SVGEditBench: A Benchmark Dataset for Quantitative Assessment of LLM's SVG Editing Capabilities

Kunato Nishina Yusuke Matsui
The University of Tokyo
{nishina, matsui}@hal.t.u-tokyo.ac.jp

Abstract

Text-to-image models have shown progress in recent years. Along with this progress, generating vector graphics from text has also advanced. SVG is a popular format for vector graphics, and SVG represents a scene with XML text. Therefore, Large Language Models can directly process SVG code. Taking this into account, we focused on editing SVG with LLMs. For quantitative evaluation of LLMs' ability to edit SVG, we propose SVGEditBench. SVGEditBench is a benchmark for assessing the LLMs' ability to edit SVG code. We also show the GPT-4 and GPT-3.5 results when evaluated on the proposed benchmark. In the experiments, GPT-4 showed superior performance to GPT-3.5 both quantitatively and qualitatively. The dataset is available at https://github.com/mti-lab/SVGEditBench.

1. Introduction

Vector graphics are popular for various applications because of the features not found in raster images. Vector graphics uses primitive shape elements such as circles and squares to represent a scene. Since vector representation expresses each element in the scene individually, they are highly editable [26]. Also, one of the most prominent features of vector graphics is that the image quality will not degrade when displayed in any size. Scalable Vector Graphics (SVG) [20] is the representative vector graphics format used as a standard in web icons and fonts.

With the recent advancements in Large Language Models (LLMs), generating and editing vector graphics is now possible with LLMs. Research shows that LLMs like ChatGPT [14] and GPT4 [13] can perform various tasks. Those tasks include generating programming code and summarizing or translating text [2, 3]. Since an SVG file is not a binary but a text file (XML), LLMs can directly handle those files. Hence, we could use LLMs to process SVG. Image generation models with diffusion have advanced in recent years [1, 17, 18], but combining such models with LLMs is still challenging. SVG processing with LLMs means we do not have to use those generation models. Also, using communicative LLMs such as ChatGPT [14], editing vector graphics can be realized through text chat. Since vector graphics editing typically requires knowledge and specialized software [25], being able to use intuitive interfaces like text chat can be a great advantage.

Research on SVG generation or editing with LLMs exists [3, 4]. However, they only provide examples and do not quantitatively show how LLMs can handle the numerous SVG editing tasks.

In this paper, we built a benchmark dataset that quantitatively evaluates LLMs' SVG editing capabilities. We selected six editing tasks whose quality can be measured easily. We also created the LLM prompt and the model response for each editing task. Comparisons of the capabilities between models will be possible with this benchmark. Additionally, we conducted experiments on GPT-4 and GPT-3.5 with the proposed benchmark. We examined its validity by comparing the results with qualitative evaluations. GPT-4 outperformed GPT-3.5 in all six editing tasks. Both quantitative and qualitative experiments confirmed this tendency.

2. Related Works

2.1. Scalable Vector Graphics

An example of an SVG code and its rendered result is shown in Figure 1. SVG uses XML format that takes an `<svg>` tag as its top-level element to represent a scene. The root `<svg>` tag contains tags representing shapes or text as its child. Those tags fall into three main categories: basic shapes such as rectangles (`<rect>`) and circles (`<circle>`), curves composed of straight lines and Bézier curves (`<path>`), and text (`<text>`). Each tag has its own set of attributes (e.g., `cx`, `fill`, `d` in Figure 1) that define the position or color of the shape. Since paths can also express basic shapes, using paths is more flexible. This expressivity of paths is why most previous works in the next section learn models that only deal with paths. However,

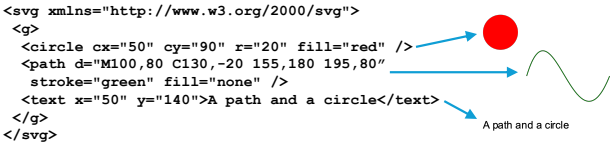

Figure 1. An example of an image represented in SVG format. Each XML element corresponds to a single shape or text block, as indicated by the blue arrows.

we cannot quickly determine the shape by looking at the path representation, especially when the shape is complex. This is because paths are represented only by combinations of the line type and the coordinates of their control points. The `<path>` elements also show no semantic information.

2.2. Recent Studies on Vector Image Processing

Regarding research related to vector graphics, especially SVG, various processing tasks and models have been proposed within the last few years. Popular tasks include:

- Vectorization: converts a raster image into a vector image
- Text-to-vector: generating a vector image conditioned by an input text
- Editing: edits the input vector image in a specific way (the focus of this paper)

Vectorization methods have advanced using the rasterization methods, especially DiffVG [10]. Im2Vec [15] uses RNN for vectorization. LIVE [11] proposed a method to progressively add the number of shapes to represent the scene. A recent method, S^2VG^2 [28], shows that a combination of Vision Transformer [7] and language models (BERT [6]) can generate human-readable SVG code.

Text-to-vector is a significant research topic in recent vector graphics processing. Its methods can be broadly classified by whether or not they use diffusion models [18]. Examples not using diffusion include IconShop [25] and StrokeNUWA [19]. IconShop generates icons using an Autoregressive Transformer [24]. StrokeNUWA learns tokens representing strokes, and an LLM uses those tokens to generate a vector image. On the other hand, research that uses diffusion models includes VectorFusion [8] and SVGDreamer [26]. They integrate DiffVG and diffusion models in a loop that optimizes the SVG parameters.

Concerning editing, Zhang et al. [27] attempt to customize vector images via a text prompt. DiffVG first renders the input image into a raster. Then, a diffusion model edits the rendered image, and SVG paths are optimized while semantically aligning with the edited raster image.

Some examples try to perform the SVG image processing tasks mentioned above with LLMs. For vectorization, StarVector [16] outputs SVG code with an LLM for code, named StarCoder [9]. Several works [3, 4] show chat-based SVG editing and generation examples. SVG manipulation using LLMs is a research area that is gaining momentum.

3. Building the Benchmark

This section provides the details of the editing task used for the benchmark. We show the method we used to select the original SVG image, the details of the six editing tasks, and the quantitative evaluation method.

3.1. Overview of the Tasks

Figure 2 shows an overview of the task used in the proposed benchmark. The prompt given to the evaluated LLM consists of the following three parts. Firstly, it explains the editing task the LLM should perform. Secondly, it provides the SVG code before the edit. Finally, it specifies the format in which LLM should respond. We regard the text between ```` ```svg ```` and ```` ``` ```` as the output image. We render the output SVG code into PNG before evaluating the editing quality numerically. For some tasks, we also use the code itself for evaluation. Refer to Section 3.3 for more details.

3.2. Selection of SVG Data

We selected Twemoji [21] as the SVG data before editing in the proposed benchmark. This decision was under the following criteria. Firstly, the data should be easily retrievable as SVG files. Secondly, the SVG images should contain both `<path>` elements and other primitive shape elements. Thirdly, the SVG file should be small, and lastly, an explanation text for each image should be available. Twemoji [21] is part of the benchmark used in SVGBench. SVGBench is a method of evaluating SVG generation models proposed with StarVector [16]. Twemoji contains 3689 pairs of SVG code and 72×72 PNG image of emojis corresponding to Unicode 14.0 [22].

We further filtered the images in Twemoji. Firstly, we removed the Regional Indicator Symbols[1] since they contain the phrase `REGIONAL INDICATOR SYMBOL LETTER` in its name. This name is unrelated to the appearance of the emoji, and we considered that this could lead to confusion by the LLM. Also, we removed ZWJ sequences[2] [23] and flags. We also removed the emojis whose names are unavailable with the `unicodedata` library in Python 3.12. The above process resulted in 1366 images. Figure 3 shows some examples of images in the benchmark and some removed images.

3.3. Evaluation Tasks and Metrics

We created the LLM prompts for performing the following six tasks. We also generated the images after correct modification (answers) using the emoji SVG data obtained above. We selected these six tasks by considering whether

[1]For instance, Regional Indicator Symbols for J and P show an emoji of the Japanese national flag.

[2]Multiple Unicode characters can be joined into a single glyph with the ZERO WIDTH JOINER (U+200D). These sequences of characters are called ZWJ sequences.

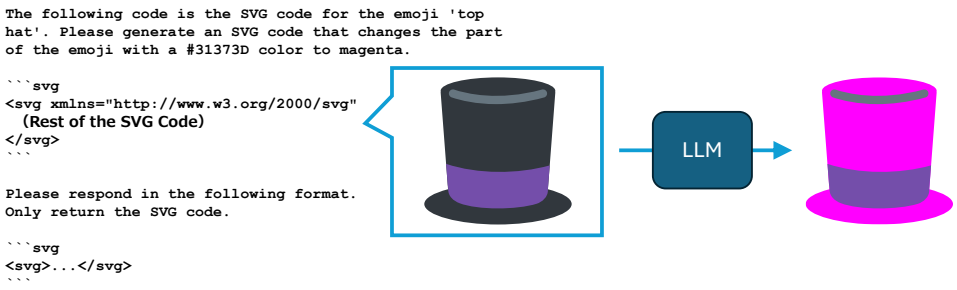

Figure 2. An overview of the tasks in the proposed benchmark and an example of the prompt in the **Change Color** task.

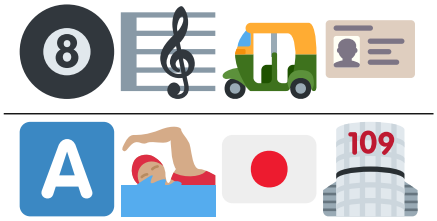

Figure 3. Sample images in the Twemoji dataset. The top row shows some images in the dataset, and the bottom row shows the ones removed.

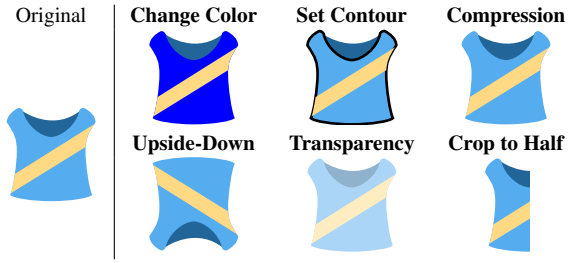

Figure 4. Examples of answers for each task used in the proposed benchmark. Note that for the **Compression** task, the rendered result should not change from the original.

we could generate the answers automatically. As shown below, most tasks here can be achieved only by changing a single attribute of the SVG code. Therefore, these tasks can test the LLMs if they know the SVG functionality. Figure 4 shows the answers for each task. We explain the structure of the prompts in more detail in the supplementary material.

Change Color We randomly selected a color from the ones specified in the `fill` attribute. The task is to change the color of the part with the picked color into another. We picked the target color randomly from red, green, blue, yellow, cyan, magenta, white, and black. Modifying the applicable `fill` attribute is adequate for this change.

Set Contour We selected a color from the image, similarly to the **Change Color** task. The task is to draw a black line around the part with the picked color. Setting the `stroke` and the `stroke-width` attributes achieves this modification.

Compression In this task, we asked the LLM to shorten the SVG code without changing the appearance. For instance, replacing the shapes bounded by four straight lines to `<rect>` or `<polygon>` may make the SVG expression shorter. The LLM has to look at the input code throughout, interpret its graphical meaning, and look for parts of the code to compress. Therefore, this task is more complex than others.

Upside-Down The task is to flip the image upside down. Adding an appropriate `transform` attribute to the root `<svg>` element can suffice the task.

Transparency The task is to make the image half transparent. Setting the `opacity` attribute of the root element to `0.5` can perform the task.

Crop to Half The task is to trim the right half of the input image and only leave the left half. This modification can be accomplished by editing the `viewBox` attribute of the `<svg>` and setting the width to half.

Except for the **Compression** task, we generated the answer by replacing or adding the attributes shown above to the SVG code in Twemoji. We converted the LLM output and the answer SVG into a 72×72 PNG using the CairoSVG library [5]. We compare the two converted images by calculating the Mean Squared Error (MSE) between the two raster images. Since the MSE does not consider the SVG code, the output is evaluated as correct if the edits by the LLMs are equivalent. For example, replacing the shapes with the ones with half the width may accomplish the **Crop to Half** task.

We set the background to white when converting to PNG. This setting ensures correct comparison with MSE for the transparent areas. Also, we standardized the pixel values to fall between 0 and 1. Since we averaged the MSE calculated for each color channel, the MSE for a single image will also be between 0 and 1.

Table 1. Results of evaluating GPT-4 and GPT-3.5 with the proposed benchmark. We also show the results when no edits are made as a reference.

Task	Metric↓	Model		
		GPT-4	GPT-3.5	No Edit
Change Color	MSE	$\mathbf{6.88 \times 10^{-5}}$	0.0134	0.0702
Set Contour	MSE	**0.0190**	0.0362	0.0286
Compression	Ratio	**94.5%**	96.1%	100%
	MSE	0.0071	**0.0023**	0
Upside-Down	MSE	**0.0463**	0.0705	0.0878
Transparency	MSE	**0.0012**	0.0122	0.0402
Crop to Half	MSE	**0.0851**	0.1068	0.1174

The answer to the **Compression** task is the SVG code in Twemoji, and the MSE was calculated similarly to the other tasks. In addition to MSE, we calculated the compression rate by comparing the length of the SVG code ((output code length) / (input code length)).

We randomly selected one hundred emojis from the 1366 emojis selected in the previous section. Prompts and answers for the six tasks were created for each emoji and used as the evaluation dataset. Therefore, a single LLM performed 600 SVG edits in total.

4. Experiments

This section presents the results of evaluating the SVG editing capabilities of GPT-4/3.5 with the proposed benchmark.

4.1. Quantitative Evaluation of GPT Models

We sent the prompts in the dataset to GPT-4 and GPT-3.5 via the OpenAI API [12]. The models used were `gpt-4-0125-preview` and `gpt-3.5-turbo-0125`. To ensure reproducibility, we set the temperature parameter to `0`. We calculated the MSE and the compression ratio using the SVG obtained from the model outputs. Then, we averaged the values over the 100 images for each task. We did not include the result in the average calculation if there were no or multiple valid SVG codes in the response. Table 1 shows the evaluation results. We also show the results without edits to the images in the table's rightmost column. If the metrics are smaller than this value, it indicates that the LLMs could perform the task.

The results reveal that GPT-4 outperformed GPT-3.5 in terms of MSE except for the **Compression** task. Even for the **Compression** task, the compression ratio is smaller with GPT-4, which means GPT-4 has higher compression capabilities. Therefore, GPT-4 shows better editing performance with the quantitative evaluation by the proposed method.

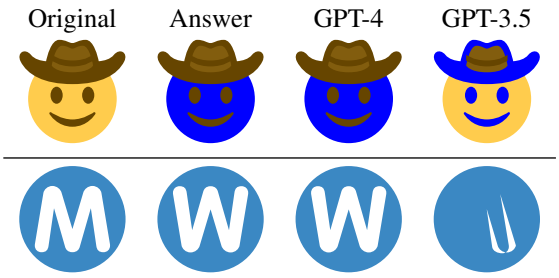

Figure 5. A qualitative evaluation of the SVG editing results generated by LLMs. The two rows show an example of the **Change Color** task and the **Upside-Down** task, respectively.

4.2. Comparison with the Qualitative Evaluation

In this section, we discuss the factors that led to the difference in performance between the two models by looking into the actual output image. We point out two differences. The first point is that GPT-4 could reflect the instructions to the output more appropriately. The upper row of Figure 5 is one example. GPT-3.5 painted the parts with a color different from the one indicated in the prompt. The second point is that GPT-3.5 often redrew the paths unnecessarily. Images in the bottom row of Figure 5 are one example from the **Upside-Down** task. As mentioned in Section 3.3, adding or editing one attribute can complete the tasks used here, except for the **Compression** task. However, GPT-3.5 rewrote the paths, which resulted in significant image corruption.

Also, for the **Compression** task, we found that GPT-4 rounded off numbers in the coordinates to shorten the representation. Although the prompts did not suggest that rounding leads to shorter code, it is interesting that GPT-4 recognized that this strategy would compress the representation while maintaining the image nearly unchanged.

The above qualitative results show that GPT-4 is superior to GPT-3.5 in SVG editing capabilities. This result is consistent with our quantitative experiments. We conclude that the metrics in the proposed benchmark reflect the LLMs' editing performance.

5. Conclusion

Considering the abilities of LLMs in handling SVG editing, we built a benchmark dataset to evaluate the SVG editing performance of LLMs quantitatively. We evaluated GPT-4 and GPT-3.5 with the proposed benchmark and showed that GPT-4 outperforms GPT-3.5 in SVG editing. This trend was also true when we compared the actual output images.

Future directions would be to improve the dataset by adding tasks, especially the ones that test semantic understanding of SVGs. Also, the benchmark can be used not only for existing models like GPT-4 and GPT-3.5 but for LLMs finetuned for SVG editing.

References

[1] James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntang Zhuang, Joyce Lee, Yufei Guo, Wesam Manassra, Prafulla Dhariwal, Casey Chu, Yunxin Jiao, and Aditya Ramesh. Improving image generation with better captions. https://cdn.openai.com/papers/dall-e-3.pdf, n.d. 1

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, pages 1877–1901, 2020. 1

[3] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023. 1, 2

[4] Mu Cai, Zeyi Huang, Yuheng Li, Haohan Wang, and Yong Jae Lee. Leveraging large language models for scalable vector graphics-driven image understanding. *arXiv preprint arXiv:2306.06094*, 2023. 1, 2

[5] CourtBouillon. CairoSVG. https://cairosvg.org/, n.d. 3

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conf. North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, 2019. 2

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021. 2

[8] Ajay Jain, Amber Xie, and Pieter Abbeel. Vectorfusion: Text-to-svg by abstracting pixel-based diffusion models. In *CVPR*, pages 1911–1920, 2023. 2

[9] Raymond Li, Loubna Ben allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia LI, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Joel Lamy-Poirier, Joao Monteiro, Nicolas Gontier, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Ben Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason T Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Urvashi Bhattacharyya, Wenhao Yu, Sasha Luccioni, Paulo Villegas, Fedor Zhdanov, Tony Lee, Nadav Timor, Jennifer Ding, Claire S Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro Von Werra, and Harm de Vries. Starcoder: may the source be with you! *Trans. Mach. Learn. Res.*, 2023. 2

[10] Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM TOG*, 39(6), 2020. 2

[11] Xu Ma, Yuqian Zhou, Xingqian Xu, Bin Sun, Valerii Filev, Nikita Orlov, Yun Fu, and Humphrey Shi. Towards layer-wise image vectorization. In *CVPR*, pages 16314–16323, 2022. 2

[12] OpenAI. https://openai.com/blog/openai-api, 2020. 4

[13] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 1

[14] OpenAI. ChatGPT. https://openai.com/chatgpt, n.d. 1

[15] Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J. Mitra. Im2vec: Synthesizing vector graphics without vector supervision. In *CVPR*, pages 7342–7351, 2021. 2

[16] Juan A Rodriguez, Shubham Agarwal, Issam H Laradji, Pau Rodriguez, David Vazquez, Christopher Pal, and Marco Pedersoli. Starvector: Generating scalable vector graphics code from images. *arXiv preprint arXiv:2312.11556*, 2023. 2

[17] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, pages 10684–10695, 2022. 1

[18] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. In *Int. Conf. Mach. Learn.*, pages 2256–2265, 2015. 1, 2

[19] Zecheng Tang, Chenfei Wu, Zekai Zhang, Mingheng Ni, Shengming Yin, Yu Liu, Zhengyuan Yang, Lijuan Wang, Zicheng Liu, Juntao Li, et al. Strokenuwa: Tokenizing strokes for vector graphic synthesis. *arXiv preprint arXiv:2401.17093*, 2024. 2

[20] The W3C SVG Working Group. Scalable Vector Graphics (SVG) 2. https://www.w3.org/TR/SVG2/, 2018. 1

[21] Twitter. https://github.com/twitter/twemoji, 2019. 2

[22] Unicode, Inc. https://unicode.org/versions/Unicode14.0.0/, 2021. 2

[23] Unicode, Inc. https://unicode.org/emoji/charts/emoji-zwj-sequences.html, n.d. 2

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 2

[25] Ronghuan Wu, Wanchao Su, Kede Ma, and Jing Liao. Iconshop: Text-guided vector icon synthesis with autoregressive transformers. *ACM TOG*, 42(6), 2023. 1, 2

[26] Ximing Xing, Haitao Zhou, Chuang Wang, Jing Zhang, Dong Xu, and Qian Yu. Svgdreamer: Text guided svg generation with diffusion model. *arXiv preprint arXiv:2312.16476*, 2023. 1, 2

[27] Peiying Zhang, Nanxuan Zhao, and Jing Liao. Text-guided vector graphics customization. In *SIGGRAPH Asia*, 2023. 2

[28] Tong Zhang, Haoyang Liu, Peiyan Zhang, Yuxuan Cheng, and Haohan Wang. Beyond pixels: Exploring human-readable svg generation for simple images with vision language models. *arXiv preprint arXiv:2311.15543*, 2023. 2