Improving the Language Representation of Prompts for Complex Text-to-Image Generation

Gal Almog (ID. 206350175)

Submitted as final project report for the NLP course, IDC, 2023

1 Introduction

Recent large-scale text-to-image models have given users the ability to intuitively generate new images from only textual prompts. However, current state-of-the-art (SOTA) text-to-image models often fail to convey the semantics of long, complex prompts. The two most common issues are neglect and incorrect attribute binding; in neglect, the model completely fails to generate one or more subjects or attributes, and incorrect attribute binding refers to the model incorrectly binding a given attribute (ex. a colour) to the corresponding subject.

Fig. 1 shows a naive baseline generation using Stable Diffusion v1.4 [6] (a SOTA text-to-image model) to generate an image from the complex prompt "A dog wearing a blue hat and sunglasses and a cat are riding a bicycle made out of spaghetti". It can be seen that the model performs very poorly on this prompt; there is catastrophic neglect of multiple tokens and attributes, and the present attributes are almost all incorrectly bound. In this work, I attempted to improve the language representation of the embedded prompt that is used for image generation to mitigate these failure cases.

Figure 1: Naive baseline: four random samples of generated images using Stable Diffusion v1.4 [6] conditioned on the full prompt "A dog wearing a blue hat and sunglasses and a cat are riding a bicycle made out of spaghetti".

In text-to-image diffusion models, token embeddings are passed into

the text encoder to condition the image generation. A text tokenizer first splits the input prompt into tokens, with a special token marking the end of the sentence (EOS). Each token corresponds to a pre-trained embedding that is retrieved from a pre-defined dictionary of vector embeddings (ie. the dictionary is a lookup table that connects each token to a unique embedding vector). After retrieving the vectors for a given sentence from the table, they are concatenated and passed to a pretrained text encoder, which processes the connections between the individual words in the sentence and outputs the encoding, which is then used to condition the diffusion model. This text conditioning is injected into the cross attention layers of the U-Net that is part of the diffusion model.

1.1 Related Works

Recent works [3, 7] have accomplished text-to-image generation with personalized concepts (ex. a user's unique dog or a specific personal object) by replacing the vector associated with the tokenized string with a new, learned embedding: S^*. After injecting it into the vocabulary with a straightforward initiation (such as a similar general concept), S^* can be found through optimization. Textual Inversion (TI) [3] uses an input set of 3-5 images of the unique concept, and at each step randomly sample an image from the set and a neutral context text (eg."A photo of S^*"), and noise the image through diffusion. Then, they feed the noisy image, timestep, and encoding into a diffusion model that predicts the noise. S^* is then optimized to describe the concept in the set of images.

I adapted this concept and attempted to split the input prompt into an optimal list of sequential mini-prompts, so that I can generate specific parts of the prompt each time, encode them as a single token, and recursively add more objects/attributes to arrive at the final result. By optimizing the exact mini-prompts, I am effectively manipulating the language representation of the prompts to try to improve how the text is encoded. For example, given the complex prompt "A dog wearing a blue hat and sunglasses and a cat are riding a bicycle made out of spaghetti", is it better to generate the dog with sunglasses first, and then add the cat? Or first the dog and cat, and later add the sunglasses and hat? Or first generate the bike because it is bigger? I attempt to find consistent patterns that can be generalized to all prompts to better understand the role that the text embeddings play in text-to-image generation and how they can be optimized to achieved the desired results. Fig. 2 demonstrates my overall pipeline of recursive mini-prompt optimization.

1.2 Motivation

I wanted to explore whether splitting and optimizing the input prompt based on constituency parsing might improve the embedding accuracy and prevent neglect of tokens or incorrect attribute binding when generating images from a text prompt. By analyzing the relations between different tokens in the prompt, it may be possible to identify patterns or dependencies that govern the cor-

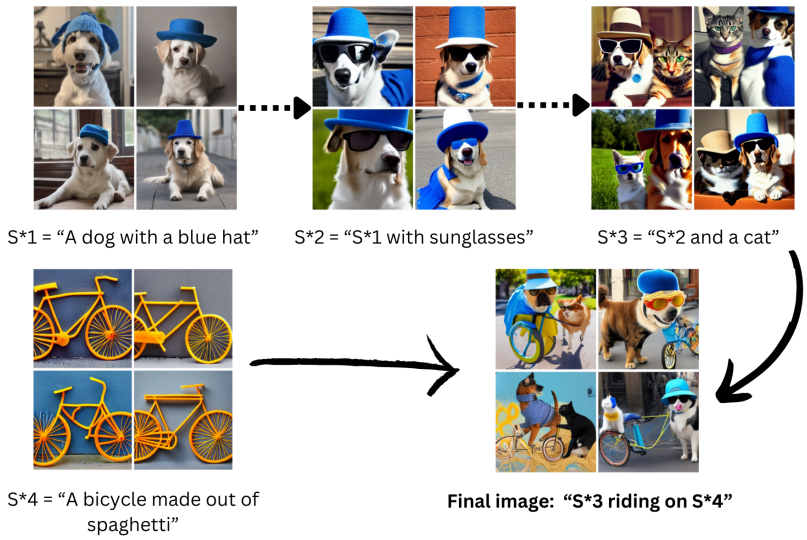

Figure 2: Visual outline of the pipeline. Tokens are recursively optimized and used to generate more complex tokens. Example is for the prompt “A dog wearing a blue hat and sunglasses and a cat are riding a bicycle made out of spaghetti”.

rect attribute binding. A deeper understanding of these patterns can guide the generation process towards more accurate embeddings, ensuring that the attributes are appropriately associated with the correct objects. As well, optimizing the order of generation with the mini-prompts enables exploration of the impact of different language elements on the generation performance. This procedure may reveal that certain attributes should be generated before/after others, with/without others, or with certain keywords attached to them in the prompt. For example, generating the dog with sunglasses first before adding the cat might lead to more coherent and accurate embeddings.

As well, by leveraging the cross-attention mechanism within the U-Net architecture, splitting the prompt into mini-prompts can improve the contextual relevance of the model during text-to-image generation. The diffusion process text conditioning works by injecting the text embedding into the cross attention of the U-Net of the diffusion model. The U-Net architecture consists of an encoder-decoder structure with skip connections that facilitate the generation of high-quality images. The addition of the cross-attention mechanism into the UNet [6] allows the model to attend to relevant information from both the image and the text embedding, enabling efficient integration of the textual context into the image generation process. Splitting the prompt into mini-prompts may enable the diffusion model to focus the cross-attention mechanism only on specific parts of the prompt at each step. By attending to individual mini-prompts

during different stages of the generation process, the model can accurately capture the relationships between objects and attributes, leading to more precise and context-aware image generation. The cross-attention mechanism also helps the model identify and utilize the most important information for generating the image. Separating the prompt into mini-prompts can help ensure coherence and consistency in the generated images. The cross-attention mechanism allows the model to attend to both the current mini-prompt and previously generated information, ensuring that the generated images align with the previously generated context and any attributes introduced in previous mini-prompts.

By identifying consistent patterns in the relationship between objects and attributes, they can be generalized to improve the overall embedding accuracy and prevent neglect or incorrect attribute binding in complex prompt generations. A successful optimization algorithm that can do this for an individual prompt will likely take a rather long time to run. However, such an algorithm can be used to create a novel dataset of complex prompt and optimal mini-prompt set. This is the ultimate goal - as this can be used to train a deep learning algorithm that can learn much more complex language structures and patterns. Once trained, it may be used to generate an image given an input complex prompt, without the lengthy optimization process of trying many permutations.

2 Solution

2.1 General approach

My goal was to build a model that receives a complex prompt, and outputs an ordered list of mini-prompts that can later be used to sequentially generate images. My pipeline begins with creating a syntax tree (parse tree) for the given prompt. A parse tree is a tree representation of the different syntactic categories of a sentence; I used this to represent the syntactical structure of a sentence and be able to take advantage of the additional information to optimize the prompt. This is similar to part-of-speech (POS) tagging, in which each token gets assigned a label which reflects its word class. In the parse tree, each sentence is represented as a tree structure which reflects how its components are related to each other. I generated my parse trees using the spaCy library [4]. Including this syntactical and relational information may help the model to identify which attributes belong to which objects. Fig. 3 visualizes the parse tree for the previous sample prompt. After analyzing many different complex prompts, I created a list of hard-coded rules to create the list of mini-prompts from the parse tree. This is the general procedure:

1. Encode all sibling leaf nodes into a single mini-prompt. Assign a unique identifier S_i^* to each i mini-prompt. Ex:

   (a) mini-prompt S_1^*: "A dog wearing a blue hat"

(b) mini-prompt S_2^*: "S1 with sunglasses"

(c) mini-prompt S_3^*: "S2 and a cat"

(d) mini-prompt S_4^*: "A bicycle made out of spaghetti"

2. Combine mini-prompts according to the parse tree. Ex:

   (a) Final mini-prompt: Combine S_3^* and S_4^* using "riding on." Resulting in "S_3^* riding on S_4^*."

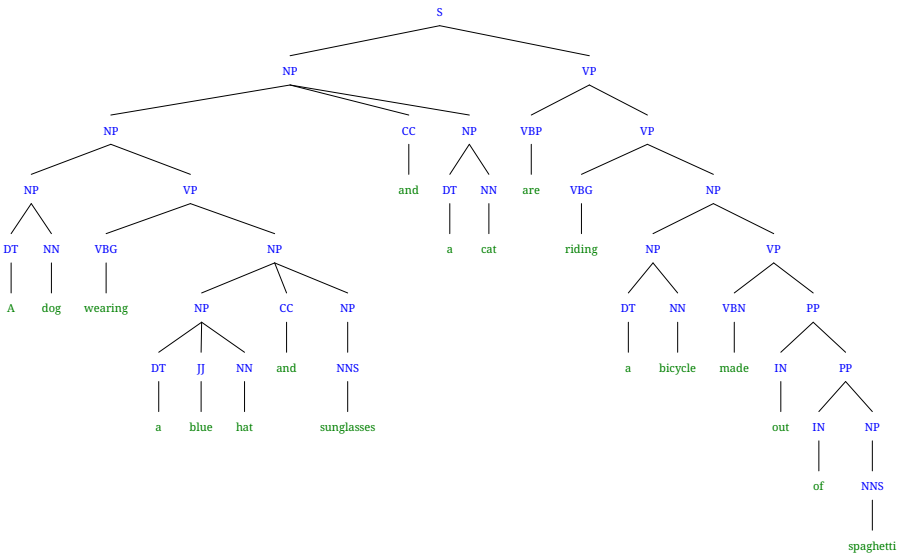

Figure 3: Visualized results of the prompt constituency parsing. This is done as the first stage in my pipeline before determining how to create and permute the mini-prompts. Graph visual generated with RSyntaxTree v1.2.9.

2.2 Design

My code is adapted from the NeTI paper [1], which is similar to TI [3] but implements a new text-conditioning space that is dependent on both the denoising process timestep and the U-Net layers, allowing for more precise generation and tuning during inference time. The code is based on the Stable Diffusion [6] code-base, and implements new classes and functions to implement the embedding optimization. A pseudocode overview of my algorithm is provided in Algorithm 1. For automatically testing which permutation of the mini-prompts results in the best generated image, I used the PickScore model [5]. PickScore is a powerful preference predictor that receives a prompt and two images and generates a human preference score for how well each image represents the given

Algorithm 1: Text-to-Image Generation with Mini-Prompts

Input: Prompt **P**, Random seed s
Output: Generated image x and list of mini-prompts, **MP**

$\mathbf{P}^* \leftarrow$ Extracted parse tree from **P**;
Split $\mathbf{P}^*$ into [**MP**];
Generate 5 random permutations of [**MP**];
$maxPickScore \leftarrow -\infty$;
$selectedImage \leftarrow$ None;
foreach perm *in* [**permutations**] **do**
    foreach *mini-prompt* **mp** *in* **perm do**
        Generate image x_{mp} through a diffusion process conditioned on **mp**;
    Generate final image x_{perm} by combining intermediate optimized tokens;
    Calculate PickScore ps for x_{perm} ;
    if $ps > maxPickScore$ **then**
        $maxPickScore \leftarrow ps$;
        $selectedImage \leftarrow x_{perm}$;
Compile $[\mathbf{MP}]_f$ used for $selectedImage$;
return $selectedImage$, $[\mathbf{MP}]_f$

prompt. Each token optimization took around 1 hour to run on a GPU (comparable to TI), and inference using the token was faster at around 30 seconds per image. With an average of 5 optimizations per prompt, this resulted in roughly 5 hours to optimize a single prompt (not including inference time between each optimization to generate the input images). I have tried implementing various papers [2, 3], but settled on NeTI due to the improved dropout tuning at inference time, which allowed me to analyze more parameters during inference from a single optimization. Unfortunately, running the optimization scripts was more challenging than anticipated, and I was not able to run everything completely automatically (script scripts/main.py). I was able to execute the optimizations one by one, and analyze them manually, which also took around a day per prompt due to the lengthy process. Some other technical challenges I had were with merging the model embeddings from two optimizations - as this required a lot of fine understanding of the model's attention mechanism.

3 Experimental results

My results for the discussed sampled prompt are shown in Fig. 4. The results demonstrate a marked improvement over the naive baseline (simple single-prompt Stable Diffusion, Fig. 1). Returning to our two main deficiencies in text-to-image generation, the improved pipeline provides an improvement on

both. In terms of catastrophic neglect, there is a large improvement in the generated images: all prompt elements are present in all images. The attribute binding, while also not perfect, is also more accurate than the baseline. We can see that the model still mixes up some attributes; for example, putting sunglasses on the cat and not only the dog, or putting spaghetti on the dog's hat. Overall, the pipeline seems to greatly improve the attribute retention when dealing with long, complex prompts. Of course, analysis on many varied prompts is required to conduct more thorough experiments, but again, this exceeded my current GPU capacity. Due to a lack of GPU resources and each experiment taking many hours to run, I was not able to complete many experiments. Once I am able to automate the pipeline successfully, this will enable many more experiments on varying prompts which will allow me to conduct a more thorough analysis.

Figure 4: Random sample of final results of the sample complex prompt generation. A clear improvement can be seen over the baseline results.

My experiments at this stage consisted mainly of attempting different mini-prompt permutations (manually) to see the impact of different linguistic structures on the resulting image. For example, placing one or more noun phrases together in the same mini-prompt, combining noun phrases and adjectives in different stages, and combining a different number of attributes in each mini-prompt. I first found that putting the regular token (non-optimized) first in the combined mini-prompt is better. In this example, "cat and S_1" provides better results than "S_1 and cat". When placing the optimized token first, there is very severe neglect ("cat" is not generated at all). This can be seen in Fig. 5. When placing the optimized token first, the model tends to give it too much weight and it overpowers the remaining tokens. As well, I have found that colors are very commonly leaked to other objects, whereas if they are added towards the end they are better attached to their corresponding object.

4 Discussion

Overall, while I was not able to accomplish the full task of optimizing the entire pipeline, I was able to still gain some interesting insights. When it comes to building complex text-to-image prompts sequentially, it is better to place the optimized tokens after the regular tokens. As well, color attributes and other

Figure 5: Effect of placing optimized token before or after other subjects. A: generation results for "S_1 and cat". B: generation results for "cat and S_1".

adjectives should be added towards the end, whereas relations should be determined earlier in the generation process. The reason for this is likely due to the nature of the token embeddings in the latent diffusion models. Important continuations for this project will be to further analyze the cross attention maps in each iteration of the generation process, to observe which tokens are attending to which pixels at each stage. This will allow a more in-depth observation on the impact of the linguistic structures on the generated images. It will also be interesting to apply these insights into the diffusion process itself - as the model can be manipulated to have different text embeddings used for different cross attention layers throughout the generation. So, for example, maybe simply applying these insights for the cross attention manipulations rather than the token optimizations can also have significant gains. There are also many important generation parameters, both during training and inference time, that have a strong impact on the generation process. I have tried to increase the number of vectors that represent each token in the optimized embeddings, but was not successful in doing so. I believe that increasing the number of vectors per token (it is currently implemented as 1) may allow the model to learn more complex representations in each token, and allow for better generation. While this will likely come at a cost of editability, I believe that for this purpose this would be preferred.

5 Code

My code and model checkpoints are available at the following repository:

https://github.com/galmog/NLP_Final_Project_2023

References

[1] Yuval Alaluf, Elad Richardson, Gal Metzer, and Daniel Cohen-Or. A neural space-time representation for text-to-image personalization. *arXiv preprint arXiv:2305.15391*, 2023.

[2] Weixi Feng, Xuehai He, Tsu-Jui Fu, Varun Jampani, Arjun Akula, Pradyumna Narayana, Sugato Basu, Xin Eric Wang, and William Yang Wang. Training-free structured diffusion guidance for compositional text-to-image synthesis. *arXiv preprint arXiv:2212.05032*, 2022.

[3] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion. *arXiv preprint arXiv:2208.01618*, 2022.

[4] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

[5] Yuval Kirstain, Adam Polyak, Uriel Singer, Shahbuland Matiana, Joe Penna, and Omer Levy. Pick-a-pic: An open dataset of user preferences for text-to-image generation. 2023.

[6] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[7] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22500–22510, 2023.