(D) TokenCompose: Diving into multi-category generation

January 12, 2025

Raphaël Bernas
ENSTA-MVA
raphael.bernas@ensta.fr

Maxime Corlay
ENSTA-MVA
maxime.corlay@ensta.fr

Abstract

Text-to-image (T2I) generation has improved a lot since regression based models. The introduction of diffusion and later, efficiency enhancement with latents space based models, leads to highly realistic text-to-image generation. But state-of-the-art models such as Stable Diffusion have shown weaknesses in multi-categories generation. Thus, Zirui Wang, Zhizhou Sha et al. introduced "TokenCompose: Text-to-Image Diffusion with Token-level Supervision". This new method for training latent diffusion models yields impressive results on this peculiar task of multi-categories generation with high realism.

1. Introduction

See GitHub references in section 5. If you are interested by Latent Diffusion Model, you can find some more information in Appendix A (Section 4), **those are not useful for the rest of this paper**.

1.1. Introduction to Latent Diffusion models

The state-of-the-art architecture used to perform diffusion operations (such as in Stable Diffusion models) is U-Net [4],[5]. It is made of a contracting path and an expanding path (hence the U). More precisely, it is made of three main parts: down, middle and up. Note that the U-Net works in the latent space: U-Net takes a latent image as input, and returns a latent image as output. What does it mean ? It means that the image was already passed by a variational autoencoder before passing it to the U-Net. In our setup, input images are (512,512). When they go through the VAE, we obtain (64,64), which is their representation in latent space. So U-Net takes (64,64) pictures as input. The important idea is that down, middle and up are very similar in their structure: they are all made of convolutions, linear layers and most importantly attention layers. The key is to obtain attention layers (to satisfy our need to compute Ltoken and Lpixel), and this process is a bit more explained in the part "Simplified implementation of TokenCompose".

1.2. Overview of TokenCompose

To begin this section, let us comment on the work presented in the paper [7]. The authors' contributions can be divided into two main parts:

- Development and implementation of a new training method: TokenCompose
- Creation of a new benchmarking method: MultiGen

1.2.1. TokenCompose: A new training method

TokenCompose process consists of adding two new losses to the $\mathcal{L}_{simple}$:

$$\mathcal{L}_{\text{pixel}} = -\mathcal{M}_u \log \mathcal{A}_u - (1 - \mathcal{M}_u) \log(1 - \mathcal{A}_u)$$

$$\mathcal{L}_{\text{token}} = \left(1 - \frac{\sum_{u \in \mathcal{B}} \mathcal{A}_u}{\sum_u \mathcal{A}_u}\right)^2$$

where $\mathcal{A}_u$ is a cross-attention map and $\mathcal{M}_u$ the attention mask.

1.2.2. MultiGen: A new benchmarking method

The authors introduced **MultiGen Benchmark**, a new and challenging benchmarking method. The process steps of this method are described in the pipeline in Figure 2.

2. Simplified implementation of TokenCompose

This part was done by Maxime Corlay.

Why recode TokenCompose ? To deepen the theoretical understanding of the model, to identify the technical challenges faced by the original authors, and to deeply understand the Stable Diffusion architecture available on Hugging Face.

The reimplementation was structured around the Stable Diffusion 1.4 architecture, isolating its fundamental components: the tokenizer, the encoder, the variational autoencoder (VAE), the U-Net and the scheduler. The architecture of TokenCompose has been entirely redeveloped around these core modules.

This approach not only provides a detailed understanding of the underlying mechanisms, but also offers precise control over all the parameters of the model.

2.1. Setup

We used Google Colab environment with an NVIDIA A100 GPU (40 GB RAM). The tokenizer and scheduler were assigned to the CPU, while the variational auto-encoder, encoder and U-Net were deployed on the GPU.

2.2. Optimizer, trained parameters and number of epochs

Model optimization was performed using the AdamW algorithm with a learning rate of 10^{-5}. Only U-Net parameters were trained. The model was trained over 200 epochs.

2.3. Batch size, dataset

In this initial experimental phase, the study focused on a restricted dataset, using 10 images for training and 2 images for validation, selected from the COCO dataset provided in the original TokenCompose repository. Although the approach is extensible to larger datasets, this reduced configuration enabled rapid iterations.

Training was carried out with a batch size of 1, a constraint imposed by the memory limitations of the A100 GPU (40 GB). Several memory optimization strategies were tried:

- Release of temporary variables with the command `del`
- Limit the activation of the `train()` mode to learning phases only, to avoid an unnecessary storage of gradients

2.4. Comparison on training methods

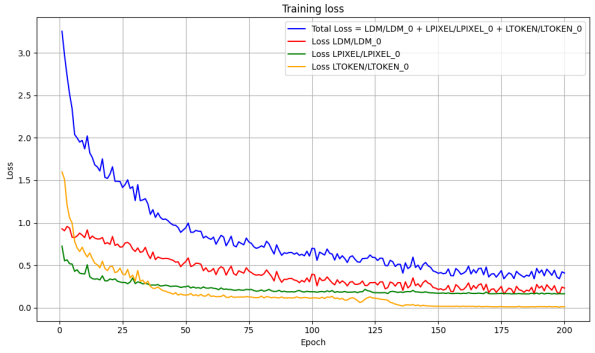
Three distinct loss functions were studied. A kind of normalization of the individual components of the loss was carried out to ensure a fair contribution of each term to the overall objective function. The following reference values were used: $LDM_0 = 20, LPIXEL_0 = 1, LTOKEN_0 = 10000$.

Three configurations were studied:

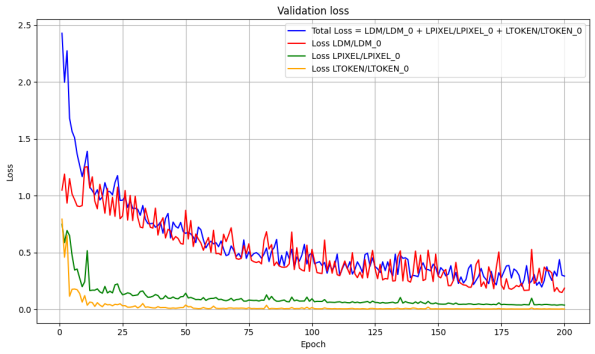
1. $L_{total} = L_{DM} + L_{token}$
2. $L_{total} = L_{DM} + L_{pixel}$
3. $L_{total} = L_{DM} + L_{pixel} + L_{token}$

As shown in Figures 3 (section 7), 4 (section 7), and 1, the token loss decreases rapidly, even faster than the pixel loss. While we initially balanced the contribution of each loss, the LDM loss becomes the most important contributor as training progresses.

In Appendix E (section 8), Figure 6 displays the cross-attention maps resulting from different training methods. The improvement in "plate" recognition between Stable Diffusion and TokenCompose is clearly visible.



(a) Train



(b) Validation

Figure 1. LDM+LTOKEN+LPIXEL

2.5. Challenges encountered

As said before, we expected that coding everything ourselves from scratch would lead to the same difficulties as the TokenCompose authors.

From a technical point of view, you have to learn how to read the Hugging Face pipeline code. It is a very dense code, on many different files.

The main difficulty we encountered was retrieving the cross attention maps. This required us to define 'hooks' (which was very instructive).

3. On extending TokenCompose

This part was done by Raphaël Bernas. As Maxime's part dived into basic TokenCompose experiments, this part won't detail further (For the full process, see Appendix C).

3.1. On a new complementary loss: Spatial loss

In TokenCompose, the authors did not focus on the spatial disposition of the multi-category objects. Thus, we introduced here the Spatial loss, a complementary loss to Token & Pixel loss. First, let us define $\mathcal{R}(u, u')$, a spatial relationship between object u and object u' (for instance, u is to the

"left" of u'):

$$\mathcal{R}(u,u') = \begin{cases} 1 & \text{if } u \textbf{ is to the left of } u', \\ 0 & \text{otherwise ,} \end{cases}$$

For all u and u' in the prompt such that $\mathcal{R}(u,u') = 1$:

$$\mathcal{L}_{spatial} := 1 - MSE\left((\mathcal{A}_u - \mathcal{A}_{u'}) \times (2\mathbf{1}_{(\mathcal{M}_u > \mathcal{M}_{u'})} - 1)\right)$$

where MSE is the mean squared error; the term $(\mathcal{A}_u - \mathcal{A}_{u'})$ represents the attention map on which the object u is positive and the object u' negative; and $(2\mathbf{1}_{(\mathcal{M}_u > \mathcal{M}_{u'})} - 1)$ denotes a vector which value is 1 when the object u' (its mask) is to the "left" of u and -1 otherwise. The higher $MSE\left((\mathcal{A}_u - \mathcal{A}_{u'}) \times (2\mathbf{1}_{(\mathcal{M}_u > \mathcal{M}_{u'})} - 1)\right)$ is, the more it is accurate to say that u' is to the left of u (then we can do the same for multiple others relationships between words by changing a bit the formula). Testing the new loss on the following relationships : "left", "above", "right" and "under" (these are not the keywords used to identify object relationship in the prompt). To see all losses together, refer to Figure 5.

3.2. On multi-category benchmarking of LDM

MultiGen: A partial test

MultiGen benchmarking process is GPU expensive. In order to make proper tests, we cannot reduce the resolution of produced images ($resolution = 512$), nor can we reduce too much the inference steps (here we will fix it to 35 steps). We computed upon $N_{prompt} = 10$ and $N_{image/prompt} = 2$. All the tests are performed on a GPU *Nvidia RTX3050 Laptop*. This is not a high level GPU which explains why we were set on reducing the computational cost. We tested seven models : Stable Diffusion v1.4 (SD1.4), SD1.4 with Token Compose (SD1.4TK), SD1.4 with Token Compose checkpoint 2000 (SD1.4TK2000), SD1.4 with only token loss checkpoint 2000 (SD1.4T2000), SD1.4 with only pixel loss checkpoint 2000 (SD1.4P2000), SD1.4 with TokenCompose + spatial loss - as described above - checkpoint 2000 (SD1.4TKS2000) and Stable Diffusion v3.5. We will not be providing MG1 and MG5 as they both do not reveal much on models (as explained in the original paper).

OptiGen: a MultiGen complementary and computationally-free benchmark for T2I As described above, MultiGen is a benchmarking process which aims to exhaustively test a T2I models (particularly LDM). Unfortunately, such a benchmarking method is computationally expensive. Indeed, it requires the generation of a vast number of images over a variety of prompts. This allows MultiGen to be a highly confident benchmarking solution and, for a well-defined prompt-dataset, an almost exhaustive method. But as models tend to increase in height, their forward path take much more time to be computed. Thus, producing thousands of images becomes computationally difficult (for example, StableDiffusion3.5 is more than three times heavier than StableDiffusion1.4). Thus, we introduce here OptiGen, a benchmarking method that tries to reduce the computation time. This process goes as follows (see Figure 2):

1. For a given word-dataset, use a language prediction model to compute for each word **A** and **B**, the probability that "**B**" follows "a picture of **A** and". This should return a matrix of probabilities $\mathcal{P}$ (to correctly use it, we need to normalize it so that it becomes a doubly stochastic matrix).
2. Take the $N_{\text{prompt}} = 5$ couples of words $(\mathbf{A};\mathbf{B})$ with the lowest $\mathcal{P}(\mathbf{B}|\mathbf{A})$, the probability matrix value. For each couple, generate $N_{\text{image/prompt}} = 10$ images using the prompt "a picture of **A** and **B**".
3. *Same as MultiGen step 3*: Use a strong open-vocabulary detector model to list the objects in each generated image.
4. *Same as MultiGen step 4*: Compute the accuracy of object class alignment with expected objects.
5. *Same as MultiGen step 5*: Compute the probability for each category to be present in the image.

Why use such a method? OptiGen is not only motivated by computational cost. Indeed, in the task of multi-category generation, we often observe that text-to-image models underperform when faced with prompts containing unrelated categories (refer to the original article [7]). Thus, the objective here is to test a model in the worst condition possible. This benchmark is computationally cheap and gives a first idea of how badly a model performs.

Results obtained On this new benchmark method we tested the same models tested on MultiGen. The OptiGen Benchmark can be extended to test on multi-category with more than 2 classes but to make it the most efficient, we use it on 2 classes. Thus, the accuracy computed corresponds to MG2 (the accuracy of generation of the worst second class possible). The results can be found on Table 1.

Conclusion: The results from Table 1 tend to show that while TokenCompose is a very efficient method, MultiGen benchmark is limited. OptiGen has reasonably ranked each models and shows that TokenCompose is a strong improvement. Finally, spatialization was sparingly implemented here but has already shown good results on OptiGen. We can expect that with a benchmark based on spatial identification, our Spatial TokenCompose model would have outperformed others.

Model	MG2	MG3	MG4	OptiGen Acc
SD1.4	0.90	0.55	0.05	0.52
SD1.4TK	0.95	0.60	0.20	0.96
SD1.4TK2000†	0.95	0.45	0.25	0.74
SD1.4T2000†	0.95	0.45	0.15	0.52
SD1.4P2000†	0.95	0.55	0.10	0.62
SD1.4TKS2000†	0.90	0.35	0.05	0.80
SD3.5 (large)	1.00	1.00	1.00	0.90

Table 1. MultiGen Benchmarking on seven models. † : refer to model we trained.

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arxiv:2006.11239*, 2020. 4

[2] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 4

[3] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021. 4

[4] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021. 1, 4

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015. 1, 4

[6] Jascha Sohl-Dickstein, Eric A Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *arXiv preprint arXiv:1503.03585*, 2015. 4

[7] Zirui Wang, Zhizhou Sha, Zheng Ding, Yilin Wang, and Zhuowen Tu. Tokencompose: Text-to-image diffusion with token-level supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8553–8564, 2024. 1, 3

4. Appendix A : Diving more into LDM

WARNING : This part is not meant to complete the introduction. It is just meant for those interested in the subject of Latent Diffusion Model.

Latent Diffusion Models (LDM) have been first introduced in [4]. But the process called "diffusion" was first presented in [6]. Then detailled to be a process for generating high quality images in [1]. The processus goes as follows [1]:
Given an image $\mathbf{x}_0$, a time step $t \in [|1, T|]$ and a corruption schedule $(\beta_t)_{t\in[|1,T|]}$ we have the transition gaussian kernel :

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t|\sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

This is the **corruption**-process of diffusion. We gradually corrupt our image so that it ultimately becomes gaussian noise from which we can easily sample. Then, begin the **denoising**-process [1]: We train a Neural Network (NN) to predict the noise added to our image. Let us note θ our denoising NN weight and ϵ_θ the model which denoise our images. To train this NN we backpropagate on the following **loss** (with $\epsilon \sim \mathcal{N}(\mathbf{0}, 1)$):

$$\mathcal{L}_{simple} := \mathbb{E}_{t,\mathbf{x}_0,\epsilon}\left(\left|\left|\epsilon - \epsilon_{\theta,t}\left(\mathbf{x}_0, \epsilon, (\beta_s)_{s\in[|1,t|]}\right)\right|\right|_2^2\right)$$

The model commonly used to perform those operations in Stable Diffusion models is U-Net [4],[5]. Its name comes from the form the model's forward path takes. It is made of a contracting path and an expanding path. Both face each other with the following process : The first contracting layer use convolution to extract features which are passed to the second contracting layer as well as the last expanding layer. And then you reciprocate this mirror process while diving more into the U.
In 2022, R. Rombach, A. Blattmann et al introduced Latent Diffusion models [4]. The main idea behind LDM is to apply the same process we described above but in a latent space. This allow to greatly reduce the computational cost while maintaining high quality results. After embedding our noisy image in a latent space and using U-Net to reconstruct it we use a decoding process to return it in the image space. Typically, this is handled by a Variational Autoencoder (VAE) [2]. Finally, to make it into a proper T2I model we need to constrain on a text our image reconstruction process. This is handled in the latent space using a text encoding model, while cross-attention layer handle the reconstructed image to make it into a proper text based generated image. Such a process is dependant of a Contrastive Language–Image Pretraining (CLIP) model [3].

5. Appendix B : Code GitHub reference

Everything we have done is reproducible. It means that you can find our code online. Most of our code is machine-agnostic and can be run on Google COLAB.
Original GitHub TokenCompose :
https://github.com/mlpc-ucsd/TokenCompose
The code of our different works can be found here :
Recoding from scratch in one notebook:
https://github.com/maximecorlay/MVA-ORCV-Final-Project
SD Fine-Tuning :
https://github.com/Raphael-Bernas/Stable-Diffusion-Fine-Tuning-UNet-
OptiGen :
https://github.com/Raphael-Bernas/OptiGen$_benchmark$
Training TokenCompose (contain the code for the Spatial loss) :
https://github.com/Raphael-Bernas/Training$_TokenCompose$

6. Appendix C : FPP

Raphaël's Part

(a) Making an easy-to-compute fine-tuning process for SD1.4 on any Hugging Face dataset.

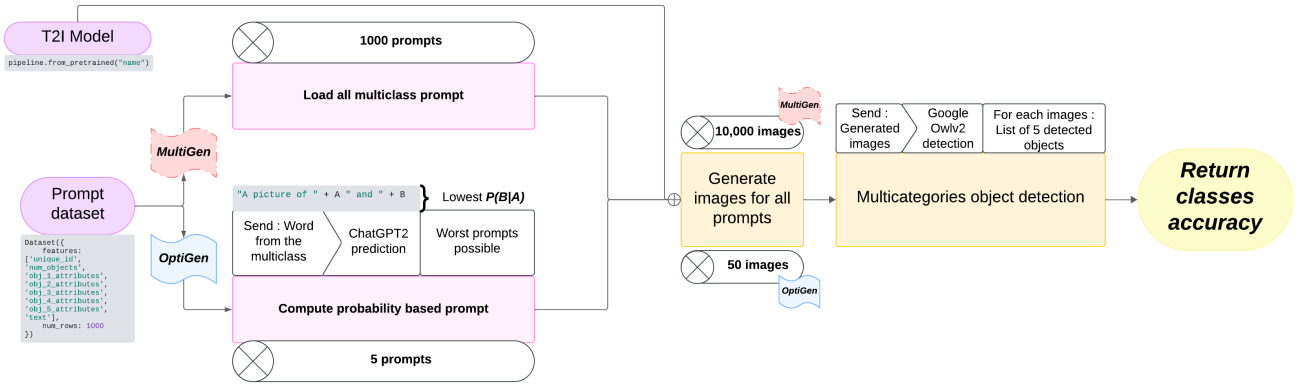


Figure 2. An **illustrative** overview of multi-category benchmarking method.

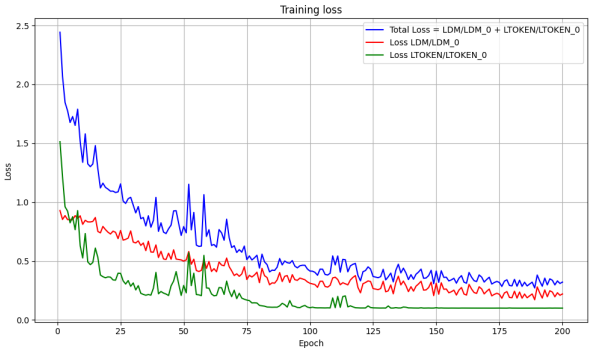
Given a prompt dataset, MultiGen will generate $10,000$ images using those prompts while OptiGen will only focus on *useful* prompts which test the model on the worst possible circumstances. They both use a detection model to obtain class accuracies.

(b) Testing TokenCompose, SD1.4, and SD3.5 with a reduced MultiGen benchmark.
(c) Training with TokenCompose method SD1.4 (with all losses, without pixel loss, without token loss).
(d) Testing the trained models from step (c) with the same MultiGen process, observing MultiGen limitations.
(e) Implementing a new benchmark method to test models on multi-category generation: **OptiGen**.
(f) Making an improvement for TokenCompose: adding a spatial loss.

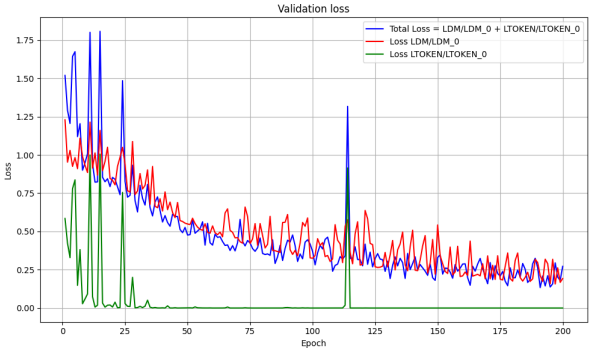
(For all the training process, it was performed on a GPU A100 from Google Colab. While the benchmarks where performed on a GPU RTX 3050 Laptop)

Maxime's Part

(a) Understand what happens precisely in the model and in the paper.
(b) Try to recode a simple version of TokenCompose based on knowledge of the Hugging Face libraries and reading of the paper.
(c) Particularly, try to understand how to get the cross-attention maps from the U-net module using a "hook".
(d) Conduct experiments on a small dataset to train the model with:
  (a) `loss = LDM + LPIXEL`
  (b) `loss = LDM + LTOKEN`
  (c) `loss = LDM + LTOKEN + LPIXEL`



(a) Train



(b) Validation

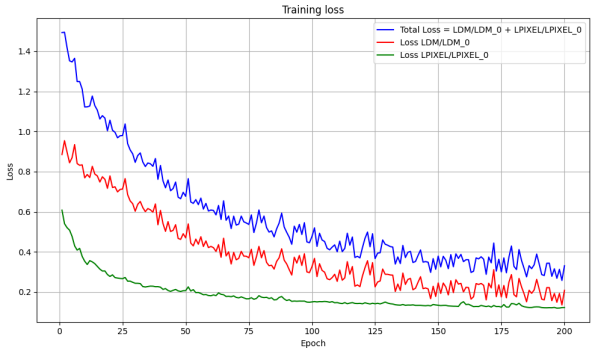
Figure 3. LDM+LTOKEN

7. Appendix D : other losses

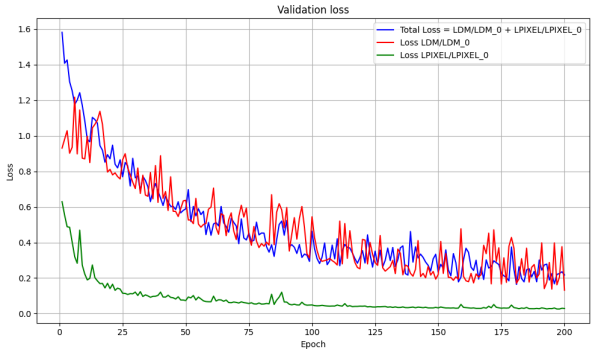
See Fig 3, 4 and 5.

8. Appendix E : cross attentions

See Fig 6 and 7.



(a) Train



(b) Validation

Figure 4. LDM+LPIXEL

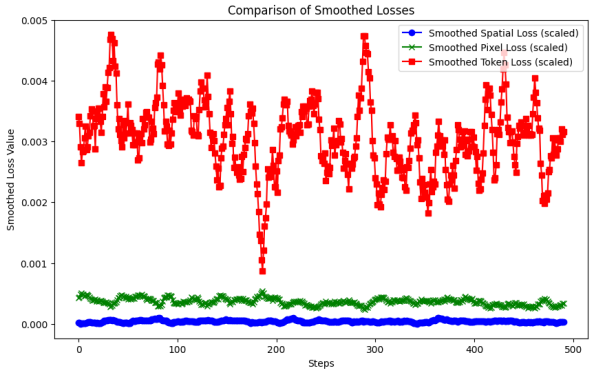
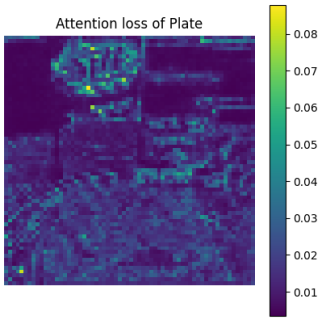
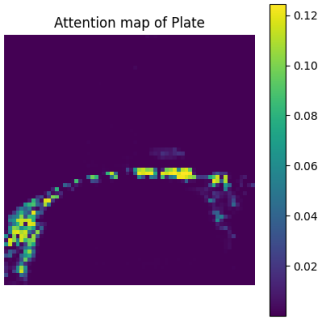


Figure 5. Three losses together scaled by their weight in the final loss and smoothed.

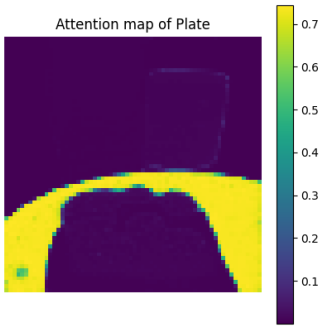
Here are three losses : Token loss, Pixel loss and Spatial loss. They were smoothed to be more readable. We can observe that Spatial loss is frequently null, this is because most of the time there is no explicit relationship between objects in the prompt.



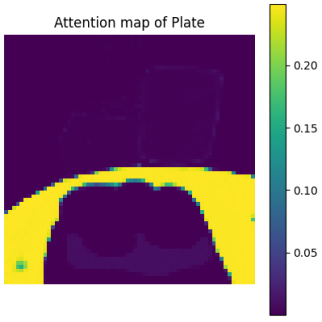
(a) SD14 (LDM only)



(b) LDM+LTOKEN

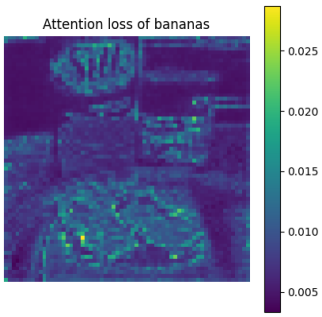


(c) LDM+LPIXEL

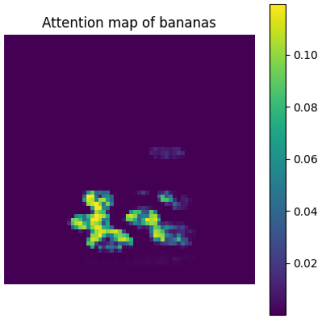


(d) LDM+LPIXEL+LTOKEN

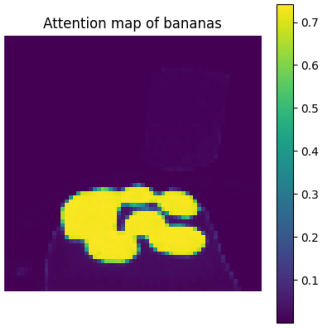
Figure 6. Cross-Attention Map of ”plate” in a picture



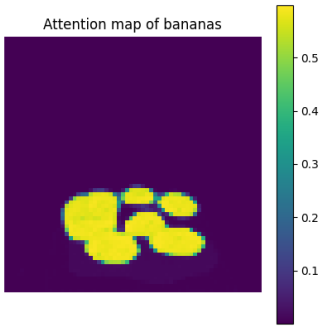
(a) SD14 (LDM)



(b) LDM+LTOKEN



(c) LDM+LPIXEL



(d) LDM+LTOKEN+LPIXEL

Figure 7. Cross-Attention Map of ”bananas” in a picture