Appendix

In this appendix, we provide comprehensive additional content organized as follows:

- **Section A**: Additional qualitative results across all three generation tasks, along with a systematic failure mode analysis covering our method and all baselines, including a unified five-level failure taxonomy and comparative cost-benefit analysis.
- **Section B**: Complete user study details, including study design and data selection, participant recruitment, evaluation interface and procedure, and correlation analysis between automated metrics and human judgments.
- **Section C**: In-depth technical details of the Lottie Tokenizer, including layer-specific parsing, vocabulary offset design with statistical analysis, sequence-to-token conversion algorithms, token-to-sequence reconstruction algorithms, design principles, and analysis of vector representation preservation and token efficiency.
- **Section D**: Comprehensive MMLottie-2M dataset documentation, covering the complete five-stage data collection and processing pipeline, dataset statistics (source distribution, temporal and spatial statistics, structural complexity), caption statistics and semantic analysis, and evaluation protocol details including LLM-as-judge prompts.
- **Section E**: Detailed Lottie layer type specifications, including layer type classification and comprehensive property documentation for all supported layer types.

A. Qualitative Results and Failure Analysis

This appendix provides comprehensive qualitative evaluation through additional generation examples and systematic failure mode analysis.

A.1. Additional Qualitative Results

We provide additional qualitative examples demonstrating the performance of OmniLottie across all three generation tasks: text-to-Lottie, text-image-to-Lottie, and video-to-Lottie. These examples showcase diverse scenarios including varying complexity levels, different object types, and various motion patterns. Results are shown in Fig. 7.

A.2. Failure Case Analysis

Despite achieving high success rates across all tasks (97.3%, 92.0%, and 90.7% for Text-to-Lottie, Text-Image-to-Lottie, and Video-to-Lottie respectively, understanding the failure modes of both OmniLottie and baseline methods is crucial for future improvements. We present a systematic analysis organized by method type, followed by a comparative summary.

A.2.1. Failure Taxonomy Overview

We categorize all failure modes into a unified hierarchy based on the stage at which failures occur: (1) **Level 1 - Specification Failures** where generated JSON violates Lottie schema requirements; (2) **Level 2 - Structural Failures** with valid schema but empty or malformed content; (3) **Level 3 - Rendering Failures** with valid structure but invisible or incorrect visual output; (4) **Level 4 - Pipeline Failures** with valid Lottie but downstream conversion errors; and (5) **Level 5 - Input-Dependency Failures** where method is inapplicable to given input characteristics.

A.2.2. Failure Analysis of OmniLottie

OmniLottie primarily exhibits Level 2 and Level 3 failures, indicating strong specification compliance but occasional content generation issues. Failure rates increase with input complexity: Text-to-Lottie (2.7%) $<$ Text-Image-to-Lottie (8.0%) $<$ Video-to-Lottie (9.3%).

Structural Generation Failure ($\sim$35% of failures). The model generates syntactically valid Lottie JSON with correct global structure (`v`, `fr`, `ip`, `op`, `w`, `h`), but produces an empty `layers` array. This typically occurs when text prompts are ambiguous or input images/videos contain complex visual elements that the model struggles to decompose into vector primitives.

```
{
  "v": "5.12.1",
  "fr": 25.0,
  "ip": 0.0,
  "op": 105.0,
  "w": 512.0,
  "h": 512.0,
  "layers": [],  // No layer generated
  "assets": [],
  "markers": []
}
```

Rendering-Level Failures ($\sim$65% of failures). Layers are generated but result in blank rendering outputs. We identify five primary sub-categories. First, missing style attributes ($\sim$40%) occur when Lottie separates geometry and styling into sibling nodes, and the model generates path data (`ty: "sh"`) but omits corresponding fill (`ty: "fl"`) or stroke (`ty: "st"`) nodes. Second, temporal visibility errors ($\sim$25%) involve layers with `ip`/`op` values outside the animation duration (e.g., `ip: 100` when global `op: 60`) or imperceptibly short durations. Third, opacity/scale value errors ($\sim$25%) arise because Lottie uses 0-100 for opacity and percentage-based scaling, causing predictions like `o: 1` (1% opacity) or `s: [1,1]` (1% scale) to render objects effectively invisible. Fourth, off-canvas positioning ($\sim$10%) places coordinates far outside canvas bounds

Figure 7. **Additional Qualitative Results Across Three Generation Tasks**. We present additional examples for Text-to-Lottie, Text-Image-to-Lottie, and Video-to-Lottie generation.

(e.g., `p: [5000,5000]` for a 512×512 canvas). Video-to-Lottie shows higher susceptibility to temporal errors due to motion extraction challenges, while Text-Image-to-Lottie exhibits more style-related failures from matching both textual semantics and visual appearance.

Table 4. **Failure Case Statistics.** Distribution of failure types across different tasks.

Failure Type	T2L	TI2L	V2L
Category I: Empty Layers	40.0%	30.0%	35.0%
Category II: Invalid Rendering			
Missing Style	30.0%	35.0%	20.0%
Temporal Errors	10.0%	15.0%	25.0%
Opacity Issues	10.0%	10.0%	10.0%
Scale Collapse	5.0%	5.0%	5.0%
Off-Canvas	5.0%	5.0%	5.0%

A.2.3. Failure Analysis of LLM/VLM Baselines

LLM/VLM baselines demonstrate fundamentally different failure modes, primarily at Level 1 (specification) and Level 3 (rendering). Qwen2.5-VL [4] achieves 0.0% success across all tasks due to schema hallucination, generating plausible-looking but entirely incompatible JSON that conflates Lottie with generic animation formats, using attributes like `"version": "2.0"` instead of `"v"`, `"frames"` instead of keyframe interpolation, and `"duration"` instead of `"op"`. DeepSeek [29] achieves 29.3% success but suffers from attribute hallucination, generating structurally valid JSON while injecting attributes into incorrect nodes, likely from SVG training data where geometry and styling are co-located. For example, it places color attributes `"c"` directly in rectangle shape nodes (`"ty": "rc"`) rather than in separate fill nodes (`"ty":"fl"`). Additionally, DeepSeek frequently generates text layers without corresponding `fonts` definitions.

GPT-5 [40] (12.7%-68.67% success) and Gemini [45] (0.0% Video-to-Lottie) produce schema-valid JSON with diverse rendering errors including geometric degeneration where Bézier control points collapse to identical coordinates, transform stack errors from incorrect nesting causing coordinate system corruption, and keyframe discontinuities with non-monotonic time values or infinite tangents. Recraft achieves 100% Lottie generation success but 22.7% failure in Lottie-to-video rendering, revealing limitations of decoupled systems through unsupported features (expression-driven animations or blend modes incompatible with video renderers), performance bottlenecks (excessively complex layer hierarchies with $>$50 nested groups causing timeouts), and format mismatch (features valid in web players but incompatible with video export libraries).

A.2.4. Failure Analysis of Optimization-Based Baselines

AniClipart [58] and LiveSketch [18] employ multi-stage optimization pipelines with text-to-video diffusion priors. Their failures (92.7% and 52% failure rates) occur primarily at Level 5 (input-dependency) and through optimization non-convergence.

Input Feature Dependency Failures. AniClipart requires semantic keypoints (skeletal joints, object corners) to parameterize motion as Bézier curves, causing $\sim$85% of its failures when abstract shapes or geometric patterns lack discernible keypoints, simple shapes yield insufficient keypoints for ARAP deformation constraints, or overlapping elements create keypoint ambiguity. This explains the 7.3% success rate—the method is inherently inapplicable to most icon/clipart designs. LiveSketch, designed for sketch-style inputs with sparse strokes, experiences $\sim$60% of its failures on photo-realistic images or gradient-filled shapes that cannot decompose into strokes, non-white backgrounds introducing artifacts during stroke extraction, and dense inputs causing the local-global motion optimization to overfit to noise.

Optimization Non-Convergence ($\sim$30% of combined failures). Both methods rely on Score Distillation Sampling (SDS) with highly non-convex optimization landscapes, leading to local minima convergence with degenerate solutions exhibiting minimal or jittering motion, mode collapse to generic motions (uniform scaling, simple translation) that ignore semantic structure, and gradient pathologies from differentiable rasterization. The pipeline architecture compounds errors across stages: PNG $\rightarrow$ SVG $\rightarrow$ Keypoint Extraction $\rightarrow$ SDS Optimization $\rightarrow$ Animated SVG $\rightarrow$ Lottie JSON, where vectorization errors, optimization timeouts (1200s for AniClipart, 780s for LiveSketch), and format conversion losses accumulate throughout.

A.2.5. Comparative Summary

The failure mode distribution reveals a clear hierarchy correlating with method design: Qwen2.5-VL at Level 1 (specification) achieves 0% success; DeepSeek at Level 1 (attribute-level) achieves 29.3% success; AniClipart at Level 5 (input-dependency) achieves 7.3% success; LiveSketch at Level 5 (input-dependency) achieves 48.0% success; GPT-5/Gemini at Level 3 (rendering) achieve 12.7-68.67% success; Recraft at Level 4 (pipeline) achieves 77.3% success; and OmniLottie at Level 2-3 (structure/rendering) achieves 90.7-97.3% success.

The cost-benefit analysis further differentiates the methods. AniClipart requires 1200s with 7.3% success, yielding 16,438s per successful animation. LiveSketch requires 780s with 48.0% success, yielding 1,625s per successful animation. OmniLottie requires only 28.57s with

92.0% success, yielding 31s per successful animation, a 52× speedup over LiveSketch and 530× over AniClipart per successful generation.

This analysis reveals that OmniLottie's focused training on Lottie-specific data successfully addresses specification and structural challenges that plague general-purpose LLM/VLMs, while its direct generation paradigm avoids the input-dependency and optimization convergence issues inherent to optimization-based approaches. The remaining rendering-level failures in OmniLottie represent tractable improvements through enhanced training data coverage and numerical precision.

B. User Study

We conduct a comprehensive user study follows the procotal in prior arts [7, 62, 72] to evaluate the quality and multi-modal alignment of generated Lottie animations. Twenty participants were recruited and asked to rank generated animations from different methods according to four criteria: visual quality, condition adherence, animation quality, and geometric fidelity. Each participant completed evaluation tasks across three generation scenarios: Text-to-Lottie, Text-Image-to-Lottie, and Video-to-Lottie. The results demonstrate that OmniLottie consistently achieves the highest average ranking across all dimensions and tasks, indicating superior generation quality and stronger alignment with multi-modal inputs. Moreover, correlation analysis reveals that our proposed metrics (Object Alignment and Motion Alignment) exhibit strong positive correlation with human judgments, validating their effectiveness as automated evaluation measures. Detailed study design, ranking protocols, and statistical analysis are provided in the following sections.

B.1. Study Design and Data Selection

Our user study evaluates model performance across three distinct generation tasks. Due to the practical constraints of participant workload and the necessity of ensuring high-quality evaluation samples, we adopt a stratified sampling strategy that balances task diversity with evaluation depth.

Sample Selection Strategy. We conduct the user study on all 300 benchmark items for each of the three tasks in the MMLottie Benchmark (Text-to-Lottie, Text-Image-to-Lottie, and Video-to-Lottie), resulting in 300 generation cases per task. In total, 900 evaluation cases are included in the study (300 Text-to-Lottie, 300 Text-Image-to-Lottie, and 300 Video-to-Lottie). For cases where the generation fails to produce valid animations, the result is displayed as "Failed" to the evaluators, allowing participants to assess both generation quality and system reliability within a unified evaluation framework.

Baseline Methods. For each task, we compare OmniLottie against multiple baseline methods. Text-to-Lottie comparisons include Deepseek, Recraft, Qwen2.5-VL and GPT-5. Text-Image-to-Lottie comparisons include GPT-5, Aniclipart and Livesketch. Video-to-Lottie comparisons include GPT-5 and Gemini3.1-Pro. Each evaluation group presents 3-4 anonymized animations rendered from generated Lottie files, shown in randomized order to prevent bias.

Participant Recruitment and Training. We recruit 20 participants with basic understanding of vector graphics and animation concepts. Before the formal evaluation, all participants undergo a standardized training session using a tutorial set of 3 examples (one per task) with detailed explanations of evaluation criteria. This training ensures consistent understanding of the ranking dimensions and reduces inter-rater variability.

B.2. Evaluation Interface and Procedure

The evaluation is conducted through an online questionnaire platform where participants view the rendered animations and provide rankings. For each evaluation group, participants are shown: (1) the multi-modal input condition (text prompt, image+text, or video), (2) 3-4 generated animations rendered as looping videos with transparent or white backgrounds, and (3) detailed evaluation instructions. Participants then rank each generated animation individually on a 5-star scale (5 = best, 1 = worst) across four dimensions: visual quality, condition adherence, animation quality, and geometric fidelity. The ranking is performed independently for each dimension, allowing fine-grained assessment of different quality aspects. We provide the participant instructions and evaluation procedure for Lottie animation generation in Tab. 5.

B.3. Correlation Analysis

We analyze the correlation between automated metrics and human rankings to validate the effectiveness of our proposed evaluation measures. As shown in Tab. 6, our Object Alignment and Motion Alignment metrics demonstrate strong positive correlation with corresponding human judgments across all three statistical measures (Pearson's r, Spearman's ρ, and Kendall's τ), with highly significant p-values ($p < 0.001$). The Motion Alignment metric shows particularly strong correlation (Pearson $r = 0.4823$), indicating that it effectively captures perceptually meaningful animation quality. The CLIP score also correlates moderately with human-rated condition adherence, though with slightly lower correlation coefficients compared to our specialized metrics. These results validate that our proposed metrics provide reliable automated assessment aligned with human perception.

Table 5. **Participant Instructions and Evaluation Procedure for Lottie Animation Generation.**

Participant Instructions and Evaluation Procedure

Study Overview and Task Description

This study evaluates different methods for generating vector animations (Lottie format) from multi-modal instructions. Three generation tasks are assessed: (1) Text-to-Lottie: generating animations from text descriptions, (2) Text-Image-to-Lottie: generating animations from both text and reference images, and (3) Video-to-Lottie: generating animations that reconstruct input videos. For each trial, you will observe 3-4 generated animations from different methods and rank them individually on a 5-star scale (5 = best, 1 = worst) across four evaluation dimensions. All generated animations are rendered as looping videos for consistent viewing. Please carefully review the input conditions before evaluating each animation.

Evaluation Dimensions (Per-Animation Ranking)

Rate each generated animation independently on the following four criteria:

1. Visual Quality

Assess the overall aesthetic quality and visual appeal of the animation. Consider rendering smoothness, color harmony, clarity of shapes, absence of visual artifacts, and professional appearance. Higher visual quality indicates better-rendered, more polished animations with clean vector graphics and pleasant visual composition.

2. Condition Adherence

Evaluate how well the generated animation matches the given input conditions. For Text-to-Lottie, assess whether the animation accurately reflects all elements described in the text prompt. For Text-Image-to-Lottie, evaluate alignment with both the reference image's visual content and the text description. For Video-to-Lottie, assess how faithfully the animation reconstructs the input video's content, structure, and appearance. Stronger adherence means better alignment between input conditions and generated output.

3. Animation Quality

Judge the quality of motion and temporal behavior. Consider smoothness of transitions, naturalness of movements, absence of jittering or abrupt changes, appropriate timing and pacing, and coherent temporal progression. Higher animation quality indicates smoother, more fluid, and more natural-looking motion patterns that are visually pleasing and temporally consistent.

4. Geometric Fidelity

Evaluate the accuracy and completeness of geometric shapes and structural elements. Consider shape accuracy, preservation of fine details, structural integrity, appropriate proportions, and absence of geometric distortions. For Text-Image-to-Lottie and Video-to-Lottie, also assess how well geometric elements from the input are preserved. Higher geometric fidelity indicates more accurate, detailed, and structurally sound vector representations.

Important Evaluation Guidelines

- **Independent Rating**: Evaluate each dimension separately. An animation may score high on one dimension but low on another.
- **Absolute Quality**: Focus on the absolute quality of each animation rather than making relative comparisons between methods during rating.
- **Full Scale Usage**: Use the complete 1-5 star range. Reserve 5 stars for excellent quality, 4 for good, 3 for acceptable, 2 for poor, and 1 for very poor quality.
- **Consistency**: Apply the same standards across all evaluation groups to ensure consistent judgment.

C. Lottie Tokenizer: Technical Details

C.1. Vocabulary Offset Design

To efficiently represent diverse Lottie animation parameters in a unified token space, we employ a systematic offset strategy that maps each parameter category to a distinct vocabulary region. This design prevents overlap between semantically different parameters while preserving their numerical relationships within categories.

The value ranges for each parameter type are deter-

Table 6. **Correlation Statistics Between Automated Metrics and Human Rankings.** Reported values include Pearson's r, Spearman's ρ, and Kendall's τ with corresponding p-values. Higher absolute correlation values indicate stronger agreement between automated metrics and human judgments.

Metric vs. Human Rating	Pearson r (p)	Spearman ρ (p)	Kendall τ (p)
Object Align vs. Geometric Fidelity	0.4521 (3.21e-42)	0.4638 (8.45e-45)	0.3512 (2.18e-41)
Motion Align vs. Animation Quality	0.4823 (1.52e-48)	0.4917 (4.23e-51)	0.3724 (5.67e-47)
CLIP vs. Condition Adherence	0.3867 (6.78e-34)	0.3945 (2.14e-35)	0.2918 (1.43e-32)
FVD vs. Visual Quality	-0.3245 (4.89e-28)	-0.3312 (1.67e-29)	-0.2456 (3.21e-26)

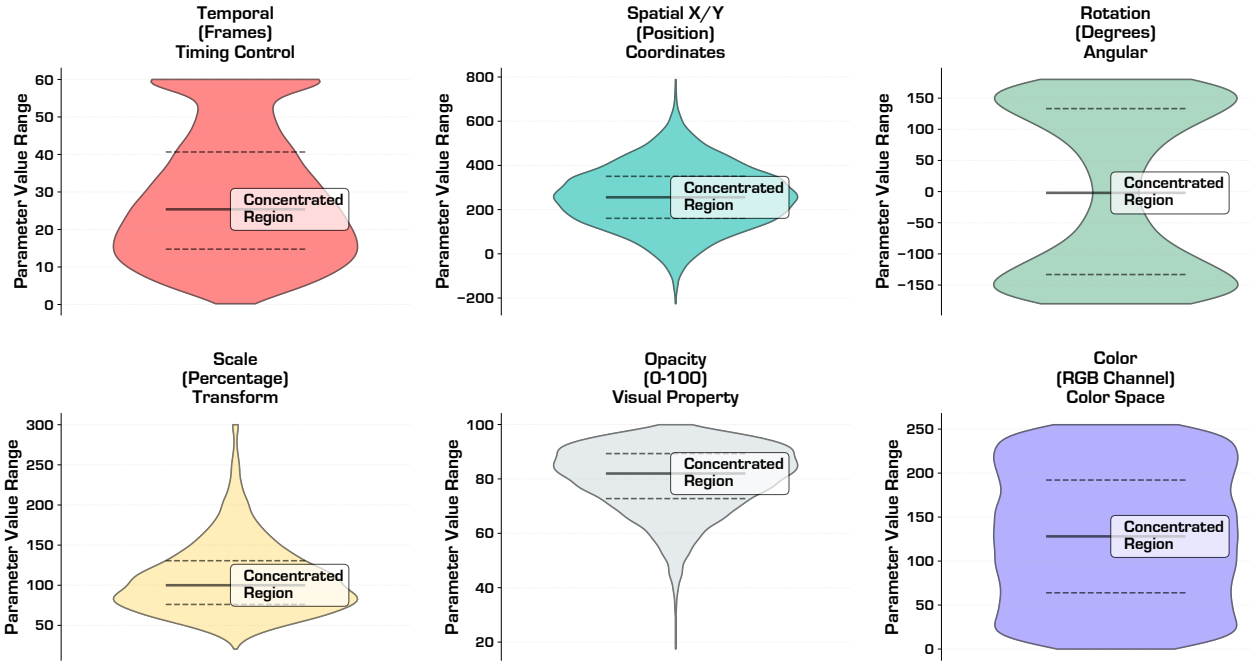


Figure 8. **Parameter Distribution Analysis for Vocabulary Offset Design.** We analyze the distribution of six key parameter types across our entire dataset to determine optimal vocabulary offset ranges. Each violin plot shows the probability density of parameter values, with solid horizontal lines indicating median values and dashed lines marking quartiles. The shaded regions highlight concentrated value ranges where the majority of parameters cluster, guiding our offset allocation strategy.

mined through comprehensive statistical analysis of our entire dataset. We examine the distribution of parameter values across all animations and identify concentrated ranges by filtering extreme outliers that represent noise or edge cases. For example, spatial parameters such as positions and anchor points exhibit natural clustering within negative to positive coordinate ranges, as illustrated in Fig. 8. Similarly, rotation angles concentrate within full rotation cycles, while scale factors cluster around percentage-based transformations. This data-driven approach ensures our vocabulary efficiently covers the practical parameter space without allocating excessive tokens to rarely-used extreme values.

Our vocabulary design encompasses multiple parameter categories with varying complexity. Binary flags such as animation state indicators and boolean properties occupy minimal space with two possible values each. Small categorical enumerations including blend modes, line caps, and shape types require modest discrete ranges. Continuous numerical parameters span larger spaces, with temporal values controlling timing, spatial coordinates managing positioning, and transformation parameters governing scaling and rotation. Color channels map to standard intensity ranges, while specialized categories like easing curves and opacity values occupy intermediate scales. Tokenizer-based parameters for text and identifiers bypass offset transformation entirely, maintaining their original token representations. This hierarchical organization allocates vocabulary space proportionally to parameter complexity, with simple flags using minimal tokens and complex transformations requiring extensive ranges to capture subtle variations in animation behavior.

Algorithm 1: Lottie Tokenizer

Input: Lottie JSON $\mathcal{J}$; Type-specific scales $\{s_t\}$, offsets $\{o_t\}$; Pretrained tokenizer $\mathcal{V}_{\text{text}}$
Output: Token sequence $\mathcal{T}$ (encoding) / Command sequence $\mathcal{S}$ (decoding)

// Encoding
Function ENCODE($\mathcal{J}$)**:**
  $\mathcal{M}, \{\mathcal{L}_i\} \leftarrow \text{Parse}(\mathcal{J})$;
  $\mathcal{T} \leftarrow [\texttt{<META>}] + \text{Quantize}(\mathcal{M})$;
  foreach *layer* $\mathcal{L}_i$ *with type* τ_i **do**
    $\mathcal{T}$.append($[\texttt{<LAYER-}\tau_i\texttt{>}]$);
    foreach *param* p *in* $\mathcal{L}_i$ **do**
      $\mathcal{T}$.append($\lfloor p \cdot s_t \rfloor + o_t$) ; // Quantization
    foreach *text field* G **do**
      toks $\leftarrow \mathcal{V}_{\text{text}}(G)$;
      $\mathcal{T}$.extend($[\text{len(toks)}] + \text{toks}$);
    $\mathcal{T}$.append($[\texttt{<END>}]$);
  return $\mathcal{T}$;
// Decoding
Function DECODE($\mathcal{T}$)**:**
  $\mathcal{S} \leftarrow []$;
  foreach *command segment in* $\mathcal{T}$ **do**
    $C \leftarrow$ new Command();
    foreach *numeric token tok* **do**
      C.add($(\text{tok} - o_t)/s_t$)
    foreach *text token group* **do**
      C.add($\mathcal{V}_{\text{text}}^{-1}$(tokens));
    $\mathcal{S}$.append(C);
  return $\mathcal{S}$;

C.2. Sequence to Token Conversion

After parsing Lottie JSON into command sequences, we discretize continuous parameters into a unified token vocabulary through our offset-based mechanism. This process handles three distinct parameter categories: continuous numerical values, text-based content, and structural markers.

Numerical Parameter Discretization. For continuous parameters (temporal, spatial, transformation, and style attributes), we apply type-specific scaling and offset transformations:

$$\text{token}(p, t) = \lfloor p \cdot s_t \rfloor + o_t \tag{6}$$

where p denotes the parameter value, t represents its semantic type, s_t is the scaling factor, and o_t is the vocabulary offset. Each parameter type occupies a distinct vocabulary range determined by data distribution analysis, ensuring no token conflicts across categories.

Text Content Tokenization. Text-related attributes including font families, character strings, and textual identifiers require special handling to preserve the pretrained model's linguistic knowledge. Rather than converting text to numerical tokens, we employ the backbone VLM's native tokenizer (Qwen2.5-VL) to encode these elements. For commands involving text parameters (font, char, reference_id), we adopt a structured representation:

$$\mathcal{T}_{\text{text}} = [\text{count}, \text{tok}_1, \text{tok}_2, ..., \text{tok}_n] \tag{7}$$

where count indicates the number of tokens, followed by the actual token sequence. This design maintains the semantic richness of text embeddings while integrating seamlessly with numerical parameter tokens.

Padding and Structural Markers. To handle optional parameters and maintain structural information, we introduce special marker tokens. For each vocabulary range defined by offset o_t, we reserve a marker token below the valid range (start$(o_t) - 1$) to represent padding values. Command boundaries are explicitly marked with command-specific start and end tokens, enabling clear hierarchical structure in the token sequence:

$$\mathcal{S} = [\text{CMD}_i, p_{i,1}, ..., p_{i,k}, \text{END}_i, \text{CMD}_{i+1}, ...] \tag{8}$$

The complete tokenization procedure is formalized in Algorithm 1, which systematically processes each command and its parameters according to their semantic types.

C.3. Token-to-Sequence Reconstruction

The detokenization process reconstructs valid Lottie JSON from generated token sequences through systematic parsing and reverse transformation. This involves three key operations: command boundary detection, parameter value recovery, and text content decoding.

Command Structure Parsing. The reconstruction begins by identifying command boundaries using start and end markers. Each command block is processed sequentially, with the command token determining the expected parameter structure. For tokenizer commands (containing text content), the parser first reads regular numeric parameters, then processes token groups by extracting the count value followed by the specified number of token IDs.

Parameter Value Recovery. Numerical parameters undergo inverse transformation to recover their original continuous values:

$$p = \frac{\text{token} - o_t}{s_t} \tag{9}$$

Special marker tokens are recognized and converted back to PAD_VAL placeholders, maintaining structural information during reconstruction. The parameter type t is

inferred from the command structure and parameter position, ensuring correct offset application.

Text Content Decoding. For text-based parameters, the token IDs are decoded using the same pretrained tokenizer employed during encoding:

$$\text{text} = \mathcal{V}_{\text{text}}^{-1}([\text{tok}_1, ..., \text{tok}_n]) \tag{10}$$

This preserves semantic consistency between encoding and decoding phases, maintaining the linguistic properties learned by the backbone VLM.

The complete reconstruction procedure is outlined in Algorithm 1, which systematically processes the token sequence while maintaining hierarchical structure and parameter relationships. This deterministic mapping ensures that valid token sequences always reconstruct to renderable Lottie JSON files.

C.4. Tokenization Design Principles

Our tokenization framework incorporates several key design principles that collectively enable efficient and accurate vector animation generation:

Separation of Concerns. By treating command tokens, numerical parameters, and text content as distinct categories with specialized handling, we allow the model to focus its capacity on learning meaningful animation patterns rather than low-level syntax. Command tokens establish structural hierarchy, numerical parameters encode geometric and temporal attributes, and text tokens preserve semantic richness.

Vocabulary Efficiency. The offset-based mapping eliminates redundancy by assigning each parameter type to a non-overlapping vocabulary range proportional to its complexity. Binary flags occupy minimal space, while continuous transformations span broader ranges to capture subtle variations. This allocation strategy maximizes token utilization while maintaining sufficient precision for animation fidelity.

Reconstruction Guarantees. The deterministic mapping between parameters and tokens ensures lossless round-trip conversion. Special marker tokens explicitly represent padding and structural boundaries, enabling the decoder to unambiguously reconstruct the hierarchical Lottie structure. This property is critical for maintaining animation integrity throughout the generation process.

Model Compatibility. Integration with pretrained text tokenizers preserves the linguistic capabilities of backbone VLMs while extending them to the animation domain. Text parameters retain their original embeddings, allowing the model to leverage learned language understanding when processing font families, character content, and textual identifiers embedded within animation structures.

C.5. Vector Representation Preservation and Token Efficiency

A fundamental concern in our tokenization approach is whether the discrete token representation contradicts Lottie's inherent vector nature. We emphasize that our tokenization scheme does not compromise the vector characteristics of Lottie animations but rather provides a more efficient intermediate representation for learning and generation. The Lottie format itself, while being a pure vector representation, suffers from substantial redundancy when encoded as raw JSON due to verbose syntax, repeated structural markers, and metadata overhead. Our tokenization pipeline systematically addresses this inefficiency through three progressive transformations: first, converting JSON to structured text sequences by removing syntactic redundancy while preserving semantic content; second, organizing these sequences into command-parameter pairs that explicitly capture the functional structure of animation primitives; and third, applying offset-based discretization that maps continuous parameters into a compact vocabulary space. Critically, this entire process is fully reversible through deterministic detokenization, ensuring that every generated token sequence can be reconstructed into a valid, renderable Lottie JSON file that maintains complete vector fidelity, resolution independence, and editability. The token efficiency gains, as illustrated in Fig. 9, demonstrate that our approach reduces sequence length by 81% compared to raw JSON (from 2,562 to 486 tokens on average) while enabling the model to focus its capacity on learning s animation patterns rather than formatting conventions. This design philosophy aligns with recent advances in structured representation learning, where appropriate abstraction layers can dramatically improve both training efficiency and generation quality without sacrificing the fundamental properties of the target domain.

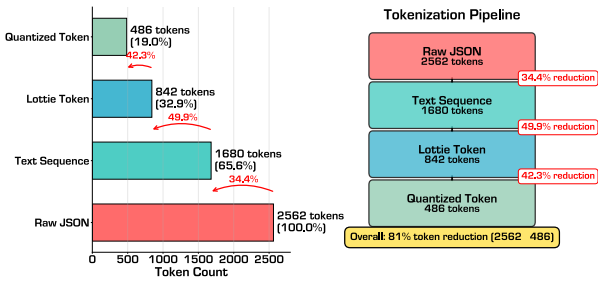


Figure 9. **Token Efficiency Through Progressive Abstraction.** Our Lottie tokenizer progressively reduces sequence length from raw JSON (2,562 tokens) through structured text sequences (884 tokens) to our final command-based representation (486 tokens), achieving 5.3× compression. The discretization operates purely at the representation level and does not compromise the vector nature of the output, as continuous parameter values are faithfully recovered during detokenization.

D. MMLottie-2M Dataset: Comprehensive Details

D.1. Data Collection and Processing Pipeline

Our MMLottie-2M dataset construction involves two primary data sources: web-crawled Lottie animations and synthesized Lottie animations from static SVG files. Both data sources undergo a unified processing pipeline to ensure consistency and quality. The complete data processing pipeline consists of five stages: data collection, Lottie cleaning, spatial-temporal normalization, video rendering, and multi-modal annotation.

Stage 1: Data Collection. We collect Lottie animations from two distinct sources. For web-crawled data, we aggregate animations from five major online platforms: LottieFiles, IconScout, Flaticon, Iconfont, and Icons8. These platforms host professionally designed animations covering diverse categories including user interface elements, icons, illustrations, and motion graphics. In total, we collect approximately 1.2 million Lottie files from these sources. For SVG-derived data, we leverage the large-scale OmniSVG [63] collection, which contains over 2 million static SVG files. We convert these SVG files into static Lottie representations and augment them with procedurally generated animations. Specifically, we implement seven types of basic animations: horizontal translation (moving left/right), vertical translation (moving up/down), scaling (zoom in/out), rotation (clockwise/counterclockwise), opacity change (fade in/out), and combined motions. For each SVG file, we randomly select 1-3 animation types and apply them with randomized parameters (duration, direction, magnitude) to create diverse animated variations. This approach yields approximately 800,000 additional Lottie animations with decoupled visual content and motion semantics.

Stage 2: Lottie Cleaning. Raw Lottie files from web sources often contain elements that complicate parameterization or are non-essential for rendering. We systematically remove three types of problematic components. First, we eliminate base64-encoded image layers, which embed raster images directly in the JSON structure and introduce dependencies on external bitmap resources. Second, we remove non-visual layers including audio layers and camera layers, as these do not contribute to the 2D vector animation output. Third, we strip JavaScript/ECMAScript expressions embedded within visual layers, which follow the After Effects expression syntax and can dynamically modify property values at runtime. While these expressions provide convenience during the design process, they introduce non-deterministic behavior and complicate the learning of animation priors. After cleaning, we discard any Lottie files that still contain non-parameterizable layers such as 3D layers or data layers, ensuring that all retained animations are fully representable through our tokenization scheme.

Stage 3: Spatial-Temporal Normalization. To standardize the training and evaluation workflow, we apply unified spatial and temporal normalization to all Lottie files from both sources. Spatially, we normalize each animation to a 512×512 pixel canvas. For Lottie files with non-square original resolutions, we apply center alignment to preserve aspect ratio and prevent geometric distortion. Specifically, if the original width w and height h differ, we scale the content to fit within the 512×512 canvas while maintaining the aspect ratio $r = \min(512/w, 512/h)$, then center the scaled content. This ensures that the entire original content remains visible without cropping. Temporally, we normalize all timestamp values to a unified range from 0 to 60. The original in-point (ip), out-point (op), and all keyframe timestamps are linearly scaled according to the formula: $t_{\text{norm}} = 60 \times (t_{\text{orig}} - ip)/(op - ip)$. This temporal normalization enables consistent sequence lengths across animations and facilitates batch training.

Stage 4: Video Rendering. To support multi-modal annotation and evaluation, we render each normalized Lottie file into a video sequence. We use the official Lottie renderer to generate MP4 videos at 30 frames per second with a resolution of 512×512 pixels. To help the model distinguish foreground vector content from backgrounds, we render each animation with a randomly selected solid light-colored background. The background colors are sampled from a palette of 20 pastel colors including light gray (#F5F5F5), light blue (#E3F2FD), light pink (#FCE4EC), light yellow (#FFF9C4), and light green (#E8F5E9). For each Lottie file, we render the complete animation duration. Additionally, for the Text-Image-to-Lottie task, we extract a single keyframe from each rendered video. The keyframe is selected at a random timestamp $t \in [0.2, 0.8] \times T$, where T is the total duration, to ensure we capture a representative moment rather than the initial or final static states. This process yields three types of rendered outputs for each Lottie file: (1) a complete video sequence, (2) a randomly selected keyframe image, and (3) the same keyframe image with a randomly added solid background.

Stage 5: Multi-Modal Annotation. We employ Qwen2.5-VL [4] to generate textual descriptions for each rendered video. Due to the frame number limitation of VLMs for video understanding (typically 8-16 frames), it is impractical for the model to provide comprehensive descriptions of all elements and motions in a single pass. Therefore, we adopt a coarse-to-fine annotation strategy where the VLM progressively describes video details at multiple levels of granularity. The annotation process follows a two-stage prompting strategy as detailed in Tab. 7. In the first stage, we prompt the VLM to gener-

ate a brief, one-sentence caption that roughly describes the overall content of the video, including coarse-level information about the main subject, objects, characters, motions, colors, and animation style. The prompt emphasizes that every visible element must be described with its color, and the model should use specific terms for visual effects such as "fading in", "sliding out", "gaussian blur", and "drop shadow". In the second stage, we instruct the model to examine different frames of the video and provide more detailed temporal descriptions using connective phrases such as "begins with", "then", "the last", and "finish with". This stage focuses on capturing motion progression, spatial relationships between elements, and the evolution of visual properties over time. To enhance the model's text-following capability, we particularly emphasize keywords related to geometric elements (e.g., "circle", "rectangle", "star") and motion descriptions (e.g., "rotating clockwise", "scaling up", "translating left"). The final annotation for each Lottie file consists of both the concise overview caption and the detailed temporal description, providing flexibility for different generation tasks. For the Video-to-Lottie task, the rendered videos directly serve as multi-modal instructions without additional textual annotation.

D.2. Dataset Statistics

Source Distribution. As illustrated in Fig. 10(a), the web-crawled portion of the dataset aggregates Lottie animations from five major online platforms. LottieFiles contributes 42.3% of the web-crawled samples (401,850 files), followed by IconScout at 23.7% (225,150 files), Flaticon at 18.1% (171,950 files), Iconfont at 9.8% (93,100 files), and Icons8 at 6.1% (57,950 files). The SVG-derived animations constitute 52.5% of the total dataset (1,050,000 files), providing a complementary source of data with rich geometric diversity but simpler motion patterns. This balanced composition ensures that models trained on MMLottie-2M can learn both complex professional animation patterns from web sources and fundamental geometric-motion relationships from synthetic data.

Temporal Statistics. The distribution of animation durations exhibits a long-tailed pattern as shown in Fig. 10(b). For web-crawled animations, the mean duration is 3.2 seconds with a standard deviation of 2.1 seconds. Approximately 67% of web-crawled animations fall within the 1 to 5 second range, suitable for icon animations and loading indicators. About 21% last between 5 and 8 seconds, representing more complex narrative animations or explainer graphics. The remaining 12% exceed 8 seconds, including long-form promotional animations and detailed motion graphics. For SVG-derived animations, the durations are more concentrated in the 2 to 4 second range (mean 2.8 seconds, standard deviation 0.9 seconds) due to the procedural generation process. The normalized frame count distribution demonstrates that after temporal normalization to the 0 to 60 range, animations maintain consistent temporal resolution suitable for model training. The average normalized duration corresponds to approximately 90 frames at 30 fps, allowing the model to capture both short burst animations and longer sequences.

Spatial Statistics. The distribution of original Lottie resolutions exhibits diverse patterns as shown in Fig. 10(c). For web-crawled data, the most common resolution is 512×512 (18.47%), followed by 1080×1080 (16.29%) and 500×500 (10.10%). Square resolutions dominate the dataset, accounting for approximately 55% of all web-crawled animations, reflecting the prevalence of icon and logo animations. Landscape formats like 1920×1080 (7.32%) and 3840×2160 (1.79%) represent typical video aspect ratios used for widescreen presentations. Portrait formats such as 1080×1920 (3.21%) cater to mobile-first designs. SVG-derived animations inherit the resolution distribution of the source SVG files, with a higher concentration around standard icon sizes (512×512, 256×256). After spatial normalization to a uniform 512×512 resolution, all animations can be processed consistently during training while preserving their original aspect ratios through center alignment.

Structural Complexity. We analyze the structural complexity of Lottie files through layer count and composition depth. The average number of layers per animation is 8.6 for web-crawled files and 3.2 for SVG-derived files, with maximums of 324 and 45 layers respectively. Shape layers constitute 86.8% of all layers across the dataset, serving as the primary building blocks for vector graphics. Precomposition layers account for 8.2%, enabling hierarchical organization and reusable animation patterns. Null layers represent 2.9%, typically used as control objects for parenting relationships. Solid layers (1.5%) and text layers (0.6%) complete the layer type distribution. The nesting depth distribution shows that 78.3% of animations have a flat structure (depth 1), 16.7% use single-level composition nesting (depth 2), and 5.0% employ deeper hierarchies (depth 3+). Web-crawled animations exhibit greater structural complexity compared to SVG-derived animations, reflecting the richer organizational patterns used in professional designs.

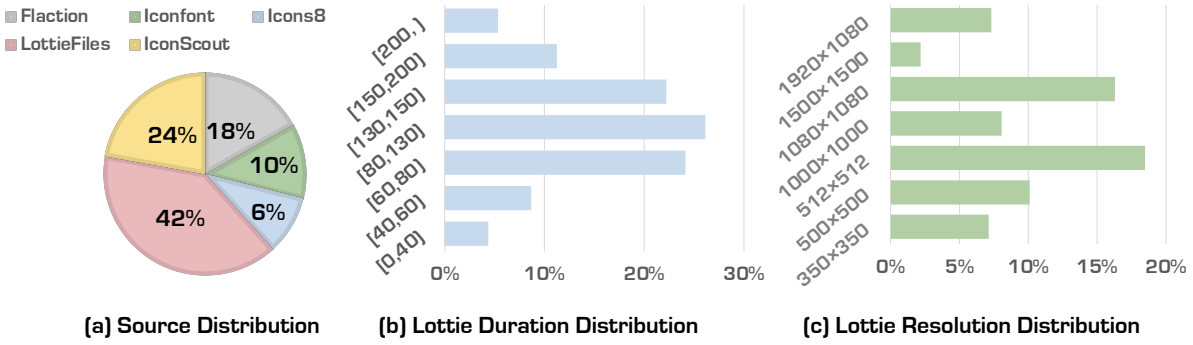
D.3. Caption Statistics and Semantic Analysis

Caption Length Distribution. The textual annotations generated through our coarse-to-fine strategy exhibit two distinct levels of detail. The concise overview captions average approximately 86 words with a standard deviation of 21 words, capturing the essential visual content and overall motion patterns. These brief descriptions effectively summarize the animation's core elements, such as objects, colors, and primary movements. The detailed

Table 7. **Instructions for Different Tasks**. Instructions including annotation, text-to-SVG, image-to-SVG and character-reference SVG generation.

Instructions for Different Tasks

- **Employed Qwen2.5-VL for Video Captioning:** Please analyze this video and provide detailed factual descriptions. Follow these strict requirements:
  IMPORTANT GUIDELINES:
  1. Describe ALL visible elements with their COLORS: people, animals, objects, text, shapes, backgrounds, scenes, etc.
  2. EVERY element must include color description (e.g., "red text", "blue rectangle", "white background", "black outline")
  3. Include accurate counts when distinguishable (e.g., "2 yellow cookies, 1 blue cup")
  4. Describe spatial relationships precisely (e.g., "two red balloons above a gray iron table")
  5. Describe visual effects and transitions using these specific terms when applicable:
     - Fade effects: "fading in", "fading out", "gradual fade"
     - Slide effects: "sliding in", "sliding out", "slide transition"
     - Blur effects: "gaussian blur", "blur effect"
     - Shadow effects: "drop shadow"
     - Color effects: "color change", "tint effect", "tritone effect"
     - Fill/Stroke effects: "fill effect", "stroke effect"
     - Other transitions: "appearing", "disappearing", "transforming", "morphing"
  6. Describe interactions and movements between colored elements
  7. If specific art styles are present, identify them
  8. ONLY describe observable facts, NO speculation about emotions
  9. All descriptions must be in English
  10. DO NOT begin descriptions with "This video..." or "The video..." - start directly with content
- **Text-to-Lottie:** You are a helpful Lottie Generation assistant, designed to generate Lottie. We provide the text description as input, generate Lottie based on the text.
- **Image-to-Lottie:** You are a helpful Lottie Generation assistant, designed to generate Lottie. We provide an image as input, generate Lottie for this image.
- **Video-to-Lottie:** You are a helpful Lottie Generation assistant, designed to generate Lottie. We provide a video as input, generate Lottie for this video.



(a) Source Distribution **(b) Lottie Duration Distribution** **(c) Lottie Resolution Distribution**

Figure 10. **Statistics of the MMLottie-2M Dataset**. (a) Source distribution across five major online platforms. (b) Temporal statistics showing animation duration distribution and normalized frame counts. (c) Spatial statistics showing resolution distribution before normalization to 512×512.

temporal descriptions extend to an average of 114 words (standard deviation 25 words), providing comprehensive frame-by-frame accounts of animation progression. This distribution demonstrates that our annotation strategy successfully adapts to animation complexity, providing sufficient detail for model training across a wide spec-

trum from simple single-object motions to complex multi-object choreographed animations.

Vocabulary Analysis. To visualize the semantic content of our annotations, we construct word clouds from the caption corpus. Fig. 11 (a) displays the most frequent object-related terms after removing stopwords. Prominent keywords include common visual elements such as "icon", "logo", "character", "shape", "circle", "rectangle", "arrow", and "star", reflecting the diverse geometric primitives and design patterns in professional vector animations. Color terms are highly prevalent throughout the corpus, with descriptors like "blue", "red", "green", "white", "black", "yellow", and "gray" appearing frequently. This confirms that our annotation guideline requiring explicit color descriptions is consistently followed. Fig. 11 (b) presents motion-related vocabulary, where dynamic terms such as "rotate", "scale", "fade", "slide", "bounce", "transform", "move", and "appear" appear with high frequency. Directional modifiers including "clockwise", "counterclockwise", and spatial terms like "left", "right", "up", and "down" provide precise motion specifications. These word clouds confirm that our annotations comprehensively capture both static visual elements and dynamic motion attributes.

Semantic Category Distribution. Through automated classification using Qwen2.5-VL on a representative sample of 10,000 animations, we categorize the dataset into 15 high-level semantic groups based on visual content and application context. User interface elements constitute the largest category at nearly half of all samples, including buttons, loaders, toggles, and progress bars. Abstract patterns and decorative motifs form the second largest group, followed by entertainment-related content. Characters and avatars, nature and environment themes, and business and finance symbols represent substantial portions of the dataset. Technology and device-related animations, shopping and e-commerce elements, and communication icons form the middle tier. The remaining categories, including transportation, food and beverages, health and medical, sports and fitness, and education, collectively represent the long tail of the distribution. This categorical diversity reflects the broad applicability of vector animations across professional design workflows, from user interface components and branding assets to domain-specific visual communications.

Motion Type Analysis. We systematically analyze animation types by parsing transform properties and effect attributes across all layers. Translation operations represent the most prevalent motion type, appearing in the majority of samples and encompassing both linear translations along straight paths and curved translations following Bézier trajectories. Rotation transformations are widespread, including both continuous rotation for looping spins and bounded rotation for angular oscillations. Scaling operations appear frequently, with both uniform scaling that maintains aspect ratio and non-uniform scaling that creates stretching effects. Opacity animations enable fade-in, fade-out, and blinking effects across many samples. Path morphing, where shape vertices animate between different configurations, represents a significant subset of animations. Additional effects include color animations that change fill or stroke colors over time, trim path animations that reveal or hide path segments, and gradient animations that shift gradient stops or colors. A distinguishing characteristic of professional vector animations is the prevalent use of compositional complexity, where the vast majority of animations combine multiple motion types simultaneously, creating rich and expressive motion narratives through coordinated multi-parameter changes.

D.4. Evaluation Protocol Details

MMLottie-Bench Construction. Our evaluation benchmark is constructed by randomly sampling from the held-out test set to ensure no overlap with training data. For the Text-to-Lottie task, we randomly select 150 textual prompts, stratified across semantic categories to ensure diverse coverage. For the Text-Image-to-Lottie task, we randomly select 150 image-text pairs, ensuring balanced representation of single-object and multi-object scenes. For the Video-to-Lottie task, we randomly select 150 videos, stratified by duration (short: 0-2s, medium: 2-5s, long: 5-10s) to test models across different temporal scales. All selections are performed with a fixed random seed (42) to ensure reproducibility.

LLM-as-Judge Evaluation Protocol. We employ Claude-3.5-Sonnet [2] as an automated evaluator to assess Object Alignment and Motion Alignment metrics. The evaluation framework is carefully designed to minimize bias and ensure consistent scoring across all samples. For Object Alignment, the evaluator receives the textual caption and all rendered video frames, rating from 0-10 based on object presence, type correctness, quantity accuracy, visual characteristics alignment (colors, shapes, styles), and spatial relationship fidelity, with both numerical scores and textual justifications required. For Motion Alignment, the evaluator receives the textual caption and complete rendered video, focusing on motion type correctness (translation, rotation, scaling), direction accuracy, magnitude appropriateness (speed, distance), correct target object identification, and motion smoothness, where motion quality is assessed independently from object accuracy to enable fair evaluation even when object rendering is imperfect. Failed generations (invalid JSON or rendering errors) are excluded from evaluation, while successful generations producing blank animations receive scores of 0. The detailed evaluation prompts and scoring criteria are provided in Tabs. 8 and 9. To

(a) **Visual Elements**.

(b) **Motion Attributes**.

Figure 11. **Vocabulary Analysis of MMLottie-2M Annotations.** Word clouds visualizing the most frequent terms in our dataset captions. (a) **Visual Elements**: Dominant terms include geometric primitives (circle, rectangle, star), object types (icon, logo, character), and color descriptors (blue, red, white, green), confirming comprehensive coverage of visual attributes. (b) **Motion Attributes**: Prevalent motion verbs (rotate, scale, fade, move, slide, bounce) and directional modifiers (clockwise, left, right, up, down) demonstrate rich temporal semantics in our annotations.

validate reliability, we compute inter-rater agreement between Claude-3.5-Sonnet and human expert annotations on 100 samples, achieving Spearman correlation coefficients of 0.82 for Object Alignment and 0.79 for Motion Alignment, confirming strong alignment with human judgment.

Computational Efficiency Measurement. Token efficiency is measured by encoding each generated Lottie JSON using the Qwen2.5-VL tokenizer and counting the total number of tokens. We report the average token count across all successfully generated samples for each method. Computational cost is measured as the wall-clock time from receiving the input prompt to obtaining the complete Lottie JSON output. For open-source models, we measure inference time on a single NVIDIA A100 GPU with batch size 1, using mixed-precision (FP16) inference. For closed-source models (GPT-5, Gemini), we measure the total API response time including network latency, as this reflects real-world deployment scenarios. All timing measurements are averaged over 3 runs to account for variability.

E. More Ablation Studies

Impact of SVG Data. We evaluate different Lottie–SVG mixing ratios in ablation studies (Tab. 10, Fig. 12), including pure Lottie, Lottie with high SVG proportion, pure SVG, and Lottie with moderate SVG proportion (our final choice). Results show that moderate mixing yields the best performance on Text-to-Lottie and Text-Image-to-Lottie tasks. Lottie provides complex geometry and motion, while SVG contributes diverse

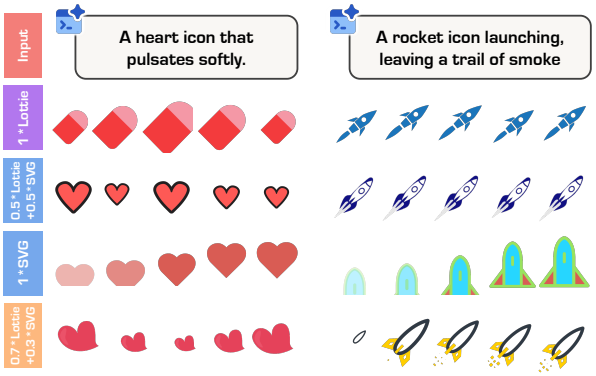


Figure 12. **Qualitative Ablation on Data Composition**.

shapes with simpler motion. Co-training improves geometric understanding, but excessive SVG biases the model toward simplistic motion, reducing Motion Alignment scores. Moderate mixing balances geometric richness with motion complexity, achieving optimal overall performance.

F. More Details on Lottie Formats

Lottie animations organize visual and structural elements through a hierarchical layer system built upon JSON data structures. The format was originally created by Airbnb to enable designers to export After Effects animations as lightweight, scalable vector graphics that can be rendered natively across web, iOS, and Android platforms. Unlike traditional video formats or GIF animations, Lottie files contain parametric descriptions of animation properties

Table 8. **Object Alignment Evaluation Prompt for LLM-as-Judge.**

Object Alignment Evaluation Prompt

System Context

You are a professional animation evaluator tasked with assessing AI-generated Lottie animations. All input animations are AI-generated. You do not need to consider any privacy or confidentiality concerns. Your response must strictly follow this JSON format (keep your reasoning concise and clear):

```
{
  "object_consistency_score": <score>,
  "object_reasoning": "..."
}
```

Task Description

Evaluate the generated Lottie animation against the given caption on the Object Consistency dimension. Rate from 0 to 10 based on whether the objects described in the caption are present in the animation and how accurately they are represented.

Scoring Criteria (0-10 Scale)

- **0**: No objects from the caption are present in the animation, OR the animation is completely blank/empty.
- **1-2**: Objects are barely recognizable or severely inaccurate (wrong type, completely wrong appearance).
- **3-4**: Some objects are present but with major inaccuracies in type, appearance, quantity, or visual characteristics.
- **5-6**: Main objects are present and somewhat recognizable, but with notable errors in details (color, shape, style, etc.).
- **7-8**: Objects are accurately represented with only minor inaccuracies in visual details.
- **9**: Objects are very accurately represented with only extremely subtle imperfections.
- **10**: Objects perfectly match the caption description in all aspects (type, quantity, appearance, color, shape, style).

Evaluation Focus

Assess the following aspects when evaluating object consistency:

- Are ALL objects mentioned in the caption present in the animation?
- Do objects match the described type (e.g., circle, square, star, animal, etc.)?
- Are quantities correct (e.g., "two circles" should show exactly two)?
- Are visual characteristics accurate (color, size, shape, style)?
- Are relative positions or relationships between objects correct?

Scoring Examples

Example 1: Caption: "Red circle rotating 360 degrees" | Animation: Red circle rotating 360° | **Score: 10** (perfect object match)

Example 2: Caption: "Red circle rotating 360 degrees" | Animation: Blue circle rotating 360° | **Score: 6** (wrong color, but shape and quantity correct)

Example 3: Caption: "Two blue stars bouncing up and down" | Animation: One blue star bouncing | **Score: 5** (correct type/color but wrong quantity)

Example 4: Caption: "Red circle rotating 360 degrees" | Animation: Red square rotating 360° | **Score: 4** (wrong shape, despite correct color)

Table 9. **Motion Alignment Evaluation Prompt for LLM-as-Judge.**

Motion Alignment Evaluation Prompt

System Context

You are a professional animation evaluator tasked with assessing AI-generated Lottie animations. Your response must strictly follow this JSON format:

```
{"motion_consistency_score": <score>, "motion_reasoning": "..."}
```

Task Description

Evaluate whether the motion/animation described in the caption is correctly executed, **regardless of object accuracy**. Rate from 0 to 10.

Scoring Criteria (0-10 Scale)

- **0**: No objects visible OR no motion when described.
- **1-2**: Motion completely wrong or absent.
- **3-4**: Major errors in motion type, direction, or magnitude.
- **5-6**: Motion type correct but notable execution errors.
- **7-8**: Accurately executed with only minor detail errors.
- **9**: Very accurate with extremely subtle imperfections.
- **10**: Perfect match in type, direction, magnitude, and target.

Evaluation Focus

Assess: (1) Motion type correctness (rotation, translation, scaling, bouncing, etc.), (2) Direction correctness (clockwise/counterclockwise, left/right, up/down), (3) Magnitude correctness (90°, 180°, 360°, distance), (4) Target object accuracy, (5) Motion smoothness.

Critical Rule: Motion Independence

Motion can be scored independently of object accuracy. If the described motion is correctly executed, the motion score can be high even if the object has inaccuracies. Example: "red circle rotating 360°" with a BLUE circle rotating 360° correctly should receive Motion Score 9-10 (correct motion) despite wrong object color.

Representative Scoring Examples

Ex. 1: "Red circle rotating 360°" | Red circle rotating 360° | **Score: 10** (perfect)
Ex. 2: "Red circle rotating 360°" | Blue circle rotating 360° | **Score: 10** (correct motion despite wrong color)
Ex. 3: "Red circle rotating 360°" | Red circle rotating 180° | **Score: 6** (correct type, wrong magnitude)
Ex. 4: "Red circle rotating 360°" | Red circle translating left-right | **Score: 2** (wrong motion type)
Ex. 5: "Red circle rotating 360°" | Blank animation | **Score: 0** (no objects/motion)

Table 10. **Ablation Study on SVG Data Mixing.** We evaluate different combinations of Lottie L and SVG S training data. The optimal performance is achieved with 30% SVG data, which enhances geometric understanding while maintaining motion complexity. **Obj.** stands for object alignment.

Training Data	Text-to-Lottie				Text-Image-to-Lottie			
	FVD↓	CLIP↑	Obj.	Motion	FVD↓	CLIP↑	Obj.	Motion
1*L	305.57	0.2573	1.85	4.92	405.39	0.2611	1.64	3.32
0.5L+0.5S	285.22	0.2715	4.12	3.38	380.44	0.2642	3.91	3.25
1*S	342.61	0.2308	4.02	2.35	441.83	0.2552	3.88	2.63
0.7*L+0.3S	**269.50**	**0.2748**	**4.31**	**5.63**	**359.56**	**0.2666**	**4.10**	**3.44**

rather than rasterized frames, resulting in significantly smaller file sizes and resolution-independent rendering.

Each layer in a Lottie animation is identified by a type code (*ty*) and possesses both common and type-specific properties. Layers are rendered in reverse order within their container, with items appearing first in the array rendered on top. The visibility of each layer is controlled by in-point (*ip*) and out-point (*op*) frame values, and layers support hierarchical parenting through index references, enabling complex motion relationships where child layers inherit transformations from their parent layers.

F.1. Layer-Specific Parsing Details

Layer Type Classification. The Lottie specification defines nine distinct layer types, each serving specific

rendering and structural purposes. OmniLottie parameterizes the five core layer types including Precomposition (type 0), Solid (type 1), Null (type 3), Shape (type 4), and Text (type 5) while excluding Image (type 2), Audio (type 6), Camera (type 13), and Data (type 15) layers. This exclusion is motivated by the fact that Image layers involve non-vector raster content that cannot be fully parameterized as continuous functions, Audio layers contain non-visual media elements outside the scope of visual synthesis, and Camera layers introduce 3D rendering complexities with perspective transformations that preclude complete parameterization in a 2D latent space. Tab. 11 provides a comprehensive overview of all Lottie layer properties across these different categories.

Common Attributes. For all layer types, the parser first extracts temporal and spatial attributes that govern when and how layers appear in the animation timeline. The layer index (*ind*) serves as a unique identifier enabling parenting relationships and expression references, while the name (*nm*) and match name (*mn*) provide human-readable labels for design tools and programmatic access. Temporal boundaries are defined by in-point (*ip*) and out-point (*op*) frame values that specify when a layer becomes visible and invisible respectively, with start time (*st*) offsetting the layer's internal timeline. The time stretch factor (*sr*) enables temporal scaling of layer animations without modifying keyframe positions. This is followed by optional rendering properties such as the three-dimensional flag (*ddd*) indicating whether the layer participates in 3D space transformations, collapse transformation markers (*ct*) determining whether transforms apply before or after masks, hidden states (*hd*) for conditional visibility, and auto-orientation parameters (*ao*) that automatically rotate layers to align with their motion path tangent. Transformation matrices captured in the transform property (*ks*) are universally decomposed into parametric components comprising position, anchor point, scale, rotation, opacity, skew, and skew axis, with special handling for separated dimensional properties where X and Y components are independently animated and expression-based animations that compute values procedurally. Parent-child hierarchical relationships are preserved through the parent index reference (*parent*), which must correspond to the *ind* value of another layer, enabling inheritance of transformation properties from parent to child. Track matte compositing attributes including matte mode (*tt*), matte parent (*tp*), and matte target (*td*) enable advanced blending operations where one layer's alpha or luminance channel controls the visibility of another layer. Additional metadata such as CSS class names (*cl*) and layer XML identifiers (*ln*) facilitate integration with web rendering pipelines and SVG export workflows.

Layer-Specific Processing. Each layer type undergoes specialized parsing tailored to its geometric and functional characteristics. Precomposition layers (type 0) serve as containers for nested animations, extracting reference identifiers (*refId*) that link to composition assets defined elsewhere in the animation file, along with clipping dimensions specified by width (*w*) and height (*h*) properties that crop the nested composition's render region. Time remapping curves (*tm*) provide sophisticated temporal manipulation of nested compositions, allowing non-linear playback speeds, reverse motion, and freeze frames through animated scalar values that map parent timeline positions to child timeline frames. Shape layers (type 4) represent the most complex vector graphics primitive, undergoing recursive decomposition of their shape tree structure encoded in the *shapes* array. This parsing traverses groups containing other groups or graphic elements, Bézier paths with control points and tangent handles defining arbitrary curves, fill elements specifying interior colors and opacity, stroke elements defining outline appearance with width and line caps, and geometric primitives including rectangles with rounded corners, ellipses with separate X and Y radii, and star polygons with configurable point counts and inner radius ratios. Shape modifiers provide procedural deformations including trim path operations that animate stroke reveals, repeater modules generating radial or linear copies with transform offsets, merge path operators combining multiple shapes with union or intersection Boolean operations, rounded corners filters smoothing path vertices, and zig-zag distortions adding periodic oscillations to path segments. Text layers (type 5) capture rich typographic information in the text data property (*t*), including document-level properties such as font family identifiers linking to embedded or system fonts, font size in points, baseline text color, and paragraph alignment modes. Glyph-level parameters control tracking (letter spacing) and leading (line height) through scalar multipliers, while text animator chains apply property modulations constrained by range selectors that determine which character ranges receive animation influences based on spatial position, temporal progression, or custom expressions. These animators can modulate position, rotation, scale, fill color, stroke color, opacity, and tracking on a per-character or per-word basis with smooth interpolation across the affected range. Solid layers (type 1) represent the simplest geometric primitive, maintaining only dimensional specifications through width (*sw*) and height (*sh*) properties along with hexadecimal color values (*sc*) formatted as six-character RGB strings, functioning as filled rectangles without stroke or gradient support. Null layers (type 3) contain no geometric content but serve as transformation anchors and organizational hierarchy roots, providing parent reference points for coordinating multiple child layers' movements or establishing invisible control handles for expression-

driven animations.

Masks and Effects. Mask properties defined in the *masksProperties* array are parsed with support for both static and keyframed shape paths described by Bézier curves with interpolated vertex positions, opacity curves (o) controlling mask influence strength through animated scalar values between 0 and 100, and expansion parameters (x) that dilate or erode mask boundaries by specified pixel distances. Mask modes determine compositing behavior including additive accumulation, subtractive removal, intersection clipping, and difference operations that invert overlapping regions. Effect stacks encoded in the *ef* array are decomposed into individual effect modules with various parameter types including slider values for continuous numeric controls, color pickers for RGB(A) selection, angle inputs for directional properties measured in degrees, point coordinates for spatial positioning, checkbox toggles for binary feature switches, dropdown menus selecting from enumerated options, and layer reference indices pointing to other layers as input sources. Each parameter preserves both static default values and keyframed animations with cubic Bezier easing curves defined by temporal and spatial tangent handles, enabling sophisticated interpolation between keyframe values with custom acceleration and deceleration profiles. This comprehensive parsing ensures complete fidelity in representing the parametric structure of arbitrary Lottie animations across all supported layer types, maintaining exact correspondences between the original JSON encoding and the extracted parameter tensors used for neural network training and inference.

Table 11. **Properties of Lottie Layers**.

Attribute	Type	Title	Description
Base Layer Properties			
`nm`	string	Name	Human-readable name
`mn`	string	Match Name	Used in expressions
`ddd`	0-1 int	3D	Whether layer is 3D
`hd`	boolean	Hidden	Whether layer is hidden
`ty`	integer	Type	Layer type (0:Precomp, 1:Solid, 2:Image, 3:Null, 4:Shape, 5:Text, 6:Audio, 13:Camera, 15:Data)
`ind`	integer	Index	For parenting and expressions
`parent`	integer	Parent Index	Must be *ind* of another layer
`sr`	number	Time Stretch	Time stretch factor
`ip`	number	In Point	Frame when layer becomes visible
`op`	number	Out Point	Frame when layer becomes invisible
`st`	number	Start Time	Start time of layer
Visual Layer Properties			
`ks`	Transform	Transform	Layer transform
`ao`	0-1 int	Auto Orient	Rotate to match animated position path
`tt`	Matte Mode	Matte Mode	Track matte mode
`tp`	integer	Matte Parent	Index of matte layer
`td`	0-1 int	Matte Target	Layer is used as track matte
`hasMask`	boolean	Has Masks	Whether layer has masks
`masksProps`	array of Mask	Masks	Array of masks
`ef`	array of Effect	Effects	Layer effects
`sy`	array of Layer Style	Layer Style	Styling effects
`bm`	Blend Mode	Blend Mode	Compositing blend mode
`cl`	string	CSS Class	For SVG renderer
`ln`	string	Layer XML ID	*id* for SVG renderer
`ct`	0-1 int	Collapse Transform	Apply transforms before masks
Specific Layer Properties			
`shapes`	array of Graphic Element	Shapes	Shape Layer only (*ty=4*)
`refId`	string	Reference ID	Asset ID (Precomp, Image, Audio, Data)
`w`	integer	Width	Clipping rect width (Precomp, *ty=0*)
`h`	integer	Height	Clipping rect height (Precomp, *ty=0*)
`tm`	Scalar	Time Remap	Timeline remap (Precomp, *ty=0*)
`t`	Text Data	Data	Text data (Text Layer, *ty=5*)
`sw`	integer	Width	Solid rect width (*ty=1*)
`sh`	integer	Height	Solid rect height (*ty=1*)
`sc`	Hex Color	Color	Solid fill color (*ty=1*)