Supplementary Materials
NeuralSVG: An Implicit Representation for Text-to-Vector Generation

1. Additional Details

Training Scheme In the pretraining stage, we train the network for up to 300 steps using a constant learning rate of 0.01. For the full training process, we train for 4000 steps, employing a learning rate scheduler that features a linear warm-up from 0 to 0.018, followed by a cosine decay to a final value of 0.012. To improve training stability, we clip the gradients using a maximum norm of 0.1. Optimization takes approximately 15 minutes in total (including both the pretraining and training stages) on an A100 GPU.

For computing the SDS loss, we utilize the Stable Diffusion 2.1 model from the diffusers library [3].

In all experiments we optimize the network using color control, where the prompts are structured using the following format:

"A minimalist vector art of [object], isolated on a [color] background."

Here, [object] specifies the desired scene to be generated, and [color] represents the background color, which is either sampled during training or provided by the user at inference.

When applying our dropout technique, the indices are sampled as follows: with a probability of 0.7, all 16 shapes are rendered. Otherwise, the truncation index, between 1 and 16, is sampled from an exponential distribution with a temperature value of 3, favoring smaller indices and thus rendering fewer shapes more often.

LoRA Fine-Tuning As detailed in the main paper, our SDS loss is applied with a LoRA adapter applied to Stable Diffusion 2.1. This adapter was pretrained on a high-quality dataset of vector art images. Specifically, the adapter was trained using 1,600 images spanning 145 different prompts, with minor variations between prompts (e.g., with different background colors). These images were generated using the Simple Vector Flux LoRA (see `renderartist/simplevectorflux` from diffusers.). Examples can be seen in Figure 1.

The LoRA adapter was trained for 15,000 steps with a rank of 4.

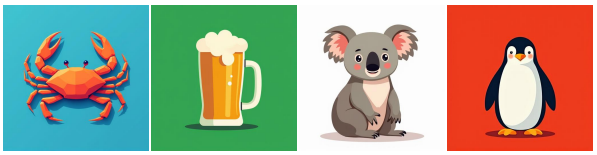

Figure 1. **LoRA Fine-Tuning data.** Example images from the dataset used for LoRA fine-tuning in our approach.

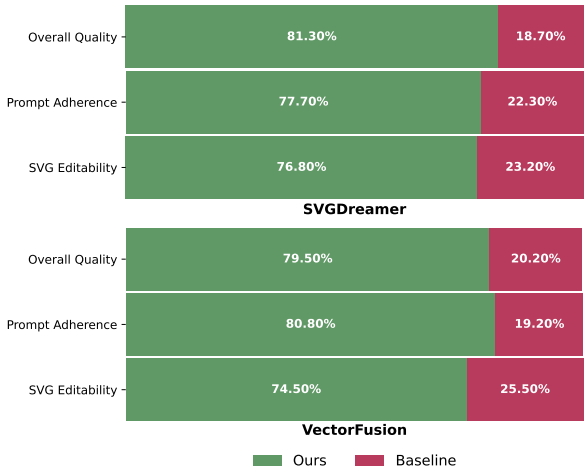

Figure 2. **User Study**. We ask respondents to evaluate the considered methods in terms of overall quality, how well the resulting SVGs match the input text prompt, and the overall editability of the SVGs. We report the percent of responses that preferred our method compared to the alternative approaches.

2. User Study

We conducted a user study to evaluate NeuralSVG based on three key aspects: (1) the overall quality of the generated SVGs, (2) how well the results aligned with the input text prompt, and (3) the editability of the SVGs. For this, we performed head-to-head comparisons between our method and the two open-source approaches, VectorFusion [1] and SVGDreamer [4], each utilizing 16 learned shapes. To assess editability, users were shown both the rendered SVGs and their corresponding outlines. For each comparison, we sampled 7 results, with a total of 210 responses collected per baseline from 30 participants.

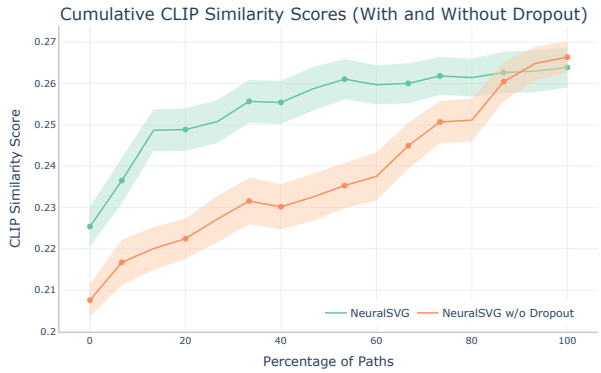

Figure 3. **Cumulative CLIP Similarities With and Without Dropout.** We show cumulative CLIP similarities achieved by NeuralSVG trained with and without dropout across 50 prompts, using 16 learnable shapes.

The complete results are presented in Figure 2, where we report the fraction of times our method was preferred over each baseline. As shown, users strongly favored our approach over existing methods in terms of overall visual quality, prompt adherence, and practical usability.

3. Ablation Studies

In the main paper, we demonstrated the advantage of training NeuralSVG with our dropout technique to achieve more recognizable outputs when rendering with a limited number of shapes. In Figure 3, we present the CLIP-space similarities between our 128 input prompts and the generated SVGs, where the shapes are selected sequentially from the learned set of 16 shapes. As shown, when using only a small number of shapes, the resulting SVGs are more easily recognized by CLIP. This suggests that the individual shapes carry more standalone meaning, aligning with our intended goal.

4. Limitations

While we have demonstrated that NeuralSVG can effectively generate high-quality, editable vector graphics, certain limitations should be considered.

First, in some cases, the network may struggle to separate shapes, sometimes mapping different indices to similar shapes. This can lead to redundancies, where multiple shapes represent the same semantic part (see Fig. 4, top). However, thanks to our ordered representation, these redundancies can be mitigated at inference time by discarding unnecessary shapes, particularly for simpler scenes requiring fewer than 16 shapes.

Second, the small number of shapes used in this work and our regularization technique can make it challenging to capture complex scenes, such as those containing multiple objects or an object within a specific background setting (see Fig. 4, bottom). Additionally, we find that incorporating our color palette and aspect ratio controls impacts model convergence, making optimization more challenging. This issue could be addressed by exploring alternative ways to introduce these controls into the optimization process, potentially incorporating additional loss objectives to help guide convergence. Finally, our method relies on an SDS-based optimization process per prompt, making it less scalable compared to feed-forward approaches. While our representation enables users to dynamically modify the learned output at inference time, we aim to explore efficient, feed-forward methods for text-to-SVG generation to further improve practical usability. We demonstrate these limitations in Figure 4.

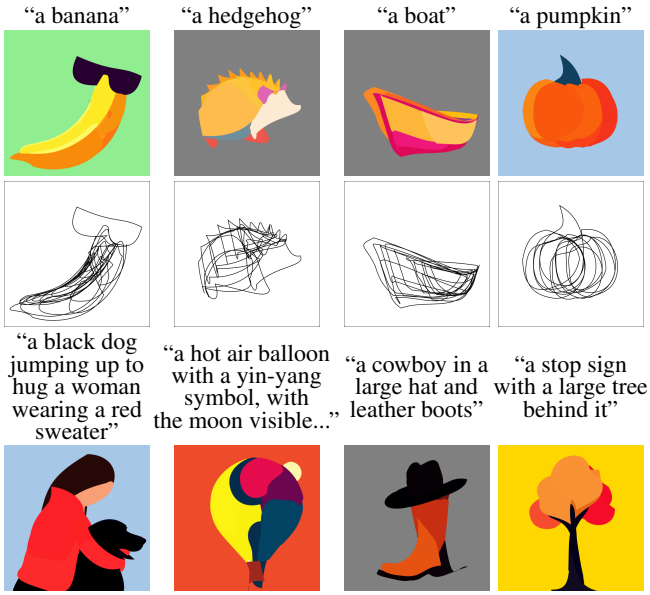

Figure 4. **Limitations.** Top: NeuralSVG may generate redundant shapes, particularly in simpler scenes that require fewer than 16 shapes. Bottom: NeuralSVG may struggle to accurately capture complex scenes containing multiple objects or actions, sometimes resulting in missing objects or misalignment with the prompt.

5. Additional Results and Comparisons

Below, we provide additional qualitative results and comparisons, as follows:

1. In Figure 5 we show additional results from optimizing NeuralSVG with aspect ratios of 1:1 and 4:1.
2. In Figures 6 and 7, we present additional qualitative results produced by NeuralSVG when applying our dropout technique during inference. Specifically, we vary the number of learned shapes included in the final rendering, showing results with 1, 4, 8, 12, and all 16 shapes.
3. In Figures 8 and 9, we provide additional qualitative comparisons to open-source text-to-vector methods VectorFusion [1] and SVGDreamer [4].
4. Following Figure 8, we provide corresponding outlines for the generated SVGs, showing that alternative methods have a tendency to produce nearly pixel-like shapes that are difficult to modify manually while NeuralSVG promotes individual shapes with more semantic meaning and order.
5. In Figure 11, we provide additional qualitative comparisons to closed-source techniques NIVeL [2] and Text-to-Vector [5] using results presented in their respective papers.
6. Next, in Figures 12 and 13, we show results obtained when rendering the learned SVG with different background colors at inference time, with both seen and unseen colors.
7. Finally, in Figure 14, we show sketch generation results obtained using our NeuralSVG framework. Sketches are rendered using a varying number of strokes by modifying the truncation index at inference time. This approach enables a single learned representation to generate sketches at multiple levels of abstraction without modifying our text-to-vector framework.

References

[1] Ajay Jain, Amber Xie, and Pieter Abbeel. Vectorfusion: Text-to-svg by abstracting pixel-based diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1911–1920, 2023. 1, 3, 7, 8

[2] Vikas Thamizharasan, Difan Liu, Matthew Fisher, Nanxuan Zhao, Evangelos Kalogerakis, and Michal Lukac. Nivel: Neural implicit vector layers for text-to-vector generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4589–4597, 2024. 3, 10

[3] Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Davaadorj, Dhruv Nair, Sayak Paul, William Berman, Yiyi Xu, Steven Liu, and Thomas Wolf. Diffusers: State-of-the-art diffusion models. https://github.com/huggingface/diffusers, 2022. 1

[4] Ximing Xing, Haitao Zhou, Chuang Wang, Jing Zhang, Dong Xu, and Qian Yu. Svgdreamer: Text guided svg generation with diffusion model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4546–4555, 2024. 1, 3, 7, 8

[5] Peiying Zhang, Nanxuan Zhao, and Jing Liao. Text-to-vector generation with neural path representation. *ACM Transactions on Graphics (TOG)*, 43(4):1–13, 2024. 3, 10

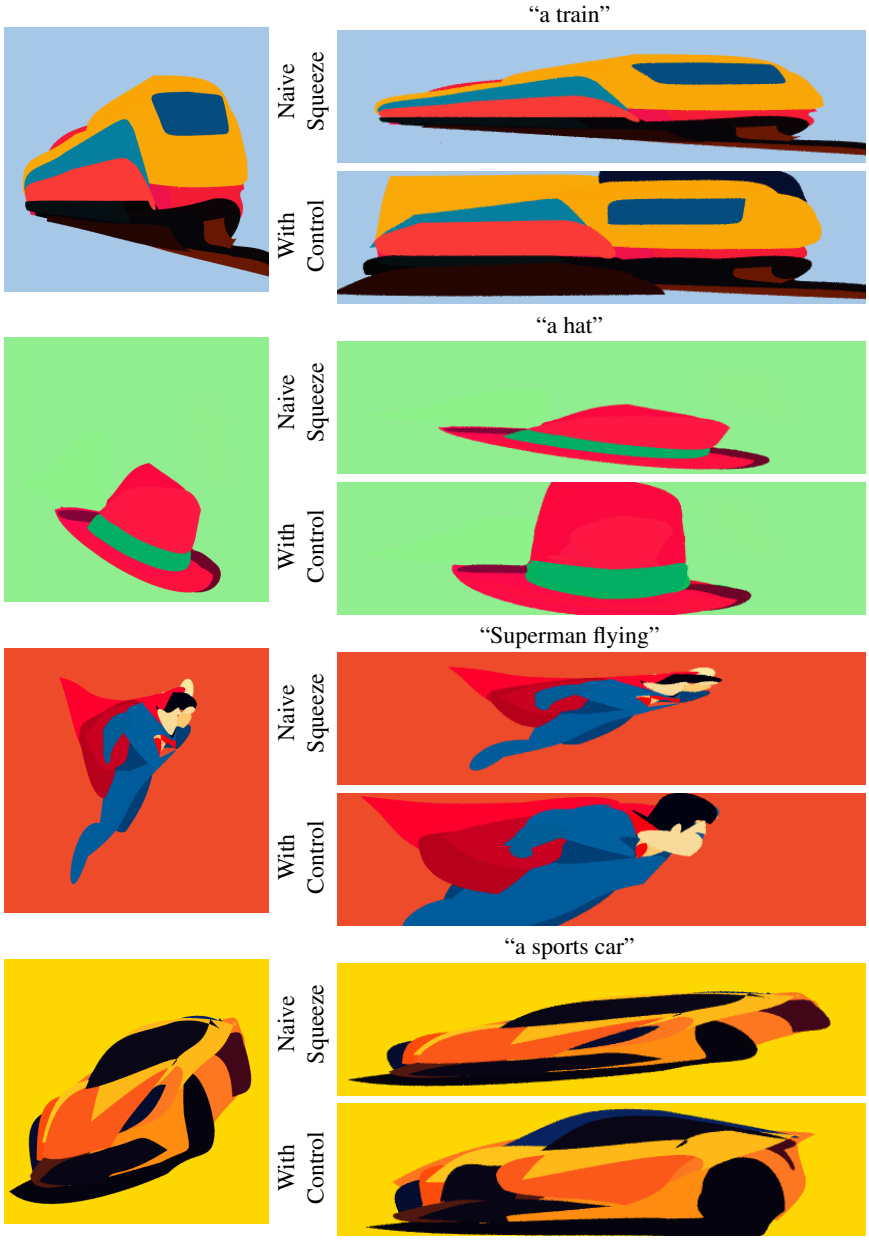

Figure 5. **Dynamically Controlling the Aspect Ratio.** Additional results from optimizing NeuralSVG with aspect ratios of 1:1 and 4:1. In each pair of results, the top row shows the naive approach of squeezing the 1:1 output into a 4:1 aspect ratio. The bottom row shows the results where our trained network directly outputs the 4:1 aspect ratio.

Figure 6. **Additional Qualitative Results Obtained with NeuralSVG.** We show results generated by our method when keeping a varying number of learned shapes in the final rendering.

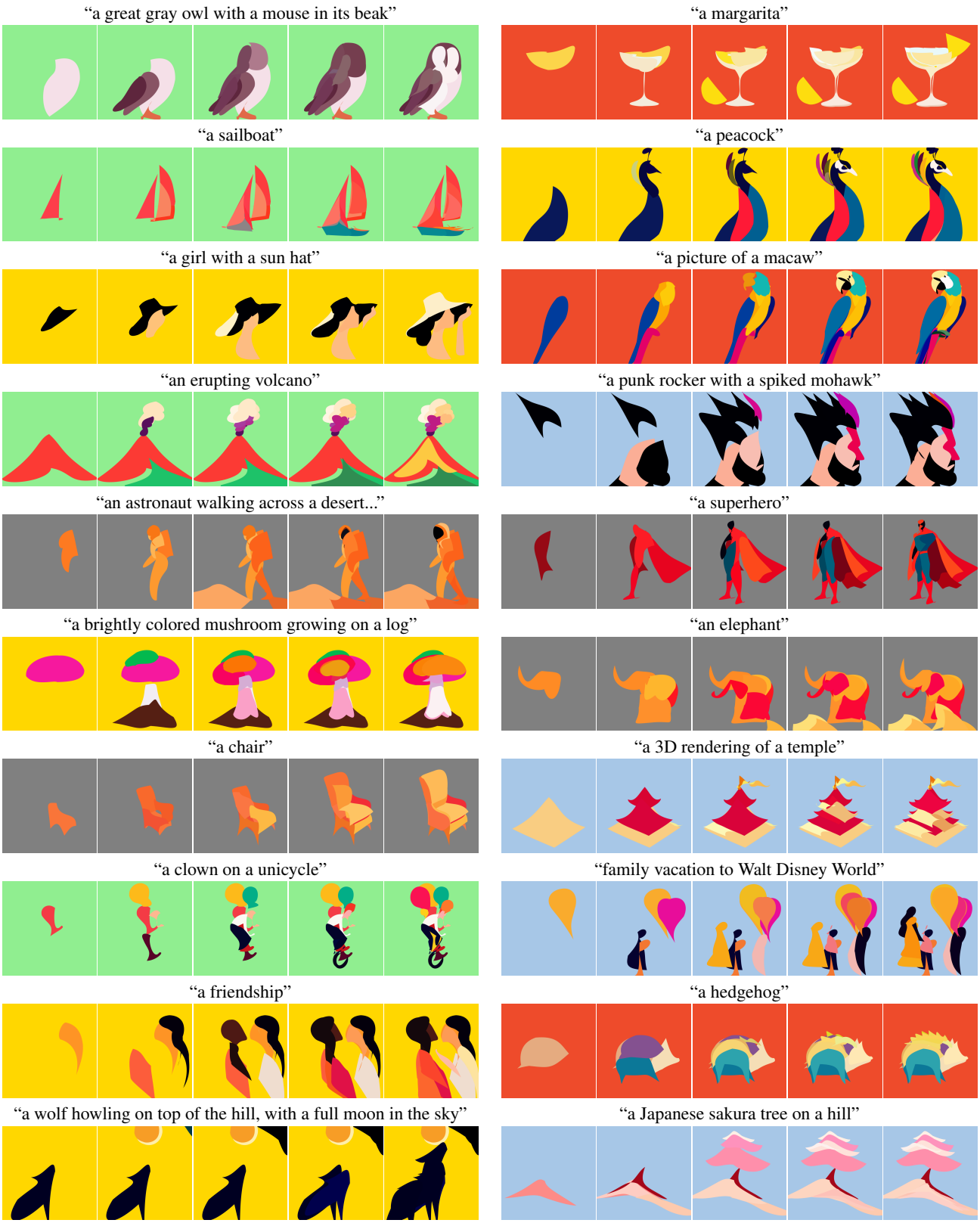

Figure 7. **Additional Qualitative Results Obtained with NeuralSVG.** We show results generated by our method when keeping a varying number of learned shapes in the final rendering.

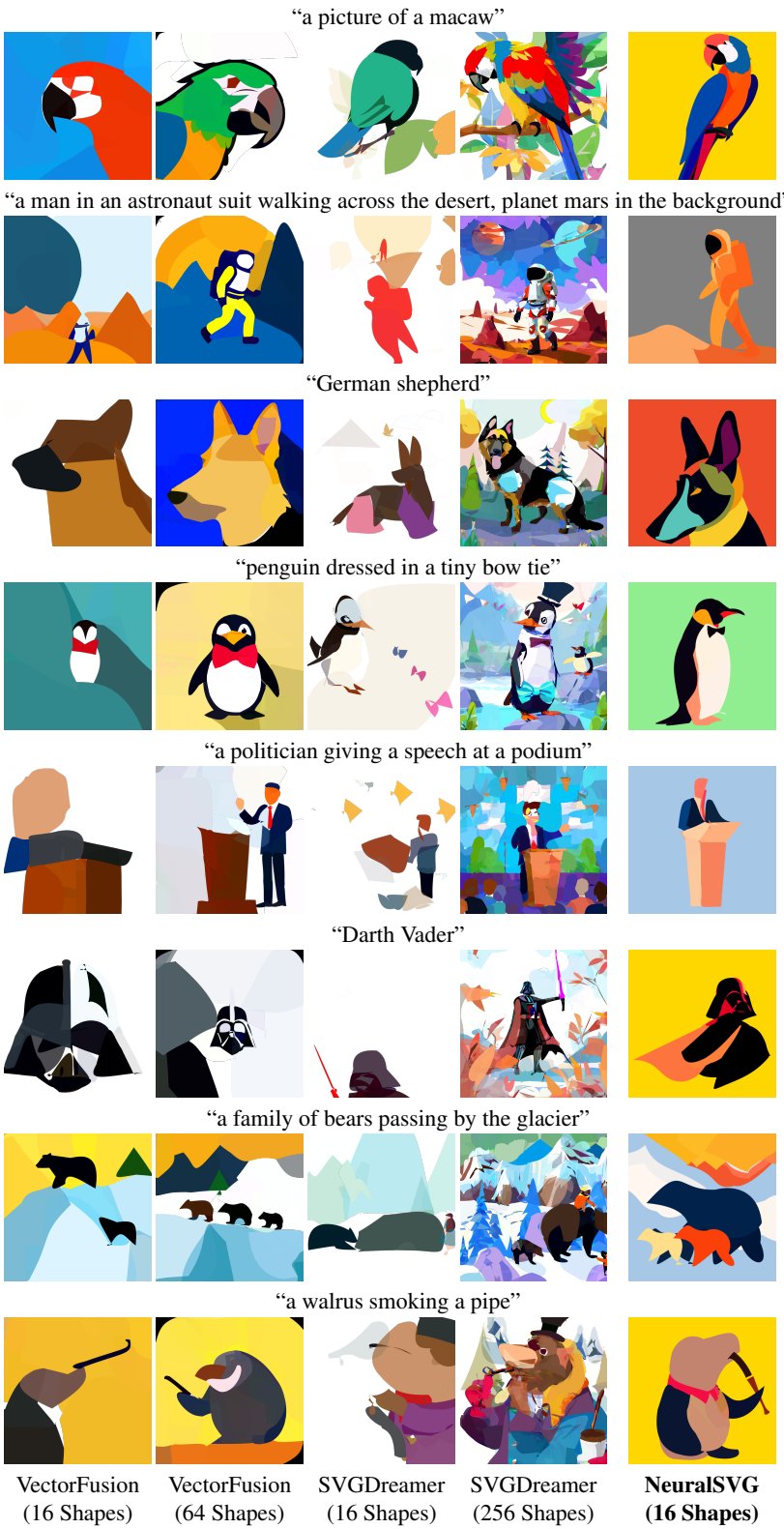

Figure 8. **Additional Qualitative Comparisons.** We provide additional visual comparisons to VectorFusion [1] and SVGDreamer [4] using a varying number of shapes.

Figure 9. **Additional Qualitative Comparisons.** We provide additional visual comparisons to VectorFusion [1] and SVGDreamer [4] using a varying number of shapes.

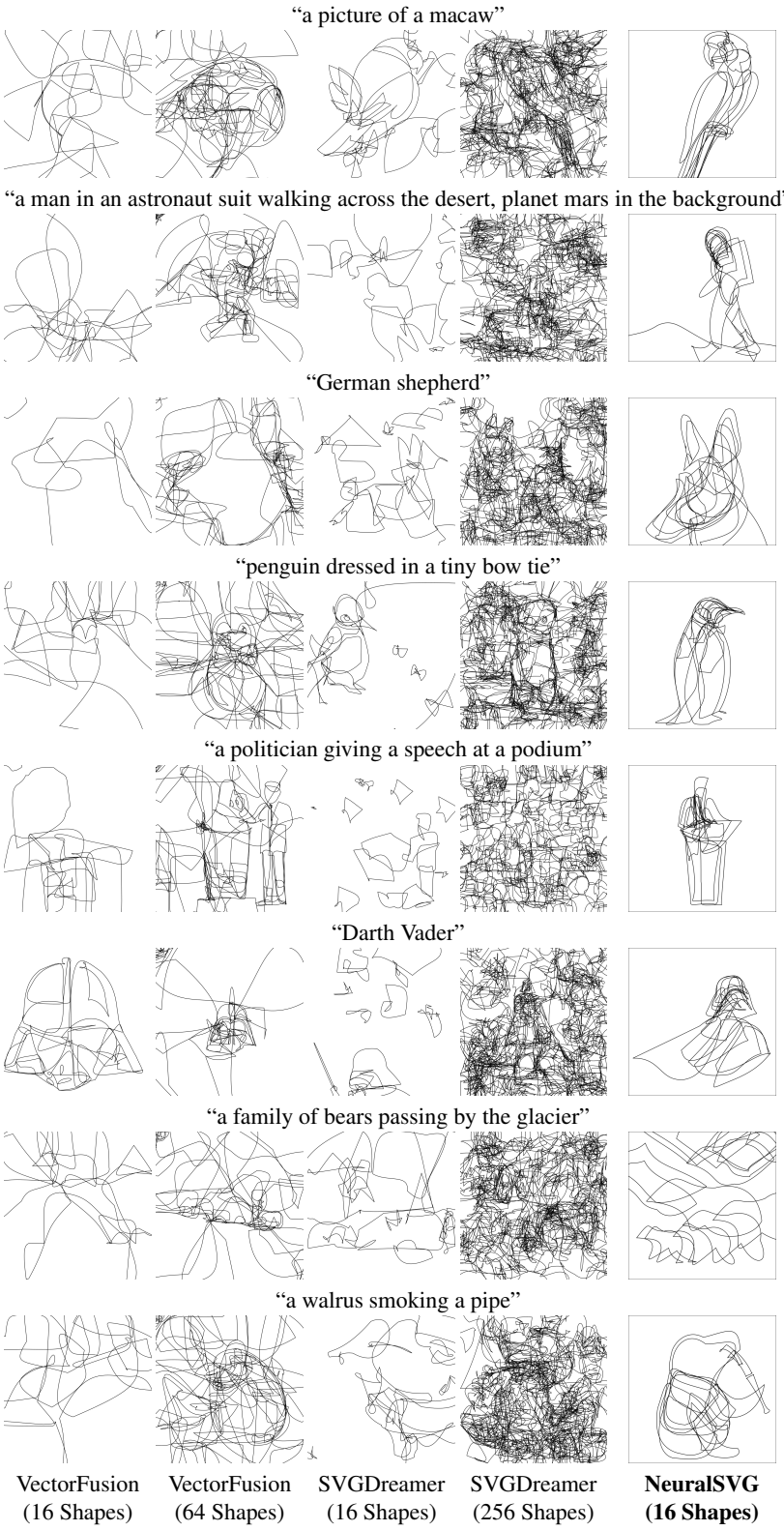

Figure 10. **Shape Outlines of the Generated SVGs.** We present the corresponding outlines of SVGs generated by NeuralSVG, VectorFusion, and SVGDreamer for the results shown in Figure 8. The alternative methods often produce nearly pixel-like shapes that are difficult to modify manually. In contrast, NeuralSVG generates cleaner SVGs, making them more editable and practical.

Figure 11. **Qualitative Comparisons.** As no code implementations are available, we provide visual comparisons to NIVeL [2] (left colunms) and Text-to-Vector [5] (right columns) using results shown in their paper.

Figure 12. **Dynamically Controlling the Color Palette.** Given a learned representation, we render the result using different background colors specified by the user, resulting in varying color palettes in the resulting SVGs. The 5 leftmost columns show colors observed during training while the 5 rightmost columns show unobserved colors.

Figure 13. **Dynamically Controlling the Color Palette.** Given a learned representation, we render the result using different background colors specified by the user, resulting in varying color palettes in the resulting SVGs. The 5 leftmost columns show colors observed during training while the 5 rightmost columns show unobserved colors.

Figure 14. **Additioanl Sketch Generation Results.** NeuralSVG can generate sketches with varying numbers of strokes using a single network, without requiring modifications to our framework.