Chat2SVG: Vector Graphics Generation with Large Language Models and Image Diffusion Models

Supplementary Material

A. Overview

This supplementary material provides additional implementation details and experimental results, including:

- More details about the optimization procedure (Section B);
- Additional ablation studies evaluating the effects of prompt design strategies, SAM-guided path additions, and various optimization stages (Section C);
- Complete LLM prompts used in SVG generation and editing (Section D);
- More generated and edited SVG examples (Section E) are available on the project page.

B. Optimization Details

Our SVG optimization consists of three sequential stages: latent inversion, latent optimization, and point optimization. Each stage runs for 500 iterations. During the latent optimization stage, we initialize the stroke color to black with a width of 0.8. We also set the gradient of the RGBA color's alpha channel to zero to forbid transparency optimization. Upon completing latent optimization, we apply transformation matrices to the points, which allows us to bypass the need for optimizing transformation matrices in the subsequent point optimization stage. Optimizing an SVG containing approximately 30 shapes requires around 10 minutes when executed on a single NVIDIA RTX 4090 GPU, consuming roughly 5GB of GPU memory.

C. Additional Ablation Study

SVG Template. We demonstrate the critical role of each component in SVG template generation, with results shown in Figure 8. (1) The prompt expansion phase is crucial for synthesizing complete SVG templates by enriching the initial brief description with essential details and components. Without it, key visual elements are missing, such as the dog's chef hat and the police car's flashing lights, leading to incomplete and semantically deficient outputs. This demonstrates how prompt expansion helps capture essential visual elements needed for a coherent design. (2) Without visual rectification, the generated SVG contains significant visual inconsistencies that impact both local and global coherence. At the local level, we observe misalignments between connected components (like the dog's ear and head), while at the global level, certain elements become unrecognizable (such as the police car's body) due to improper spatial relationships and proportions. These issues highlight how visual rectification serves as a crucial quality control step in ensuring the generated SVG maintains proper visual structure and semantic clarity.

Detail Enhancement. The SAM-guided path addition enhances the visual richness of the optimized SVG by incorporating decorative elements and fine details. Without this step, the generated SVG appears simpler. As shown in the fourth column of Figure 8, compared to the complete method in the first column, decorative elements such as the intricate patterns on the dog's chef hat and the detailed windows of the police car are absent, resulting in reduced visual sophistication and artistic appeal.

SVG Optimization. Our experiments validate the importance of each optimization stage: (1) Without latent optimization, relying solely on point optimization produces shapes with undesirable artifacts like self-intersections in the dog's hat and distorted corners in the car's window; (2) Conversely, omitting point optimization yields excessively smooth contours in elements like the hat and window, resulting in a lack of geometric precision. We further analyze the impact of individual loss terms. The curvature loss plays a vital role in maintaining smooth and natural path contours - when excluded, the optimization generates shapes with noticeably irregular boundaries. Similarly, the path IoU loss is crucial for spatial consistency, anchoring elements in their designated positions. Without this constraint, components like the dog's ear and window frame drift from their intended locations, compromising the overall compositional integrity.

D. LLM Prompts

In this section, we present the detailed prompts used in our Chat2SVG. First, we introduce the system prompt (Table 3), which provides global guidance to the LLM. We then list three prompts used to create an SVG template: prompt expansion (Table 4), SVG script generation (Table 5), and visual rectification (Table 6). Additionally, we include the prompt that guides the LLM to perform accurate editing based on natural language descriptions (Table 7).

E. More Results

To provide a more user-friendly visualization, we have put all generated SVG, including the text-guided generation and editing results, on the project website. Please visit the https://chat2svg.github.io/ to view more outputs.

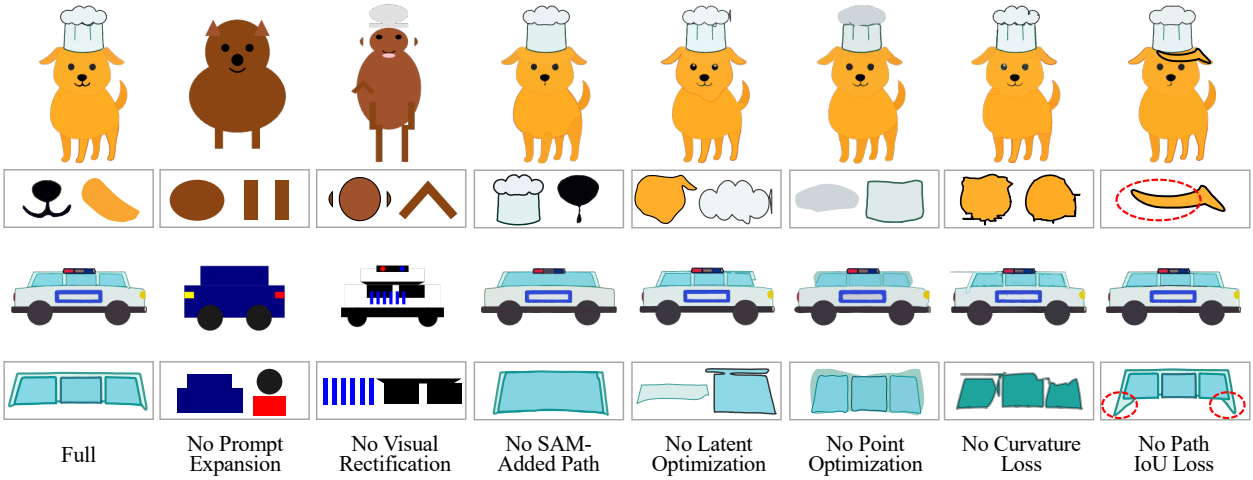

Figure 8. Qualitative results of ablation study.

System Prompt
You are a vector graphics designer tasked with creating Scalable Vector Graphics (SVG) from a given text prompt. Your tasks: • Task 1: **Expand Text Prompt**. The provided prompt is simple, concise and abstract. The first task is imagining what will appear in the image, making it more detailed. • Task 2: **Write SVG Code**. Using the expanded prompt as a guide, translate it into SVG code. • Task 3: **Code Improvement**. Although the SVG code may align with the text prompt, the rendered image could reveal oddities from human perception. Adjust the SVG code to correct these visual oddities. Constraints: 1. SVG Elements: Use only the specified elements: `rect`, `circle`, `ellipse`, `line`, `polyline`, `polygon`, and short `path` (up to 5 commands). 2. Canvas Details: The SVG canvas is defined by a 512×512 unit viewBox. Coordinates start at (0, 0) in the top-left and extend to (512, 512) at the bottom-right. 3. Element Stacking Order: The sequencing of SVG elements matters; elements defined later in the code will overlap earlier ones. 4. Colors: Use hexadecimal color values (e.g., #FF0000). For layers fully enclosed by others, differentiate with distinct colors. 5. Simplicity: Keep the SVG code simple and clear. 6. Realism: While using simple shapes, strive to create recognizable and proportionate representations of objects. Note: The SVG you create will serve as an initial draft using simple shapes rather than a fully polished final product with complex paths. Focus on creating a recognizable representation of the prompt using basic geometric forms.

Table 3. System Prompt for SVG Template Generation

Prompt Expansion

Expand the given text prompt to detail the abstract concept. Follow these steps in your response:

1. **Expand the Short Text Prompt**. Start by expanding the short text prompt into a more detailed description of the scene. Focus on talking about what objects appear in the scene, rather than merely talking the abstract idea. For example, for the short prompt "A spaceship flying in the sky", expand it to "A silver spaceship soars through the galaxy, with distant planets and shimmering stars forming a vivid backdrop".
2. **Object Breakdown and Component Analysis**.
   (a) For every object in the expanded prompt, add more details to describe the object. You can include color, size, shape, motion, status, or any other relevant details. For example, "A silver spaceship" can be expanded into "A silver spaceship with two large wings, ejecting flames from its thrusters".
   (b) Then, break down each object into its individual components. For instance, the spaceship's components could be "a body (rectangle), two triangular wings (polygon), a window (circle), and flames (polyline) emitting from the rear thrusters (rectangle)". You need to list **ALL** parts of each object. If you ignore any part, the system will assume it's not present in the scene. When listing components, explain how each component can be depicted using the specified SVG elements.
3. **Scene Layout and Composition**. Propose a logical and visually appealing layout for the final scene. For each object and each component, describe their positions on the canvas, relative/absolute sizes, colors, and relative spatial arrangement (e.g., the hand is connected to the arm, the moon is behind the mountain).

When expanding the prompt, follow these guidelines:

1. When expanding the short text prompt, avoid adding excessive new objects to the scene. Introduce additional objects only if they enhance the completeness of the scene. If the prompt mentions just a single object, you can choose to not introduce new objects and focus instead on enriching the description with more details about that object.
2. When add details to describe objects, the description can be detailed and vivid, but the language should be clear and concise. Avoid overly complex or ambiguous descriptions.
3. When breaking down objects into individual components, ensure you list all essential parts typically comprising that object, even if they are not explicitly mentioned in the initial object description.

A Unicorn Example:
#####
Input Text Prompt: "A unicorn is eating a carrot."

Expanded Prompt:

1. Scene Description: "The pink unicorn is standing in side view. The unicorn's mouth is open, bitting an orange carrot."
2. Object Detail:
   - Unicorn: The unicorn, standing in side view, has a pink body and a yellow horn. Its brown tail swings, while its four legs stand still. It has a pink head and a small black eye.
   - Carrot: The carrot is orange with two green leaves at the top.
3. Component Breakdown:
   - Unicorn: horizontal pink body (ellipse), pink head (ellipse), thin neck (rectangle), four vertical legs (rectangles), brown curved tail (polyline), yello triangle horn atop the head (polygon), round eye (circle), mouth on the head (path)
   - Carrot: elongated orange triangle body (triangle), two small green triangular leaves (triangle)

Key Components Layout:

1. Unicorn
   (a) Body: An ellipse centered around (256, 256) with rx=90 and ry=60. The body is pink.
   (b) Head: An ellipse centered at (342, 166), 30 units wide and 25 units high, oriented to suggest the unicorn's gaze forward.
   (c) Neck: A rectangle, 50 units long and 10 units wide, positioned at (312, 168) connecting the head and the body.
   (d) Legs: Four rectangles, each 10 units wide and 80 units tall, positioned at (185, 296), (220, 311), (293, 307), and (316, 296) to suggest stability.
   (e) Tail: A polyline starting from the back of the body ellipse at (168, 258) and curving to points (122, 298) and (142, 252) to suggest a flowing tail.
   (f) Horn: A polygon with points at (331, 140), (336, 110), and (341, 140) to represent the unicorn's triangular horn. The horn is yellow.
   (g) Eye: A small circle with a radius of 5 units, placed at (352, 164) on the head.
   (h) Mouth: A small curved line positioned at (342, 178) on the head
2. Carrot
   (a) Body: An elongated triangle with points at (369, 180), (348, 200), and (356, 172). The carrot body is orange (#FFA500).
   (b) Leaves: Two small triangles positioned at the top of the carrot body, centered around (363, 174). The leaves are green (#00FF00).

#####

Refer to the Unicorn example for response guidance and formatting. Avoiding any unnecessary dialogue in your response.

Here is the text prompt: TEXT_PROMPT

Table 4. Detailed Guidance for the Prompt Expansion Stage

SVG Script Generation

Write the SVG code following the expanded prompt and layout of key components, adhering to these rules:

1. SVG Elements: Use only the specified elements: `rect`, `circle`, `ellipse`, `line`, `polyline`, `polygon`, and short `path` (up to 5 commands). Other elements like `text`, `Gradient`, `clipPath`, etc., are not allowed. If there is `path`, the final command should be `Z`. If a path only contains a single command, you need to increase the `stroke-width` to make it visible.
2. Viewbox: The viewbox should be 512 by 512.
3. Stacking Order: Elements defined later will overlap earlier ones. So if there is a background, it should be defined first.
4. Colors: Use hexadecimal color values (e.g., #FF0000). For layers fully enclosed by others, differentiate with distinct colors.
5. Comments: Include concise phrase to explain the semantic meaning of each element.
6. Shape IDs: **Every** shape element should have a unique "id" starting with "path_num".

The translation from the Unicorn's expanded prompt to SVG code is provided below:

#####

```
<svg viewBox="0 0 512 512" xmlns="http://www.w3.org/2000/svg">
    <!-- Body -->
    <ellipse id="path_1" cx="256" cy="256" rx="90" ry="60" fill="#ffc0cb"/>
    <!-- Legs -->
    <rect id="path_2" x="185" y="296" width="10" height="80"
    ↪ fill="#d3d3d3"/>
    <rect id="path_3" x="220" y="311" width="10" height="80"
    ↪ fill="#d3d3d3"/>
    <rect id="path_4" x="293" y="307" width="10" height="80"
    ↪ fill="#d3d3d3"/>
    <rect id="path_5" x="316" y="296" width="10" height="80"
    ↪ fill="#d3d3d3"/>
    <!-- Neck -->
    <rect id="path_6" x="312" y="168" width="10" height="50"
    ↪ fill="#ffc0cb"/>
    <!-- Head -->
    <ellipse id="path_7" cx="342" cy="166" rx="30" ry="25" fill="#ffc0cb"/>
    <!-- Eye -->
    <circle id="path_8" cx="352" cy="164" r="5" fill="#000000"/>
    <!-- Tail -->
    <polyline id="path_9" points="168,258 122,298 142,252" fill="none"
    ↪ stroke="#a52a2a" stroke-width="8"/>
    <!-- Horn -->
    <polygon id="path_10" points="331,140 336,110 341,140" fill="#ffff00"/>
    <!-- Unicorn mouth -->
    <path id="path_11" d="M 337 178 Q 342 183 347 178" fill="none"
    ↪ stroke="#000000" stroke-width="2"/>
    <!-- Carrot body -->
    <polygon id="path_12" points="369 180 348 200 356 172 369 180"
    ↪ fill="#ffa500"/>
    <!-- Carrot leaves -->
    <polygon id="path_13" points="363 174 364 163 373 168 363 174"
    ↪ fill="#00ff00"/>
    <polygon id="path_14" points="363 174 356 166 375 173 363 174"
    ↪ fill="#00ff00"/>
</svg>
```

#####

In your answer, avoid any unnecessary dialogue, and include the SVG code in the following format:

```
```svg
  svg_code
```
```

Table 5. Detailed Prompt for the SVG Script Generation Stage

Visual Rectification

The SVG code you provide might have a critical issue: while it adheres to the text prompt, the rendered image could reveal real-world inconsistencies or visual oddities. For example:

1. Misalignments: The unicorn's legs may appear detached from the body.
2. Hidden elements: The snowman's arms could be hidden if they blend with the body due to identical colors and overlapping elements, making them indistinguishable.
3. Unrecognizable object: The SVG code includes a tiger, but the rendered image is unrecognizable due to a disorganized arrangement of shapes.
4. Disproportionate scaling: The squirrel's tail might appear overly small compared to its body.
5. Color: If a shape is purely white and placed on a white background, it may seem invisible in the final image. If there is no background, try to avoid using white for the shape.
6. Incorrect path order: Incorrect path order can cause unintended overlaps, such as the face being completely covered by the hair or hat.

These issues may not be evident in the SVG code but become apparent in the rendered image.

The provided image is rendered from your SVG code. You need to do the following:

1. First, carefully examine the image and SVG code to detect visual problems. Please list ALL the visual problems you find. If the image is severely flawed/unrecognizable, consider rewriting the entire SVG code.
2. Second, adjust the SVG code to correct these visual oddities, ensuring the final image appears more realistic and matches the expanded prompt.
3. When adding/deleting/modifying elements, ensure the IDs are unique and continuous, starting from "path_1", "path_2", etc. For example, if you delete "path_3", rename "path_4" to "path_3" and "path_5" to "path_4" to maintain continuity.

Your task is NOT to modify the SVG code to better match the image content, but to identify visual oddities in the image and suggest adjustments to the SVG code to correct them. You are not permitted to delete any SVG elements unless rewriting is involved.

In your answer, include the SVG code in the following format:

```
```svg
  svg_code
```
```

Table 6. Detailed Prompt for the Visual Rectification Stage

Editing

Analyze the given editing prompt to understand the user's intention. Classify the editing instruction into one of (or a combination of) the following categories:
1. **Object Addition**: Adding a new object to the scene.
2. **Object Removal**: Removing an existing object from the scene.
3. **Object Modification**: Changing an existing object in the scene (e.g., color, size, position, pose, layout).

Follow these steps based on the type of editing instruction:
- **Object Addition**:
  1. Detailed Description: For each new object, add more details to describe the object. You can include color, size, shape, motion, status, or any other relevant details. For example, "A silver spaceship" can be expanded into "A silver spaceship with two large wings, ejecting flames from its thrusters".
  2. Component Breakdown: Break down each object into its individual components. For instance, the spaceship's components could be "a body (rectangle), two triangular wings (polygon), a window (circle), and flames (polyline) emitting from the rear thrusters (rectangle)". You need to list **ALL** parts of each object. If you ignore any part, the system will assume it's not present in the scene. When listing components, explain how each component can be depicted using the specified SVG elements.
  3. Global Layout: Propose a global layout for each new object, i.e., describing its spatial relationship to existing elements.
  4. Local Components Layout: Propose a local components layout, describing arrangement of its components, including their relative sizes and positions.
  5. Stacking Order: Specify the layering order of new elements, especially for overlapping objects, to ensure the correct visual effect.
- **Object Removal**:
  1. Identify the object(s) to be removed with precise descriptions.
  2. Specify any adjustments needed for remaining elements to maintain scene coherence.
- **Object Modification**:
  1. Identify the specific part(s) of the object that need to be changed.
  2. Describe modifications in detail, including exact size, colors (in hexadecimal), and positions where applicable.
  3. For complex modifications, consider treating them as a combination of object removal and addition.

Guidelines for expanding the prompt:
1. When add details to describe objects, the description can be detailed and vivid, but the language should be clear and concise. Avoid overly complex or ambiguous descriptions.
2. When breaking down objects into individual components, ensure you list all essential parts typically comprising that object, even if they are not explicitly mentioned in the initial object description.
3. For object modifications, provide exact specifications (e.g., "Increase the unicorn's horn length by 20 units").
4. Consider the overall composition and balance of the scene when adding or modifying elements.

Operation Summary:
In the original SVG code, each shape has an id attribute. After generating the edited SVG code, summarize the operations performed in the following format:
1. Element Modification: [id_1, id_2, ...] (for elements that were modified but kept the same stacking order)
2. Element Removal: [id_1, id_2, ...] (for elements that were removed)
3. Element Addition: start_path_id, [id_1, id_2, ...] (for newly added elements; start_path_id is the id of the path in the original SVG code, indicating that new elements should be inserted after the element with id start_path_id. If inserting at the beginning, set start_path_id to an empty string "".)

Note: The ids in the lists refer to the ids from the original SVG code, not the edited SVG code.

Table 7. Detailed Prompt for the Editing Stage