LottieGPT: Tokenizing Vector Animation for Autoregressive Generation

Supplementary Material

Contents

1. Comparison of the Tokenizer

To demonstrate the efficiency of our proposed Lottie tokenizer, we compare it with existing tokenization approaches across different datasets. We evaluate the compression performance on both MMSVG-icon and LottieAnimation datasets using four tokenization methods: QwenVL tokenizer, OmniSVG tokenizer, our Lottie tokenizer, and Lottie tokenizer with numeric quantization.

1.1. Token Count Comparison

As shown in Tab. 1, our Lottie tokenizer achieves superior compression ratios compared to QwenVL and OmniSVG tokenizers. On the MMSVG-icon dataset, our tokenizer reduces the average token count from 2.6k (QwenVL w. SVG) to 1.3k, achieving a 50% compression ratio. Through numerical quantization, this ratio is further improved to 15.4%, with the average token count reduced to 0.4k. (Note that the OmniSVG tokenizer already incorporates numerical quantization.) For the more complex LottieAnimation dataset, this advantage becomes even more pronounced. Our tokenizer achieves a 63.3% compression ratio (17.4k tokens vs. QwenVL's 27.5k tokens), and with quantization, the compression ratio reaches 24% (6.6k tokens). The significant reduction in token count not only accelerates training and inference but also enables the model to handle longer and more complex animations within the same context window. Unlike the difficulty-based Lottie data segmentation by token count mentioned in the main text, the average token count calculation here includes numerical values to provide a fairer comparison with QwenVL and OmniSVG.

Table 1. Tokenizer comparison on MMSVG-icon and LottieAnimation datasets. Our Lottie tokenizer achieves significantly better compression ratios while maintaining generation quality.

Tokenizer	Avg. File Size			Tokens	
	PNG/ MP4	SVG/ JSON	Comp. Ratio	Avg.	Comp. Ratio
MMSVG-icon Dataset					
SVG Code	5.46 KB	2.74 KB	50.2%	N/A	N/A
Lottie Json	5.46 KB	2.16 KB	39.6%	N/A	N/A
QwenVL w. SVG	N/A	N/A	N/A	2.6 k	100%
OmniSVG	N/A	N/A	N/A	2.2 k	84.6%
QwenVL w. JSON	N/A	N/A	N/A	1.6 k	61.5%
Lottie	N/A	N/A	N/A	1.3 k	50.0%
Lottie w. Quant.	N/A	N/A	N/A	**0.4 k**	**15.4%**
LottieAnimation Dataset					
Original	194.11 KB	60.72 KB	31.3%	N/A	N/A
Optimized	194.11 KB	39.87 KB	20.5%	N/A	N/A
QwenVL	N/A	N/A	N/A	27.5 k	100%
Lottie	N/A	N/A	N/A	17.4 k	63.3%
Lottie w. Quant.	N/A	N/A	N/A	**6.6 k**	**24.0%**

1.2. Animation File Size Comparison

We find that using Lottie JSON as the representation for vector graphics and vector animations yields higher compression ratios. For example, as shown in Tab. 1, on the MMSVG-icon dataset, the average size of original SVG files is 2.74KB, which is 50.2% of the average PNG image size of 5.46KB. However, vector graphics stored in Lottie JSON format have an average size of only 2.16KB, merely 39.6% of the original PNG image size. This can also be reflected in token counts. For example, using the QwenVL default tokenizer on the MMSVG-icon dataset with SVG files as training instructions, the average token length is 2.6k (QwenVL w. SVG), while using Lottie JSON files as training instructions, the average token length is only 1.6k, which is 61.5% of the former, with no difference in rendering results between the two. On the LottieAnimation dataset, the average size of MP4 files rendered from original Lottie animations is 194.11KB, while animations saved in Lottie JSON format have an average size of 60.72KB, achieving a compression ratio of 31.3%. After our simplification process, the average Lottie JSON size is further reduced to 39.87KB, improving the compression ratio to 20.5%. This demonstrates that using Lottie JSON for representing vector graphics and vector animations achieves higher compression ratios than SVG without sacrificing rendering quality.

1.3. Support of Numeric Quantization

In the Lottie JSON simplification process, we simplify the numerical components by compressing floating-point values to four significant digits. This approach significantly reduces the size of Lottie JSON without sacrificing rendering quality. It is important to note that the current version of LottieGPT presented in the main text was trained *without* numerical quantization to ensure maximum fidelity. However, our experiments show that models trained with quantized tokens achieve comparable performance while being more efficient. Inference is performed on a single NVIDIA H20 GPU (140GB), with LottieGPT-7B achieving a generation speed of approximately 60 tokens per second.

2. Static Vector Graphics Task Results

We compared our method with state-of-the-art SVG generation models, including StarVector [6] and OmniSVG [7], conducting experiments with both text input and image input. Qualitative results are shown in Fig.1 and Fig.2. We randomly sampled examples from LottieBench. As illustrated in the figures, although LottieGPT may still fail on some complex cases, it is capable of generating static vector graphics for the vast majority of scenarios. Notably, our current version was trained on only 1/3 of the training data used by OmniSVG.

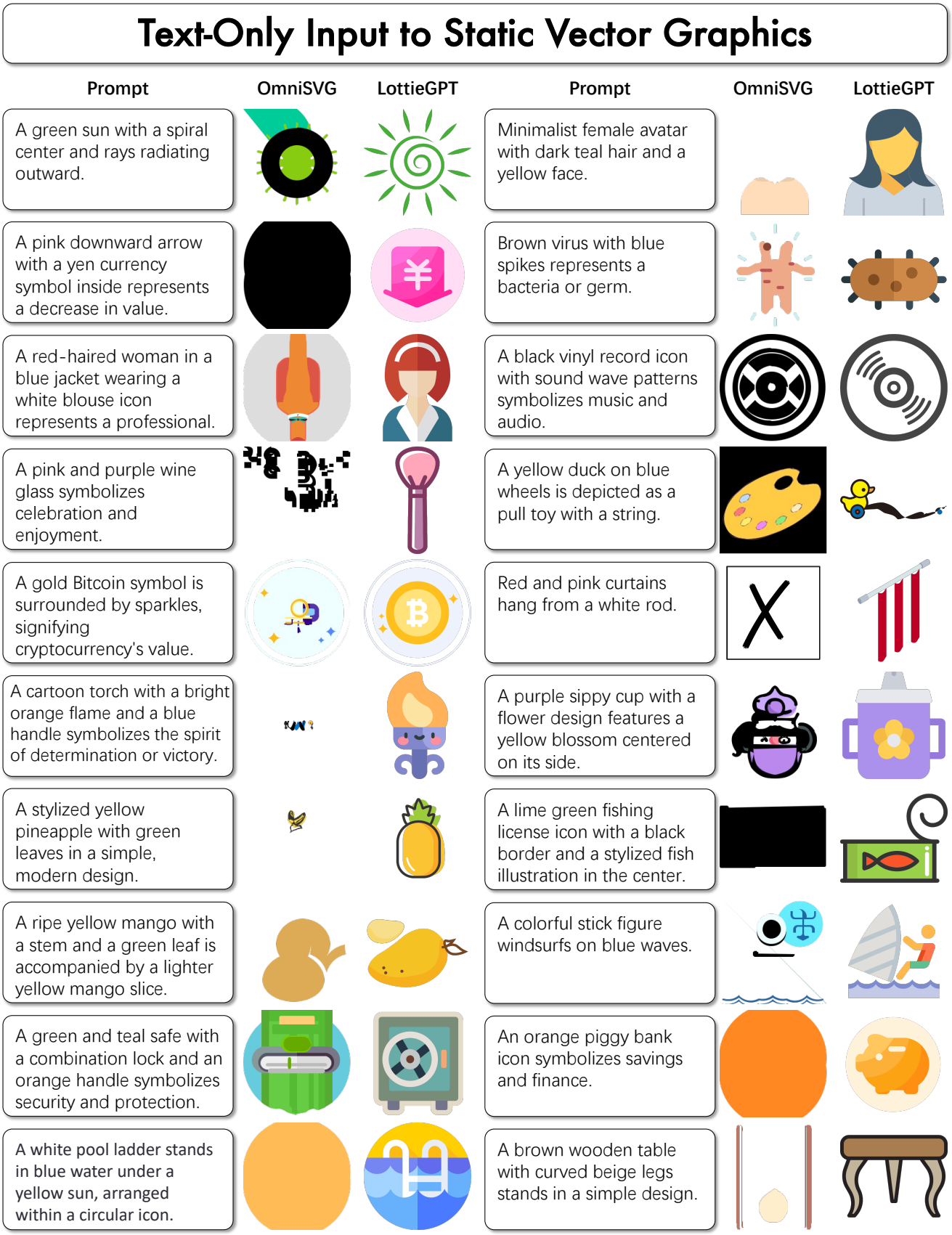


Figure 1. Performance of OmniSVG and LottieGPT on the text-to-vector graphics generation task.

Figure 2. Performance of OmniSVG, StarVector, and LottieGPT on the image-to-vector graphics generation task.

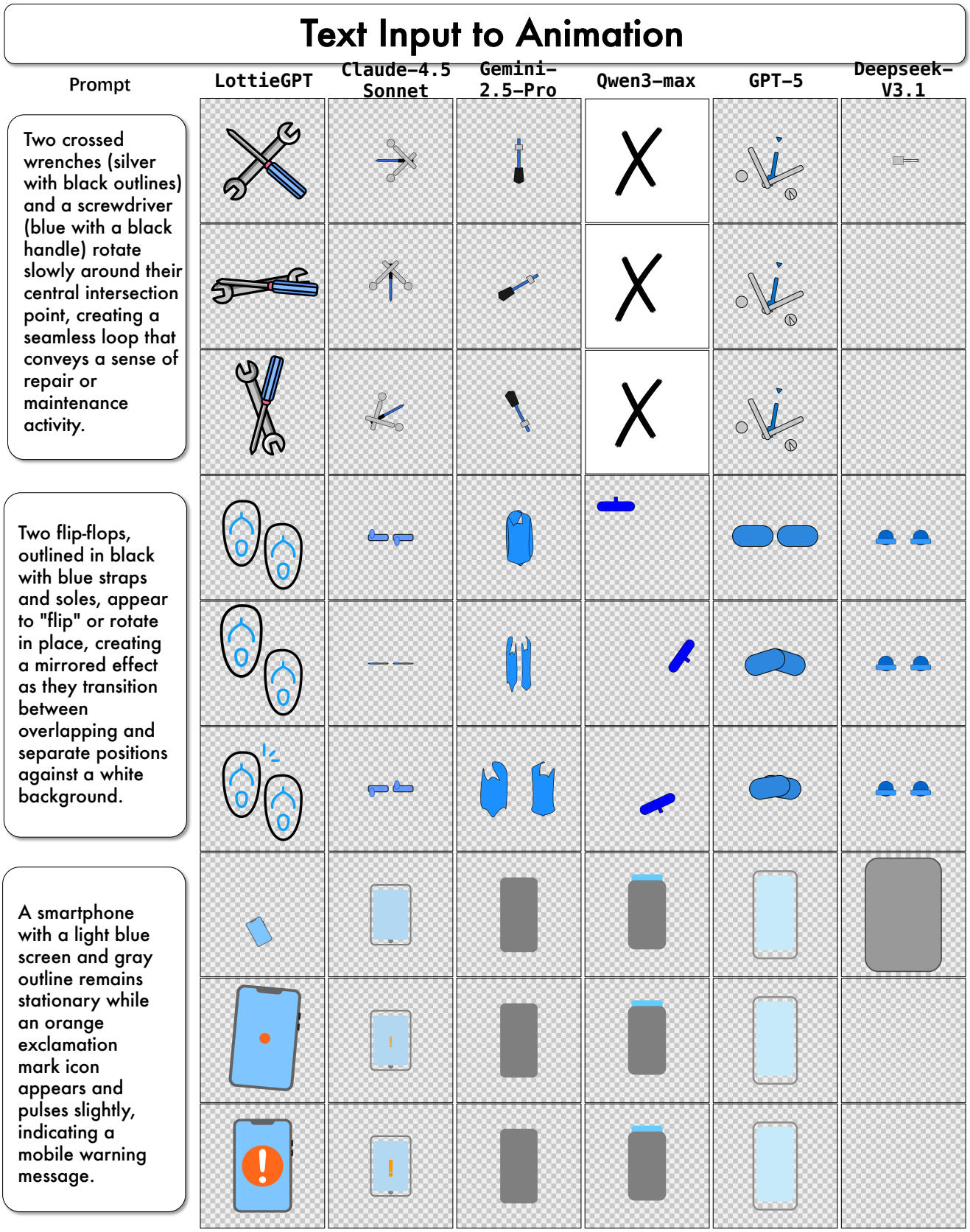


Figure 3. For the Text-to-Animation task, all LLM baselines were provided with identical 3-shot examples. ✗ indicates that no renderable Lottie JSON was obtained even after the fifth attempt. **More results on vector animation can be found in the supplementary video and on the project website.**

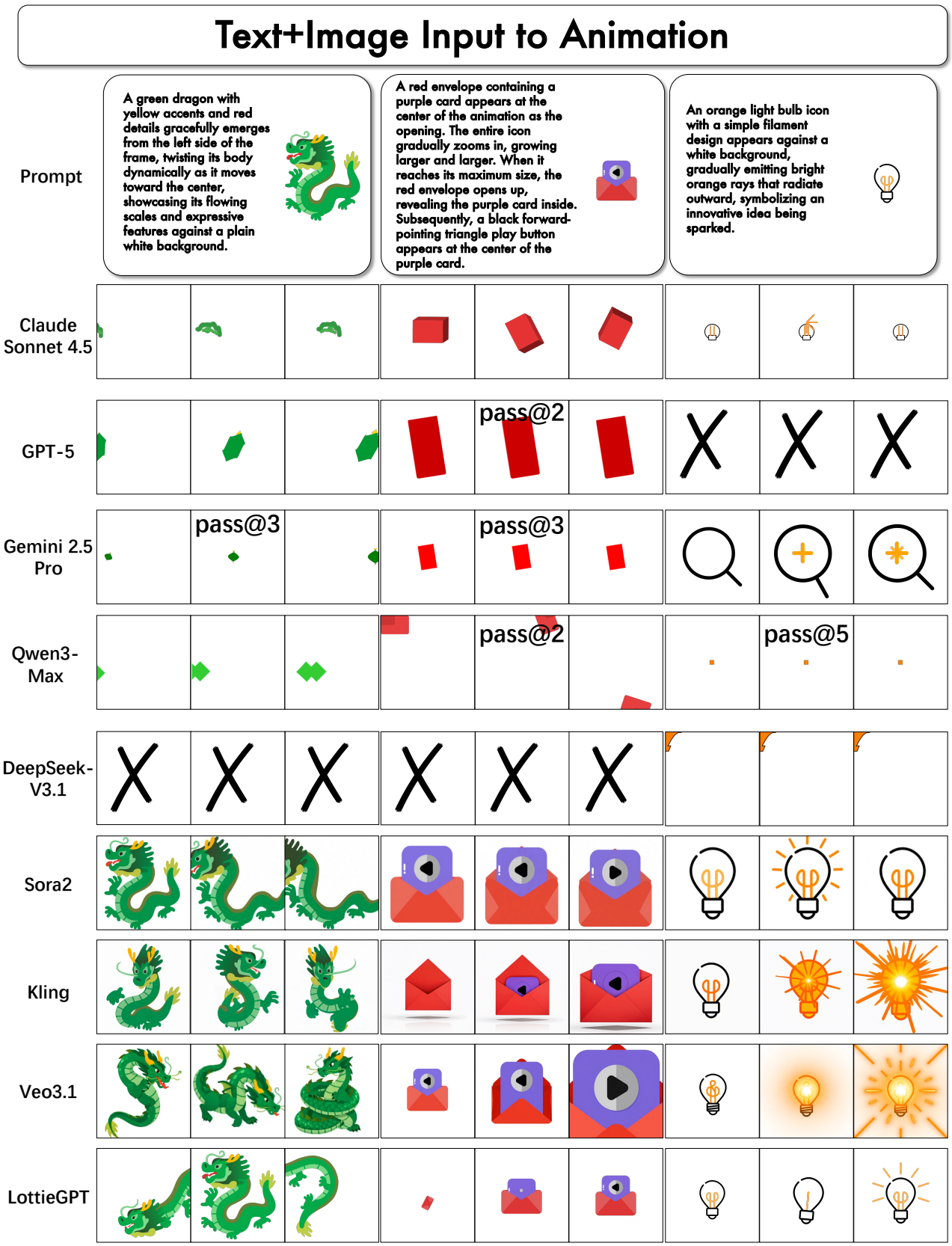


Figure 4. Using Text+Image as input to generate animations. Few-shot refers to providing three description-Lottie JSON data pairs. Except for Deepseek which does not support image input, all other methods use a single image and text description as input. pass@x indicates that x attempts were required to generate a renderable Lottie JSON. ✗ indicates that no renderable Lottie JSON was obtained even after the fifth attempt.

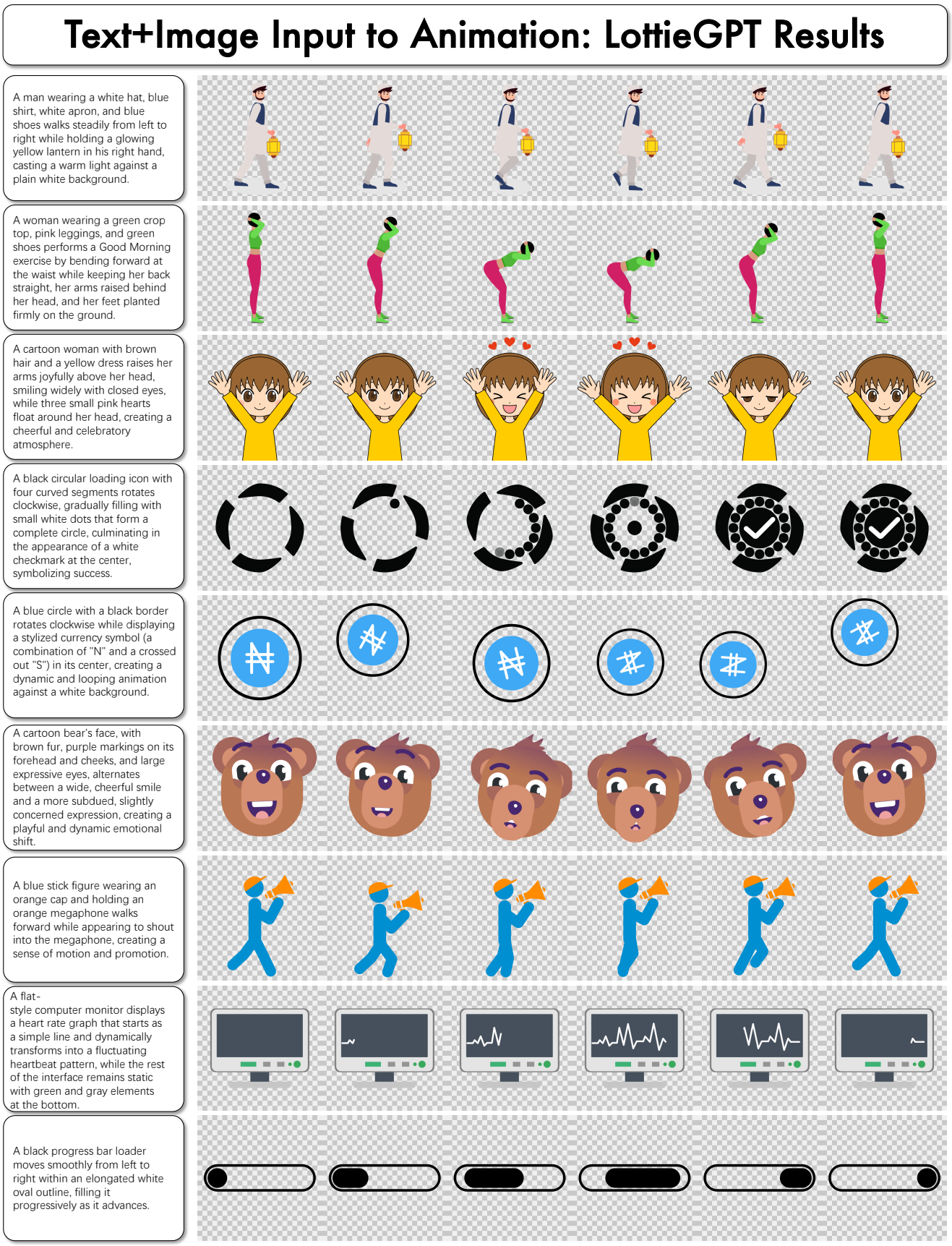


Figure 5. LottieGPT results on in-the-wild text-only inputs.

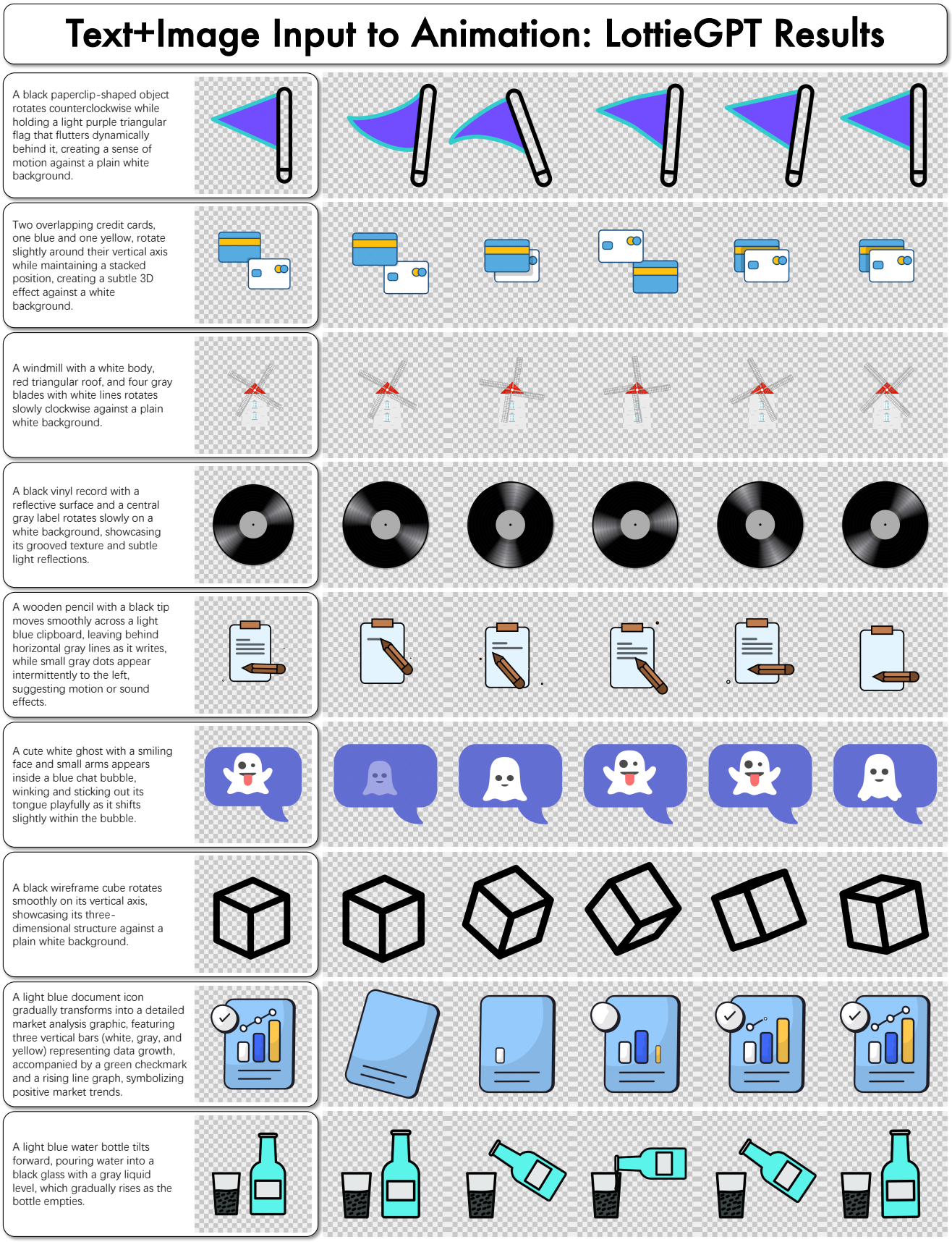


Figure 6. LottieGPT results on in-the-wild text-image inputs.

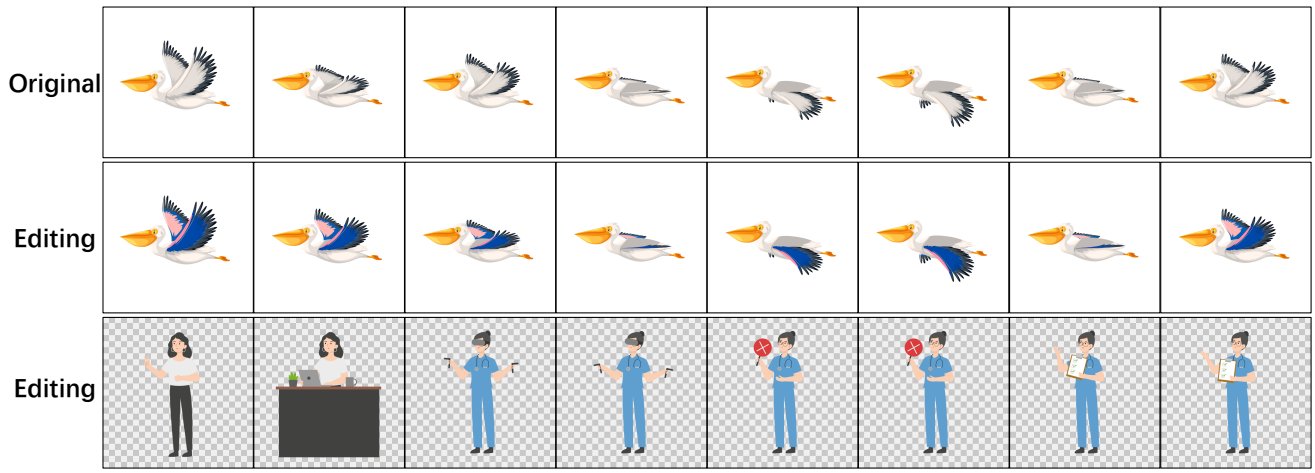


Figure 7. A manually edited Lottie animation where we modified the wing color using LottieLab.

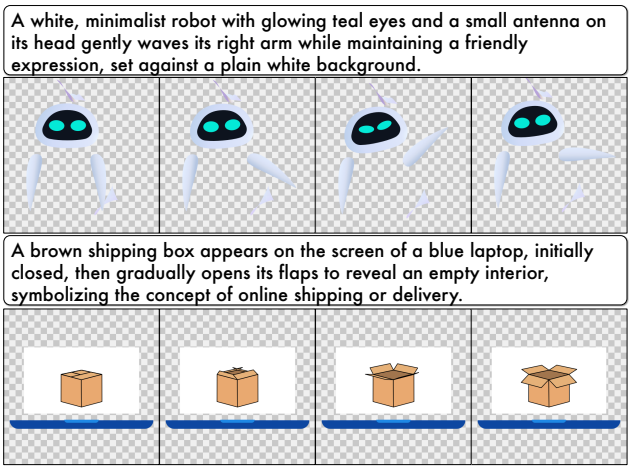


Figure 8. LottieGPT may still generate cases that are renderable but visually inconsistent with expectations, typically manifesting as extraneous or missing shapes relative to the intended design.

3. Vector Animation Task Results

All baselines were invoked through their official APIs, and we selected cases where the majority of baselines could successfully render. Comparison results are presented in Fig. 3 and Fig. 4. Since zero-shot LLM / VLM cannot generate any valid renderable Lottie JSON files, we do not present these results in our quantitative and qualitative evaluations, and only show the few-shot LLM / VLM results.

In our experimental results, most LLM-generated outputs failed to render properly. We present more LottieGPT results on in-the-wild inputs in Fig. 5 and Fig. 6.

Please refer to the supplementary materials including the project website and demo video for more animation generation comparison results.

4. Examples of Editing

Lottie animations can be created and edited using various professional tools. The most common workflow involves using Adobe After Effects [1] with the Bodymovin plugin[2] to export animations as Lottie JSON format. For online editing, LottieLab [3] and LottieFiles[4] provides a comprehensive ecosystem including animation editors, preview tools, and asset libraries. Additionally, tools such as Haiku Animator[5] and Cavalry[6] also support direct creation and export of Lottie animations. To provide a more intuitive illustration of the editability and flexibility advantages of Lottie Animation over traditional raster videos, we present an example from the dataset in Fig. 7 with various editing results.

5. Failure Cases

Some failure cases of LottieGPT are shown in Fig. 2 and Fig. 8. The Valid Rate in Tab. 2 of the main text refers to the rendering success rate rather than the exact match rate. Most Lottie JSON files generated by VLMs cannot be rendered properly (they do not conform to the standard Lottie JSON format). Rendering failures in LottieGPT typically arise from two scenarios. First, the presence of non-numeric content, such as unexpected characters, in numerical values during Lottie token generation causes detokenization to fail. Second, when the generated Lottie token sequence exceeds the maximum context length, it becomes truncated, preventing successful detokenization and valid Lottie JSON generation. Similarly, the rendering failures of OmniSVG shown

[1] https://adobe.com/products/aftereffects.html
[2] https://aescripts.com/bodymovin/
[3] https://www.lottielab.com/?home
[4] https://lottiefiles.com/lottie-editor
[5] https://www.haikuanimator.com/
[6] https://cavalry.scenegroup.co/

in Fig. 2 are attributed to the generation of unexpected characters that cause detokenization failures.

6. Lottie Dataset

We introduce three comprehensive datasets for vector graphics and animation generation: LottieSVG-10M, LottieImage-15M, and LottieAnimation-660K. These datasets provide paired data of vector code, textual descriptions, and rendered images/videos, enabling multimodal learning for vector graphics generation. **Please refer to the supplementary materials for dataset examples.**

6.1. Details of Data Curation Pipeline

While pixel-based visual data has become abundant with the rapid development of generative models for images and videos, and several large-scale static vector graphics datasets (e.g., SVG) are widely used, existing resources focus almost exclusively on static content, lacking systematically curated large-scale vector animation datasets.

Due to SVG's limited support for complex timeline-based animations, obtaining high-quality vector animations directly is challenging. We therefore turn to the After Effects (AE) ecosystem, leveraging the Bodymovin plugin[7] to export AE animations into Lottie JSON format. As a lightweight, cross-platform vector animation representation, Lottie supports dynamic properties such as keyframes, path morphing, and color gradients in JSON format, making it well-suited for autoregressive model training.

We collect a large number of AE source files from public resources and convert them to Lottie format via an automated pipeline. Additionally, we integrate over 15 million static vector graphics, uniformly converted to Lottie representation, to support a progressive "static-first, then dynamic" training strategy. To enable multimodal generation, we use BLIP-2 [4] to generate text descriptions for static vector graphics, and employ Qwen2.5-VL 32B [1] to produce detailed, temporally aligned descriptions after rendering vector animations into videos. Fig. 2 in main text illustrates our dataset construction pipeline and sample examples.

During dataset construction, we apply rigorous filtering and standardization. First, we remove all animations with rendering failures or visual anomalies (e.g., blank frames, misalignment, flickering). Second, we simplify Lottie JSON structures by removing redundant fields that do not affect rendering (e.g., comments, unused layer properties, debug metadata), and unify version formats and key path representations to ensure data consistency and training stability. The right side of Fig. 3 in main text shows the word cloud of text descriptions and file size distribution for the LottieAnimation dataset. Our simplification reduces sequence length by approximately 34% without affecting animation rendering quality.

LottieImage and LottieAnimation cover diverse graphic design assets including vector icons, infographics, animated illustrations, cartoon character animations, and UI motion effects, exhibiting rich semantic and stylistic variation. We present the first systematically curated, large-scale dataset of paired vector graphics and animations in **Lottie format**. Tab. 1 in main text compares our datasets with existing vector graphics datasets. This dataset provides a novel structured representation foundation for vector generation, multimodal understanding, and text-to-animation synthesis, advancing generative models toward scalable, editable, and lightweight dynamic content.

6.2. LottieSVG-10M Dataset

We collect and filter 10 million unique SVG images from the internet, ensuring no overlap with existing SVG datasets. Following the annotation methodology of OmniSVG, we employ BLIP-2 with the instruction template shown in Fig. 12 to generate textual descriptions for each SVG image. The LottieSVG-10M dataset provides triplets of *(SVG code, Lottie Json, text description, rendered PNG image)*.

Category Distribution. The dataset comprises four main categories based on creator designations, as shown in Fig. 9. Icon-designer content dominates with 54.82%, followed by illustrator (18.96%), motion-designer (14.64%), and 3d-designer (11.58%). Representative examples from each category are visualized in Fig. 9.

File Size Distribution. Fig. 10 shows the file size distribution of SVG images in the 0-20KB range, which covers the majority of the dataset. The distribution exhibits a right-skewed pattern with a mean size of 4.65KB and median of 2.25KB, indicating that most SVG files are compact and efficient for storage and transmission.

Tag Distribution. To analyze the semantic content of our dataset, we visualize the tag distribution as a word cloud in Fig. 11. The most frequent tags include "business" (535,098), "food" (407,321), "money" (316,625), "technology" (314,205), and "finance" (288,782), reflecting the diverse application domains covered by our dataset.

6.3. LottieImage-15M Dataset

We construct the largest vector graphics dataset to date by combining LottieSVG-10M, SVG-Stack-2M, OmniSVG-2M, and SVGX-1M, totaling 15 million SVG images with detailed textual annotations.

To enable Lottie-based generation, we develop a conversion pipeline that transforms SVG static images into Lottie static images. The conversion process includes:

- Parsing SVG elements and attributes
- Mapping SVG primitives to Lottie shape layers

[7] https://aescripts.com/bodymovin/

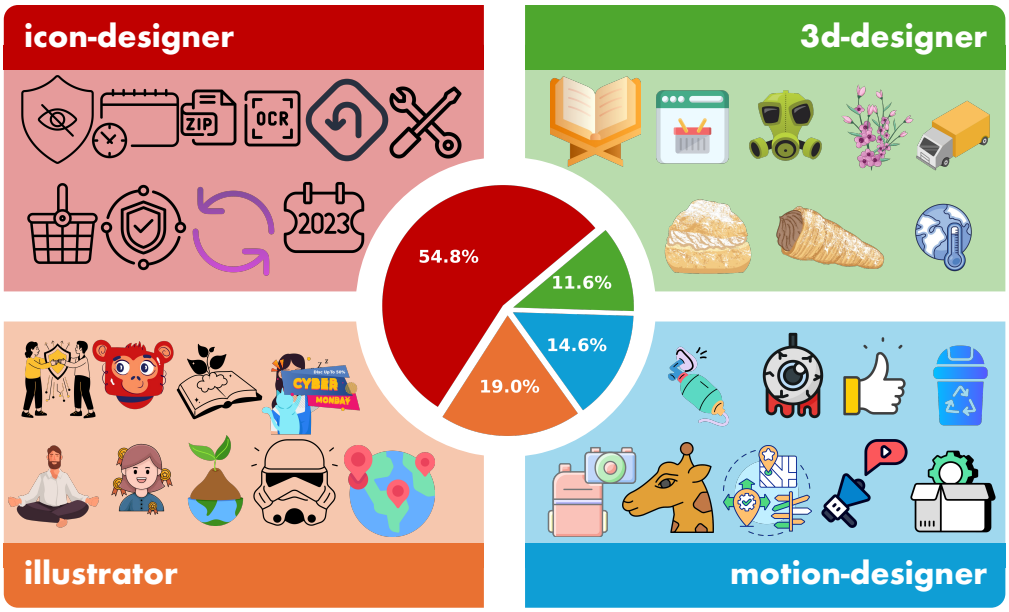


Figure 9. Representative examples from the four categories in LottieSVG-10M dataset. From top to bottom: icon-designer, illustrator, motion-designer, and 3d-designer. Each category exhibits distinct visual characteristics and complexity levels.

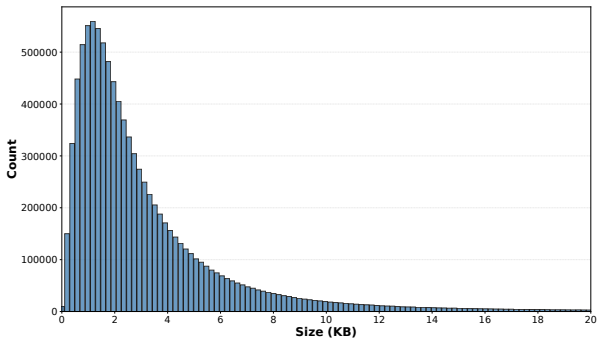


Figure 10. File size distribution of LottieSVG-10M dataset in the 0-20KB range. The distribution shows a mean of 4.65KB and median of 2.25KB, demonstrating the compact nature of SVG representations.

- Converting coordinate systems and transformations
- Validating rendering consistency

We filter out samples where the rendered output differs between SVG and Lottie formats, ensuring high-quality paired data. The LottieImage-15M dataset provides quadruplets of *(SVG code, Lottie JSON code, text description, rendered PNG image)*.

6.4. LottieAnimation-660K Dataset

The LottieAnimation-660K dataset contains 671,121 Lottie animations with comprehensive temporal annotations. The dataset comprises a total of 70,261,325 frames, spanning 608 hours of animation content. On average, each animation contains 104.7 frames with a median of 90 frames,

Figure 11. Word cloud visualization of tag distribution in LottieSVG-10M dataset. The size of each word corresponds to its frequency, with top tags including business, food, money, technology, and finance.

ranging from 1 to 3,600 frames. In terms of duration, the animations average 3.26 seconds with a median of 3.00 seconds, ranging from 0.02 to 91.40 seconds.

Duration Distribution. As shown in Tab. 2, the animation durations exhibit a concentrated distribution pattern. The majority of animations (70.55%) fall within the 2-5 sec-

- **SVG data Captioning:** You are a helpful assistant. Your task is to describe this image in a single sentence, including the object, its color, and its overall arrangement. For example: "Yellow cheers with glasses of alcohol drinks." / "Heart emojis represent love on Valentine's Day."

Figure 12. The instruction for SVG data captioning.

ond range, which is typical for UI animations and micro-interactions. Short animations (1-2s) account for 10.29%, while very short animations (0-1s) represent only 1.51%. Longer animations (5-10s) comprise 16.19%, and ultra-long animations exceeding 10 seconds constitute merely 1.46% of the dataset, indicating a focus on concise, purposeful animations rather than extended narrative sequences.

Table 2. Duration distribution of LottieAnimation-660K dataset.

Duration Range	Count	Percentage	Total Duration
0-1s (Very short)	10,107	1.51%	1.98 hours
1-2s (Short)	69,075	10.29%	27.35 hours
2-3s (Medium)	228,808	34.09%	135.76 hours
3-5s (Long)	244,724	36.46%	234.14 hours
5-10s (Very long)	108,633	16.19%	176.93 hours
10s+ (Ultra long)	9,774	1.46%	31.90 hours

Simplification and Optimization. To reduce token count while maintaining visual quality, we apply a multi-step optimization process:

1. **Expression removal:** Remove After Effects expressions that are not supported in standard Lottie players
2. **Field pruning:** Remove unused fields including `nm` (name), `mn` (match name), `hd` (hidden), `ix` (index), and `cix` (undocumented)
3. **Precision reduction:** Round numerical values to 4 significant digits (excluding color values)
4. **Color compression:** Convert color values to hexadecimal representation
5. **Metadata update:** Update Lottie JSON keys for backward compatibility

We utilize the `lottie-optim`[8] tool for automated optimization. As shown in Tab. 1, this simplification approach reduces the average Lottie JSON length by approximately 34% (from 60.72 KB to 39.87 KB) on the LottieAnimation dataset while maintaining rendering quality with no perceptible visual degradation.

Annotation. We employ Qwen2.5-VL to generate textual descriptions for Lottie animations using the instruction template shown in Fig. 13. The annotation process considers both spatial content and temporal dynamics, producing descriptions that capture motion patterns, timing, and visual effects.

[8] https://github.com/levibuzolic/lottie-optim

- **Lottie Animation data Captioning:** You are a helpful assistant. Your task is to describe this animation in a single sentence, including the objects involved, their colors, positions, actions, and overall layout. This description should enable a designer unfamiliar with the animation to understand its content, style, and structure and create a similar one. For example: "A blue arrow moves from left to right with a faint blue flicker effect." "A red character rotates around a circular path, waving a flag in hand." The title of this video is '{*video_title*}', please describe this animation in a single sentence, including the objects involved, their colors, positions, actions, and overall layout. Be sure to include any movements or dynamic changes occurring in the video.

Figure 13. The instruction for Lottie animation data captioning.

6.5. Instruction Templates

We design task-specific instruction templates for different generation scenarios. Figures 14 through 18 present the complete set of instruction templates used in our framework:

- **Static image generation:** Text-to-SVG (Fig. 14) and text+image-to-SVG (Fig. 15)
- **Animation generation:** Text-to-Lottie (Fig. 16), text+image-to-Lottie (Fig. 17), and text+video-to-Lottie (Fig. 18)

All generation instructions emphasize the use of compressed token format with special tokens, explicitly prohibiting direct JSON output to ensure compatibility with our tokenization scheme.

- **Text to Lottie Image Instruction:** You are a helpful Animation Generator. You output animations in compressed token format using special tokens. You never output JSON format. Generate an static animation based on this description: "{*caption*}"

Figure 14. The instruction for text to Lottie image generation.

- **Text and Image to Lottie Image Instruction:** You are a helpful Animation Generator. You output animations in compressed token format using special tokens. You never output JSON format. Generate an static animation based on the given image and this description: "{*caption*}"

Figure 15. The instruction for text and image to Lottie image generation.

- **Text to Lottie Animation Instruction:** You are a helpful Lottie animation Generator. You output animations in compressed token format using special tokens. You never output JSON format. Generate Lottie animation based on the description: "{*caption*}"

Figure 16. The instruction for text to Lottie animation generation.

- **Text and Image to Lottie Animation Instruction:** You are a helpful Lottie animation Generator. You output animations in compressed token format using special tokens. You never output JSON format. Generate Lottie animation based on the given image and this description: "{*caption*}"

Figure 17. The instruction for text and image to Lottie animation generation.

- **Text and Video to Lottie Animation Instruction:** You are a helpful Lottie animation Generator. You output animations in compressed token format using special tokens. You never output JSON format. Generate Lottie animation based on the given video and this description: "{*caption*}"

Figure 18. The instruction for text and video to Lottie animation generation.

7. User Study

To validate that LottieBench aligns with human preferences, we conduct a user study following OmniSVG's protocol [7]. We recruit 20 participants with design backgrounds to evaluate outputs from LottieGPT and baselines on three aspects: overall preference, visual vividity, and alignment with input prompts. Participants rate shuffled outputs (without method labels) on a 5-point Likert scale. Tab. 3 shows that LottieGPT achieves the highest scores across all metrics for both static graphics and animation tasks, demonstrating that LottieBench metrics correlate strongly with human judgment.

Table 3. User study results (5-point scale, 20 participants). LottieGPT achieves the highest ratings across all metrics.

Method	**Static Graphics**			**Animation**		
	Pref.	Vivid.	Align.	Pref.	Vivid.	Align.
OmniSVG [7]	3.6	3.8	3.7	-	-	-
StarVector [6]	3.4	3.5	3.3	-	-	-
Sora2 [5]	-	-	-	3.8	3.3	3.6
Veo 3.1 [2]	-	-	-	3.9	3.5	3.5
Kling [3]	-	-	-	3.4	3.8	3.5
LottieGPT	**4.1**	**4.2**	**4.0**	**3.8**	**3.9**	**3.7**

8. Keyframe Easing Interpolation

Lottie achieves smooth animations through keyframe interpolation with cubic Bézier easing curves. This section explains how time progress is transformed into animation progress using concrete examples from our dataset.

8.1. Core Concept: Time vs. Animation Progress

The key to understanding easing is distinguishing between two types of progress: **(1) Time Progress** (t_{norm}): Linear progression of time from 0 to 1. **(2) Animation Progress** (t_{eased}): Non-linear progression of the animated value from 0 to 1. The Bézier easing curve maps time progress to animation progress, creating natural-looking motion. Fig. 19 illustrates this relationship.

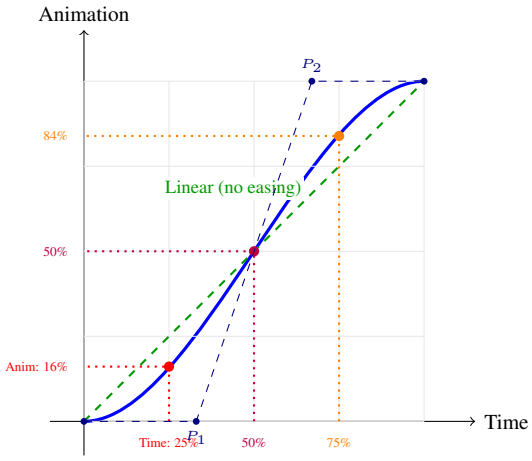


Figure 19. Bézier easing curve transforms time progress (x-axis) into animation progress (y-axis). At 25% time, animation has only progressed 16% (slow start); at 75% time, animation has reached 84% (slow end).

8.2. Bézier Curve Definition

The easing function maps normalized time progress to animation progress:

$$t_{\text{eased}} = f(t_{\text{norm}}), \quad t_{\text{norm}}, t_{\text{eased}} \in [0,1] \tag{1}$$

This function is defined implicitly via a cubic Bézier curve with fixed endpoints $P_0 = (0,0)$, $P_3 = (1,1)$ and control points $P_1 = (o_x, o_y)$, $P_2 = (i_x, i_y)$:

$$\begin{aligned}\mathbf{B}(u) =& (1-u)^3 P_0 + 3(1-u)^2 u P_1 \\ &+ 3(1-u)u^2 P_2 + u^3 P_3, \quad u \in [0,1]\end{aligned} \tag{2}$$

Expanding with $P_0 = (0,0)$ and $P_3 = (1,1)$:

$$t_{\text{norm}} = x(u) = 3(1-u)^2 u \cdot o_x + 3(1-u)u^2 \cdot i_x + u^3 \tag{3}$$

$$t_{\text{eased}} = y(u) = 3(1-u)^2 u \cdot o_y + 3(1-u)u^2 \cdot i_y + u^3 \tag{4}$$

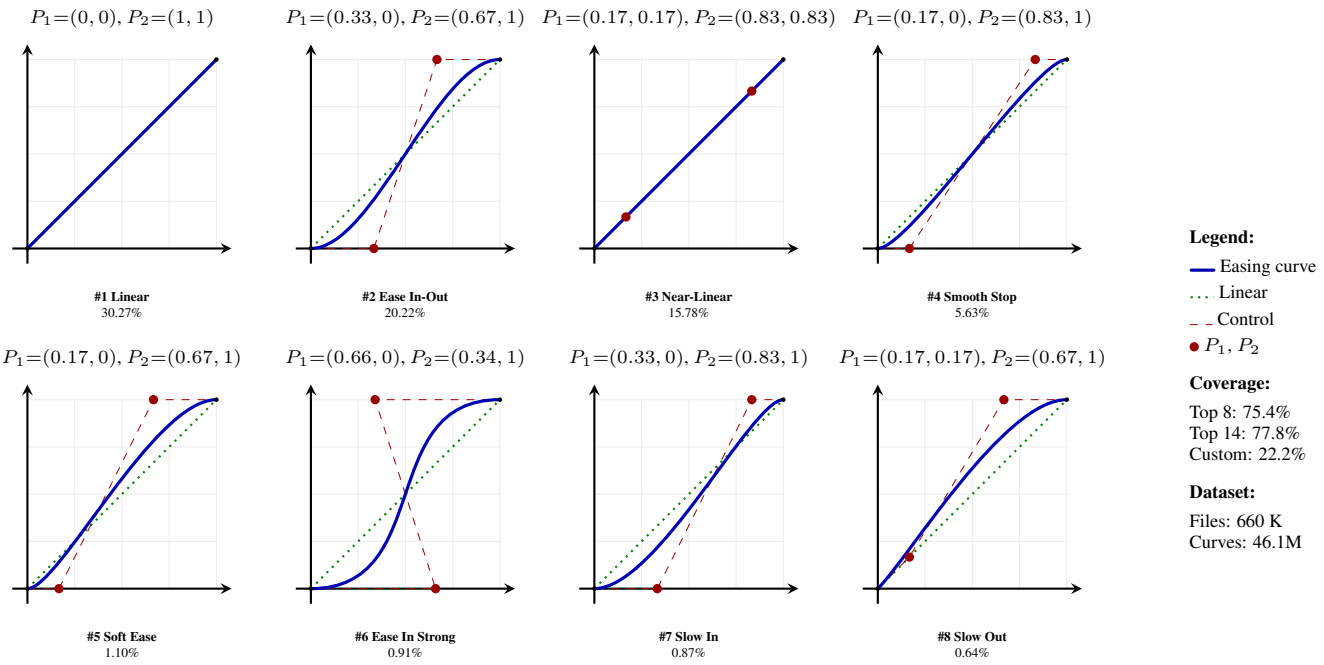


Figure 20. Top 8 most common Bézier easing curves in Lottie animations, covering 75.4% of all usage. Each curve is defined by control points $P_1 = (o_x, o_y)$ and $P_2 = (i_x, i_y)$, with fixed endpoints at $(0, 0)$ and $(1, 1)$. Green dotted line shows linear interpolation for comparison. The legend on the right provides dataset statistics and visual element descriptions. The third most common pattern (15.78%) uses control points $(0.167, 0.167)$ and $(0.833, 0.833)$, which lie on the linear diagonal, making it functionally identical to #1 Linear. This redundancy represents a significant compression opportunity.

To evaluate $f(t_{\text{norm}})$: solve Eq. (3) for u, then compute Eq. (4). This is necessary because the Bézier curve provides a parametric (not explicit) definition of f.

8.3. Top 8 Easing Curves in Lottie Animations

Fig. 20 shows the eight most common easing curves from our analysis of 46.1M curves across 660K animation files. These patterns account for 75.4% of all usage, revealing clear preferences and encoding redundancies: Linear (30.27%) uses identity control points, while the third pattern (15.78%) is mathematically equivalent to linear despite explicit Bézier parameters $(0.167, 0.167)$ and $(0.833, 0.833)$. Together, these represent 46.05% linear motion. The second pattern (20.22%) provides standard ease-in-ease-out motion with control points $(0.333, 0)$ and $(0.667, 1)$, mimicking real-world physics.

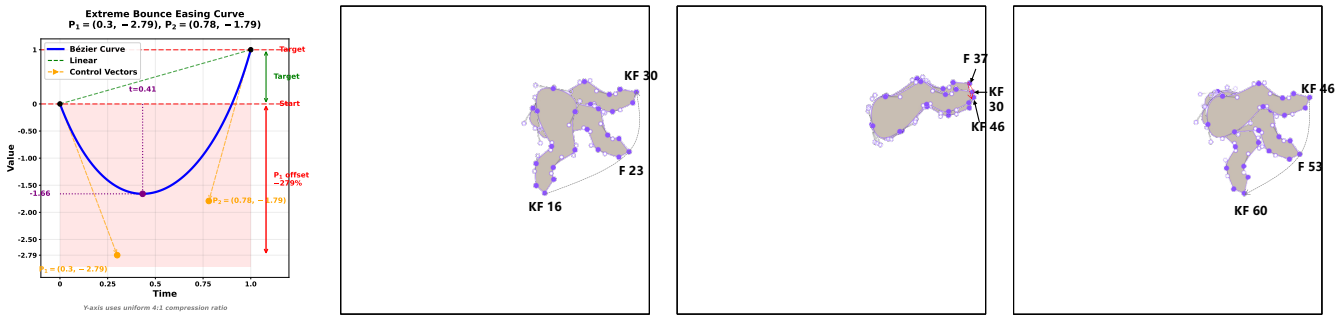
This high repetition rate presents a significant compression opportunity. The top 14 presets cover 77.8% of all curves, suggesting that a token-based encoding scheme could substantially improve storage density by replacing redundant Bézier parameters with compact preset identifiers.

8.4. Bounce Easing Example

Lottie's Bézier easing curves support control points with y-coordinates outside $[0, 1]$, enabling advanced effects such as bounce and overshoot animations. This section demonstrates how such curves are interpolated to create dramatic spring-like motion. We demonstrate the easing interpolation function using the example from Fig. 4 in main text, focusing on the interval between frame 30 and frame 46. Fig. 21 illustrates the Bézier curve of this spring motion effect in detail. Consider a rotation animation from -113.4 to -109.5 over frames 30-46 (16 frames) with extreme bounce easing defined by control points $P_1 = (0.3, -2.79)$ and $P_2 = (0.78, -1.79)$:

```
"r": {
  "a": 1,
  "k": [
    {"t": 30, "s": [-113.4],
     "o": {"x": [0.3], "y": [-2.79]},
     "i": {"x": [0.78], "y": [-1.79]}},
    {"t": 46, "s": [-109.5],
     "o": {"x": [0.35], "y": [0.16]},
     "i": {"x": [0.83], "y": [0.87]}}
  ]
}
```

Note that the first keyframe uses extreme negative y-values in its outgoing control points, creating the bounce effect, while the second keyframe uses standard positive values for the next segment.



(a) Bézier curve with extreme undershoot

(b) Keyframes 16 and 30, interpolated frame 23

(c) Keyframes 30 and 46, interpolated frame 37 (bounce)

(d) Keyframes 46 and 60, interpolated frame 53

Figure 21. Bounce easing animation demonstrating extreme undershoot. (a) Bézier curve with $P_1 = (0.3, -2.79)$ and $P_2 = (0.78, -1.79)$ reaching minimum $y = -1.6583$ at $t_{\text{norm}} = 0.4134$ (Y-axis compressed 4:1 for visibility). (b) Normal interpolation between keyframes 16 and 30. (c) Bounce segment between keyframes 30 and 46, showing extreme undershoot at frame 37 where the 3.9 target rotation is amplified to a 6.47 reverse motion (165.8% overshoot, actual minimum at frame 36.61). (d) Settling segment between keyframes 46 and 60. We removed the layer-level animations and shape-level position animations from the original animation, retaining only the shape-level rotation animations.

Interpolation Process. To compute the rotation at frame $f = 37$, we follow a four-step process:

Step 1: Normalize time progress

$$t_{\text{norm}} = \frac{f - t_{\text{start}}}{t_{\text{end}} - t_{\text{start}}} = \frac{37 - 30}{46 - 30} = \frac{7}{16} = 0.4375 \tag{5}$$

Step 2: Solve for curve parameter u Find u such that $x(u) = t_{\text{norm}}$ using Eq. (3). For $t_{\text{norm}} = 0.4375$:

$$3(1-u)^2 u \cdot 0.3 + 3(1-u)u^2 \cdot 0.78 + u^3 = 0.4375 \tag{6}$$

Using Newton-Raphson iteration yields $u \approx 0.4603$.

Step 3: Evaluate animation progress Compute $t_{\text{eased}} = y(u)$ using Eq. (4) with the negative control point y-values:

$$\begin{aligned} t_{\text{eased}} &= 3(1-0.4603)^2 \cdot 0.4603 \cdot (-2.79) \\ &\quad + 3(1-0.4603) \cdot (0.4603)^2 \cdot (-1.79) + (0.4603)^3 \\ &\approx -1.6531 \end{aligned} \tag{7}$$

The negative value indicates the animation has reversed by 165.31% of its total range.

Step 4: Interpolate final value The rotation range is $\Delta r = -109.5 - (-113.4) = 3.9$:

$$\begin{aligned} \text{rotation}(37) &= s_{\text{start}} + (s_{\text{end}} - s_{\text{start}}) \times t_{\text{eased}} \\ &= -113.4 + 3.9 \times (-1.6531) \\ &= -113.4 - 6.45 \\ &= -119.85 \end{aligned} \tag{8}$$

Despite the animation's target being a small 3.9 clockwise rotation (from -113.4 to -109.5), the bounce effect causes the object to first rotate 6.45 counterclockwise to -119.85 before rebounding to the final position. The actual minimum occurs at frame 36.61 ($t_{\text{norm}} = 0.4134$) with $t_{\text{eased}} = -1.6583$ and rotation -119.87.

Table 4. Bounce animation progression with extreme undershoot easing

Frame	t_{norm}	t_{eased}	Rotation	Phase
30	0.0000	0.0000	−113.40	Initial state
35	0.3125	−1.5663	−119.51	↺
36	0.3750	−1.6453	−119.82	↺
36.61	0.4134	−1.6583	−119.87	↺
37	0.4375	−1.6531	−119.85	Velocity ≈ 0
38	0.5000	−1.5925	−119.61	↻
40	0.6250	−1.2781	−118.38	↻
43.6	0.8500	−0.1279	−113.90	↻
46	1.0000	1.0000	−109.50	Settle at target

Animation Phases. Tab. 4 shows the complete bounce trajectory through distinct phases:

- **Undershoot phase** (frames 30-36.61): Despite a target of only 3.9 clockwise rotation, the object rotates 6.47 counterclockwise (165.8% overshoot), reaching maximum at frame 36.61 where $t_{\text{eased}} = -1.6583$
- **Velocity zero point**: At frame 37 ($t_{\text{norm}} = 0.4375$, $u \approx 0.4603$), the curve's derivative $\frac{dy}{du} \approx 0$, marking the transition from CCW to CW motion. This occurs slightly after the minimum point.
- **Rebound phase** (frames 37-43.6): Clockwise acceleration crosses the starting position at frame 43.6
- **Settling phase** (frames 43.6-46): Deceleration to final position at -109.5

This spring-like behavior is achieved through negative y-values in control points, creating animation progress values outside $[0, 1]$. The extreme undershoot of 165.8% demonstrates how Bézier easing can amplify small rotations into dramatic bounce effects, commonly used for impact animations, dramatic transitions, and physics-based UI feedback.

9. Limitations and Future Work

While Lottie Animation can compress After Effects animations into a compact representation and achieve higher compression ratios compared to SVG, and our proposed Lottie Tokenizer has been validated on existing VLMs to generate Lottie animations from text and image inputs, there remain several key limitations that warrant future investigation.

Limited Color Representation. A fundamental constraint of all vector graphics and vector animations (including SVG, Lottie, and HTML/CSS animations) is their inability to express the full range of colors present in real-world imagery. Vector graphics typically represent colors by filling shapes with solid colors or regular gradients, which limits their capacity to capture complex color gradients, textures, and photorealistic details. This inherent limitation makes vector-based representations less suitable for tasks requiring high-fidelity color reproduction or photorealistic rendering. Future work could explore hybrid representations that combine the compactness of vector formats with richer color modeling capabilities, such as incorporating procedural texture representations.

Complex Animation Effects. While the current Lottie format is powerful for many animation scenarios, it has limitations in representing certain advanced effects commonly used in professional animation workflows, such as complex particle systems, intricate shape paths, advanced blending modes, and 3D transformations. The existing Lottie tokenizer still has low compression efficiency for these components, and future work will need to extend the Lottie tokenizer accordingly.

Temporal Coherence and Long Animations. Although our model can generate animations with reasonable temporal coherence, generating very long and complex animations remains challenging due to the context length limitations of VLM generation. Future work could explore hierarchical generation strategies to better model long-range dependencies in animation sequences.

Future Work. Despite these limitations, After Effects animations and the Lottie format continue to find widespread applications across numerous domains, including UI/UX design, web frontend development, 2D animated films, motion graphics, and interactive media. The compact representation, editability, and scalability of Lottie animations make them particularly valuable for these applications. Moving forward, we aim to extend LottieGPT to express more complex and realistic animations while maintaining the advantages of vector-based representations. We believe that continued advancement in vector animation generation will unlock new possibilities for creative content creation and interactive media applications.

References

[1] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025. 10

[2] Google. Veo 3.1: Video generation, 2025. 13

[3] Kuaishou. Kling, 2025. 13

[4] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pages 19730–19742. PMLR, 2023. 10

[5] OpenAI. Sora 2 system card, 2025. 13

[6] Juan A Rodriguez, Abhay Puri, Shubham Agarwal, Issam H Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. Starvector: Generating scalable vector graphics code from images and text. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 16175–16186, 2025. 2, 13

[7] Yiying Yang, Wei Cheng, Sijin Chen, Xianfang Zeng, Fukun Yin, Jiaxu Zhang, Liao Wang, Gang Yu, Xingjun Ma, and Yu-Gang Jiang. Omnisvg: A unified scalable vector graphics generation model. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. 2, 13