with Gaussian noise, move, or resize.

SLD is effective in some cases, but in terms of the prospect of future potential improvements, its reliance on disparate black-box models makes it less scalable relative to using a single model end-to-end or even a family of related models, since overall improvement relies on improvements in the individual (disparate) components. Motivated by this and the expectation that the vision understanding embodied in VLMs may give a performance boost for SLD, we propose making the pipeline components more homogenous (towards a single VLM pipeline). Ultimately, in our

flagship model, we successfully replace the parser, controller, and detector with models in the VLM family and show results that are on-par or even better than baseline SLD.

2.3 GPT-4o mini & PaliGemma

In choosing appropriate VLMs to replace the pipeline components, we sought a few design requirements. In particular, (1) the VLM to replace the parser and controller must have well-documented in-context prompting functionality, (2) the VLM to replace the detector and (potentially) the segmentor must be able to output bounding boxes and masks in an easy-to-visualize way (to facilitate initial testing) and be small enough to finetune efficiently, given our GPU/time resources, and (3) both VLMs must be inexpensive to run.

VLM GPT-4o mini has documented prompting functionality, achieving 82% on MMLU (OpenAI), and PaliGemma's vocabulary supports both detection and segmentation, using tokens such as `<loc####>` and `<seg###>` to represent bounding boxes coordinates and segmentation masks, represented by 16 `seg` tokens. In particular, `paligemma-3b-mix-224` is a version of the model that takes prompts in the form of "detect [object] ; [object] ; ..." and "segment [object] ; [object] ; ..." for detection/segmentation of multiple instances of the same object type. For example, to detect two cats, the prompt needs to contain at least two mentions of cat, like so: "detect cat ; cat." So, it is necessary to know the number of instances in advance, which makes it less practical as a replacement for OWL-ViT v2. To bypass this, we use our knowledge of the evaluation benchmarks (i.e. the maximum specified object count is 10) to set an upper-limit on the number of instances we prompt for, and to account for duplicate bounding boxes, we filter out IoU $>= 0.75$. PaliGemma is open-source, and GPT-4o mini is relatively inexpensive.

On notation: **detect** $\{\mathbf{o}\}^{\mathbf{n}}$ is shorthand for specifying n instances of o (e.g. **detect** $\{\mathbf{cat}\}^{\mathbf{2}}$ is shorthand for **detect cat ; cat**).

3 Methodology

We explore two ways of replacing SLD pipeline components with VLM models. In Section 3.1, we describe an approach in which pretrained VLMs are used to replace individual components of the SLD pipeline. In Section 3.3, we present an alternative approach in which we attempt to finetune a single VLM to perform the SLD guidance task in an end-to-end manner.

3.1 Component-based VLM replacement

In this approach, we replace the various components of SLD with pre-trained VLMs. To make higher quality, spatial-aware corrections, we replace SLD's LLM controller with the VLM GPT-4o mini. Towards a more homogenous pipeline, we replace SLD's open-vocab detector with VLM PaliGemma and the LLM parser with GPT-4o mini. The open-vocab detector takes arbitrary object categories as well as the initial generated image to detect bounding boxes, which is a task that the `paligemma-3b-mix-224` flavor of PaliGemma has been finetuned to perform well at. We also consider replacing SAM with PaliGemma as segmentor, but empirically find this change be unbeneficial. Instead, we use PaliGemma as segmentor as an intermediate step to improve PaliGemma predicted bounding boxes (we refer to this as **self-feedback**—see appendix section A.2 for brief details). See Figure 4 for a summary of our component-based VLM replacements.

The VLM parser and controller use the same in-context prompting as baseline SLD. Incorporating PaliGemma as a detector and segmentor requires a few addition steps to convert the PaliGemma-specific grounding tokens into bbox coordinates and binary masks. Details in appendix section A.1.

In practice, we use nucleus(0.3) decoding for PaliGemma output generation.

3.2 Evaluation

We evaluate our methods on three datasets, focusing on count-specifying prompts: LMD (Lian et al., 2024), CoCo-Count (Binyamin et al., 2024), and T2I-Compbench (Huang et al., 2023). LMD has 100 prompt-image pairs, and CoCoCount and T2I-Compbench each have 200, for a total test size of 500 prompts. We measure performance using accuracy and FID score, where accuracy is defined as the percent of generated images with the correct count of specified object types, computed by an off-the-shelf detector (OWL-ViT v2 for LMD and YOLOv9 for the latter two). FID is computed using 1000 random images from ImageNet-1k as the real set.

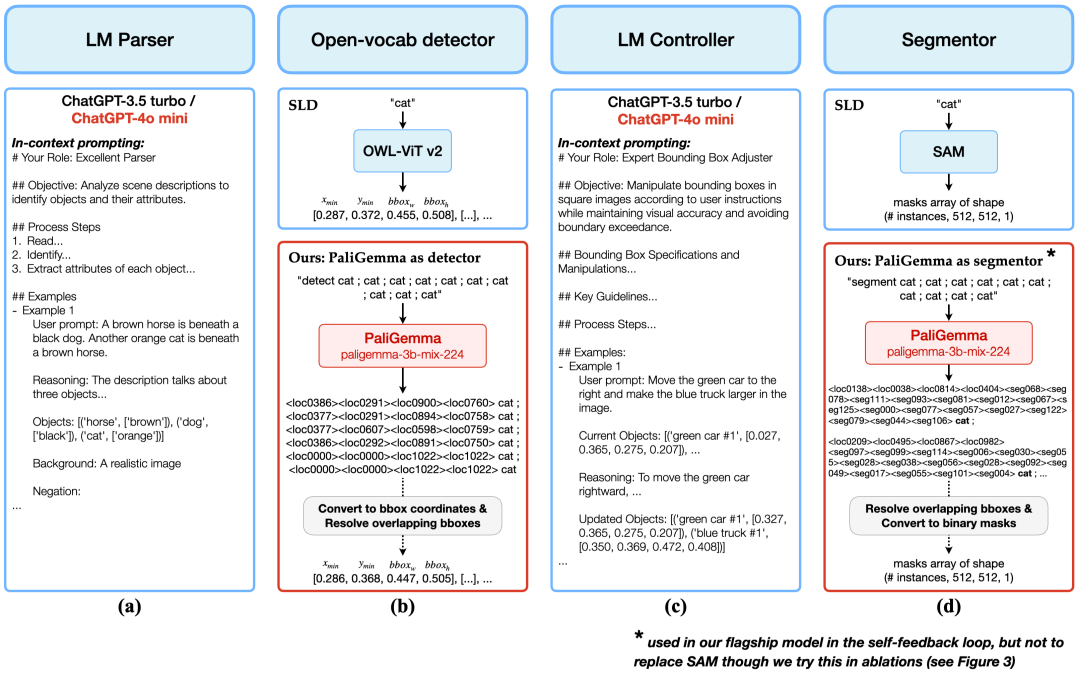


Figure 4: The components we try to ablate and replace with VLMs. **Our adjustments are in red. (a),(c)** The same kind of in-context prompting is used for the LLM/VLM flavor of the parser and controller. **(b)** `paligemma-3b-mix-224` outputs PaliGemma bbox tokens (`<loc####>`), which we (1) convert to bbox coordinates and (2) resolve for duplicates via IoU thresholding. **(d)** `paligemma-3b-mix-224` outputs PaliGemma bbox and segmenation mask tokens (`<seg###>`), which we (1) convert to bbox coordinates and resolve for duplicates and (2) convert to binary masks.

3.3 End-to-end fine-tuning

Because we replace multiple consecutive components in SLD with independent VLMs, one natural follow-up idea is to instead have a single VLM perform all of the SLD components' tasks in an end-to-end manner. To test this, we experiment with replacing the portion of the SLD pipeline between the LLM parser and the LLM controller with a single VLM. We try finetuning this VLM in two separate ways, which we refer to as **finetuning strategy #1 and #2**. In strategy #1, we provide the VLM with input consisting of I_{init} and prompt P, and train it to predict the entire text transcript of each component in the original SLD pipeline. In finetuning strategy #2, we give the VLM an additional input: the bounding box/segmentation masks in text format and train the VLM to output layout suggestions (see Figures 8 and 9 for a sample portion of the training transcript).

3.3.1 Data collection

On a high-level, we use SLD as our data collection pipeline to collect the prompt, image, and suggested image corrections. To facilitate fine-tuning PaliGemma, we incorporate PaliGemma as much as possible into the pipeline in order to leverage PaliGemma-specific tokens (e.g. `<loc####>` and `<seg###>`). More specifically, we run the pipeline with a VLM parser and controller, PaliGemma as OWL-ViT v2 replacement with self-feedback, and PaliGemma as SAM replacement. In addition, where possible, we convert bounding box coordinates to PaliGemma `<loc####>` tokens. We run the pipeline on all 500 prompts from LMD, CoCoCount, and T2I-Compbench, ultimately using an 80/20 split for training/testing. Below are the specific intermediate values that we extract from the pipeline as part of data collection for finetuning PaliGemma.

- Initial generated image I_{init}
- Prompt P
- PaliGemma detects bounding boxes and segmentation masks (refined via self-feedback) $\rightarrow$ List of bboxes and masks $D^{PG}_{bbox;mask}$

- VLM Parser → List of objects and their attributes O_{attr} specified by prompt, Image background specified by prompt B, Negation prompt P_N
- VLM Controller → List of bbox suggestions converted to PaliGemma tokens $O_{bbox}^{corrected;PG}$
- Resolve differences between controller-suggestions and detected bboxes → Convert to PaliGemma tokens → List of object bboxes to preserve $O_{preserve}^{PG}$; List of object bboxes to add O_{add}^{PG}; List of object bboxes to delete O_{del}^{PG}; List of object bboxes to reposition O_{repos}^{PG}; List of object bboxes to apply attribute modifications to $O_{attrmod}^{PG}$
- PaliGemma (replacing SAM) → List of masks to remove M_{del}^{PG}; List of masks to preserve $M_{preserve}^{PG}$

We finetune PaliGemma with pairs (X, Y). Per fine-tuning strategies #1 and #2, we formulate X and Y as follows.

1. $X = \{I_{init}, P\}$;
   $Y = \{O_{attr}, B, P_N, O_{bbox}^{corrected;PG}, O_{preserve}^{PG}, O_{add}^{PG}, O_{del}^{PG}, O_{repos}^{PG}, O_{attrmod}^{PG}, M_{del}^{PG}, M_{preserve}^{PG}\}$
2. $X = \{I_{init}, P, O_{attr}, B, P_N, D_{bbox;mask}^{PG}\}$;
   $Y = \{O_{bbox}^{corrected;PG}, O_{preserve}^{PG}, O_{add}^{PG}, O_{del}^{PG}, O_{repos}^{PG}, O_{attrmod}^{PG}, M_{del}^{PG}, M_{preserve}^{PG}\}$

4 Experiments

4.1 Our model vs. baseline SLD

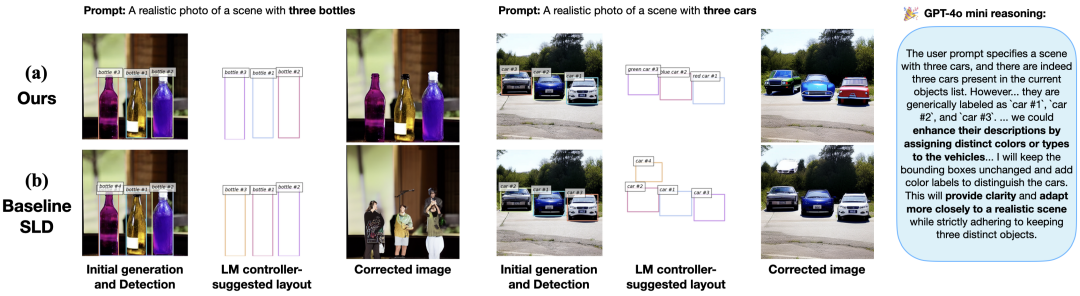


Figure 5: Baseline SLD vs Ours.

Our flagship model, which incorporates the VLM parser and controller, along with PaliGemma as detector with self-feedback, yields both quantitative and qualitative improvements over base SLD across all three benchmarks in numeracy and across two of the three benchmarks in FID (Table 1). Furthermore, from Figure 5, we see two examples where our model on row (a) has fewer artifacts than the baseline on row (b). In particular, the PaliGemma detector shows improved accuracy over OWL-ViT v2, VLM GPT-4o mini is effective at suggesting visual enhancements (shown on right), while adhering to the prompt.

	Numeracy ↑			**FID ↓**		
Method	LMD	CoCoCount	T2I-Compbench	LMD	CoCoCount	T2I-Compbench
Base SLD	0.750	0.545	0.170	238.992	198.222	**191.575**
Ours	**0.810**	**0.580**	**0.190**	**236.750**	**198.192**	192.217

Table 1: Baseline SLD vs. Ours. Our flagship model components are the VLM parser, PaliGemma detector with self-feedback, and VLM controller.

4.2 Ablation: LLM/VLM parser or controller or both?

In Table 2, we run ablations on replacing the parser and/or controller with a VLM. We find that in two out of three of the benchmarks, using a VLM parser improves numeracy and FID, so for section 4.3 ablation experiments, where we

seek to determine to what extent we should incorporate PaliGemma into the pipeline, we use a VLM parser and forego the use of VLM as controller. Furthermore, in Table 2, we report our ablation experiments on choice of output decoder for PaliGemma, and find that nucleus(0.3) is the best choice, which we use in all further experiments.

Comparing Base SLD against SLD w/ VLM controller in Table 2, we see that the introduction of VLM as controller improves numeracy on all three benchmarks and FID on CoCoCount. Figure 6 shows two examples where despite correct bbox predictions in both cases, the LLM controller incorrectly suggests adding a fourth object, despite the prompt specifying a count of three. The VLM controller (b) suggests a correct layout.

	Numeracy ↑			FID ↓		
Method	LMD	CoCoCount	T2I-Compbench	LMD	CoCoCount	T2I-Compbench
Base SLD	0.750	0.545	0.170	238.992	198.222	191.575
w/ VLM parser	**0.800**	0.570	**0.205**	237.888	**195.379**	**190.950**
w/ VLM controller	0.760	**0.585**	0.185	239.969	198.085	191.583
w/ VLM for both	0.780	0.570	0.185	**236.853**	198.838	191.213
PaliGemma w/ greedy decoder	0.490	0.300	0.105	246.459	200.835	204.072
PaliGemma w/ nucleus(0.3) decoder	**0.660**	**0.430**	**0.170**	**244.315**	**197.182**	**198.498**

Table 2: We find that with regards to the baseline, replacing the LLM parser with a VLM is most effective. We also experiment on choice of decoder for PaliGemma. Across the board, the nucleus(0.3) decoder outperforms the greedy decoder.

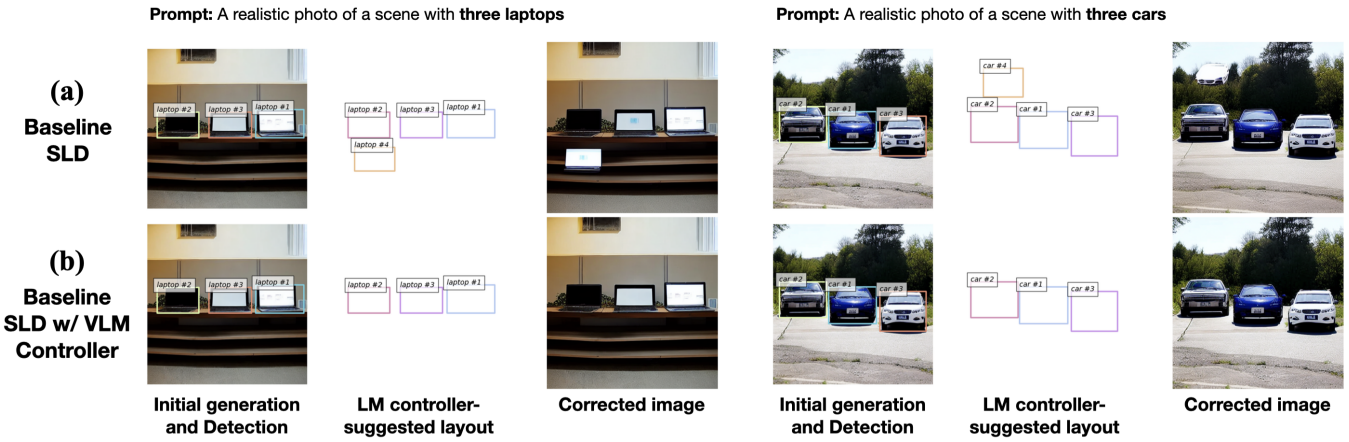


Figure 6: SLD vs. SLD w/ VLM Controller.

4.3 Ablation: the extent of PaliGemma incorporation

In Table 3, we see that PaliGemma detector w/ self-feedback and VLM parser outperforms all other combinations of PaliGemma detector/segmentor/feedback. And a deeper look shows that it even performs on par with Base SLD w/ VLM parser in Table 2, which is significant, because this means the PaliGemma detector reaches on-par performance with the OWL-ViT v2 detector and sometimes even yields a lower (better) FID in the case of LMD and T2I-Compbench.

We point out one other significant finding. The singular addition of PaliGemma segmentor (result row 2 in Table 3) to PaliGemma detector w/ VLM parser (result row 1) decreases performance across numeracy and FID across all benchmarks. Furthermore, the singular addition of PaliGemma segmentor (result row 4 in Table 3) to PaliGemma detector + self-feedback w/ VLM parser (result row 3) also decreases performance across scores and benchmarks, except one: numeracy on T2I-Compbench, where it reports the same accuracy. This leads us to believe PaliGemma is not effective enough to replace SAM completely.

Method	Numeracy ↑			FID ↓		
	LMD	CoCoCount	T2I-Compbench	LMD	CoCoCount	T2I-Compbench
PaliGemma Detector w/ VLM Parser	0.580	0.435	0.165	242.837	197.040	192.957
w/ PaliGemma Segmentor	0.440	0.325	0.090	275.719	213.247	231.880
w/ Self-feedback	**0.780**	**0.565**	**0.185**	**235.975**	**196.067**	**190.228**
w/ Self-feedback + PaliGemma Segmentor	0.700	0.510	**0.185**	244.436	206.740	208.126

Table 3: Ablation on the extent to which to incorporate PaliGemma.

4.4 Final ablation: revisiting which LLM/VLM parser/controller combination is best

Given that PaliGemma detector w/ self-feedback and VLM parser is the best-performing, VLM-centered pipeline, we make one final check to verify whether the additional introduction of a VLM controller may help in this scenario (though it was not the best performing in the baseline case). In Table 4, we see that there is a new winner, our flagship model! **PaliGemma detector with self-feedback and VLM for both parser and controller yields the highest numeracy across all benchmarks** (i.e. the addition of VLM controller relative to using just a VLM parser in the presence of PaliGemma detector (result row 3 relative to row 1 in Table 4) improved numeracy). This is interesting, since the similar addition of VLM controller relative to using just a VLM parser in the baseline SLD case (result row 4 relative to row 2 in Table 2) dropped numeracy. It may be interesting to look into this in future work.

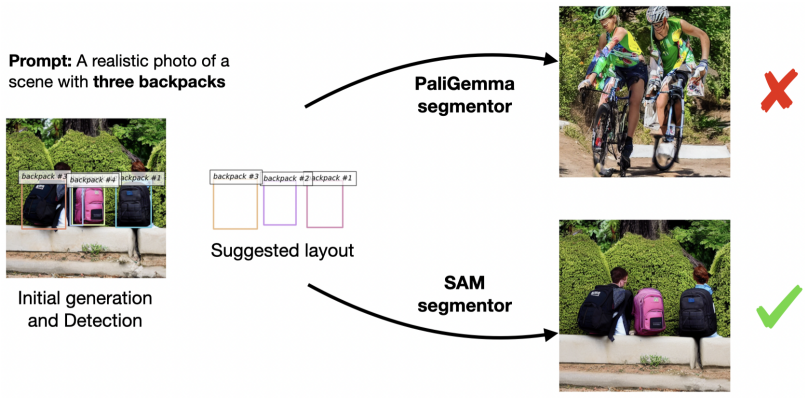


Figure 7: PaliGemma detector w/ self-feedback and VLM parser vs. with the addition of PaliGemma as SAM replacement.

Lastly, we try a final ablation introducing the PaliGemma segmentor into our best model and find that performance drops across all scores and benchmarks, which reinforces that PaliGemma is not effective enough to replace SAM completely. Figure 7 supports the same. Furthermore, we conclude that the performance boost afforded by the VLM controller can only be taken full advantage of when paired with a downstream, capable segmentor that can utilize the added boost in layout suggestions accurately.

Method	Numeracy ↑			FID ↓		
	LMD	CoCoCount	T2I-Compbench	LMD	CoCoCount	T2I-Compbench
PaliGemma Detector w/ self-feedback + VLM parser	0.780	0.565	0.185	**235.975**	**196.067**	190.228
PaliGemma Detector w/ self-feedback + VLM controller	0.780	0.550	**0.190**	237.306	198.397	**189.732**
PaliGemma Detector w/ self-feedback + VLM for both	**0.810**	**0.580**	**0.190**	236.750	198.192	192.217
w/ PaliGemma Segmentor	0.670	0.510	0.185	244.358	207.487	208.449

Table 4: Revisiting VLM/LLM parser/controller combinations with our yet highest-performing PaliGemma incorporation.

5 Discussion

5.1 Benefits of replacing SLD components with VLMs

Our flagship model improves upon the SLD baseline by replacing multiple components in the original pipeline with VLMs such as GPT-4o mini and PaliGemma. From our ablation studies, we can also see that individual replacements of components also yields strong performance boosts. Starting with the base SLD model, when we only replace the LLM parser with GPT-4o mini, numeracy accuracy improves across all datasets while FID scores remain comparable. Similarly, if we only replace the LLM controller in SLD with GPT-4o mini, numeracy accuracy once again increases across the board, and FID value drops. In another ablation in which we use PaliGemma detector with self-feedback, nucleus(0.3) sampling, and a VLM parser as a base method, adding an additional VLM controller also improves the numeracy performance while maintaining similar FID.

Our ablation studies also help show that replacing SAM within the SLD pipeline with a VLM segmentor (at least with PaliGemma) is not beneficial. For example, when starting with a VLM parser, nucleus(0.3) sampling, and a PaliGemma detector as a base model, replacing SAM with PaliGemma resulted in a sharp drop in numeracy accuracy and increased FID score. Adding self-feedback to the base model resulted in similar trends, in which using PaliGemma in place of SAM decreases model performance in terms of numeracy and FID. When comparing the performance of our flagship model versus a version of it with a PaliGemma segmentor, performance drop is also very apparent. This suggests that the advantages granted by an improved VLM controller cannot be fully expressed in final model performance without a strong enough segmentor such as SAM.

5.2 Importance of fine-tuning strategy for an end-to-end VLM

Both of our finetuning strategies are intended to help a single VLM make end-to-end corrections to diffusion-generated images. The first strategy, in which we ask the VLM to take in only the image and text prompt and generate the entire textual outputs of SLD's original components is inspired by chain-of-thought (CoT) reasoning. While it is plausible to have the VLM directly predict the final layout corrections, having it also predict intermediate text transcripts of SLD's components may help it better learn and reason about the complex task.

Our second finetuning strategy, in which we provide the VLM with additional information such as the parsed prompt and bounding box/seg masks, outperforms our first finetuning strategy (see the presence of less red in Figure 11 relative to Figure 10). Although the overall objective of this VLM still remains the same as before, namely generating modifications to the latent space of the generated image, the complexity of the problem drastically decreases. This is because in the previous finetuning setup, the VLM needs to accurately predict the behavior of the SLD pipeline's choice of parser and open-vocab detector. The inherent bias in these components may not be directly inferrable from the original image and text prompt–instead, by taking these as input to the VLM rather than as prediction target, we can make the VLM focus more on the **correction** aspect of the task, rather than the parser and open-vocab detector result generation.

6 Conclusion

In this paper, we explored the idea of using VLMs instead of LLMs within the SLD pipeline to improve diffusion models' numeracy ability with respect to the given text prompt. We show that by replacing SLD's LLM parser, open-vocab detector, and LLM controller with pretrained VLMs, we are able to achieve better numeracy accuracy and lower FID.

Given the promising results of replacing individual components in the SLD pipeline with VLMs, we also explored the idea of replacing the bulk of the SLD pipeline with a single, finetuned VLM. While this approach did not yield as practical results compared to the original SLD pipeline, we believe that these problems can be overcome by scaling up the size of the dataset, and by further refining the training strategy used for the VLM. Having a single, unified VLM that guides diffusion-based image generation could lead to many exciting possibilities that LLM-based guidance cannot offer, such as more pixel-level image modification.

References

Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, Thomas Unterthiner, Daniel Keysers, Skanda

Koppula, Fangyu Liu, Adam Grycner, Alexey A. Gritsenko, Neil Houlsby, Manoj Kumar, Keran Rong, Julian Eisenschlos, Rishabh Kabra, Matthias Bauer, Matko Bosnjak, Xi Chen, Matthias Minderer, Paul Voigtlaender, Ioana Bica, Ivana Balazevic, Joan Puigcerver, Pinelopi Papalampidi, Olivier J. Hénaff, Xi Xiong, Radu Soricut, Jeremiah Harmsen, and Xiaohua Zhai. Paligemma: A versatile 3b vlm for transfer. *CoRR*, abs/2407.07726, 2024. URL https://doi.org/10.48550/arXiv.2407.07726.

Lital Binyamin, Yoad Tewel, Hilit Segev, Eran Hirsch, Royi Rassin, and Gal Chechik. Make it count: Text-to-image generation with an accurate number of objects. *arXiv preprint arXiv:2406.10210*, 2024.

Junsong Chen, Jincheng YU, Chongjian GE, Lewei Yao, Enze Xie, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, and Zhenguo Li. Pixart-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=eAKmQPe3m1.

Aniket Rajiv Didolkar, Anirudh Goyal, Nan Rosemary Ke, Siyuan Guo, Michal Valko, Timothy P Lillicrap, Danilo Jimenez Rezende, Yoshua Bengio, Michael Curtis Mozer, and Sanjeev Arora. Metacognitive capabilities of LLMs: An exploration in mathematical problem solving. In *AI for Math Workshop @ ICML 2024*, 2024. URL https://openreview.net/forum?id=0MsI3bSmmD.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=FPnUhsQJ5B.

Kaiyi Huang, Kaiyue Sun, Enze Xie, Zhenguo Li, and Xihui Liu. T2i-compbench: A comprehensive benchmark for open-world compositional text-to-image generation. *Advances in Neural Information Processing Systems*, 36: 78723–78747, 2023.

Wonjun Kang, Kevin Galim, and Hyung Il Koo. Counting guidance for high fidelity text-to-image synthesis. *arXiv preprint arXiv:2306.17567*, 2023.

Chen Li, Weiqi Wang, Jingcheng Hu, Yixuan Wei, Nanning Zheng, Han Hu, Zheng Zhang, and Houwen Peng. Common 7b language models already possess strong math capabilities. *arXiv preprint arXiv:2403.04706*, 2024.

Long Lian, Boyi Li, Adam Yala, and Trevor Darrell. LLM-grounded diffusion: Enhancing prompt understanding of text-to-image diffusion models with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=hFALpTb4fR. Featured Certification.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

OpenAI. Gpt-4o mini: advancing cost-efficient intelligence. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/. Accessed: 2024-12-13.

OpenAI. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. SDXL: Improving latent diffusion models for high-resolution image synthesis. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=di52zR8xgf.

Vedant Shah, Dingli Yu, Kaifeng Lyu, Simon Park, Jiatong Yu, Yinghui He, Nan Rosemary Ke, Michael Mozer, Yoshua Bengio, Sanjeev Arora, and Anirudh Goyal. Ai-assisted generation of difficult math questions. *arXiv preprint arXiv:2407.21009*, 2024.

Ayush Singh, Mansi Gupta, Shivank Garg, Abhinav Kumar, and Vansh Agrawal. Beyond captioning: Task-specific prompting for improved vlm performance in mathematical reasoning. *arXiv preprint arXiv:2410.05928*, 2024.

Zayne Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. To cot or not to cot? chain-of-thought helps mainly on math and symbolic reasoning. *arXiv preprint arXiv:2409.12183*, 2024.

Tsung-Han Wu, Long Lian, Joseph E Gonzalez, Boyi Li, and Trevor Darrell. Self-correcting llm-controlled diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6327–6336, 2024.

A Appendix

A.1 Converting PaliGemma-specific grounding tokens into bounding box coordinates and binary masks

For detection: positionally, each quadruplet of <loc####> can be interpreted as follows: <loc{y_{min}}><loc{x_{min}}><loc{y_{max}}><loc{x_{max}}>. To convert to SLD bbox coordinates, we divide the y_{min}, x_{min}, y_{max}, x_{max} by 1024, multiply by the respective image dimensions (height for y and width for x), and divide by 512. For segmentation, each quadruplet of <loc####> followed by 16 <seg###> tokens can be interpreted as such: <loc{y_{min}}><loc{x_{min}}><loc{y_{max}}><loc{x_{max}}>{16 tokens which can be decoded to a seg mask}. Bbox tokens are converted similar to before, while the seg tokens are converted using a PaliGemma-specific codebook.

A.2 PaliGemma detector w/ self-feedback

After running the PaliGemma detector and resolving duplicate bounding boxes, we are left with n detections of object o. Leveraging this information, we prompt `paligemma-3b-mix-224` to **segment** $\{\mathbf{o}; \}^{\mathbf{n}}$, resolve duplicate predictions once again, and use the bounding boxes surrounding the unique masks as the final detected results (i.e. final bounding boxes).

```
Prompt: Image + A realistic photo of a scene with five bowls ????
Response: Objects: [['bowl', [None, None, None, None, None]]] #### Background: A
realistic image #### Negation: #### LLM Suggestions: <loc386><loc235><loc559><loc409>
bowl ; <loc387><loc043><loc553><loc209> bowl ; <loc388><loc627><loc554><loc795> bowl
; <loc389><loc816><loc547><loc982> bowl ; <loc387><loc422><loc561><loc608> bowl ; ####
Preservation: <loc389><loc816><loc547><loc982> bowl ; <loc387><loc422><loc561><loc608>
bowl ; <loc387><loc043><loc553><loc209> bowl ; <loc386><loc235><loc559><loc409> bowl ;
<loc388><loc627><loc554><loc795> bowl ; #### Addition: #### Deletion: #### Repositioning: ####
Attribute Modification: #### Remove Masks: #### Preservation Masks: ####

Prompt: Image + A photo of ten clocks ????
Response: Objects: [['clock', ['ten']]] #### Background: A photo #### Negation: #### LLM
Suggestions: <loc231><loc229><loc404><loc398> clock ; <loc225><loc029><loc392><loc198>
clock ; <loc222><loc847><loc402><loc1001> clock ; <loc000><loc000><loc424><loc1023>
clock ; <loc230><loc636><loc398><loc786> clock ; <loc224><loc427><loc399><loc591> clock
; <loc029><loc624><loc187><loc773> clock ; <loc206><loc006><loc395><loc187> clock ;
<loc235><loc458><loc401><loc598> clock ; <loc228><loc830><loc408><loc981> clock ; ####
Preservation: <loc029><loc624><loc187><loc773> clock ; <loc228><loc830><loc408><loc981>
clock ; <loc222><loc847><loc402><loc1001> clock ; <loc225><loc029><loc392><loc198>
clock ; <loc224><loc427><loc399><loc591> clock ; <loc231><loc229><loc404><loc398>
clock ; <loc230><loc636><loc398><loc786> clock ; <loc235><loc458><loc401><loc598>
clock ; <loc206><loc006><loc395><loc187> clock ; <loc000><loc000><loc424><loc1023>
clock ; #### Addition: #### Deletion: <loc029><loc015><loc204><loc198> clock ;
<loc037><loc219><loc194><loc384> clock ; <loc000><loc000><loc1010><loc1023> clock ;
<loc246><loc791><loc400><loc928> clock ; #### Repositioning: #### Attribute Modification: ####
Remove Masks: <loc0004><loc0000><loc0211><loc0201><seg064><seg004><seg057><seg071><seg022>
<seg066><seg026><seg022><seg022><seg066><seg106><seg082><seg072><seg041><seg089>
<seg075> ; <loc0035><loc0222><loc0199><loc0388><seg106><seg004><seg009><seg106>
<seg022><seg026><seg023><seg104><seg022><seg026><seg066><seg082><seg095><seg041>
<seg089><seg073> ; <loc0000><loc0000><loc1023><loc1018><seg067><seg053><seg115>
<seg053><seg067><seg053><seg109><seg067><seg036><seg067><seg002><seg060>
<seg077><seg067><seg040><seg012> ; <loc0218><loc0830><loc0400><loc1000><seg066>
<seg004><seg009><seg106><seg042><seg029><seg029><seg042><seg022>
<seg023><seg066><seg082><seg095><seg041><seg089><seg075> ; #### Preservation Masks: ####

Prompt: Image + A realistic photo of a scene with two books ????
Response: Objects: [['book', [None, None]]] #### Background: A realistic image #### Negation:
#### LLM Suggestions: <loc391><loc515><loc712><loc774> book ; <loc390><loc204><loc707><loc460>
book ; #### Preservation: <loc391><loc515><loc712><loc774> book ; #### Addition:
<loc390><loc204><loc707><loc460> book ; #### Deletion: #### Repositioning: ####
Attribute Modification: #### Remove Masks: #### Preservation Masks:", "Segmentation":
"<loc0391><loc0515><loc0712><loc0774><seg052><seg079><seg060><seg023><seg022>
<seg091><seg028><seg042><seg022><seg041><seg066><seg042><seg098><seg041> <seg005><seg051> book ;
####
```

Figure 8: Sample training transcript for finetuning strategy #1.

```
Prompt:
Image + A realistic photo of a scene with five bowls\n
Objects: [['bowl', [None, None, None, None, None]]]\n
Background: A realistic image\n
Negation: \n
<loc0386><loc0235><loc0559><loc0409><seg092><seg004><seg044><seg106><seg022>
<seg026><seg054><seg020><seg022><seg026><seg023><seg082><seg095><seg041>
<seg044><seg073> bowl ; <loc0387><loc0043><loc0553><loc0209><seg066><seg011>
<seg028><seg066><seg104><seg054><seg042><seg000><seg022><seg106><seg064>
<seg120><seg015><seg041><seg044><seg073> bowl ; <loc0388><loc0627>
<loc0554><loc0795><seg066><seg077><seg068><seg092><seg034><seg023><seg056>
<seg086><seg022><seg091><seg064><seg082><seg058><seg041><seg039><seg099> bowl
; <loc0389><loc0816><loc0547><loc0982><seg023><seg004><seg009><seg066><seg022>
<seg023><seg023><seg022><seg022><seg106><seg066><seg082><seg072><seg041>
<seg045><seg075> bowl ; <loc0387><loc0422><loc0561><loc0608><seg037><seg004>
<seg091><seg037><seg007><seg023><seg023><seg034><seg022><seg064><seg092>
<seg082><seg085><seg041><seg045><seg073> bowl ;\n
Response:
LLM Suggestions: <loc386><loc235><loc559><loc409> bowl ; <loc387><loc043><loc553><loc209>
bowl ; <loc388><loc627><loc554><loc795> bowl ; <loc389><loc816><loc547><loc982> bowl ;
<loc387><loc422><loc561><loc608> bowl ;\n
Preservation: <loc389><loc816><loc547><loc982> bowl ; <loc387><loc422><loc561><loc608>
bowl ; <loc387><loc043><loc553><loc209> bowl ; <loc386><loc235><loc559><loc409> bowl ;
<loc388><loc627><loc554><loc795> bowl ;\n
Addition: \n
Deletion: \n
Repositioning: \n
Attribute Modification: \n
Remove Masks: \n
Preservation Masks: \n

Prompt:
Image + A realistic photo of a scene with two books\n
Objects: [['book', [None, None]]]\n
Background: A realistic image\n
Negation: \n
<loc0391><loc0515><loc0712><loc0774><seg052><seg079><seg060><seg023><seg022>
<seg091><seg028><seg042><seg022><seg041><seg066><seg042><seg098><seg041> <seg005><seg051> book
;\n
Response:
LLM Suggestions: <loc391><loc515><loc712><loc774> book ;<loc390><loc204><loc707><loc460> book ;\n
Preservation: <loc391><loc515><loc712><loc774> book ;\n
Addition: <loc390><loc204><loc707><loc460> book ;\n
Deletion: \n
Repositioning: \n
Attribute Modification: \n
Remove Masks: \n
Preservation Masks: \n
```

Figure 9: Sample training transcript for finetuning strategy #2.

```
Prompt: Image + A realistic photo of a scene with three bowls ????
Response: Objects: [['bowl', [None, None, None]]] #### Background: A realistic photo ####
Negation: #### LLM: <loc332><loc1023><loc1023>####loc487loc714><loc1023><loc298> bowl ;
<loc137><loc1023> by<loc1023> about has <seg075><loc0000> <loc0000><loc1023><loc1023><seg000>
; <loc327><loc1023><loc1023><loc1010> by object hall of ‘loc159‘<loc1023><loc1023>
\<loc1023><loc1023> <loc1023><loc1023> of ’loc159’ in ‘loc 298‘ bowl ; ####
Preservation: <loc332><loc1023><loc1023><loc0000><loc0437><seg039><loc1023><loc1023>
‘loc 159‘ in ’loc 298‘ bowl ; <loc137><loc1023><loc1023><loc1023><loc1023><loc1023>
<loc1015><loc1023><loc1023><loc1018><loc1018><loc1023><loc1018> <loc1018><loc1018><loc1018><loc1018>
#### Preservation: #### Addition: #### Preservation: #### Preservation: #### [...]

Prompt: Image + A photo of two sports balls on the grass ????
Response: Objects: [['sports ball', [None]]] #### Background: A realistic image
#### Negation: #### LLM Suggestions: <loc462><loc453><loc543><loc684> sports ball ;
<loc445><loc000><loc541><loc201> sports ball ; <loc461><loc973><loc538><loc048> sports ball
; <loc466><loc667><loc536><loc1055> sports ball ; #### Preservation: #### Preservation:
#### (<loc0671><loc0513><loc0988><loc1023><loc1023><seg088><loc1022><seg045><seg025>
<seg013><seg001><seg065><seg036>) ; #### Preservation Masks: <loc462><loc453><loc543><loc684>
sports ball ; #### Preservation Masks: <loc466><loc667><loc536><loc1055> sports ball ; #### Iteration
masks: <loc445><loc000><loc541><loc201> sports ball ; #### Deletion Masks: #### Removal Masks:
<loc461><loc973><loc536><loc048> sports ball ; #### Deletion Masks: #### Preservation Masks:
```

Figure 10: Sample output transcript from fine-tuning strategy #1. In red are the artifacts.

```
Prompt:
Image + A realistic photo of a scene with three bowls\n
Objects: [['bowl', [None, None, None]]]\n
Background: A realistic photo\n
Negation:\n
<loc0395><loc0712><loc0587><loc0919><seg066><seg011><seg028><seg066><seg104>
<seg029><seg042><seg000><seg022><seg106><seg106><seg082><seg095><seg041>
<seg089><seg073> bowl ; <loc0403><loc0414><loc0592><loc0592><seg021><seg075>
<seg059><seg021><seg042><seg026><seg029><seg042><seg022><seg029><seg029>
<seg090><seg015><seg008><seg044><seg073> bowl ; <loc0400><loc0119>
<loc0591><loc0292><seg066><seg004><seg009><seg066><seg022>
<seg023><seg062><seg042><seg022><seg026><seg023><seg090><seg015> <seg041><seg044><seg073> bowl
;\n
Response:
LLM Suggestions: <loc400><loc716><loc591><loc722> bowl ; <loc404><loc593><loc591><loc726> bowl ;\n
Preservation: <loc400><loc716><loc591><loc722> bowl ; <loc404><loc593><loc591><loc726> bowl ;\n
Addition:\n
Deletion:\n
Repositioning:\n
Attribute Modification:\n
Remove Masks:\n
Preservation Masks:\n[...]
```

Figure 11: Sample output transcript from finetuning strategy #2.