SVGThinker: Instruction-Aligned and Reasoning-Driven Text-to-SVG Generation

Yongtae Lee

2025. 02. 27

INFONET Journal Club Presentation

Paper

SVGThinker: Instruction-Aligned and Reasoning-Driven Text-to-SVG Generation

Hanqi Chen
Shanghai Jiao Tong University
SJTU Paris Elite Institute of Technology
Shanghai, China
caffinities@sjtu.edu.cn

Zhongyin Zhao
Shanghai Jiao Tong University
Shanghai, China
zhao_zhongyin@sjtu.edu.cn

Ye Chen
Shanghai Jiao Tong University
Shanghai, China
chenye123@sjtu.edu.cn

Zhujin Liang
PhiGent Robotics
Beijing, China
zhujin.liang@phigent.ai

Bingbing Ni*
Shanghai Jiao Tong University
Shanghai, China
nibingbing@sjtu.edu.cn

- Published in
  - 33rd ACM International Conference on Multimedia (MM '25)
- Date of Publication
  - Oct, 2025.
- Latest Update
  - https://dl.acm.org/doi/10.1145/3746027.3755392

Introduction

- Text-to-SVG task
  - Scalable Vector Graphics(SVG) – an image format widely adopted in modern graphic design
    - Vector image: represents graphics through geometric entities rather than pixel grids
  - Merits
    - Resolution-independent and structurally editable
    - Represented as hierarchical XML-based code
    - Widely used in design, UI, icons, and industrial graphic
  - Challenges
    - Hard to understand its syntax or master its creation methods
    - The visual representation and semantic information of the code are complex

Introduction

- Existing Approaches & Limitations
  - Existing paradigms in Text-to-SVG generation
    - Optimization-based method (generate images first, then optimize paths):

      - Produce redundant and chaotic path structures

      - Lacks structural and semantic interpretability

      - Poor editability despite visual similarity

    - Auto-regressive approach (use LLM to directly generate code representing SVG):

      - Sometimes restrict primitive vocabulary

      - Fail to model hierarchical SVG structure

      - Weak alignment between text instruction and primitive-level construction

Introduction

- Motivation of Paper
  - Observation
    - Existing methods consider SVG generation as image approximation or direct code prediction, ignoring the reasoning process behind visual construction.
  - Core Idea of the Paper
    - Text-to-SVG generation should be modeled as a step-by-step reasoning process aligned with how humans construct graphics.
      - Learn how to construct the graphic
      - Align semantic description with structural generation
      - Generate SVG through explicit reasoning before writing code

Introduction

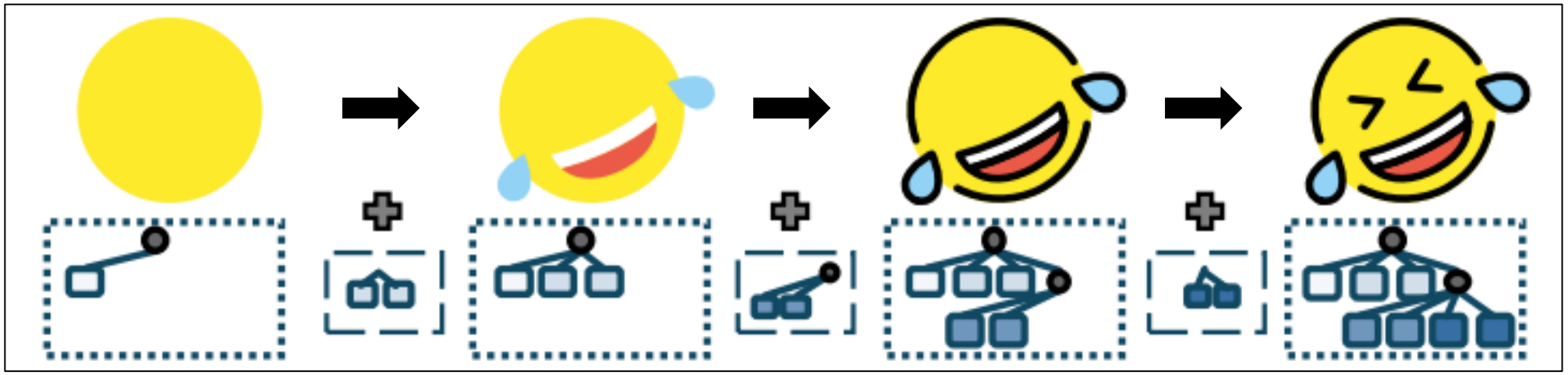
- Contributions of the Paper
  - SVGThinker
    - An LLM-based text-to-SVG generation model
    - Annotation pipeline for hierarchical reconstruction and explicit alignment of SVG primitive and its visual-semantic meaning
    - Integration of CoT reasoning and logical construction
    - Fine-grained, prompt-based editing capability (e.g., position, color, elements, structure)
    - Outperforms prior SOTA models in text-SVG alignment, coherence, and editability

Methodology

- Data Collection and Curation
  - 400k Vector Graphic data
  - Source: Kaggle + SVGRepo(free icon)
  - Semantic matching accuracy of these data was manually calibrated to ensure comprehensive semantic coverage
  - Used SVGO(a Nodejs library) to optimize SVGs
  - Excessively long samples were removed

Methodology

- SVG Structure Reconstruction
  - Parse SVG into structured primitives
  - Render SVG incrementally, adding one primitive at a time
    - Each intermediate image reflects visual change caused by one primitive addition



Methodology

- Sequential Annotation
  - Challenges in SVG annotation
    - Difficult correspondence between SVG code and visual elements
    - Sequential rendering induces layering effects
    - Increased complexity for language models
  - Limitations of Previous Methods
    - Based on a single rendered image
    - Inconsistent descriptions (detailed localization vs vague summaries)
    - Reduced editability and generation stability

Methodology

- Sequential Annotation Strategy

1. Global Reference
    - Generate a comprehensive description from fully rendered SVG

2. Stepwise Image Sequence
    - Construct image sequence from first instruction to final rendering

3. Difference-Based Annotation
    - At each step, the model compares the current image with the previous one.
    - Descibes only the newly added/modified elements.
    - Use both the full description and the final image as references

Methodology

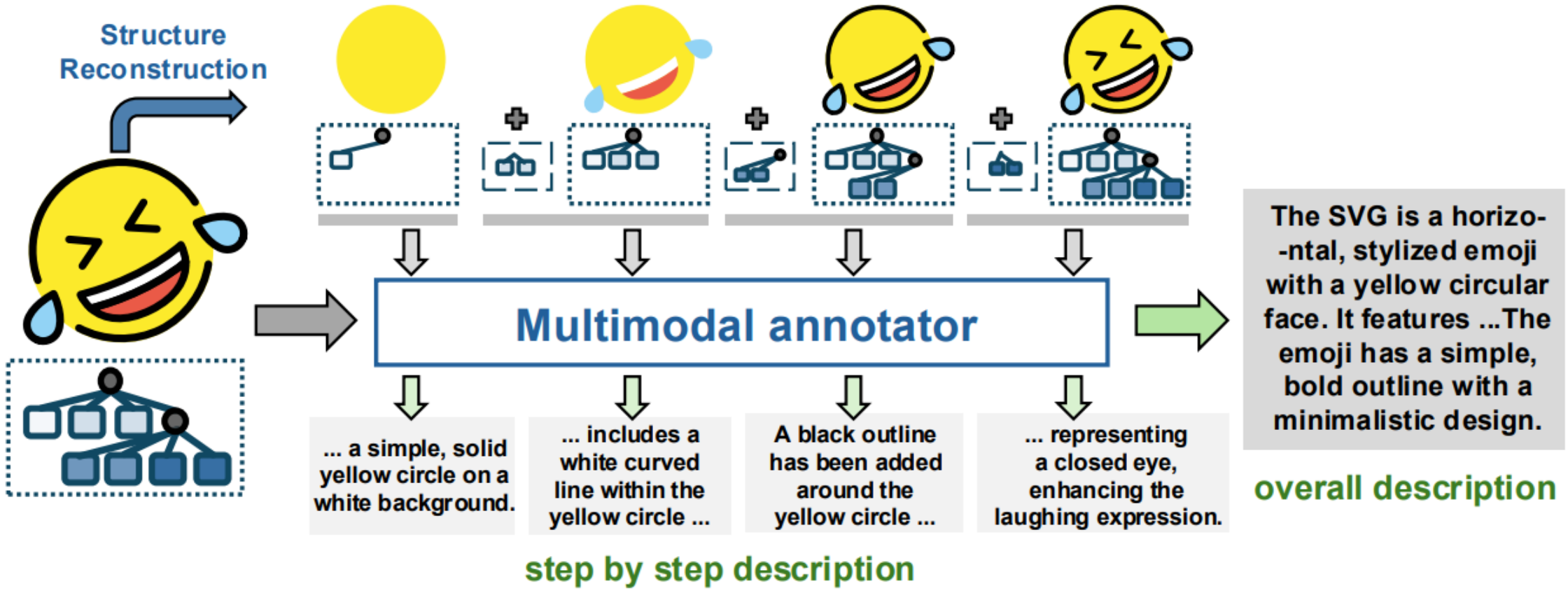


Figure 2: The process of sequential textual annotation for SVG. First, we perform an overall description of the SVG. Then, by parsing and reconstructing the SVG structure, we obtain a set of textual descriptions related to the sequence of SVG primitives, based on the visual changes at each step.

Methodology

- Training of LLM for Reasoning-Driven SVG Generation
  - Goal: Align the LLM's reasoning process with layer-by-layer drawing process
  - Instead of directly generating SVG code:

    1. the model first formulate structured drawing steps,

    2. then generates the final SVG instructions.

  - Data Structure
    - Assume one SVG is composed of a sequence of primitive instructions (s_1, s_2, … s_n)
    - There are complete textual description t_g and sequence of intermediate descriptions (t_1, t_2, … t_n) from annotation pipeline

Methodology

- Training of LLM for Reasoning-Driven SVG Generation
  - Step 1. Reasoning Sequence Modeling
    - The model predicts intermediate reasoning steps conditioned on t_g

$$\mathbb{P}((t_1, t_2, \ldots, t_{n-1}) \mid t_g) = \prod_{i=1}^{n-1} \mathbb{P}(t_i \mid (t_j)_{\forall j<i}, t_g) \quad (2)$$

  - Step 2. Modeling SVG Instruction Sequence
    - Conditioned on the t_1~t_n, the model learns to generate the corresponding SVG instruction sequence

$$\mathbb{P}((s_1, s_2, \ldots, s_n) \mid (t_1, t_2, \ldots, t_{n-1}), t_g) = \prod_{i=1}^{n} \mathbb{P}\left(t_i \mid (s_j)_{\forall j<i}, (t_1, t_2, \ldots, t_{n-1}), t_g\right) \quad (3)$$

  - Training follows the autoregressive negative log-likelihood loss function used in language models
    - Applied over: Reasoning sequence generation and SVG instruction generation

Experiments

- SOTA Comparison Experiments
  - Goal
    - Performance comparison with baseline models
    - Highlights the proposed model's precise instruction editability and compare it with SOTA methods
  - SVGThinker Setup
    - Backbone architecture: Qwen 2.5-7B

      + Initialized with DeepSeek CoT distillation weights

Experiments

- Experimental Setup
  - Dataset
    - 270,436 SVG-text pairs constructed from sequential annotation
    - 1000 randomly selected samples for testing (no training overlap)
  - Baselines
    - Closed-source LLMs: GPT-4o, DeepSeek-R1 (API-based evaluation)
    - Open-source SVG generation methods: SVGDreamer, IconShop, LayerTracer
  - Evaluation Metrics
    - FID: distribution similarity (↓)
    - CLIP score: text-image alignment (↑)
    - FID-CLIP: CLIP-based distribution metric (↓)

Experiments

- Quantitative Comparison

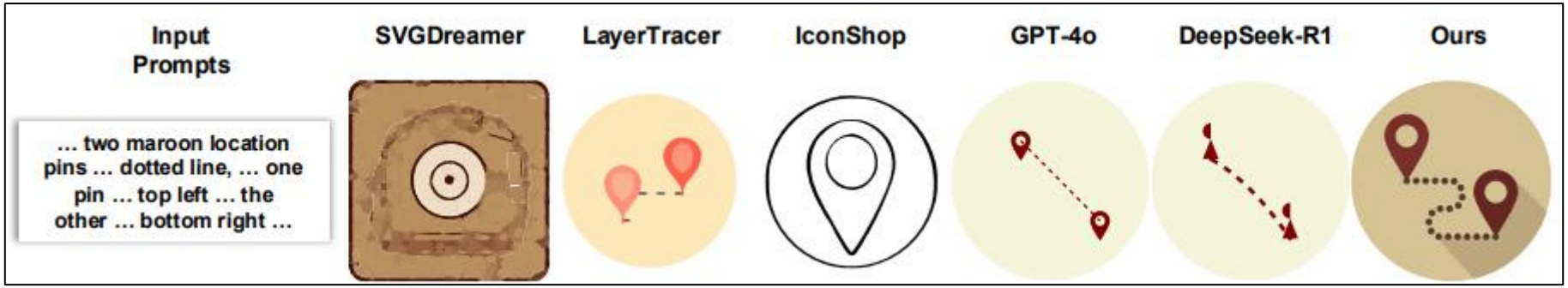
Table 1: Quantitative comparison with SOTA models. We also compared the file size, the number of primitives used, and the variety of supported primitives. Our method supports all primitive types, generating higher-quality, more efficient, and compact SVGs with a more concise use of primitives.

Methods	FID↓	CLIP Score↑	FID-CLIP↓	Primitives Support	File Size (KB)	Primitives Used
LayerTracer [25]	54.75	0.2290	30.46	path	16.25	17.83
SVGDreamer [34]	240.87	0.1923	150.34	path	282.13	513.0
IconShop [32]	89.24	0.2672	53.79	path	3.14	1.042
GPT-4o-2024-11-20 [1]	62.56	0.1715	43.93	all	0.67	5.62
DeepSeek-R1 [8]	153.04	0.1160	111.42	all	0.71	5.30
SVGThinker	**34.06**	**0.2765**	**21.08**	all	1.16	3.707

- SVGThinker outperforms all baselines across all 3 metrics
- Demonstrates superior distributional similarity and text-image alignment

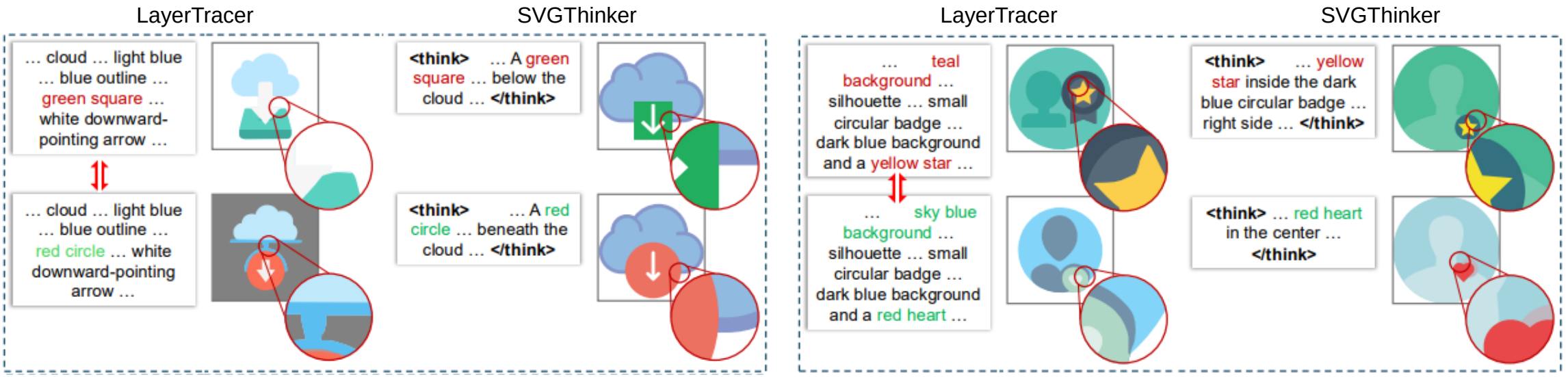
Experiments

- Qualitative Comparison
  - Image-based methods
    - Overuse of complex path, Redundant primitives
  - Commercial LLMs
    - Results are often lacking visual coordination and fail to align with generated instructions
  - SVGThinker
    - Clean and well-structured
    - Visually smooth and coherent output



Experiments

- Precise Prompt Editing
  - Controllable Editing Capability
    - Small modification in text → precise structural update
    - SVGThinker can precisely edit generated result based on prompt
    - Other models encounter difficulties during re-generating the whole image



Experiments

- Ablation on Method
  - Same architecture, different initialization

    Qwen + DeepSeek CoT distillation vs original Qwen(directly trained)

Table 2: Component Analysis. We compared our method with the directly trained method. Our method significantly improved the performance compared to both the original weights and the directly trained method.

Methods	FID↓	CLIP↑	FID-CLIP↓
Qwen2.5-7B (zeroshot)	81.62	0.2305	65.42
Qwen2.5-7B (trained)	41.57	0.2345	25.92
R1-7B (zeroshot)	124.87	0.2027	99.26
SVGThinker	34.06	0.2765	21.08

Experiments

- User Study
  - Setup
    - 67 participants compared the results generated by SVGThinker and SOTA baselines
    - Evaluation criteria: Usability, Aesthetic quality, and Instruction alignment
    - Ranking-based evaluation

Table 4: User study on models comparison.

Methods	Usability↑	Appearance↑	Alignment↑
SVGThinker	3.58	3.33	3.78
SVGDreamer	1.25	2.69	2.58
IconShop	3.54	3.06	3.18
LayerTracer	2.91	3.85	3.55
GPT-4o	3.76	2.16	1.91

    - Best instruction alignment, Second-best Usability and Appearance

Conclusions

- SVGThinker: A Reasoning-Driven SVG Generation Framework
  - Bridges textual instructions and visual output
  - Sequential and hierarchical SVG generation
  - Supports all SVG primitives
- Core Contributions
  - Chain-of-Thought reasoning for structured generation
  - Multimodal alignment between code and rendering
  - Stable, high-quality, and editable SVG output
  - Model outperforms SOTA methods and is preferred in user study

References

- Hanqi Chen, Zhongyin Zhao, Ye Chen, Zhujin Liang, and Bingbing Ni. 2025. SVGThinker: Instruction-Aligned and Reasoning-Driven Text-to-SVG Generation. In Proceedings of the 33rd ACM International Conference on Multimedia (MM '25). Association for Computing Machinery, New York, NY, USA, 11004–11012. https://doi.org/10.1145/3746027.3755392