An Investigation on LLMs' Visual Understanding Ability Using SVG for Image-Text Bridging (Supplementary Materials)

Mu Cai[*1], **Zeyi Huang**[*1], **Yuheng Li**[1], **Utkarsh Ojha**[1], **Haohan Wang**[2], **Yong Jae Lee**[1]

[1]University of Wisconsin–Madison [2]University of Illinois Urbana-Champaign

1. Qualatative Chat Results

Image Recognition and Manipulation. In this section, we provide examples for chat-based image recognition and manipulation using GPT4 [8]. The qualitative results are shown in Figure 1: (a) SVG representation empowers robust in-context digit recognition capability given different background and foreground colors, (b) GPT4 can recognize and depict the details of a dog with the prompt: `"a stylized bear or a similar mammal with a round face and ears."` Furthermore, GPT-4 can identify the location of the dog's left eye and remove it. (c) GPT4 is capable of recognizing a natural image from the CIFAR-10 dataset.

Referring Segmentation The objective of the task is to label pixels in an image or video that correspond to an object instance referred by a linguistic expression. SVG representation has two advantages. First, language instruction is naturally embedded within the prompt, thus a separate design of the image segmentation model is not needed. Second, a large corpus of text and programming languages including XML are seen during pretraining, benefiting the vision-language understanding ability.

SVG is typically composed of several colored polygons, where each of them can correspond to a part of the object. Therefore, we can use the referring segmentation instructions to guide the LLM in finding the corresponding SVG code. Shown in Figure 6 (b) and (d) in the main paper, LLM can localize the object decently well. In (b), the majority of the airplane is selected as foreground, while in (d), not only is the lettuce recognized, but also the two pieces of cheese are localized and subsequently removed.

2. Robustness to Permutations

Next, to evaluate the robustness of LLMs against variations in SVG data, we conduct three distinct experiments, where we: (i) shuffle the order of paths, (ii) randomize path coordinate replacement, and (iii) randomize string replacement. Each experiment is designed to mimic real scenarios of imperfections in SVG data, providing insights into using SVG with LLMs under challenging conditions.

Path Shuffle Experiment. SVG data consists of multiple path elements, each representing an object or line in the image. In this experiment, we test the model's ability to interpret hand-drawn SVG data from the MNIST dataset when the sequence of path elements is shuffled. Note that, by the very nature of SVG data, this alteration will not change the final image; and hence, it is important to test if the LLM can remain invariant to this change, even though we have not explicitly provided it with this knowledge. The goal is to assess the model's ability to correctly interpret the digit, regardless of the order of path elements. The results in Table 1 indicate that LLMs are robust to path shuffle at test time, maintaining a comparable performance regardless of the path ordering.

	Without Shuffle	With Shuffle
Accuracy (%)	99.10	98.74

Table 1. Model performance with and without shuffled SVG path elements.

Random Path Coordinate Replacement. We further introduce a subtle form of noise by randomly altering the coordinates in the SVG path data. Each numerical value within the path commands is randomly translated within a specific range. For example, for, `... <path d="M0 0 C18 0 17..."> might become ... <path d="M1 0 C18 1 18...">`. We test several variations: (i) a minor adjustment within a 1/28th range (reflecting the 28x28 resolution of MNIST) and (ii) a more significant alteration within a 5/28th range. This simulates real-world scenarios of minor inaccuracies in SVG data, such as those resulting from conversion errors or imprecise digitization. The accuracy

[*]Equal Contribution. Order determined by random dice rolling.

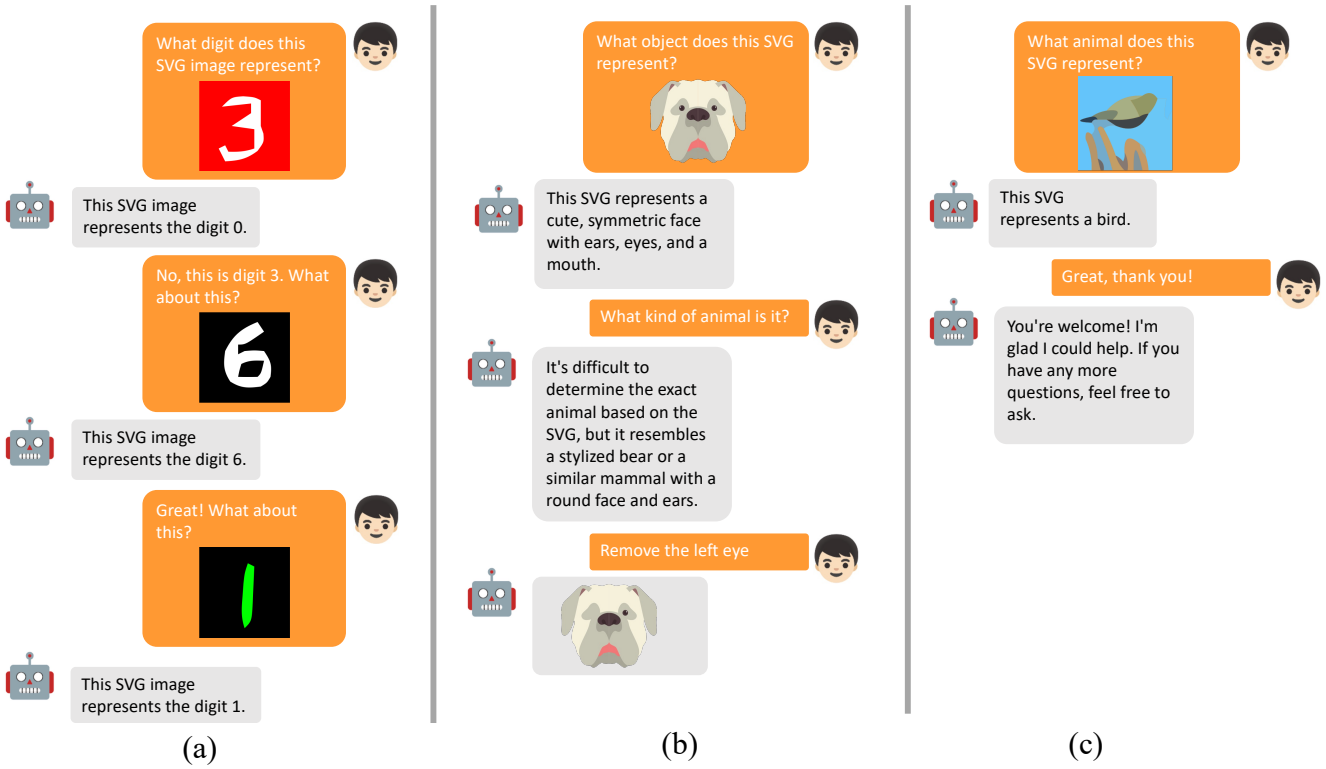

Figure 1. More qualitative results of chat-based image recognition and manipulation. (a) In-context digit recognition in Colored-MNIST-(B). (b) GPT can explain and manipulate the dog SVG image. (c) GPT4 can also recognize the bird from a CIFAR-10 example.

under different noise scales is presented in Table 2. Results indicate that LLMs are decently robust to the perturbation of the coordinate values.

	0/28	1/28	2/28	5/28
Accuracy (%)	99.10	98.97	97.91	87.56

Table 2. Model performance under different noise scales.

Random String Replacement. Finally, we design the most aggressive test of robustness, where we replace random characters in the SVG strings with any English alphabet letter, digit, or special symbol, regardless of whether they are numerical values or part of SVG commands (like `transform="translate(0,0)"`). The experiment is conducted with varying probabilities for character replacement, as shown in Table 3. Surprisingly, even after replacing 20% of a string with random characters, LLMs maintains a high accuracy rate of 90.79%. This suggests that LLMs can handle a wide range of perturbations in SVG data.

In conclusion, the results from these experiments demonstrate that despite the introduction of perturbations in the SVG data, LLMs can still perform well under challenging conditions.

	0%	1%	5%	10%	20%	50%
Acc	99.10	99.06	98.40	97.40	90.79	39.38

Table 3. Accuracy (%) with different probabilities of random string replacement.

3. Style and Content Extrapolation

In this section, we assess if LLMs can extrapolate SVG codes with more challenging transformations, such as content and style.

Style generation: We present LLMs with sample SVG letters. The first task is to figure out the style in the given examples. Then, given a new test query, the second task is to transform this given query so that it adheres to the same stylistic conventions as the example letters. The qualitative results can be found in the appendix. We observe that GPT-4 is capable of replicating styles by analyzing the correlation between given example SVG letter pairs and using this analysis to generate the corresponding test key letter.

Content generation: LLMs are shown two examples of SVG code pairs. Each pair consists of a query and key pair (both are numbers), where the query describes an SVG code of a number, and the key describes the SVG code of another

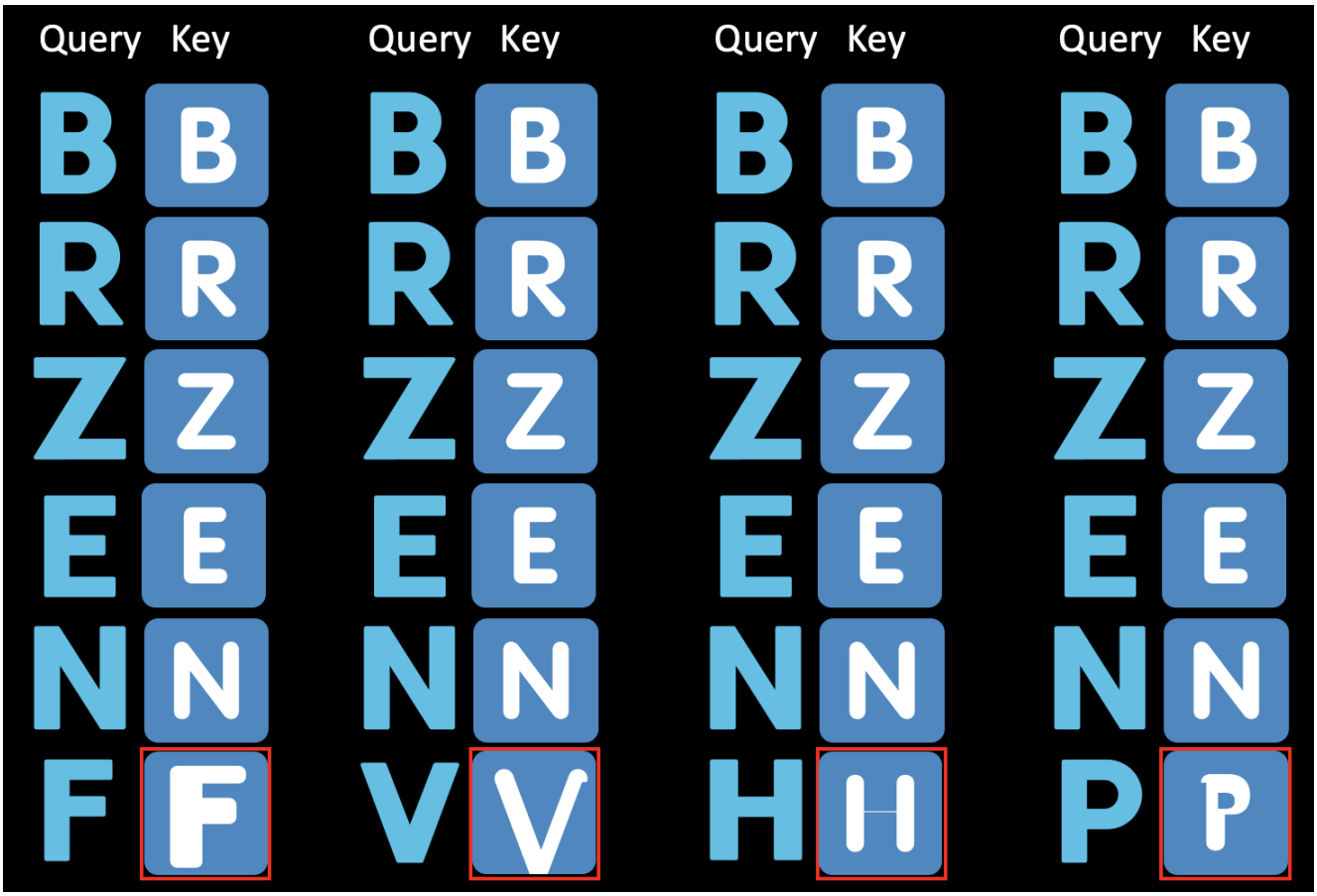

Figure 2. More qualitative results of style extrapolation. The generation results of our method are annotated with a red square.

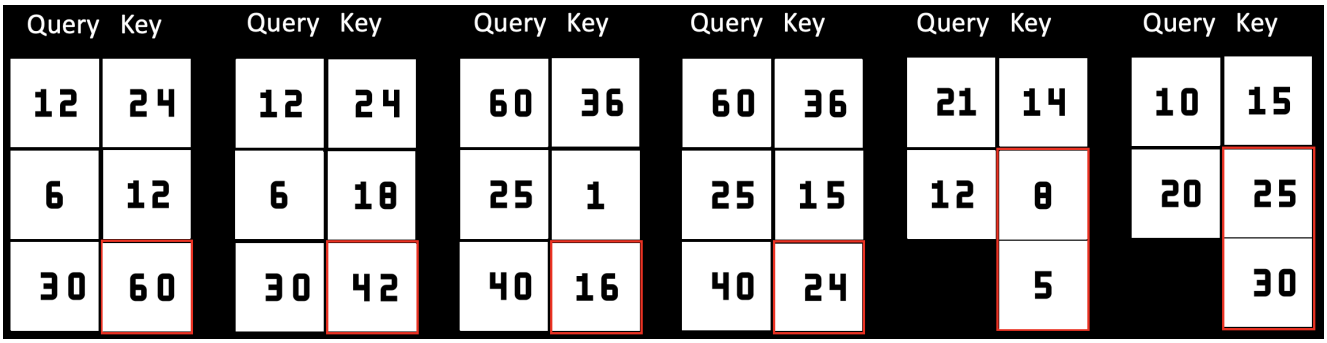

Figure 3. Understanding SVG content through the lens of GPT-4: GPT-4 demonstrates its ability to generate accurate content by analyzing the correlation between provided example number pairs, and subsequently applying this relationship to ascertain the corresponding test key number. Remarkably, in scenarios where the relationship exhibits ambiguity, GPT-4 can identify multiple possible interpretations.The generation results of our method are annotated with a red square.

number with an introduced mathematical operation. The operation can consist of add, subtract, multiply, and divide. The mathematical operation should be held in both example pairs. The first task is to figure out the mathematical operation in the two examples. Then, given a new test query SVG number, the second task is to identify what number it is and follow the mathematical operation discovered to generate the corresponding test key number. GPT-4 showcases its capability in content generation by analyzing correlations in example SVG number pairs and applying these relationships to identify corresponding test key numbers. Impressively, in cases of ambiguity, GPT-4 can discern multiple potential

interpretations. We include qualitative results in Figure 3. The prompt details can be found in the appendix.

4. Visual Reasoning Results of More Large Multimodal Models

Here we evaluate GPT-4V [8] and recent open-sourced multi-modal large language models, including LLaVA [7], InstructBLIP [2], BLIP2 [6], mPLUG_owl [11], and MiniGPT4 [12]. As the result in Table 4 shows, all current open-sourced multimodal models struggles at this fundamental reasoning task. Besides, we observe that LLaVA frequently defaults to 'yes' for yes/no queries and often resorts to random guessing for counting tasks. This behavior underscores the limitations of current large multimodal models in structured and sophisticated reasoning.

Style Extrapolation: LLMs are provided with five example pairs and are tasked with deciphering the stylistic attributes inherent in these examples. Following this, a new test query is presented to the LLMs. Their objective is to modify this query into the corresponding key, ensuring that it aligns with the same stylistic principles showcased in the example pairs. The qualitative results are shown in figure 2. The specific prompt utilized for this purpose is detailed below:

```
''Please perform the following task carefully. In this task, you will be shown five examples of Scalable Vector Graphics (SVG) code pairs. Each pair consists of a query and key pair (both are English letter), where the query describes the SVG code of the original image, and the key describes the SVG code of the transformed image. Each will be named \Example Query #" and \Example Key #" respectively. Your first task is to figure out the common transformation in the five examples. The transformation can consist of color, shape, size, style, font, and background changes, or any combination thereof. Even though you cannot see the images, and only their SVG codes, you need to discover the transformations that are happening at the image level and not just at the code level. Be detailed, and try to discover every change, and the most important change is that the paths in the SVG code between each query and key is different due to the common transformation but the shapes of the letters that query and key are representing remain the same. Then, given a new test query SVG code (named \Test Query"), your second task is to transform that query into the corresponding key SVG code (named \Test Key"), following the common transformation that you discovered in the five example pairs. To help you better understand the transformation, I will also inform you of what letter each query and key represent. You need to find the shape of each query and key by analyzing their path. Here are the five example query and key pairs: Example Query 1 (letter B):; Example Key 1 (letter B):<SVG code here>; Example Query 2 (letter R):<SVG code here>; Example Key 2 (letter R):<SVG code here>; Example Query 3 (letter Z):<SVG code here>; Example Key 3 (letter Z):<SVG code here>; Example Query 4 (letter E):<SVG code here>; Example Key 4 (letter E):<SVG code here>; Example Query 5 (letter N):<SVG code here>; Example Key 5 (letter N):<SVG code here>; Here is the test query and key pair: Test Query (letter #):; Test Key: ''
```

5. Experiment Details

5.1. Dataset

Human Designed SVG Dataset We collect a dataset from the public collection of SVG images.[1] Specifically, we collect the digits and icons to demonstrate image recognition and generation capabilities. Examples are shown in Figure 4 (a) and (b).

Convert Raster Images to SVG 1) Directly convert using curve tracing. Given the rich set of natural images in raster format, we utilize the curve tracing algorithm to convert RGB images into the SVG format.[2] Specifically, we convert MNIST [5] to SVG format using this approach, shown in Figure 4 (c).

5.2. Raster Images to SVG Conversion

One of the most fundamental pieces of information for visual perception is object shape. Our method can be conceptualized as selectively diminishing details from an image, prioritizing the extraction of less significant shapes. This guided process of reduction offers a quantitative way to

[1] https://www.svgrepo.com/, https://www.kaggle.com/datasets/victorcondino/svgicons

[2] https://github.com/visioncortex/vtracer

Question type	GPT4-brief	GPT-CoT	GPT-4V	LLaVA	CNN+MLP	Relation Networks	InstructBLIP	BLIP2	mPLUG_owl	MiniGPT4
Format	SVG	SVG	PNG	PNG	PNG	PNG	PNG	PNG	PNG	PNG
Unary	0.50	0.90	0.75	0.60	0.65	0.89	0.53	0.50	0.38	0.53
Binary	0.90	0.95	0.74	0.60	0.75	0.80	0.53	0.53	0.63	0.55
Ternary	0.10	0.88	0.28	0.10	0.55	0.55	0.10	0.30	0.30	0.30
Average	0.50	0.89	0.59	0.43	0.65	0.75	0.38	0.44	0.43	0.46

Table 4. Category-wise accuracy on the Sort-of-Clevr dataset.

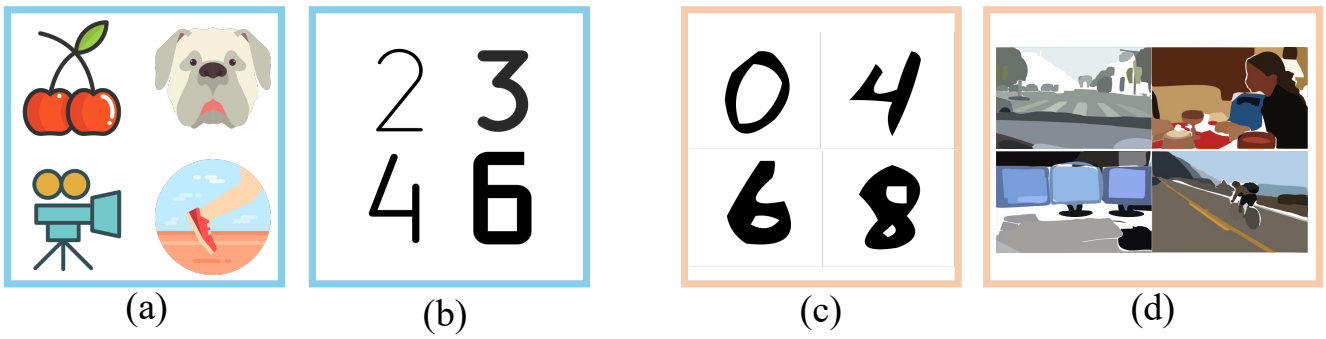

Figure 4. Visualization of our datasets. (a) and (b) are human-designed SVG vectors and icons. (c) and (d) are converted from raster images. Specifically, (c) is generated using curve tracing from MNIST [5], while (d) is generated using SAM [4] and curve tracing sequentially.

manage the amount of visual data present within an image. Within this framework, we perceive segmentation as an example of extreme simplification, whereas vectorization serves as a more moderate form of the same. Here we introduce how we use such two approaches to convert the raster images to SVG.

Image Vectorization. The vector tracing algorithm operates in a sequential three-step process. Initially, pixels are transformed into a defined path. Subsequently, this path is condensed into a simplified polygonal representation. Lastly, the polygon is refined and approximated using a curve-fitting (tracing) technique, which enhances its smoothness.

There are several online tools to convert the raster images (jpg and png) into vector graphics (SVG), such as Adobe Illustrator [1], Inkscape [3], and VTracer [10]. We experiment with all of them and found that VTracer leads to the best balance between SVG complexity (code length) and rich semantic representation.

In MNIST [5], we use the default hyperparameters during conversion. Specifically, we (i) first binarize the MNIST pixel value from the continuous range [0, 255] to the binary set $\{0, 255\}$ using the threshold 127.5, (ii) set the foreground to black, and the background to white, and (iii) apply the vector tracing algorithm VTracer.

Segmentation Prior. As mentioned earlier, segmentation can provide a strong prior for object shape. We want a generalist model that can segment any image, i.e., not trained and thus biased towards a certain dataset. The Segment Anything (SA) [4] project introduces such an image segmentation model, the Segment Anything Model (SAM), and a large-scale dataset, SA-1B, with the aim of achieving powerful generalization and zero-shot transfer across diverse segmentation tasks, demonstrating competitive results often surpassing prior fully supervised methods. We use the default hyper-parameters of SAM to segment the whole image into a set of masks without class labels, where the color of each mask is represented by the average value of the pixels within the mask. Specifically, we sample 32 query points per side (1024 points overall) to generate the image mask. Then we select the top 20 masks with the highest area as the final representation for an image.

We then use VTracer to transform the mask into SVG format. Note that, to reduce the final SVG, we adjust several settings: we set the number of significant bits to use in an RGB channel to 0; we set the minimum angle displacement degree to splice a spline to 90; we set the color difference between gradient layers to be 35; we consider a corner to have a minimum momentary angle of 0 degrees; we discard patches smaller than 16 pixels in size; and we perform iterative subdivide smoothing until all segments are shorter than 10 pixels.

5.3. Fine-tuning Dataset for Vicuna

We use the same JSON format in Vicuna [9] to construct the fine-tuning dataset. We use all the training samples in MNIST, translating to 60,000 SVG images. For each sample, we construct one round of conversation: (i) From human:

``Which digit does the following SVG reflect? <SVG code here>'', and (ii) From GPT: ``<label>''. Here <label> denotes the digit label, which ranges from 0 to 9. Then we use this dataset to fine-tune Vicuna using the default hyper-parameters [3] for 3 epochs.

5.4. Prompt Engineering

In this section, we provide the details of prompt engineering for each task. The prompt is designed to figure out the common transformation in the SVG example pairs first (each example pair consists of a query and a key) and then transform the query into the corresponding key SVG by following the discovered common transformation.

In-context Image Classification. In this task, in-context examples are aimed to provide more context information using several image-label pairs, thus facilitating the final classification. The specific prompt utilized for this purpose using 3 in-context examples is detailed below: ``Instruction: please predict the digit number for each of the following SVG images. Please think step by step, and closely look at the key identifying image characteristics. Please just tell me the image class, no other information is needed. Q: What digit does this SVG image represent? <SVG code here> A: This SVG image represents digit <label> Q: What digit does this SVG image represent? <SVG code here> A: This SVG image represents digit <label> Q: What digit does this SVG image represent? <SVG code here> A: This SVG image represents digit <label> Q: What digit does this SVG image represent? <SVG code here> A: This SVG image represents digit .

Synthetic Data Study: In this task, the objective is to conduct an analytical evaluation of the provided two example pairs, examining changes that occur in aspects such as color, shape, and size. The insight gathered from this analysis will then be used to adapt the given query into its corresponding key. The specific prompt utilized for this purpose is detailed below: ``Please perform the following task carefully. In this task, you will be shown two examples of Scalable Vector Graphics (SVG) code pairs. Each pair consists of a query and key pair, where the query describes the SVG code of the original image, and the key describes the SVG code of the transformed image. Each will be named ``Example Query #" and ``Example Key #" respectively. Your first task is to figure out the common transformation in the two examples. The transformation can consist of color, shape, size, or any combination thereof. Then, given a new test query SVG code (named \Test Query"), your second task is to transform that query into the corresponding key SVG code (named \Test Key"), following the common transformation that you discovered in the two example pairs. Here are the two example query and key pairs: Example Query 1: <SVG code here>; Example Key 1:<SVG code here>; Example Query 2:<SVG code here>; Example Key 2:<SVG code here>; Here are the test query and key pair: Test Query:<SVG code here>; Test Key:''

Content Extrapolation: In this task, LLMs are presented with SVG code pairs, each containing a query-key set that depicts numbers. The key introduces a consistent mathematical operation (addition, subtraction, multiplication, or division) to the query number. The tasks are to identify this operation in the examples and apply it to new test queries to generate corresponding test keys. To facilitate a more comprehensive understanding of SVG number codes for the LLM, we initially present the SVG codes for numbers 0 through 9 to the LLM prior to posing specific queries. The specific prompt utilized for this purpose is detailed below: ``Please perform the following task carefully. In this task, you will be shown two examples of Scalable Vector Graphics (SVG) code pairs. Each pair consists of a query and key pair, where the query describes an SVG code of an integer number, and the key describes the SVG code of another integer number with an introduced mathematical operation. Each will be named \Example Query #" and \Example Key #" respectively. In addition to the example pairs, you will be shown a new test query SVG code (named \Test Query"). Your first task is to identify which number each example query, example key, and test query represents. Your second task

[3] https://github.com/lm-sys/FastChat

is to figure out all the possible mathematical operations that are held for all given example pairs. The operation could be add, subtract, multiply, and divide (the subtract or multiply factor could be a fraction). Then, according to the numbers of example pairs and test query you identified, your third task is to predict the corresponding test key number (named \Test Key"), following all the mathematical operations that you discovered in the given example pairs. Finally, you need to generate the corresponding SVG code of the test key number. Here are the two example query and key pairs: Example Query 1: <SVG code here>; Example Key 1:<SVG code here>; Example Query 2:<SVG code here>; Example Key 2:<SVG code here>; Here are the test query and key pair: Test Query: <SVG code here>; Test Key: (Note: think about four operations one by one, and the operation should be consistent for all given example pairs)''

6. Prompt Engineering for SVG QA Dataset Curation

We use the following prompt to curate the SVG QA pairs by leveraging GPT-4V:

Generate a JSON object containing a quiz question based on an image derived from an SVG file, such as icons or emojis. The image filename is also provided to help you better curate the question, but note that this filename is not leaked to the observer. If the filename does not clearly correspond to the image, just discard that filename, never over-rely on the filename. The question should be designed to test the observer's perception to the image by making the correct answer evident only upon seeing the image. Include four answer options, ensuring that the correct answer is straightforward to identify for someone who actually view this image. The question should relate to the image's category, distinctive features, or its usuage. Provide the JSON structure with fields for the question, the four options (labeled A, B, C, D), and the correct answer indicated. Below are two examples of how to structure the question and answers within the JSON format.

{ "question": "Which category does this SVG icon best represent?", "options": { "A": "Technology", "B": "Nature", "C": "Sports", "D": "Food" }, "correct_answer": "A" }

{ "question": "The SVG icon does not use which of the following geometric shapes?", "options": { "A": "Circles", "B": "Squares", "C": "Triangles", "D": "Hexagons" }, "correct_answer": "D" }

Given this image and its filename file_path, your JSON:

7. Limitations (Extended)

Our focus was to demonstrate whether LLMs can understand images, and we used the SVG representation as a bridge to enable our studies. If one were to develop an approach out of this pipeline, then there are several limitations. One major limitation of SVG representation is the loss of fine details: Though converting raster images into SVG format and leveraging XML-based textual descriptions allows for efficient processing of crisp graphics and designs, it is not as effective in handling photographic content. As a result, fine-grained details, such as image textures, may be lost during conversion.

Conversely, when the SVG code incorporates an excessive level of detail, its sequence length can become prohibitively long, which can pose challenges for the training and inference of current Transformer-based LLMs. Developing hybrid representations that can retain the advantages of both discrete and continuous data, while preserving finer details, is a potential area for future exploration. For example, in LLMs, the processing unit is the token, which can correspond to one or several words. However, in SVG, we would prefer to have a specific embedding module for each geometric primitive in SVG, such as circles, polygons, and so on.

Furthermore, we empirically observed that LLMs can not handle low-level image manipulation tasks such as rotating the overall image by a certain angle and scaling it by a ratio. For example, we prompt GPT4 10 SVG images to conduct the following tasks: (1) enlarge the width and height by one time, (2) shrink the width and height to half, (3) rotate clock-close by 90 degrees, (4) rotate 90 degrees. Results indicate that none of the trials succeeded. Conducting such low-level image manipulation tasks needs to update the majority content of SVG code, where current LLMs are not capable of handling. Additionally, our empirical tests highlighted certain areas where LLMs fall short, particularly in

handling low-level image manipulation tasks. For instance, when prompted to manipulate SVG images in tasks like enlarging dimensions, shrinking dimensions, or rotations, LLMs like GPT-4 displayed inadequate proficiency. Such operations, which mandate considerable updates to the SVG code, currently lie outside the proficiency range of these models.

In summary, while LLMs do present limitations, it offers promising initial results for the integration of LLMs and SVG for visual tasks. Addressing these limitations could lead to more powerful image representation algorithms and pave the way for more versatile and comprehensive artificial intelligence systems.

References

[1] Adobe Inc. Adobe illustrator. https://adobe.com/products/illustrator, 2019. 5

[2] Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning. In *NeurIPS*, 2023. 4

[3] Inkscape Project. Inkscape. https://inkscape.org, 2020. 5

[4] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023. 5

[5] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010. 4, 5

[6] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. 2023. 4

[7] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *arXiv:2304.08485*, 2023. 4

[8] OpenAI. Gpt-4 technical report. 2023. 1, 4

[9] Vicuna. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. https://vicuna.lmsys.org/, 2023. 5

[10] VTracer. Vtracer. https://www.visioncortex.org/vtracer-docs, 2020. 5

[11] Qinghao Ye, Haiyang Xu, Guohai Xu, Jiabo Ye, Ming Yan, Yiyang Zhou, Junyang Wang, Anwen Hu, Pengcheng Shi, Yaya Shi, et al. mplug-owl: Modularization empowers large language models with multimodality. *arXiv preprint arXiv:2304.14178*, 2023. 4

[12] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*, 2023. 4