Expressive Text-to-Image Generation with Rich Text

Songwei Ge\textsuperscript{1} \quad Taesung Park\textsuperscript{2} \quad Jun-Yan Zhu\textsuperscript{3} \quad Jia-Bin Huang\textsuperscript{1}

\textsuperscript{1}University of Maryland, College Park \quad \textsuperscript{2}Adobe Research \quad \textsuperscript{3}Carnegie Mellon University

https://rich-text-to-image.github.io/

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figures/teaser.png}
\caption{Plain text (left image) vs. Rich text (right image). Our method allows a user to describe an image using a rich text editor that supports various text attributes such as font family, size, color, and footnote. Given these text attributes extracted from rich text prompts, our method enables precise control of text-to-image synthesis regarding colors, styles, and object details compared to plain text.}
\end{figure}

Abstract

Plain text has become a prevalent interface for text-to-image synthesis. However, its limited customization options hinder users from accurately describing desired outputs. For example, plain text makes it hard to specify continuous quantities, such as the precise RGB color value or importance of each word. Furthermore, creating detailed text prompts for complex scenes is tedious for humans to write and challenging for text encoders to interpret. To address these challenges, we propose using a rich-text editor supporting formats such as font style, size, color, and footnote. We extract each word’s attributes from rich text to enable local style control, explicit token reweighting, precise color rendering, and detailed region synthesis. We achieve these capabilities through a region-based diffusion process. We first obtain each word’s region based on attention maps of a diffusion process using plain text. For each region, we enforce its text attributes by creating region-specific detailed prompts and applying region-specific guidance, and maintain its fidelity against plain-text generation through region-based injections. We present various examples of image generation from rich text and demonstrate that our method outperforms strong baselines with quantitative evaluations.

1. Introduction

The development of large-scale text-to-image generative models \cite{96, 98, 97, 95} has propelled image generation to an unprecedented era. The great flexibility of these large-scale models further offers users powerful control of the generation through visual cues \cite{4, 17, 77} and textual inputs \cite{7, 19}. Without exception, existing studies use plain text encoded by a pretrained language model to guide the generation. However, in our daily life, it is rare to use only plain text when working on text-based tasks such as writing blogs or editing essays. Instead, a rich text editor \cite{68, 71} is the more popular choice providing versatile formatting options for writing and editing text. In this paper, we seek to introduce accessible and precise textual control from rich text editors to text-to-image synthesis.

Rich text editors offer unique solutions for incorporating conditional information separate from the text. For example, using the font color, one can indicate an arbitrary color. In contrast, describing the precise color with plain text proves more challenging as general text encoders do...
not understand RGB or Hex triplets, and many color names, such as ‘olive’ and ‘orange’, have ambiguous meanings. This font color information can be used to define the color of generated objects. For example, in Figure 1, a specific yellow can be selected to instruct the generation of a marble statue with that exact color.

Beyond providing precise color information, various font formats make it simple to augment the word-level information. For example, reweighting token influence [19] can be implemented using the font size, a task that is difficult to achieve with existing visual or textual interfaces. Rich text editors offer more options than font size – similar to how font style distinguishes the styles of individual text elements, we propose using it to capture the artistic style of specific regions. Another option is using footnotes to provide supplementary descriptions for selected words, simplifying the process of creating complex scenes.

But how can we use rich text? A straightforward implementation is to convert a rich-text prompt with detailed attributes into lengthy plain text and feed it directly into existing methods [54, 19, 7]. Unfortunately, these methods struggle to synthesize images corresponding to lengthy text prompts involving multiple objects with distinct visual attributes, as noted in a recent study [12]. They often mix styles and colors, applying a uniform style to the entire image. Furthermore, the lengthy prompt introduces extra difficulty for text encoders to interpret accurate information, making generating intricate details more demanding.

To address these challenges, our insight is to decompose a rich-text prompt into two components (1) a short plain-text prompt (without formatting) and (2) multiple region-specific prompts that include text attributes, as shown in Figure 2. First, we obtain the self- and cross-attention maps using a vanilla denoising process with the short plain-text prompt to associate each word with a specific region. Second, we create a prompt for each region using the attributes derived from rich-text prompt. For example, we use “mountain in the style of Ukiyo-e” as the prompt for the region corresponding to the word “mountain” with the attribute “font style: Ukiyo-e”. For RGB font colors that cannot be converted to the prompts, we iteratively update the region with region-based guidance to match the target color. We apply a separate denoising process for each region and fuse the predicted noises to get the final update. During this process, regions associated with the tokens that do not have any formats are supposed to look the same as the plain-text results. Also, the overall shape of the objects should stay unchanged in cases such as only the color is changed. To this end, we propose to use region-based injection approaches.

We demonstrate qualitatively and quantitatively that our method generates more precise color, distinct styles, and accurate details compared to plain text-based methods.

2. Related Work

Text-to-image models. Text-to-image systems aim to synthesize realistic images according to descriptions [82, 42]. Fueled by the large-scale text-image datasets [60, 8], various training and inference techniques [20, 62, 21, 22], and scalability [51], significant progress has been made in text-to-image generation using diffusion models [4, 51, 45, 56, 17], autoregressive models [52, 76, 11, 15], GANs [59, 28], and their hybrids [54]. Our work focuses on making these models more accessible and providing precise controls. In contrast to existing work that uses plain text, we use a rich text editor with various formatting options.

Controllable image synthesis with diffusion models. A wide range of image generation and editing applications are achieved through either fine-tuning pre-trained diffusion models [55, 32, 77, 3, 72, 30, 41, 35] or modifying the denoising process [43, 13, 19, 46, 5, 12, 2, 4, 26, 6, 58, 78, 9, 48, 48, 73, 16]. For example, Prompt-to-prompt [19] uses attention maps from the original prompt to guide the spatial structure of the target prompt. Although these methods can be applied to some rich-text-to-image applications, the results often fall short, as shown in Section 4. Concurrent with our work, Mixture-of-diffusion [26] and MultiDiffusion [6] propose merging multiple diffusion-denoising processes in different image regions through linear blending. Instead of relying on user-provided regions, we automatically compute regions of selected tokens using attention maps. Gradient [24] and Universal [5] guidance control the generation by optimizing the denoised generation at each time step. We apply them to precise color generation by designing an objective on the target region to be optimized.

Attention in diffusion models. The attention mechanism has been used in various diffusion-based applications such as view synthesis [37, 66, 70], image editing [19, 12, 47, 46, 32], and video editing [38, 49, 10, 40]. We also leverage the spatial structure in self-attention maps and alignment information between texts and regions in cross-attention maps for rich-text-to-image generation.

Rich text modeling and application. Exploiting information beyond the intrinsic meanings of the texts has been previously studied [44, 63, 75, 34]. For example, visual information, such as underlining and bold type, have also been extracted for various document understanding tasks [75, 34]. To our knowledge, we are the first to leverage rich text information for text-to-image synthesis.

Image stylization and colorization. Style transfer [18, 81, 39] and Colorization [53, 64, 74, 33, 79, 80] for editing real images have also been extensively studied. In contrast, our work focuses on local style and precise color control for generating images from text-to-image models.
3. Rich Text to Image Generation

From writing messages on communication apps, designing websites [57], to collaboratively editing a document [36, 25], a rich text editor is often the primary interface to edit texts on digital devices. Nonetheless, only plain text has been used in text-to-image generation. To use formatting options in rich-text editors for more precise control over the black-box generation process [1], we first introduce a problem setting called rich-text-to-image generation. We then discuss our approach to this task.

3.1. Problem Setting

As shown in Figure 2, a rich text editor supports various formatting options, such as font styles, font size, color, and more. We leverage these text attributes as extra information to increase control of text-to-image generation. We interpret the rich-text prompt as JSON, where each text element consists of a span of tokens $e_i$ (e.g., ‘church’) and attributes $a_i$; describing the span (e.g., ‘color: #FF9900’). Note that some tokens $e_i$ may not have any attributes. Using these annotated prompts, we explore four applications: 1) local style control using font style, 2) precise color control using font color, 3) detailed region description using footnotes, and 4) explicit token reweighting with font sizes.

Font style is used to apply a specific artistic style $\alpha_i$, e.g., $\alpha_i = \text{‘Ukiyo-e’}$, to the synthesis of the span of tokens $e_i$. For instance, in Figure 1, we apply the Ukiyo-e painting style to the ocean waves and the style of Van Gogh to the sky, enabling the application of localized artistic styles. This task presents a unique challenge for existing text-to-image models, as there are limited training images featuring multiple artistic styles. Consequently, existing models tend to generate a uniform mixed style across the entire image rather than distinct local styles.

Font color indicates a specific color of the modified text span. Given the prompt “a red toy”, the existing text-to-image models generate toys in various shades of red, such as light red, crimson, or maroon. The color attribute provides a way for specifying a precise color in the RGB color space, denoted as $\alpha_i^c$. For example, to generate a toy in fire brick red, one can change the font color to “a toy”, where the word “toy” is associated with the attribute $\alpha_i^c = [178, 34, 34]$. However, as shown in the experiment section, the pretrained text encoder cannot interpret the RGB values and have difficulty understanding obscure color names, such as lime and orange.

Footnote provides supplementary explanations of the target span without hindering readability with lengthy sentences. Writing detailed descriptions of complex scenes is tedious work, and it inevitably creates lengthy prompts [29, 27]. Additionally, existing text-to-image models are prone to ignoring some objects when multiple objects are present [12], especially with long prompts. Moreover, excess tokens are discarded when the prompt’s length surpasses the text encoder’s maximum length, e.g., 77 tokens for CLIP models [30]. We aim to mitigate these issues using a footnote string $\alpha_i^f$.

Font size can be employed to indicate the importance, quantity, or size of an object. We use a scalar $\alpha_i^s$ to denote the weight of each token.
Figure 3. **Region-based diffusion.** For each element of the rich-text input, we apply a separate diffusion process to its region. The attributes are either decoded as a region-based guidance target (e.g. re-coloring the church), or as a textual input to the diffusion UNet (e.g. handling the footnote to the garden). The self-attention maps and feature maps extracted from the plain-text generation process are injected to help preserve the structure. The predicted noise \( \epsilon_{t,a_i} \), weighted by the token map \( M_{e_i} \), and the guidance gradient \( \frac{\partial L}{\partial M_{e_i}} \) are used to denoise and update the previous generation \( x_t \) to \( x_{t+1} \). The noised plain text generation \( x^\text{plain}_t \) is blended with the current generation to preserve the exact content in those regions of the unformatted tokens.

### 3.2. Method

To utilize rich text annotations, our method consists of two steps, as shown in Figure 2. First, we compute the spatial layouts of individual token spans. Second, we use a new region-based diffusion to render each region’s attributes into a globally coherent image.

**Step 1. Token maps for spatial layout.** Several works [65, 40, 4, 19, 12, 47, 67] have discovered that the attention maps in the self- and cross-attention layers of the diffusion UNet characterize the spatial layout of the generation. Therefore, we first use the plain text as the input to the diffusion model and collect self-attention maps of size \( 32 \times 32 \times 32 \times 32 \) across different heads, layers, and time steps. We take the average across all the extracted maps and reshape the result into \( 1024 \times 1024 \). Note that the value at \( i^{th} \) row and \( j^{th} \) column of the map indicates the probability of pixel \( i \) attending to pixel \( j \). We average the map with its transpose to convert it to a symmetric matrix. It is used as a similarity map to perform spectral clustering [61, 69] and obtain the binary segmentation maps \( \hat{M} \) of size \( K \times 32 \times 32 \), where \( K \) is the number of segments.

To associate each segment with a textual span, we also extract cross-attention maps for each token \( w_j \):

\[
m_j = \frac{\exp(s_j)}{\sum_k \exp(s_k)},
\]

(1)

where \( s_j \) is the attention score. We first interpolate each cross-attention map \( m_j \) to the same resolution as \( \hat{M} \) of \( 32 \times 32 \). Similar to the processing steps of the self-attention maps, we compute the mean across heads, layers, and time steps to get the averaged map \( \hat{M}_j \). We associate each segment with a texture span \( e_i \) following Patashnik et al. [47]:

\[
M_{e_i} = \{ \hat{M}_k \mid \hat{M}_k \cdot \frac{\hat{m}_j - \min(\hat{m}_j)}{\max(\hat{m}_j) - \min(\hat{m}_j)} > \epsilon, \forall j \text{ s.t. } w_j \in e_i \},
\]

(2)

where \( \epsilon \) is a hyperparameter that controls the labeling threshold, that is, the segment \( \hat{M}_k \) is assigned to the span \( e_i \) if the normalized attention score of any tokens in this span is higher than \( \epsilon \). We associate the segments that are not assigned to any formatted spans with the unformatted tokens \( e_U \). Finally, we obtain the token map in Figure 2 as below:

\[
M_{e_i} = \sum_i \sum_{j \in e_i} \frac{\hat{M}_j}{\sum_j \sum_{j \in e_i} \hat{M}_j}
\]

(3)

**Step 2. Region-based denoising and guidance.** As shown in Figure 2, given the text attributes and token maps, we divide the overall image synthesis into several region-based denoising and guidance processes to incorporate each attribute, similar to an ensemble of diffusion models [32, 6]. More specifically, given the span \( e_i \), the region defined by its token map \( M_{e_i} \), and the attribute \( a_i \), the predicted noise \( \epsilon_{t,a_i} \) for noised generation \( x_t \) at time step \( t \) is

\[
\epsilon_t = \sum_i M_{e_i} \cdot \epsilon_{t,a_i} = \sum_i M_{e_i} \cdot D(x_t, f(e_i, a_i), t),
\]

(4)

where \( D \) is the pretrained diffusion model, and \( f(e_i, a_i) \) is a plain text representation derived from text span \( e_i \) and attributes \( a_i \) using the following process:

1. Initially, we set \( f(e_i, a_i) = e_i \).
2. If footnote \( a_i^f \) is available, we set \( f(e_i, a_i) = a_i^f \).
3. The style \( a_s^i \) is appended if it exists. 
   \[ f(e_i, a_s^i) = f(e_i, a_i) + \text{"in the style of"} + a_s^i. \]
4. The closest color name (string) of font color \( \hat{c}_i^e \) from a predefined set \( C \) is prepended. 
   \[ f(e_i, a_i) = \hat{c}_i^e + f(e_i, a_i). \]
   For example, \( \hat{c}_i^e = \text{‘brown’} \) for RGB color \( \alpha_i^e = [136,68,20] \).

We use \( f(e_i, a_i) \) as the original plain text prompt of Step 1 for the unformatted tokens \( e_U \). This helps us generate a coherent image, especially around region boundaries.

**Guidance.** By default, we use classifier-free guidance [23] for each region to better match the prompt \( f(e_i, a_i) \). In addition, if the font color is specified, to exploit the RGB values information further, we apply gradient guidance [24, 14, 3] on the current clean image prediction:

\[
\hat{x}_0 = \frac{x_t - \sqrt{1 - \alpha_t} e_t}{\sqrt{\alpha_t}},
\]

where \( x_t \) is the noisy image at time step \( t \), and \( \alpha_t \) is the coefficient defined by noise scheduling strategy [20]. Here, we compute an MSE loss \( \mathcal{L} \) between the average color of \( \hat{x} \) weighted by the token map \( M_{e_i} \), and the RGB triplet \( \alpha_i^e \).

The gradient is calculated below,

\[
\frac{d\mathcal{L}}{dx_t} = \frac{d\| \sum_p (M_{e_i} \cdot \hat{x}_0) / \sum_p M_{e_i} - \alpha_i^e \|^2_2}{dx_t} \frac{1}{\sqrt{\alpha_t}dx_0},
\]

where the summation is over all pixels \( p \). We then update \( x_t \) with the following equation:

\[
x_t \leftarrow x_t - \lambda \cdot M_{e_i} \cdot \frac{d\mathcal{L}}{dx_t},
\]

where \( \lambda \) is a hyperparameter to control the strength of the guidance. We use \( \lambda = 1 \) unless denoted otherwise.

**Token reweighting with font size.** Last, to re-weight the impact of the token \( w_j \) according to the font size \( a_j^e \), we modify its cross-attention maps \( m_j \). However, instead of applying direct multiplication as in Prompt-to-Prompt [19] where \( \sum_j a_j^e m_j \neq 1 \), we find that it is critical to preserve the probability property of \( m_j \). We thus propose the following reweighting approach:

\[
\hat{m}_j = \frac{a_j^e \exp(s_j)}{\sum_k a_k^e \exp(s_k)}. \tag{8}
\]

We can compute the token map (Equation 3) and predict the noise (Equation 4) with the reweighted attention map.

**Preserve the fidelity against plain-text generation.** Although our region-based method naturally maintains the layout, there is no guarantee that the details and shape of the objects are retained when no rich-text attributes or only the color is specified, as shown in Figure 12. To this end, we follow Plug-and-Play [67] to inject the self-attention maps and the residual features extracted from the plain-text generation process when \( t > T_{\text{pnp}} \) to improve the structure fidelity. In addition, for the regions associated with the unformatted tokens \( e_U \), stronger content preservation is desired. Therefore, at certain \( t = T_{\text{blend}} \), we blend the noised sample \( x_t^\text{plain} \) based on the plain text into those regions:

\[
x_t = M_{e_U} \cdot x_t^\text{plain} + (1 - M_{e_U}) \cdot x_t \tag{9}
\]

4. **Experimental Results**

**Implementation details.** We use Stable Diffusion V1-5 [53] for our experiments. To create the token maps, we use the cross-attention layers in all blocks, excluding the first encoder and last decoder blocks, as the attention maps in these high-resolution layers are often noisy. We discard
A night sky filled with stars (1st Region: Van Gogh) above a turbulent sea with giant waves (2nd Region: Ukiyo-e).

The awe-inspiring sky and sea (1st Region: J.M.W. Turner) by a coast with flowers and grasses in spring (2nd Region: Monet).

Figure 5. Qualitative comparison on style control. We show images generated by Prompt-to-Prompt, InstructPix2Pix, and our method using prompts with multiple styles. Only our method can generate distinct styles for both regions.

Figure 6. Quantitative evaluation of local style control. We report the CLIP similarity between each stylized region and its region prompt. Our method achieves the best stylization.

Figure 7. Quantitative evaluation on precise color generation. Distance against target color is reported (lower is better). Our method consistently outperforms baselines.

Font style evaluation. We compute CLIP scores \[50\] for each local region to evaluate the stylization quality. Specifically, we create prompts of two objects and styles. We create combinations using 7 popular styles and 10 objects, resulting in 420 prompts. For each generated image, we mask it by the token maps of each object and attach the masked output to a black background. Then, we compute the CLIP score using the region-specific prompt. For example, for the prompt “a lighthouse (Cyberpunk) among the turbulent waves (Ukiyo-e)”, the local CLIP score of the lighthouse region is measured by comparing its similarity with the prompt “lighthouse in the style of cyberpunk.” We refer to “lighthouse” as the first region and “waves” as the second region in this example.

Font color evaluation. To evaluate a method’s capacity to understand and generate a specific color, we divide colors into three categories. The Common color category contains 17 standard names, such as “red”, “yellow”, and “pink”. The HTML color names are selected from the web color names \[1\] used for website design, such as “sky blue”, “lime green”, and “violet purple”. The RGB color category contains 50 randomly sampled RGB triplets to be used as “color of RGB values [128, 128, 128]”. To create a complete

\[1\]https://simple.wikipedia.org/wiki/Web_color
A coffee table\(^1\) sits in front of a sofa\(^2\) on a cozy carpet. A painting\(^3\) on the wall. Cinematic lighting, trending on artstation, 4k, hyperrealistic, focused, extreme details.

\(^1\)A rustic wooden coffee table adorned with scented candles and many books.
\(^2\)A plush sofa with a soft blanket and colorful pillows on it.
\(^3\)A painting of wheat field with a cottage in the distance, close up shot, trending on artstation, HD, calm, complimentary color, realistic lighting, by Albert Bierstadt, Frederic Church.

<table>
<thead>
<tr>
<th>Stable Diffusion (Plain-Text)</th>
<th>Stable Diffusion (Full-Text)</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attend-and-Excite</td>
<td>Prompt-to-Prompt</td>
<td>InstructPix2Pix</td>
</tr>
</tbody>
</table>

![Stable Diffusion (Plain-Text)](image1)
![Stable Diffusion (Full-Text)](image2)
![Ours](image3)

Figure 8. Qualitative comparison on detailed description generation. We show images generated by Attend-and-Excite, Prompt-to-Prompt, InstructPix2Pix, and our method using complex prompts. Our method is the only one that can generate all the details faithfully.

Prompt, we use 12 objects exhibiting different colors, such as “flower”, “gem”, and “house”. This gives us a total of 1,200 prompts. We evaluate color accuracy by computing the mean L2 distance between the region and target RGB values. We also compute the minimal L2 distance as sometimes the object should contain other colors for fidelity, e.g., the “black tires” of a “yellow car”.

**Baselines.** For font color and style, we quantitatively compare our method with two strong baselines, Prompt-to-Prompt [19] and InstructPix2Pix [7]. When two instructions exist for each image in our font style experiments, we apply them in parallel (InstructPix2Pix-par) and sequential manners (InstructPix2Pix-seq). More details are in Appendix B. We also perform a human evaluation with these two methods in Appendix Table 1. For re-weighting token importance, we visually compare with Prompt-to-Prompt [19] and two heuristic methods, repeating and adding parentheses. For complex scene generation with footnotes, we also compare with Attend-and-Excite [12].

### 4.1. Quantitative Comparison

We report the local CLIP scores computed by a ViT-B/32 model in Figure 6. Our method achieves the best overall CLIP score compared to the two baselines. This demonstrates the advantage of our region-based diffusion method for localized stylization. To further understand the capacity of each model to generate multiple styles, we report the metric on each region. Prompt-to-Prompt and InstructPix2Pix-par achieve a decent score on the 1st Region, i.e., the region first occurs in the sentence. However, they often fail to fulfill the style in the 2nd Region. We conjecture that the Stable Diffusion model tends to generate a uniform style for the entire image, which can be attributed to single-style training images. Furthermore, InstructPix2Pix-seq performs the worst in 2nd Region. This is because the first instruction contains no information about the second region, and the
Ours: A pizza with pineapples, pepperonis, and mushrooms

Prompt-to-Prompt

A pizza with pineapples, pepperonis, and mushrooms, mushrooms, mushrooms, mushrooms

A pizza with pineapples, pepperonis, and (((((mushrooms)))))

Figure 9. **Qualitative comparison on token reweighting.** We show images generated by our method and Prompt-to-Prompt using token weight of 13 for ‘mushrooms’. Prompt-to-Prompt suffers from artifacts due to the large weight. Heuristic methods like repeating and parenthesis do not work well.

**Figure 10. Ablation of token maps.** Using solely cross-attention maps to create token maps leads to inaccurate segmentations, causing the background to be colored in an undesired way.

second region’s content could be compromised when we apply the first instruction.

We show quantitative results of precise color generation in Figure 7. The distance of HTML color is generally the lowest for baseline methods, as they provide the most interpretable textual information for text encoders. This aligns with our expectation that the diffusion model can handle simple color names, whereas they struggle to handle the RGB triplet. Our rich-text-to-image generation method consistently improves on the three categories and two metrics over the baselines.

### 4.2. Visual Comparison

**Precise color generation.** We show qualitative comparison on precise color generation in Figure 4. InstructPix2Pix [7] is prone to create global color effects rather than accurate local control. For example, in the flower results, both the vase and background are changed to the target colors. Prompt-to-Prompt [19] provides more precise control over the target region. However, both Prompt-to-Prompt and InstructPix2Pix fail to generate precise colors. In contrast, our method can generate precise colors for all categories and prompts.

**Local style generation.** Figure 5 shows a visual comparison of local style generation. When applying InstructPix2Pix-seq, the style in the first instruction dominates the entire image and undermines the second region. Figure 13 in Appendix shows that this cannot be fully resolved using different hyperparameters of classifier-free guidance. Similar to our observation in the quantitative evaluation, our baselines tend to generate the image in a globally uniform style instead of distinct local styles for each region. In contrast, our method synthesizes the correct styles for both regions. One may suggest applying baselines with two stylization processes independently and composing the results using token maps. However, as shown in Figure 12 (Appendix), such methods generate artifacts on the region boundaries.

**Complex scene generation.** Figure 8 shows comparisons on complex scene generation. Attend-and-Excite [12] uses the tokens missing in the full-text generation result as input to fix the missing objects, like the coffee table and carpet in the living room example. However, it still fails to generate all the details correctly, e.g., the books, the painting, and the blanket. Prompt-to-Prompt [19] and InstructPix2Pix [7] can edit the painting accordingly, but many objects, like the colorful pillows and stuff on the table, are still missing. In contrast, our method faithfully synthesizes all these details described in the target region.

**Token importance control.** Figure 9 shows the qualitative comparison on token reweighting. When using a large weight for ‘mushroom,’ Prompt-to-Prompt generates clear artifacts as it modifies the attention probabilities to be unbounded and creates out-of-distribution intermediate features. Heuristic methods fail with adding more mushrooms, while our method generates more mushrooms and preserves the quality. More results of different font sizes and target tokens are shown in Figures 23 - 25 in Appendix.
Figure 11. Our workflow. (top left) A user begins with an initial plain-text prompt and wishes to refine the scene by specifying the color, details, and styles. (top center) Naively inputting the whole description in plain text does not work. (top right) InstructPix2Pix [7] fails to make accurate editing. (bottom) Our method supports precise refinement with region-constrained diffusion processes. Moreover, our framework can naturally be integrated into a rich text editor, enabling a tight, streamlined UI.

Figure 12. Ablation of injection method. We show images generated based on plain text and rich text with or without injection methods. Injecting features and noised samples help preserve the structure of the church and unformatted token regions.

Interactive editing. In Figure 11, we showcase a sample workflow to illustrate our method’s interactive strength and editing capacity over InstructPix2Pix [7].

4.3. Ablation Study

Generating token maps solely from cross-attention. The other straightforward way to create token maps is to use cross-attention maps directly. To ablate this, we first take the average of cross-attention maps across heads, layers, and time steps and then take the maximum across tokens. Finally, we apply softmax across all the spans to normalize the token maps. However, as shown by the example in Figure 10, since the prompt has no correspondence with the background, the token map of “shirt” also covers partial background regions. Note that simple thresholding is ineffective as some regions still have high values, e.g., the right shoulder. As a result, the target color bleeds into the background. Our methods obtain more accurate token maps and, consequently, more precise colorization.

Ablation of the injection methods. To demonstrate the effectiveness of our injection method, we compare image generation with and without it in Figure 12. In the font color example, we show that applying the injection effectively preserves the shape and details of the target church and the structure of the sunset in the background. In the footnote example, we show that the injection keeps the looking of the black door and the color of the floor.

5. Discussion and Limitations

In this paper, we have expanded the controllability of text-to-image models by incorporating rich-text attributes as the input. We have demonstrated the potential for generating images with local styles, precise colors, different token importance, and complex descriptions. Nevertheless, numerous formatting options remain unexplored, such as bold/italic, hyperlinks, spacing, and bullets/numbering. Also, there are multiple ways to use the same formatting options. For example, one can use font style to characterize the shape of the objects. We hope this paper encourages further exploration of integrating accessible daily interfaces into text-based generation tasks, even beyond images.

Limitations. As we use multiple diffusion processes and two-stage methods, our method can be multiple times slower than the original process. Also, our way to produce token maps relies on a thresholding parameter. More advanced segmentation methods like SAM [31] could be exploited to further improve the accuracy and robustness.

Acknowledgment. We thank Mia Tang, Aaron Hertzmann, Nupur Kumari, Gaurav Parmar, Ruihan Gao, and Aniruddha Mahapatra for their helpful discussion and paper reading. This work is partly supported by NSF grant No. IIS-239076, as well as NSF grants No. IIS-1910132 and IIS-2213335.
References


[41] Yue Ma, Yingqing He, Xiaodong Cun, Xintao Wang, Ying Shan, Xia Li, and Qifeng Chen. Follow your pose: Pose-guided text-to-video generation using pose-free videos, 2023.


Expressive Text-to-Image Generation with Rich Text Appendix

In this appendix, we provide additional experimental results and details. In section A, we show the images generated by our model, Attend-and-Excite [12], Prompt-to-Prompt [19], and InstructPix2Pix [7] with various RGB colors, local styles, and detailed descriptions via footnotes. In section B, we provide additional details on the implementation and evaluation.

A. Additional Results

In this section, we first show additional results of rich-text-to-image generation on complex scene synthesis (Figures 15, 16, and 17), precise color rendering (Figures 18, 19, and 20), local style control (Figures 21 and 22), and explicit token re-weighting (Figure 23, 24, and 25). We also show an ablation study of the averaging and maximizing operations across tokens to obtain token maps in Figure 26. We present additional results compared with a composition-based baseline in Figure 27. Last, we show an ablation of the hyperparameters of our baseline method InstructPix2Pix [7] on the local style generation application in Figure 28.

A car¹ driving on the road. A bicycle² nearby a tree³. A cityscape⁴ in the background.

¹A sleek sports car gleams on the road in the sunlight, with its aerodynamic curves and polished finish catching the light. ²A bicycle with rusted frame and worn tires. ³A dead tree with a few red apples on it. ⁴A bustling Hongkong cityscape with towering skyscrapers.

Figure 13. Additional results of the footnote. We show the generation from a complex description of a garden. Note that all the methods except for ours fail to generate accurate details of the mansion and fountain as described.
A lush garden\(^1\) with a fountain\(^2\). A grand mansion\(^3\) in the background.

\(^1\)A garden is full of vibrant colors with a variety of flowers.
\(^2\)A fountain made of white marble with multiple tiers. The tiers are intricately carved with various designs.
\(^3\)An impressive two-story mansion with a royal exterior, white columns, and tile-made roof. The mansion has numerous windows, each adorned with white curtains.

Figure 14. **Additional results of the footnote.** We show the generation from a complex description of a garden. Note that all the methods except for ours fail to generate accurate details of the mansion and fountain as described.
A small chair\(^1\) sits in front of a table\(^2\) on the wooden floor. There is a bookshelf\(^3\) nearby the window\(^4\).

\(^1\)A black leather office chair with a high backrest and adjustable arms.
\(^2\)A large wooden desk with a stack of books on top of it.
\(^3\)A bookshelf filled with colorful books and binders.
\(^4\)A window overlooks a stunning natural landscape of snow mountains.

---

Figure 15. **Additional results of the footnote.** We show the generation from a complex description of an office. Note that all the methods except ours fail to generate accurate window overlooks and colorful binders as described.
Figure 16. Additional results of the font color. We show the generation of different objects with colors from the Common category. Prompt-to-Prompt has a large failure rate of respecting the given color name, while InstructPix2Pix tends to color the background and irrelevant objects.
Figure 17. Additional results of the font color. We show the generation of different objects with colors from the HTML category. Both methods fail to generate the precise color, and InstructPix2Pix tends to color the background and irrelevant objects.
Figure 18. **Additional results of the font color.** We show the generation of different objects with colors from the RGB category. Both baseline methods cannot interpret the RGB values correctly.
Figure 19. Additional results of the font style. We show images generated with different style combinations and prompt “a beautiful garden in front of a snow mountain”. Each row contains “snow mountain” in 7 styles, and each column contains “garden” in 7 styles. Only our method can generate distinct styles for both objects.
Figure 20. **Additional results of the font style.** We show images generated with different style combinations and prompt “a small pond surrounded by skyscraper”. Each row contains “skyscraper” in 7 styles, and each column contains “pond” in 7 styles. Only our method can generate distinct styles for both objects.
Repeating: A pizza with pineapples, pepperonis, and mushrooms, mushrooms, mushrooms.
Parenthesis: A pizza with pineapples, pepperonis, and ((mushrooms)).
Ours: A pizza with pineapples, pepperonis, and mushrooms.

Figure 21. **Additional results of font sizes.** We use a token weight evenly sampled from 1 to 20 for the word ‘mushrooms’ with our method and Prompt-to-Prompt. For parenthesis and repeating, we show results by repeating the word ‘mushrooms’ and adding parentheses to the word ‘mushrooms’ for 1 to 10 times. Prompt-to-Prompt suffers from generating artifacts. Heuristic methods are not effective.

Repeating: A pizza with pineapples, pepperonis, and mushrooms, pineapples, pineapples, pepperonis, and mushrooms.
Parenthesis: A pizza with pineapples, pepperonis, and ((pineapples)).
Ours: A pizza with pineapples, pepperonis, and mushrooms.

Figure 22. **Additional results of font sizes.** We use a token weight evenly sampled from 1 to 20 for the word ‘pineapples’ with our method and Prompt-to-Prompt. For parenthesis and repeating, we show results by repeating the word ‘pineapples’ and adding parentheses to the word ‘pineapples’ for 1 to 10 times. Prompt-to-Prompt suffers from generating artifacts. Heuristic methods are not effective.
Figure 23. **Additional results of font sizes.** We use a token weight evenly sampled from 1 to 20 for the word ‘pepperonis’ with our method and Prompt-to-Prompt. For parenthesis and repeating, we show results by repeating the word ‘pepperonis’ and adding parentheses to the word ‘pepperonis’ for 1 to 10 times. Prompt-to-Prompt suffers from generating artifacts. Heuristic methods are not effective.
Figure 24. **Comparison with a simple composed-based method using different random seeds.** Since the methods like Prompt-to-Prompt [19] cannot generate multiple styles on a single image, one simple idea to fix this is to apply the methods on two regions separately and compose them using the token maps. However, we show that this leads to sharp changes and artifacts at the boundary areas.
Prompt: “A camel (Cyber Punk, futuristic) in the dessert (Vincent Van Gogh).”

Figure 25. Ablation of the classifier free guidance of InstructPix2Pix. We show that InstructPix2Pix fails to generate both styles with different image and text classifier-free guidance (cfg) weights. When image-cfg is low, the desert is lost after the first editing. We use image-cfg = 1.5 and text-cfg = 7.5 in our experiment.
Ablation on the hyperparameter $\lambda$ in Equation (7). We report the trade-off of CLIP similarity and color distance achieved by sweeping the strength of color optimization $\lambda$.

![Graph showing CLIP Similarity vs. Distance to Target Color](image)

Figure 26. Ablation on the hyperparameter $\lambda$ in Equation (7). We report the trade-off of CLIP similarity and color distance achieved by sweeping the strength of color optimization $\lambda$.

---

**Ablation of the color guidance weight.** Changing the guidance strength $\lambda$ allows us to control the trade-off between fidelity and color precision. To evaluate the fidelity of the image, we compute the CLIP score between the generation and the plain text prompt. We plot the CLIP similarity vs. color distance in Figure 26 by sweeping $\lambda$ from 0 to 20. Increasing the strength always reduces the CLIP similarity as details are removed to satisfy the color objective. We find that larger $\lambda$ first reduces and then increases the distances due to the optimization divergence.

**Constrained Prompt-to-Prompt.** The original Attention Refinement proposed in Prompt-to-Prompt [19] does not apply any constraint to newly added tokens’ attention maps, which may be the reason that it fails with generating distinct styles. Therefore, we attempt to improve Prompt-to-Prompt by injecting the cross-attention maps for the newly added style tokens. For example, in Figure 27, we use the cross attention map of “garden” for the style “Claude Monet”. However, the method still produces a uniform style.

**Human Evaluation** We conduct a user study on crowdsourcing platforms. We show human annotators a pair of generated images and ask them which image more accurately expresses the reference color, artistic styles, or supplementary descriptions. To compare ours with each baseline, we show 135 font color pairs, 167 font style pairs, and 21 footnote pairs to three individuals and receive 1938 responses. As shown in the table below, our method is chosen more than 80% of the time over both baselines for producing more precise color and content given the long prompt and more than 65% of the time for rendering more accurate artistic styles. We will include a similar study at a larger scale in our revision.

<table>
<thead>
<tr>
<th>Table 1. Human evaluation results.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>Ours vs. Prompt-to-Prompt</td>
</tr>
<tr>
<td>Ours vs. InstructPix2Pix</td>
</tr>
</tbody>
</table>

---

![Images showing different styles](images)

Figure 27. Improved Prompt-to-Prompt. Further constraining the attention maps for styles does not resolve the mixed style issue.
B. Additional Details

This section details our quantitative evaluation of the font style and font color experiments.

Font style evaluation. To compute the local CLIP scores at each local region to evaluate the stylization quality, we need to create test prompts with multiple objects and styles. We use seven popular styles that people use to describe the artistic styles of the generation, as listed below. Note that for each style, to achieve the best quality, we also include complementary information like the name of famous artists in addition to the style.

styles = [
    'Claud Monet, impressionism, oil on canvas',
    'Ukiyoe',
    'Cyber Punk, futuristic',
    'Pop Art, masterpiece, Andy Warhol',
    'Vincent Van Gogh',
    'Pixel Art, 8 bits, 16 bits',
    'Abstract Cubism, Pablo Picasso'
]

We also manually create a set of prompts, where each contains a combination of two objects, for stylization, resulting in 420 prompts in total. We generally confirm that Stable Diffusion [54] can generate the correct combination, as our goal is not to evaluate the compositionality of the generation as in DrawBench [56]. The prompts and the object tokens used for our method are listed below.

candidate_prompts = [
    'A garden with a mountain in the distance.': ['garden', 'mountain'],
    'A fountain in front of an castle.': ['fountain', 'castle'],
    'A cat sitting on a meadow.': ['cat', 'meadow'],
    'A lighthouse among the turbulent waves in the night.': ['lighthouse', 'turbulent waves'],
    'A stream train on the mountain side.': ['stream train', 'mountain side'],
    'A cactus standing in the desert.': ['cactus', 'desert'],
    'A dog sitting on a beach.': ['dog', 'beach'],
    'A solitary rowboat tethered on a serene pond.': ['rowboat', 'pond'],
    'A house on a rocky mountain.': ['house', 'mountain'],
    'A rustic windmill on a grassy hill.': ['rustic', 'hill'],
]

Font color evaluation. To evaluate precise color generation capacity, we create a set of prompts with colored objects. We divide the potential colors into three levels according to the difficulty of text-to-image generation models to depend on. The easy-level color set contains 17 basic color names that these models generally understand. The complete set is as below.

COLORS_easy = {
    'brown': [165, 42, 42],
    'red': [255, 0, 0],
    'pink': [253, 108, 158],
    'orange': [255, 165, 0],
    'yellow': [255, 255, 0],
    'purple': [128, 0, 128],
    'green': [0, 128, 0],
    'blue': [0, 0, 255],
    'white': [255, 255, 255],
    'gray': [128, 128, 128],
    'black': [0, 0, 0],
    'crimson': [220, 20, 60],
    'maroon': [128, 0, 0],
    'cyan': [0, 255, 255],
}
The medium-level set contain color names that are selected from the HTML color names \(^2\). These colors are also standard to use for website design. However, their names are less often occurring in the image captions, making interpretation by a text-to-image model challenging. To address this issue, we also append the coarse color category when possible, e.g., “Chocolate” to “Chocolate brown”. The complete list is below.

```
COLORS_medium = {
    'Fire Brick red': [178, 34, 34],
    'Salmon red': [250, 128, 114],
    'Coral orange': [255, 127, 80],
    'Tomato orange': [255, 99, 71],
    'Peach Puff orange': [255, 218, 185],
    'Moccasin orange': [255, 228, 181],
    'Goldenrod yellow': [218, 165, 32],
    'Olive yellow': [128, 128, 0],
    'Gold yellow': [255, 215, 0],
    'Lavender purple': [230, 230, 250],
    'Indigo purple': [75, 0, 130],
    'Thistle purple': [216, 191, 216],
    'Plum purple': [221, 160, 221],
    'Violet purple': [238, 130, 238],
    'Orchid purple': [218, 112, 214],
    'Chartreuse green': [127, 255, 0],
    'Lawn green': [124, 252, 0],
    'Lime green': [50, 205, 50],
    'Forest green': [34, 139, 34],
    'Spring green': [0, 255, 127],
    'Sea green': [46, 139, 87],
    'Sky blue': [135, 206, 235],
    'Dodger blue': [30, 144, 255],
    'Steel blue': [70, 130, 180],
    'Navy blue': [0, 0, 128],
    'Slate blue': [106, 90, 205],
    'Wheat brown': [245, 222, 179],
    'Tan brown': [210, 180, 140],
    'Peru brown': [205, 133, 63],
    'Chocolate brown': [210, 105, 30],
    'Sienna brown': [160, 82, 4],
    'Floral White': [255, 250, 240],
    'Honeydew White': [240, 255, 240],
}
```

The hard-level set contains 50 randomly sampled RGB triplets as we aim to generate objects with arbitrary colors indicated in rich texts. For example, the color can be selected by an RGB slider.

```
COLORS_hard = {
    'color of RGB values [68, 17, 237]': [68, 17, 237],
    'color of RGB values [173, 99, 227]': [173, 99, 227],
    'color of RGB values [48, 131, 172]': [48, 131, 172],

\(^2\)https://simple.wikipedia.org/wiki/Web_color
To write a complete prompt, we create a list of 12 objects and simple prompts containing them as below. The objects would naturally exhibit different colors in practice, such as “flower”, “gem”, and “house”.

candidate_prompts = {
    'a man wearing a shirt': 'shirt',
    'a man holding a flower': 'flower',
    'a man wearing a gem': 'gem',
    'a man wearing a house': 'house',
    ...
}
Baseline. We compare our method quantitatively with two strong baselines, Prompt-to-Prompt [19] and InstructPix2Pix [7]. The prompt refinement application of Prompt-to-Prompt allows adding new tokens to the prompt. We use plain text as the base prompt and add color or style to create the modified prompt. InstructPix2Pix [7] allows using instructions to edit the image. We use the image generated by the plain text as the input image and create the instructions using templates “turn the [object] into the style of [style],” or “make the color of [object] to be [color].” For the stylization experiment, we apply two instructions in both parallel (InstructPix2Pix-para) and sequence (InstructPix2Pix-seq). We tune both methods on a separate set of manually created prompts to find the best hyperparameters. In contrast, it is worth noting that our method does not require hyperparameter tuning.

Running time. The inference time of our models depends on the number of attributes added to the rich text since we implement each attribute with an independent diffusion process. In practice, we always use a batch size of 1 to make the code compatible with low-resource devices. In our experiments on an NVIDIA RTX A6000 GPU, each sampling based on the plain text takes around 5.06 seconds, while sampling an image with two styles takes around 8.07 seconds, and sampling an image with our color optimization takes around 13.14 seconds.