Chart2SVG: Editable SVG Generation from Raster Chart Images

Jinning Cui, Lu Chen, Haoyan Shi, Yue He, Chenglong Wang, Mengyu Zhou, Weidong Huang, and Yunhai Wang

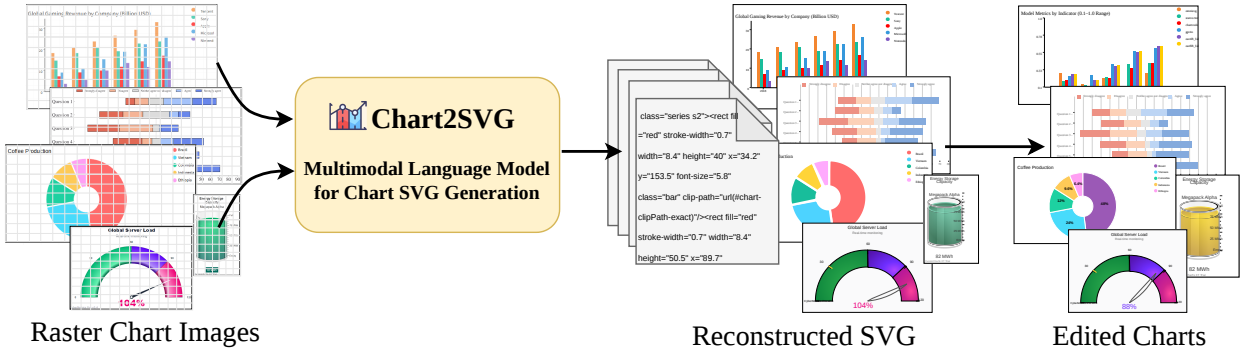

Fig. 1: Chart2SVG processes raster chart images using a multimodal LLM architecture and converts a wide range of visualizations, including bar charts, pie charts, and custom designs, into high-quality Scalable Vector Graphics (SVG) code. This enables downstream tasks such as data exploration, chart repurposing, and layout reuse. (Left) Inputs: diverse raster chart images. (Center) Our multimodal language model trained on large-scale chart corpus. (Right) Outputs: generated SVG source code that represents semantically structured and editable vectorized charts that support language-guided manipulation for downstream tasks.

Abstract—We present Chart2SVG, a multimodal large language model that converts static raster charts into structurally organized, semantically enriched SVGs that support programmatic editing. By incorporating chart-specific semantic tokens into a vision-language model, Chart2SVG captures both geometric primitives and their functional roles. To support robust structural recovery, we introduce Beagle+, a dataset of 33K canonicalized and structurally distilled chart samples. Our approach combines specialized training objectives with a rendering-aware post-training phase, producing SVGs that are both visually accurate and structurally consistent. To facilitate higher-level manipulations, we construct a Chart Structure Graph (CSG) that exposes visual dependencies, enabling tasks such as interactive exploration, chart repurposing, and layout reuse. Experiments show that Chart2SVG substantially outperforms baselines in reconstruction fidelity and downstream editing utility, advancing the development of intelligent and interactive visualization tools.

Index Terms—Chart Reverse Engineering, Scalable Vector Graphics, Vision-Language Models

1 Introduction

Data visualizations, such as charts, are a cornerstone of communication, distilling complex insights across scientific research, business analytics, and journalism. To facilitate broad knowledge dissemination, these visualizations are often rendered and shared as static raster images. However, this flattening process removes their underlying structure and data relationships, making reuse or modification notoriously difficult. Consequently, many advanced reuse techniques focus exclusively on scalable vector graphics (SVGs) [7, 11, 31]. At the same time, charts frequently need to be adapted to meet evolving requirements, from localized updates, such as correcting a specific data value, to global stylistic adjustments, such as refining color palettes for accessibility. This continual refinement is central to the reuse and modification process, ensuring charts remain accurate, effective, and communicatively relevant as their context evolves [2, 4].

Traditionally, reusing and modifying high-fidelity rasterized charts has relied on labor-intensive manual workflows, typically requiring users to convert images into SVGs using professional tools like Adobe Illustrator [12], followed by extensive cleanup. To bypass these bottlenecks, a substantial body of work has explored reverse-engineering visualizations directly from bitmap images to recover underlying data and visual encodings, rather than reconstructing the original SVG. Early systems, such as ReVision [28], pioneered automated chart classification and graphical mark extraction. Subsequent approaches refined these pipelines for greater reliability. For example, Poco and Heer [24] focused on extracting precise encoding mappings, while ChartSense [15] introduced interactive, human-in-the-loop workflows. Deep learning methods, such as ChartOCR [18], further advanced the field by leveraging object detection and optical character recognition (OCR) to robustly reconstruct data tables and chart structures.

In parallel, another line of research in the machine learning community explores direct vectorization, converting raster images into SVGs. Early deep learning approaches, such as Im2Vec [25] and LIVE [19], treat charts as generic graphics, capturing visual paths but losing the semantic binding between marks and underlying data. The emergence of multimodal large language models (MLLMs) [41] enables end-to-end parsing, generating structured representations of data and visual encodings without separate extraction steps. For example, StarVector [26] outputs compilable SVG syntax, while OneChart [8] improves numerical reasoning using auxiliary tokens. Chart-to-code methods, such as ChartCoder [42], extend this paradigm by generating executable

- *Jinning Cui, Yue He and Yunhai Wang are with Renmin University of China, Beijing, China. Email: {jinningcui,heyue25,wang.yh}@ruc.edu.cn.*
- *Lu Chen is with State Key Lab of CAD&CG, Zhejiang University, Zhejiang, China. Email: lu.chen@zju.edu.cn*
- *Haoyan Shi is with Shandong University, Shandong, China. Email: 202315173@mail.sdu.edu.cn*
- *Chenglong Wang is with Microsoft Research, Redmond, Washington, United States. Email: chenglong.wang@microsoft.com*
- *Mengyu Zhou is with Qwen Large Model Application Team, Alibaba, Beijing, China. Email: zhoumengyu.zmy@alibaba-inc.com*
- *Weidong Huang is with University of Technology Sydney, Sydney, NSW, Australia. Email: Weidong.Huang@uts.edu.au*
- *Yunhai Wang is the corresponding author.*

Manuscript received xx xxx. 202x; accepted xx xxx. 202x. Date of Publication xx xxx. 202x; date of current version xx xxx. 202x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.202x.xxxxxxx

visualization code directly from images. Despite these advances, none of these methods directly recover the original chart SVG structure, so precise graphical elements, styling, and spatial relationships are often lost. As a result, automated reproduction or programmatic modification may fail to fully preserve the chart's visual fidelity and structure.

To address these challenges, we introduce Chart2SVG, a foundational MLLM for chart SVG generation and editing. Chart2SVG leverages a vision-language model (VLM) adapted from Qwen3-VL [1] to reconstruct structural SVG charts from raster inputs (as shown in Figure 1), capturing both geometric primitives and their underlying semantics. To mitigate the stochasticity in raw vector data, we introduce a chart SVG normalization pipeline that canonicalizes geometry, coordinate systems, and styling, improving training stability and reducing variability. During supervised fine-tuning, the model is augmented with SVG semantic tokens to explicitly encode chart components and their relationships, enabling more structured and accurate generation. The model is further optimized using rendering-aware Group Relative Policy Optimization (GRPO) [27, 30], which incorporates execution-based visual feedback to penalize structural inconsistencies and invalid code. This ensures outputs are both visually faithful and programmatically correct, providing a stable foundation for downstream editing.

Once the SVG is reconstructed, we apply heuristics from [7, 31] to label code primitives as semantic components such as axes, legends, titles, and marks. Building on these labels, we introduce a Chart Structure Graph (CSG), which lifts the primitive-level SVG annotations into a structured edit-propagation graph. The CSG captures dependencies in data encoding and spatial layout among primitives, providing a grounded reasoning substrate for an LLM to plan coordinated edits across spatially scattered yet inherently connected SVG elements. This enables complex chart-editing tasks, such as recoloring data points (while updating the legend) or adjusting axes (while repositioning marks), to be performed automatically while preserving structural consistency and visual integrity. By combining execution-guided SVG recovery with graph-based semantic reasoning, Chart2SVG bridges high-level user intent and low-level vector manipulation, enabling robust, programmatic editing of complex charts.

We evaluate Chart2SVG along two dimensions: reconstruction quality and downstream utility. For reconstruction, we compare against representative baselines on held-out in-distribution benchmarks covering multiple chart-generation toolchains, as well as an out-of-distribution benchmark of real-world charts. Our evaluation measures not only pixel fidelity and perceptual similarity, but also foreground alignment, edge consistency, and rendering failure rate, reflecting both visual quality and executability. We further conduct ablations to isolate the effects of chart-specific semantic tokens and rendering-aware GRPO. Beyond reconstruction, we use case studies to examine the practical value of the recovered representation, showing that Chart2SVG supports language-guided chart analysis, semantic editing, chart reuse with new data, and multi-chart composition.

In summary, our contributions are threefold:

- We present Chart2SVG, a MLLM for reconstructing editable SVG charts from raster images, trained on Beagle+, a 33K-sample canonicalized chart SVG corpus designed to regularize heterogeneous web-crawled SVGs into a more learnable representation;
- We introduce Chart Structure Graph, an LLM-facing interface for structure-aware direct manipulation that lifts reconstructed SVG primitives into a semantic dependency space for propagating edits automatically and consistently across related SVG elements;
- We evaluate Chart2SVG and Chart Structure Graph through real-world reconstruction benchmarks, ablation studies, and downstream editing tasks, demonstrating strong reconstruction fidelity and practical editing utility.

2 Related Work

2.1 Reverse-Engineering Visualizations

Prior work on chart reverse engineering has primarily focused on recovering chart structures, visual encodings, or underlying data from static rendered images. Early systems, such as ReVision [28], treated chart images as bitmap inputs for classification and mark extraction, demonstrating that visualizations can be partially repurposed even without access to the original data. Similarly, Poco and Heer [24] targeted the recovery of Vega-Lite representations, using a pipeline of OCR and mark-type recognition to systematically infer data domains and encoding ranges. To improve the accuracy of fully automated pipelines, ChartSense [15] introduced an interactive, human-in-the-loop workflow, enabling users to refine extracted elements and achieve precise data recovery. Beyond general reuse, these techniques are also critical for accessibility; for example, Choi et al. [10] demonstrated that recovering structured data from static charts enables alternative modalities, such as tactile graphics or sonified descriptions, for visually impaired users.

Subsequent work strengthened these pipelines with learned detectors and multimodal reasoning. Subsequent work strengthened these pipelines with learned detectors: ChartOCR [18] predicts keypoints to reconstruct data with chart-specific rules; LineEX [23] recovers lines and scales via pose estimation; and ChartReader [9] integrates transformer-based detection with pretrained VLMs for chart-to-table and question answering tasks. More recently, MLLMs [33] have enabled end-to-end generative reconstruction via code. OneChart [8] parses data and encodings in a single pass using structured tokens and auxiliary numerical grounding tokens. Benchmarks such as Plot2Code [35] and ChartX [36] evaluate MLLMs on their ability to reproduce chart data and textual content, while ChartCoder [42] generates executable visualization code directly from images. Despite these advances, such methods often fail to preserve the precise spatial and structural fidelity of the original chart, producing non-renderable outputs or distorted geometries.

Despite significant progress, existing methods typically recover tables or specifications rather than editable vector graphics, leaving visually faithful and reusable SVG reconstruction as an open challenge.

2.2 Vector Graphics Generation

Vector graphics generation studies how to synthesize scalable vector representations from raster inputs, typically in the form of SVG programs composed of geometric primitives and drawing commands. Classical vectorization tools, such as Potrace [29], recover smooth curves from raster bitmaps through shape-driven tracing. Neural and differentiable methods recast this problem as learning vector representations: DiffVG [17] introduces a differentiable rasterizer enabling gradient-based editing and learning over vector primitives, while Neural methods recast vectorization as learning problems: DiffVG [17] enables gradient-based editing via differentiable rendering; DeepSVG [5] disentangles shapes from drawing commands for icon generation; Im2Vec [25] and LIVE [19] synthesize vectors from raster supervision using differentiable rendering and progressive layer-wise fitting respectively. While these methods improve fidelity and flexibility over classical tracing, they largely treat vector graphics as collections of geometric primitives, with limited modeling of semantic object structures.

More recently, multimodal foundation models have reframed SVG generation as a code-generation task. StarVector [26] employs a multimodal large language model to generate SVG code from images and text, supporting primitives such as ellipses, polygons, and text, and introduces SVG-Stack and SVG-Bench for broader evaluation. LLM4SVG [39] leverages learnable semantic tokens, a modular architecture for vector instructions, and large-scale SVGX datasets to reduce hallucinations and improve semantic alignment. RLRF [27] further enhances autoregressive SVG generation by optimizing rendered outputs with reinforcement learning, combining rewards for reconstruction fidelity, semantic consistency, and code efficiency.

Despite these advances, prior work targets generic graphics such as icons and illustrations, without handling chart-specific structures such as axes, legends, and grouped marks.

2.3 Visualization Reuse and Modification

A lot of work focuses on the reuse and augmentation of existing visualizations, as designers often find it more natural to modify existing graphics than to start from scratch [2, 4]. While traditional template-based

systems offer a user-friendly entry point, they are frequently limited by the fixed expressivity and quantity of templates provided by developers. To overcome these constraints, research has investigated how to transform existing visualizations into reusable templates without manual intervention. D3 Deconstructor [13] and iVoLVER [22] pioneered data-driven reuse from D3-based and heterogeneous sources; Ivy [21] and Cui et al. [11] further converted specifications into parameterized templates and supported mixed-initiative infographic retargeting.

Building on these foundations, DIVI [31] and Mystique [7] deconstruct SVG charts to recover semantic components such as axes, legends, and data encodings. Mystique further decomposes complex rectangle-based layouts into mark groups, spatial relationships, data encodings, and graphical constraints. By formalizing these relationships without source data or D3.js metadata, it enables the reuse of advanced layouts like small multiples and nested groupings. Structural recovery also supports accessibility; Choi et al. [10] showed that understanding a chart's structure is essential for tactile graphics. Recent datasets like VisAnatomy [6] provide fine-grained labels for element roles and hierarchical grouping. DataWink [37] allows users to adapt high-quality SVG visualizations using LLMs to extract data encodings and edit appearance via a conversational interface and on-demand widgets.

Despite these advances, the vast majority of reuse systems assume access to structured vector sources and cannot operate on raster images, a gap that Chart2SVG directly addresses.

3 CHART2SVG

We study the problem of reconstructing editable SVG charts from raster chart images. Our approach, Chart2SVG, combines a chart-oriented data preparation pipeline with a vision-language reconstruction model trained in two stages. We first build a large-scale paired corpus of raster charts and SVG code, then canonicalize heterogeneous raw SVGs into a unified representation. This format is more regular for learning while preserving the structural cues needed for downstream editing. On top of this representation, we adapt a pretrained vision-language model with SVG-specific semantic tokens and train it using supervised fine-tuning (SFT) followed by rendering-aware reinforcement learning (RL).

3.1 Data Preparation Pipeline

High-quality SVG supervision is essential for chart-to-SVG reconstruction. However, raw web SVGs are noisy, tool-dependent, and often poorly aligned with the rendered chart appearance. We therefore construct a paired chart corpus and transform all SVG targets into a canonical chart-oriented representation before model training.

Data Source. Our primary requirement is real-world chart data featuring aligned raster images and SVG code. However, existing datasets are limited in scale: VisAnatomy [6] contains only 942 real-world pairs curated for semantic diversity, while the REV [24] corpora include fewer than 500 charts from Quartz/Atlas and academic papers. To address this scarcity, we construct *Beagle+*, our main training corpus, based on Beagle [3]. This collection comprises over 41K SVG visualizations mined from five web-based repositories across 24 chart types. We further broaden coverage by crawling 276 additional examples from the ECharts [16] gallery. Notably, we exclude the D3 subset from Beagle; its long tail of bespoke designs and irregular structural patterns makes it less suitable for establishing a stable reconstruction model during the initial training stage.

While SVG serves as a versatile vector graphics format, it is semantically agnostic and possesses no inherent concept of chart elements such as bars or axes. Consequently, visually similar charts may be encoded with vastly different document object model (DOM) structures, primitive choices, style systems, and transform hierarchies depending on the authoring tool used. In our corpus, some tools generate deeply nested <g> structures while others produce much flatter SVG trees; similarly, some encode appearance through inline attributes, whereas others rely heavily on global <style> blocks and CSS selectors. We also observe substantial variation in geometric syntax, including the use of absolute versus relative path commands and multi-level transform chains.

Such heterogeneity introduces significant noise for sequence modeling, as the model must learn tool-specific implementation details alongside chart semantics. However, simply flattening SVGs into geometry-only representations using tools like SVGO [32] is undesirable, as it discards the grouping and primitive information essential for downstream editing. We therefore seek a representation that suppresses accidental cross-tool variation while preserving the structural cues that maintain chart editability.

SVG Charts Canonicalization. To reduce this heterogeneity without sacrificing editability, we canonicalize all raw SVG charts into a unified chart-oriented representation. Our goal is not generic SVG compression, but a representation that is both more regular for sequence modeling and still structurally meaningful for downstream chart editing. To this end, we preserve chart-relevant structure whenever possible, including meaningful <g> hierarchies, primitive tags, and useful `class` attributes, rather than collapsing the entire chart into anonymous paths.

1. *Canvas Standardization:* injecting standard namespaces and a `viewBox`, and unifying the canvas dimensions to ensure consistent rendering across diverse environments;
2. *Style Normalization:* converting tool-specific CSS styles into element-level attributes and removing inline style blocks to eliminate priority conflicts;
3. *Geometric Canonicalization:* standardizing path representations using relative coordinates and flattening nested `transform` operations to simplify geometric structures, thereby facilitating position computation and local editing; and
4. *Structural Distillation:* removing redundant placeholder elements, unnecessary attributes, and embedded bitmap content, while preserving the original hierarchical structure and primitive semantics (e.g., `line`, `rect`, `circle`) to improve clarity without sacrificing meaning.

The steps of style normalization and geometric canonicalization enable consistent and efficient global modifications of the converted SVG files, while the structural distillation step ensures that the final representation remains lightweight and semantically transparent for downstream sequence modeling. Compared with raw web-generated SVGs, the resulting representation is substantially more compact, typically reducing the size to 46%–72% of the original.

Since real-world datasets often contain overly dense and large SVG files (e.g., maps or large-scale scatter plots), it is challenging for the model to learn complete and well-formed SVG sequences. To address this, we filter out samples whose token length exceeds the maximum context length (8192), resulting in a curated dataset of 33K valid examples, which we denote as *Beagle+*. Beagle+ is derived primarily from the Beagle dataset [3] and is produced through our chart-oriented canonicalization pipeline.

3.2 Network Architecture and Training

As shown in Fig. 2, our reconstruction model combines a pretrained vision-language backbone, a structured SVG tokenization scheme, and a two-stage training strategy. The first stage learns chart-to-SVG generation by supervised fine-tuning (SFT), while the second stage refines the model using rendering-aware reinforcement learning to better align generated SVGs with their rendered appearance.

Problem Formulation and Backbone. We formulate chart-to-SVG reconstruction as a conditional sequence generation task. Given a raster chart image I, the model predicts a canonicalized SVG token sequence $\hat{S} = [s_1, \ldots, s_L]$ in an autoregressive manner. Concretely, the visual encoder extracts image features from I, which are projected into the language model as visual tokens. The model then decodes SVG tokens sequentially conditioned on both the visual context and previously generated tokens:

$$P(\hat{S} \mid I) = \prod_t P(s_t \mid I, s_{<t}). \tag{1}$$

This formulation enables unified modeling of chart structure, geometry, and styling within a single VLM framework.

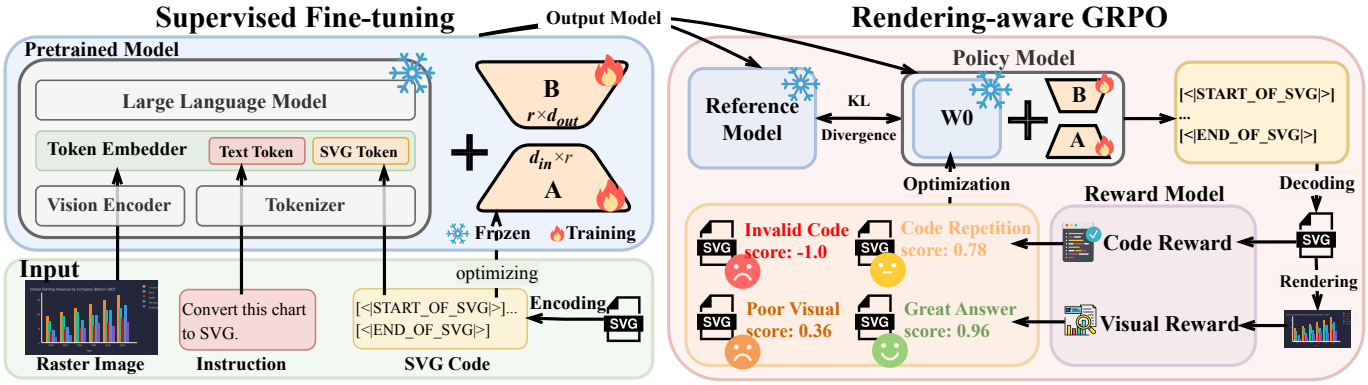

Fig. 2: Training pipeline. (Left) SFT: a vision encoder and LLM are aligned via parameter-efficient adapters to generate structured SVG sequences. (Right) Rendering-aware GRPO: SVG candidates are evaluated by code validity and visual fidelity rewards, with KL divergence against a reference model for regularization.

We instantiate our model on top of Qwen3-VL [1], a large-scale vision-language model designed for unified multimodal understanding. Qwen3-VL is well-suited to this task, as chart reconstruction requires capturing both fine-grained visual details and long-range structural relationships. In this context, the model must accurately interpret subtle elements such as ticks and labels while preserving the global semantic coherence necessary to produce a functional and editable SVG.

SFT with SVG Grammar Tokens. To adapt the backbone to chart reconstruction, we extend the tokenizer with 115 SVG-specific special tokens and perform supervised fine-tuning on paired raster chart-SVG data. Rather than treating SVG as plain text, we represent it with structured token units designed to better match the semantic organization of chart graphics. Formally, we extend the tokenizer vocabulary as

$$\mathcal{V} = \mathcal{V}_{\text{base}} \cup \mathcal{V}_{\text{struct}} \cup \mathcal{V}_{\text{prim}} \cup \mathcal{V}_{\text{attr}} \cup \mathcal{V}_{\text{geom}}, \tag{2}$$

where $\mathcal{V}_{\text{base}}$ denotes the original tokenizer vocabulary of the pretrained Qwen3-VL model, and the added SVG-specific tokens are organized into four groups, i.e., structural containers, graphical primitives, visual channel attributes, and geometric opcodes, following the chart-oriented taxonomy of VisAnatomy [6], excluding element-type semantics and reference elements:

1. *Structural containers and anchors* ($\mathcal{V}_{\text{struct}}$), including tokens such as `<svg>`, `<g>`, `<defs>`, and `<clipPath>`, which capture document scope and grouping structure;
2. *Graphical primitives* ($\mathcal{V}_{\text{prim}}$), including `<rect>`, `<circle>`, `<line>`, `<path>`, `<text>`, and `<use>`, which represent the basic chart elements;
3. *Visual channel attributes* ($\mathcal{V}_{\text{attr}}$), including geometric and stylistic tokens such as `x`, `y`, `width`, `height`, `fill`, `stroke`, `font-size`, `opacity`, and `transform`; and
4. *Geometric opcodes* ($\mathcal{V}_{\text{geom}}$), including path commands such as `M`, `L`, `C`, `Q`, and `Z`, which encode complex trajectories such as polylines, curves, and area boundaries.

Under this tokenization scheme, the model predicts a canonicalized SVG sequence composed of semantically meaningful units rather than fragmented XML strings. A complete definition of all SVG-specific special tokens is provided in the supplementary materials.

For each newly introduced token, we assign its embedding as the average embedding of a short natural-language description of its SVG meaning, so that semantically related tokens occupy nearby positions in the pretrained language space. Concretely, if a token is associated with a description sequence $\{w_j\}_{j=1}^{n}$, its embedding is defined as

$$\mathbf{e} = \frac{1}{n} \sum_{j=1}^{n} \mathbf{E}(w_j), \tag{3}$$

where $\mathbf{E}(\cdot)$ denotes the pretrained embedding lookup of the base tokenizer. We keep the embedding layer fixed during training.

We fine-tune the model with the standard autoregressive objective to maximize the likelihood of the target SVG token sequence given the input chart image. Instead of fully tuning the entire backbone, we use LoRA [14] to update a small set of low-rank adaptation parameters:

$$W = W_0 + \Delta W, \qquad \Delta W = BA, \tag{4}$$

where W_0 is the frozen pretrained weight and B, A are trainable low-rank factors. This design substantially reduces training cost and memory usage, while preserving the strong multimodal priors of the pretrained Qwen3-VL model. The SFT stage teaches the model both the grammar of SVG and the mapping from chart appearance to structured vector representations, including hierarchical grouping, graphical primitives, and encoding attributes.

Rendering-aware GRPO. While SFT optimizes token-level likelihood, it does not directly enforce the rendered quality of the generated SVG. In practice, small token-level errors, such as misplaced coordinates or duplicated primitives, can lead to large perceptual artifacts after rendering. To better align training with the actual reconstruction objective, we further refine the model using rendering-aware reinforcement learning based on Group Relative Policy Optimization (GRPO) [30]. Given an input chart image I, the current policy π_θ samples a group of G candidate SVG sequences $\{S_i\}_{i=1}^{G}$. Each candidate is rendered into a raster image $\tilde{I}_i$, and rewards are computed from both the SVG code and the rendered output.

We first define a **code-level reward** to encourage syntactically valid and compact SVG generation. Unlike natural language, SVG must satisfy strict structural constraints to be executable. We therefore assign a validity reward $R_{\text{valid}}(S_i)$ based on whether the generated SVG can be successfully parsed and rendered. In addition, we penalize unnecessarily long or redundant sequences to discourage degenerate behaviors such as repeatedly drawing overlapping shapes. The code-level reward is defined as

$$R_{\text{code}}(S_i) = R_{\text{valid}}(S_i) - \lambda_{\text{len}} \mathscr{P}_{\text{len}}(S_i). \tag{5}$$

We then define a **visual reward** that directly measures how well the rendered SVG matches the target chart. Since charts are structured and information-dense, no single image metric is sufficient. We therefore combine three complementary measures: (i) *Foreground Intersection over Union* (IoU), which evaluates overlap on non-background regions and focuses on chart content such as marks, axes, and text; (ii) *Structural Similarity Index* (SSIM), which captures local structural consistency and is sensitive to thin lines and textual elements; and (iii) *Peak Signal-to-Noise Ratio* (PSNR), which measures overall pixel-level fidelity and color consistency. The visual reward is

$$R_{\text{vis}}(\tilde{I}_i, I) = \frac{1}{3}\Big(\text{IoU}_{\text{fg}}(\tilde{I}_i, I) + \text{SSIM}(\tilde{I}_i, I) + \text{PSNR}(\tilde{I}_i, I)\Big). \tag{6}$$

Note that our visual reward design differs from RLRF [27] because chart reconstruction places greater emphasis on precise structural fi-

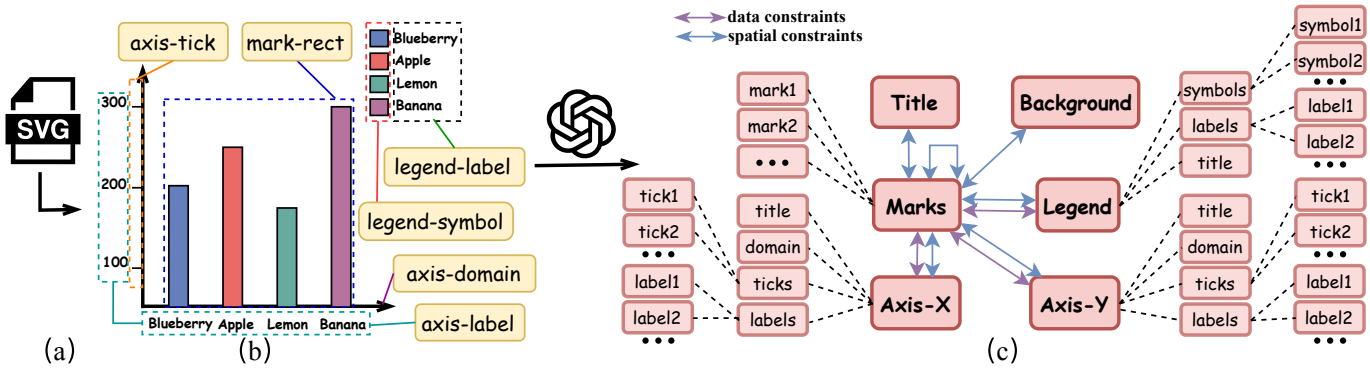

Fig. 3: CSG construction pipeline. From a reconstructed SVG (a), chart components are extracted (b) and organized into a graph (c) with data constraints (governing mark-axis-legend relationships) and spatial constraints (defining layout relative to marks).

delity than on high-level semantic similarity. In generic SVG generation, perceptual rewards are useful for encouraging semantic alignment, but for charts, small geometric errors in marks, ticks, labels, or legends can alter the conveyed data while remaining semantically similar at a coarse level. We therefore use chart-oriented visual rewards based on foreground IoU, SSIM, and PSNR, so that optimization better reflects the fine-grained geometric requirements of editable chart SVGs. The detailed computation of the reward functions is provided in Appendix C.

The final reward balances code-level validity and visual fidelity by combining the two components:

$$R(S_i, I) = \lambda_{\text{code}} R_{\text{code}}(S_i) + \lambda_{\text{vis}} R_{\text{vis}}(\tilde{I}_i, I). \tag{7}$$

Here, λ_{code} and λ_{vis} control the relative importance of code correctness and visual accuracy. The code-level reward ensures that outputs remain executable, compact, and structurally well formed, while the visual reward encourages faithful reconstruction of chart geometry and appearance. In practice, we assign greater weight to the visual term, while retaining a non-negligible code reward to avoid invalid or excessively verbose SVG outputs.

Following GRPO [30], we sample a group of G candidate SVG sequences for each input and normalize their rewards within the group to obtain relative advantages A_i. Specifically, A_i is computed by standardizing rewards across the group, encouraging the model to prefer relatively better candidates rather than relying on absolute reward scales. The policy is then updated using the clipped objective

$$\mathscr{L}_{\text{GRPO}} = -\frac{1}{G}\sum_{i=1}^{G} \min(r_i A_i,\ \text{clip}(r_i, 1-\delta, 1+\delta) A_i), \tag{8}$$

where $r_i = \pi_\theta(S_i \mid I)/\pi_{\theta_{\text{old}}}(S_i \mid I)$ is the likelihood ratio between the current and previous policies, and δ controls the clipping range to stabilize updates.

By optimizing rewards defined on rendered outputs rather than only token sequences, this stage better aligns training with the perceptual and structural objectives of chart reconstruction, leading to higher visual fidelity and more reliable SVG structure.

4 Semantic Structure-Aware Chart Manipulation

The goal of Chart2SVG is not only to reconstruct charts as SVGs, but also to make vectorized charts directly editable through language-guided manipulation. This interaction paradigm empowers users to issue high-level instructions rather than manually modifying low-level vector code. Prior work such as DirectGPT [20] improves the efficiency of vector graphic editing by combining natural language with direct manipulation. However, it operates on vector primitives, where users select elements and apply isolated edits. While effective for generic vector graphics, this approach is insufficient for charts, which are inherently governed by strict structural regularity and semantic organization.

The primary challenge in chart understanding lies in bridging the semantic gap between low-level SVG primitives and high-level chart constructs. This challenge is twofold. First, individual primitives are fundamentally ambiguous in isolation. A single `<line>` might denote an axis domain, a gridline, or a tick mark, just as a `<rect>` could represent a data bar, a legend entry, or a background bounding box. Recent systems such as DIVI [31] and Mystique [7] provide robust heuristics to map raw SVG elements to high-level chart components, effectively addressing this labeling problem.

Second, chart elements are not independent; they are tightly coupled through complex structural and semantic dependencies. Modifying one component invariably necessitates coordinated updates to others. For instance, altering a data series requires updating the legend, and adjusting layout properties like bar width dictates shifts in spacing and alignment across the entire chart. Capturing these interdependencies remains largely unaddressed.

To bridge this gap, we build on the heuristically labeled components from DIVI and leverage large language models (LLMs) to automatically infer additional structural and semantic rules. Based on these relationships, we transform the labeled SVG into an explicit *Chart Structure Graph* (CSG), formally defined as $G = (V, E)$, as illustrated in Fig. 3. Serving as a semantic interface between the SVG and the model, the CSG allows users to interact directly with structured chart components, their mapped attributes, and topological dependencies rather than disjoint low-level primitives.

4.1 Constructing Chart Structure Graphs

Before constructing the CSG, we first annotate the reconstructed SVG with chart-level semantic roles using DIVI's scene-graph vocabulary [31] as the base component schema and its heuristic grouping rules. Specifically, low-level SVG elements are assigned role-instance identifiers, such as marks, axis ticks, legend symbols, and text labels, which serve as semantic anchors for grouping elements and locating edit targets. Given the tagged SVG and the schema, an LLM then constructs the vertex set V by grouping tagged elements that share the same semantic role or belong to the same component group and instantiating them as concrete chart-component nodes. Each node is assigned a typed component label and a semantic identifier, which contains fields such as `data_key`, `scale_ref`, `svg_selector`. Because the DIVI schema restricts the node vocabulary to a fixed set of chart component types, the LLM is prevented from introducing unsupported role types. In parallel, each node retains the corresponding SVG element set or subtree as its low-level realization. This separation allows the CSG to reason over chart semantics at the graph level while preserving a concrete link to the underlying SVG elements, so that graph-level operations can be translated back into concrete SVG attribute updates.

The next step is to define the edge set E, which captures semantic and geometric dependencies among chart-component vertices. Given the instantiated vertices, their semantic attributes, and their associated SVG element sets or subtrees, we prompt an LLM to instantiate edges under a predefined constraint schema rather than freely generating arbitrary relationships. Each edge connects one or more source vertices to a target vertex and is assigned a constraint type, the affected visual attributes, and the semantic basis for the dependency. We consider two categories of constraints: data constraints and spatial constraints.

Data Constraints. These directed edges encode how data semantics determine visual encodings. They are instantiated when vertices

share data keys, scale references, legend-category bindings, or value-dependent geometric attributes. The LLM infers from three core rules:

- *Axis Scaling:* Constrains the mapping between the data domain and the coordinate system. Changes in the data range require rescaling the axes and remapping all dependent elements.
- *Value Proportionality:* Ensures that geometric attributes (e.g., height, length, or area) remain proportional to the underlying data values.
- *Category Consistency:* Maintains uniform visual encoding (e.g., color, texture, shape) for all elements within the same data category.

Spatial Constraints. These edges encode layout dependencies among vertices based on their bounding boxes, relative positions, and grouping relationships. The LLM reasons based on four key rules:

- *Element Alignment:* Ensures graphical elements and gridlines align precisely with axis ticks.
- *Group Spacing:* Maintains uniform spacing among related elements, requiring coordinated updates within groups.
- *Stacking Order:* Encodes cumulative positioning, where each element depends on preceding elements in the stack.
- *Overlap Prevention:* Avoids overlaps between text and graphical elements by triggering positional adjustments when conflicts arise.

For example, in Fig. 3, there is a data constraint between marks and axes: their coordinates are jointly determined by the same scale, so their positions remain consistent. Similarly, a data constraint exists between marks and the legend: visual style (primarily color) must be consistent, ensuring that elements of the same data category share the same appearance. Regarding spatial constraints, marks maintain layout relationships with each other and are typically aligned with axes (e.g., the bottoms of bars align with the x-axis in a bar chart). The title is positioned relative to the marks.

The chart structure graph enables consistent chart manipulation. When a user issues a high-level editing command, the system traverses the graph to propagate updates along these dependencies, preserving both semantic correctness and visual coherence. To maintain consistency under layout changes, the CSG employs a bidirectional constraint propagation mechanism: local modifications that increase spatial demand (e.g., enlarging elements or adding data) propagate upward, expanding axes and the canvas to prevent overlap or clipping, while global constraints (e.g., reducing canvas size) propagate downward, compressing element sizes, spacing, and typography. This bidirectional propagation ensures both local edits and global adjustments yield structurally valid and visually consistent chart layouts. The CSG inference prompt template is provided in Appendix B.

4.2 CSG-aware Chart Manipulation

CSG serves as the core intermediate representation for chart manipulation. Given a natural-language editing instruction, the LLM first grounds the instruction to one or more target vertices in the CSG using their component types, semantic attributes, textual labels, and geometric properties. This graph-level grounding restricts edits to existing chart components rather than arbitrary SVG primitives. The system then follows the dependency edges connected to the target vertices to identify other components that should be updated jointly, such as axes, legends, or mark groups. Based on this affected subgraph, the LLM produces a graph-level edit plan that specifies the vertices to update and the corresponding attribute changes. The plan is then executed on the SVG element sets or subtrees associated with these vertices, and the modified SVG DOM is written back as editable SVG code.

For example, the command of "increase the bar width for the banana data in Fig. 3(b)" triggers a graph operation that updates the relevant vertices representing the bars. Thanks to bidirectional constraint resolution, this local modification automatically propagates spatial adjustments to dependent vertices, such as expanding axes, shifting intra-group spacing, and updating legends. Similarly, "highlight all points in Category A" synchronizes attribute updates across all vertices in that data group, respecting the *category binding* rules encoded in the CSG.

In contrast to naive SVG editing, where a model must infer element coupling from raw SVG structure and geometry, CSG-guided editing exposes chart-component dependencies explicitly, allowing the LLM to plan coordinated updates over semantically related components. Our implementation uses an off-the-shelf LLM executor (GPT-5.1) for graph-edit planning, but the CSG interface is model-agnostic and can be paired with other models.

5 Evaluation

5.1 Metrics and Baselines

Metrics. We evaluate reconstruction quality by rasterizing both the predicted SVG and the ground-truth SVG at the same resolution and comparing the resulting images. Since chart images often contain large uniform background regions, evaluating similarity over the entire canvas can be misleading: a prediction may match the background well while still failing to reconstruct the chart itself. We therefore adopt a foreground-aware evaluation protocol tailored to chart images.

Specifically, we estimate the background color from the top-left pixel of the rendered image and identify foreground pixels as those whose color differs from the background by more than a small threshold (`tol=5`). The resulting foreground mask is used to automatically crop each chart to its content region before computing image similarity metrics. This reduces the influence of large blank margins and makes the evaluation more sensitive to the reconstructed chart elements.

Let I and $\hat{I}$ denote the cropped rasterized ground-truth and predicted images, respectively, and let M and $\hat{M}$ denote the corresponding foreground masks. We report the following metrics:

1. **PSNR**, computed from the mean squared pixel error between I and $\hat{I}$, which measures low-level reconstruction fidelity. Specifically, it is computed as $\mathrm{PSNR} = 10\log_{10}(\mathrm{MAX}^2/\mathrm{MSE})$, where $\mathrm{MAX} = 255$ for 8-bit raster images.
2. **SSIM**, which measures structural similarity between the reconstructed chart and the ground truth by comparing local luminance, contrast, and structural patterns.
3. **LPIPS**, computed with the AlexNet backbone, which measures perceptual dissimilarity between the reconstructed chart and the ground truth in a deep feature space.
4. **Foreground IoU**, defined as $\mathrm{IoU}_{\mathrm{fg}} = |M \cap \hat{M}|/(|M \cup \hat{M}| + \varepsilon)$, measures overlap between predicted and ground-truth chart foregrounds while reducing background influence.
5. **Edge Consistency**, computed from Canny edge maps with light dilation and smoothing, quantifies agreement between chart edges, emphasizing thin but semantically critical elements like axes, ticks, gridlines, and line marks.
6. **Failure Rate**, the fraction of predictions that cannot be rendered or whose background occupies more than 97% of the image, indicating failure to produce meaningful chart.

The first three metrics (PSNR, SSIM, and LPIPS) are standard in prior SVG generation models like StarVector [26], while the latter three are particularly useful for chart-specific evaluation. For PSNR, SSIM, and Foreground IoU, higher values indicate better performance, whereas lower LPIPS and Failure Rate are better.

Baselines. We compare Chart2SVG against four representative baselines. OmniSVG [40] is an end-to-end SVG generation model using discrete SVG tokenization for multimodal synthesis. StarVector [26] is a multimodal LLM that directly produces compilable SVG code from images or text. ChartCoder [42] is a dedicated chart-to-code model using a code-oriented backbone for executable chart restoration. We further include GPT-4o as a strong proprietary baseline capable of accepting image inputs and generating structured code outputs.

Implementation Details. We train two variants of Chart2SVG based on Qwen3-VL [1], with 4B and 8B parameters, sharing the same architecture and training pipeline. Input images are resized to 512×512, with maximum output length set to 8,192 tokens after canonicalization. For SFT, we use AdamW with learning rate 1×10^{-4}, 2 epochs, batch size

Table 1: In-distribution evaluation on Beagle+ test subsets. ↑: higher is better; ↓: lower is better. Bold: best among model-based methods; underline: second-best among model-based methods.

Model	ChartBlocks Subset						Fusion Subset						Plotly Subset					
	PSNR ↑	SSIM ↑	LPIPS ↓	IoU_{fg} ↑	Edge Cons. ↑	Failure Rate ↓	PSNR ↑	SSIM ↑	LPIPS ↓	IoU_{fg} ↑	Edge Cons. ↑	Failure Rate ↓	PSNR ↑	SSIM ↑	LPIPS ↓	IoU_{fg} ↑	Edge Cons. ↑	Failure Rate ↓
Adobe Illustrator†	22.9034	0.9107	0.1736	0.9187	0.7276	0.0000	21.3574	0.8813	0.2461	0.9257	0.5082	0.0000	20.8586	0.8838	0.2174	0.9217	0.5202	0.0000
OmniSVG [40]	2.3979	0.1757	0.7943	0.2498	0.3705	0.2667	4.1156	0.2162	0.7501	0.3027	0.2814	0.1615	2.7260	0.0605	0.8878	0.1872	0.1359	0.5455
StarVector [26]	9.9510	0.4935	0.6870	0.4454	0.2497	0.2667	9.1777	0.4795	0.7316	0.3304	0.2161	0.1385	9.6040	0.4933	0.6955	0.2882	0.2240	0.1273
ChartCoder [42]	6.3438	0.4462	0.7897	0.1946	0.2872	0.1917	3.5305	0.2095	0.8778	0.2129	0.1008	0.5077	2.1856	0.1358	0.8838	0.2543	0.1172	0.4091
GPT-4o	9.2612	0.5410	0.6406	0.4250	0.2825	0.0667	7.1797	0.4449	0.7012	0.3469	0.2018	0.1692	7.5700	0.3446	0.7571	0.2715	0.1479	0.3545
o3	7.4921	0.4714	0.7072	0.2955	0.2808	0.1917	10.5211	0.5490	0.6689	0.3537	0.2005	0.1308	10.7894	0.4870	0.6521	0.3481	0.1725	0.1545
Gemini-2.5-Flash	9.9435	0.5984	0.6509	0.3733	0.3126	0.0167	**12.5728**	**0.6173**	0.5986	0.4388	0.2296	0.0077	**13.6669**	**0.6051**	**0.5520**	0.3938	0.2262	0.0182
Chart2SVG-4B	15.1260	0.7034	0.2710	0.6915	0.4991	**0**	8.7804	0.5020	**0.5432**	0.5237	0.3088	**0**	7.4312	0.4974	0.6099	**0.4588**	0.2665	**0**
Chart2SVG-8B	**15.5634**	**0.7106**	**0.2600**	**0.7075**	**0.5148**	**0**	8.7341	0.5111	0.5569	**0.5520**	**0.3140**	**0**	6.6080	0.4358	0.6095	0.4419	**0.2895**	**0**

Table 2: Out-of-distribution evaluation on VisAnatomy [6]. ↑: higher is better; ↓: lower is better. Bold: best among model-based methods; underline: second-best among model-based methods.

Model	PSNR ↑	SSIM ↑	LPIPS ↓	IoU_{fg} ↑	Edge Cons. ↑	Failure Rate ↓
Adobe Illustrator†	15.8100	0.7440	0.4726	0.3873	0.3691	0.0000
OmniSVG [40]	3.6713	0.2736	0.7947	0.1345	0.2755	0.2390
StarVector [26]	8.6618	0.4116	0.7714	0.1214	0.1685	0.3558
ChartCoder [42]	7.8008	0.4115	0.7898	0.0988	0.1864	0.4026
GPT-4o	12.6482	0.6014	0.6066	0.2552	0.2423	0.0597
o3	11.8976	0.5831	0.5808	**0.2821**	0.2551	0.1039
Gemini-2.5-Flash	12.8705	0.6288	0.5669	0.2667	0.2664	0.0364
Chart2SVG-4B	14.0211	0.6951	0.5758	0.2640	0.2919	**0.0052**
Chart2SVG-8B	**14.2012**	**0.7021**	**0.5633**	0.2801	**0.2992**	0.0078

Table 3: Data accuracy by chart type on VisAnatomy [6]. For each chart, we retrieve the encoded data array and report the array-length matching rate (Len.) and the numerical accuracy over matched data (Acc.).

Model	Bar Chart		Pie Chart		Line Chart	
	Len. ↑	Acc. ↑	Len. ↑	Acc. ↑	Len. ↑	Acc. ↑
StarVector [26]	0	0	0	0	0	0
OmniSVG [40]	0.0210	0	0.1039	0.0018	0	0
GPT-4o	0.5944	0.0651	0.7013	0.0914	0.0870	0.0046
o3	**0.7413**	0.1456	0.6623	0.1243	0.3913	0.1763
Gemini-2.5-Flash	0.6643	0.1545	**0.7922**	**0.1664**	0.6087	0.2158
Chart2SVG-8B	0.6713	**0.1807**	0.7532	0.1316	**0.6522**	**0.2173**

16, and LoRA rank $r = 8$. For RL, we use group size $G = 4$, clipping parameter $\delta = 0.05$, sampling temperature $T = 1.0$, and $\lambda_{\text{code}} = 0.2$, $\lambda_{\text{vis}} = 0.6$, $\lambda_{\text{len}} = 0.2$. All models are trained on 8 NVIDIA L20 GPUs for approximately 9 hours. More detailed baseline configuration and comparative analysis are provided in the Appendix.

5.2 Quantitative Results

In-Distribution Test. Table 1 reports in-distribution results on three held-out subsets in Beagle+, generated using ChartBlocks, Fusion, and Plotly. Overall, both Chart2SVG variants substantially outperform prior baselines across the three subsets, and, most importantly, achieve a zero failure rate on all of them. This indicates that the proposed chart-oriented representation and training strategy improve not only visual similarity, but also the basic validity of generated SVG charts.

On ChartBlocks, Chart2SVG-8B achieves the best result on all metrics (15.56 PSNR, 0.71 SSIM, 0.26 LPIPS) with zero failure rate. On Fusion and Plotly, where StarVector leads on PSNR due to its sensitivity to spatial alignment, Chart2SVG models remain strongest on structural metrics and are the only methods with zero failures across all three subsets. For chart images, where many semantically important elements such as axes, ticks, and line marks are thin structures, even minor positional offsets can lead to substantial pixel error. As a result, a method may obtain high PSNR by matching coarse raster appearance, yet still reconstruct chart structure less faithfully than our method, as also reflected in the qualitative comparisons.

The *Plotly* subset is the most challenging. Here, StarVector achieves the highest PSNR, but both Chart2SVG models remain strongest on most other metrics and are again the only methods with zero failures. Chart2SVG-4B performs best on SSIM and IoU_{fg}, while Chart2SVG-8B performs best on LPIPS and edge consistency. This split suggests that the two model scales make slightly different trade-offs on Plotly charts, but both remain substantially more reliable than prior baselines.

Out-of-Distribution Test. Table 2 reports out-of-distribution results on the VisAnatomy [6] test set. All neural methods experience a noticeable performance drop compared with the in-distribution setting, confirming that cross-dataset generalization remains challenging. Adobe Illustrator achieves high visual fidelity metrics owing to its pixel-faithful path tracing, but produces no semantic structure and a zero failure rate only because it always renders something, regardless of chart correctness. Among learning-based methods, Chart2SVG-8B achieves the best results across all five metrics and maintains a failure rate of only 0.78%, outperforming GPT-4o by 12.28% in PSNR and 24.48% in edge consistency, and remaining competitive with the stronger proprietary models o3 and Gemini-2.5-Flash while producing semantically structured, directly editable SVG output.

Data Value Accuracy. Table 3 reports data accuracy on VisAnatomy by chart type, measuring array-length matching rate (Len.) and numerical accuracy over successfully matched data (Acc.). StarVector and OmniSVG produce almost no valid output across all chart types. Among learning-based methods, Chart2SVG-8B achieves superior accuracy: on bar charts it leads all baselines in Acc. (0.1807 vs. at most 0.1545), on pie charts it is competitive with o3 and GPT-4o while Gemini-2.5-Flash achieves the highest Acc. (0.1664), and on line charts Chart2SVG-8B achieves the highest Acc. (0.2173) among all models. These results suggest that Chart2SVG better preserves quantitative structure when the recovered data arrays can be aligned, although array-length mismatches remain a major source of error.

Summary. Taken together, Chart2SVG consistently improves reconstruction fidelity and robustness across both in-distribution and out-of-distribution settings, achieving the lowest failure rate among all learning-based methods and remaining competitive with stronger proprietary models. Note that although Adobe Illustrator achieves strong reconstruction performance, its output SVG code consists primarily of fitted Bézier-curve coordinates, which lack explicit chart semantics and are therefore difficult to interpret, edit, or reuse for downstream chart understanding and data extraction.

5.3 Qualitative Analysis

We present a qualitative comparison in Fig. 4 spanning standard charts and highly customized pictorial charts across three dimensions: *structural fidelity*, *data accuracy*, and *aesthetic preservation*.

Structural Fidelity. Accurate reconstruction requires both precise geometry and correct spatial organization. GPT-4o often struggles with continuous geometry, producing jagged curves in doughnut charts and unintended spacing between grouped bars. ChartCoder preserves the basic structure of standard charts but struggles to generalize to complex or non-canonical layouts. StarVector-8B and OmniSVG-3B frequently yield fragmented or incomplete outputs, particularly for charts with long token sequences or multi-element compositions. Adobe Illustrator

Fig. 4: Qualitative comparison of our model with general-purpose and state-of-the-art chart and SVG foundation MLLMs on chart vectorization.

Table 4: Ablation studies on the 4B model. We take the full model as the reference and remove one component at a time. Higher is better for PSNR, SSIM, Foreground IoU (IoU_{fg}), and Edge Cons., while lower is better for LPIPS and Failure Rate.

Variant	PSNR ↑	SSIM ↑	LPIPS ↓	IoU_{fg} ↑	Edge Cons. ↑	Failure Rate ↓
Chart2SVG-4B	**14.0211**	**0.6951**	**0.5758**	0.2640	**0.2919**	**0.0052**
w/o SVG tokens	12.2325	0.6028	0.6358	0.1801	0.2585	0.1481
w/o GRPO	6.8122	0.4672	0.6263	**0.3771**	0.2579	0.0260
w/o IoU_{fg} in GRPO	12.5594	0.6342	0.6255	0.1795	0.2616	0.0987

achieves high geometric fidelity via path tracing, but all text labels and titles are rendered illegible or entirely missing, as anti-aliased character outlines collapse into noise rather than recoverable typography. In contrast, Chart2SVG consistently generates well-aligned primitives and preserves strict layout coherence, capturing both local geometric details and global compositional structures, even for highly customized pictorial charts where spatial precision is essential for readability.

Aesthetic Preservation. ChartCoder reverts to generic library defaults, with severe degradation on pictorial charts. GPT-4o struggles with complex shading and text alignment. Chart2SVG better preserves custom color palettes, advanced effects, and text layout, maintaining visual fidelity across both standard and metaphorical mark styles.

Summary. Chart2SVG achieves a superior balance across structure, data, and aesthetics, producing robust reconstructions that bridge the gap between raster images and semantically structured vector representations that support programmatic editing workflows.

5.4 Ablation Studies

Table 4 reports ablation results on the 4B model, where the full Chart2SVG-4B serves as the reference, and one component is removed at a time. Overall, both chart-specific SVG tokenization and the rendering-aware GRPO objective substantially contribute to reconstruction quality and output validity.

Effect of SVG Grammar Tokens. Removing SVG grammar tokens degrades all metrics, with PSNR dropping from 14.02 to 12.23 and failure rate rising from 0.52% to 14.81%, confirming that chart-oriented tokenization is critical for visual fidelity and rendering reliability.

Effect of GRPO Training. Omitting GRPO causes the largest drop overall: PSNR falls from 14.02 to 6.81 and SSIM from 0.6951 to 0.4672. Notably, IoU_{fg} slightly increases without GRPO, highlighting that foreground overlap alone does not capture reconstruction quality—a prediction can cover the foreground yet produce incorrect structure.

Effect of Foreground IoU Reward. Removing the foreground IoU reward raises the failure rate from 0.52% to 9.87% and reduces IoU_{fg} from 0.2640 to 0.1795, demonstrating that this term complements rendering feedback by preserving the spatial extent of chart content.

Table 5: Controlled editing study on 100 charts sampled from VisAnatomy. Each chart is paired with four manually specified editing tasks covering color, axis, data-value, and layout edits. CSG denotes whether CSG-inferred constraints are used during editing, and Semantic Label denotes whether the SVG is heuristically tagged before CSG construction. Success rate, construction accuracy, and edit-propagation F1 score are computed against manually verified annotations and reference edits.

CSG	Semantic Label	Success Rate ↑	CSG Acc. ↑	Propagation Precision ↑	Propagation Recall ↑	Propagation F1 ↑
✗	✗	0.5000	—	—	—	—
✓	✗	0.6300	0.8389	0.6194	0.4798	0.5407
✓	✓	**0.7400**	**0.8919**	**0.7025**	**0.4913**	**0.5782**

Effect of CSG on Chart Editing. Beyond reconstruction, we conduct a small controlled editing study to ablate the contribution of CSG to downstream manipulation. We select 100 charts from VisAnatomy to cover diverse chart types. For each chart, we obtain a CSG through LLM-assisted annotation followed by manual correction, yielding verified chart components and dependency edges. Each chart is paired with four manually created editing tasks—color update, axis rescaling, data value modification, and layout adjustment—together with manually edited reference SVGs. This study evaluates three aspects of the pipeline: whether CSG-guided editing improves success over direct SVG editing, whether DIVI-based semantic labels lead to accurate CSG construction, and whether the LLM uses the task-relevant CSG edges during edit propagation. All editing experiments use GPT-5.1 as the executor, while the CSG representation itself is model-agnostic and can be paired with other capable instruction-following models.

Table 5 reports results across three configurations. Without CSG, the LLM completes only 50.0% of tasks. Adding CSG-inferred constraints raises success rate to 63.0% with edge propagation F1 of 0.5407, suggesting that explicit dependency edges help coordinate edits across related components. When the heuristically tagged SVG is additionally provided as semantic context, all metrics improve further: success rate reaches 74.0%, CSG construction accuracy reaches 89.2%, and propagation F1 increases to 0.5782. These results indicate that both the CSG structure and the underlying semantic annotations contribute to more reliable chart editing, with semantic labels improving the construction and use of task-relevant dependency edges.

Summary. These ablation results support three key conclusions: (1) chart-specific SVG tokenization is crucial for learning structured chart syntax, (2) rendering-aware GRPO is a major driver of reconstruction quality, with the foreground IoU term providing additional gains in

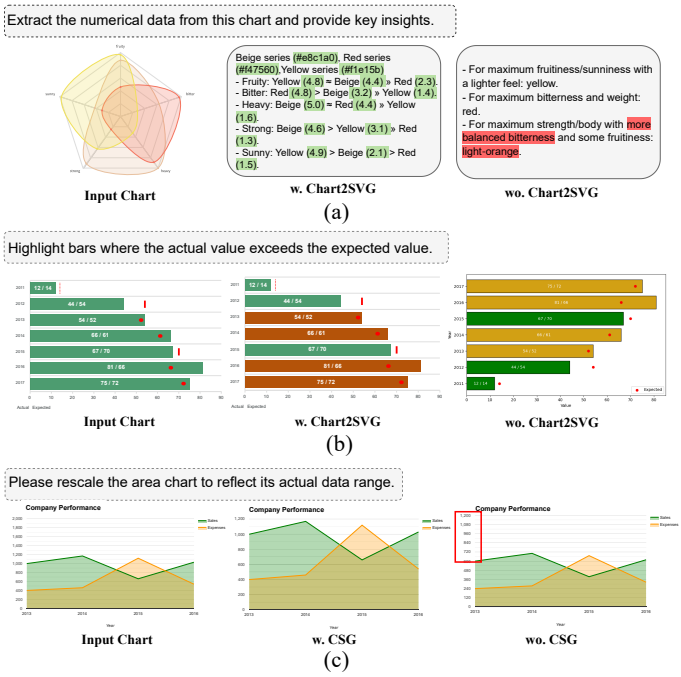

Fig. 5: Interactive chart exploration. (a) Data extraction: Chart2SVG enables precise value recovery; baseline hallucinates ranges. (b) Semantic querying: correct mark highlighting vs. baseline misidentification. (c) Rescaling: CSG updates geometric paths; baseline only edits text labels.

structural fidelity and output validity, and (3) the CSG provides consistent, measurable benefit to downstream editing, validating it as a distinct and necessary contribution beyond reconstruction.

6 Case Study

By transforming raster chart images into manipulable semantic representations, Chart2SVG enables three types of language-driven interactions: *interactive exploration*, *chart repurposing*, and *layout reuse*. In all experiments, Chart2SVG first reconstructs the chart SVG and semantic structure from the raster image, after which GPT-5.1 is used solely downstream to interpret user instructions and generate editing actions over the reconstructed representation.

Interactive Exploration. Chart2SVG transforms chart interaction from passive viewing into active visual exploration by allowing the LLM to operate over structured chart representations rather than raw pixels. To evaluate this, we compare tasks in Fig. 5(a) and (b) against the same LLM applied directly to raster images.

In Fig. 5(a), access to the reconstructed SVG enables the LLM to recover exact numerical values, such as minimums and peak totals, even when explicit labels are absent. In contrast, when operating on raster inputs, the LLM must rely on visual estimation, leading to vague approximations and hallucinated ranges (highlighted in red).

Beyond analysis, our approach supports semantic querying by treating visual marks as structured, addressable entities. In the conditional highlighting task (Fig. 5(b)), the LLM accurately identifies and highlights bars where actual values exceed expected values. Without the structured representation, the raster baseline fails to follow this logic and misidentifies the relevant data points.

To isolate the role of explicit structure, Fig. 5(c) compares edits generated with and without the CSG. When asked to rescale the y-axis to the true data range, the LLM leverages the dependencies encoded in the CSG to update geometric paths and maintain consistency between axes and marks. Without the CSG, the model only modifies textual labels, leaving the underlying geometry unchanged and breaking the visual-data correspondence.

Chart Repurposing. Charts can be transformed to re-encode existing data into new visual forms for more effective communication. In Fig. 6(a), a pie chart is converted into a pictorial bar chart. Guided by CSG dependencies, the model preserves numerical fidelity while

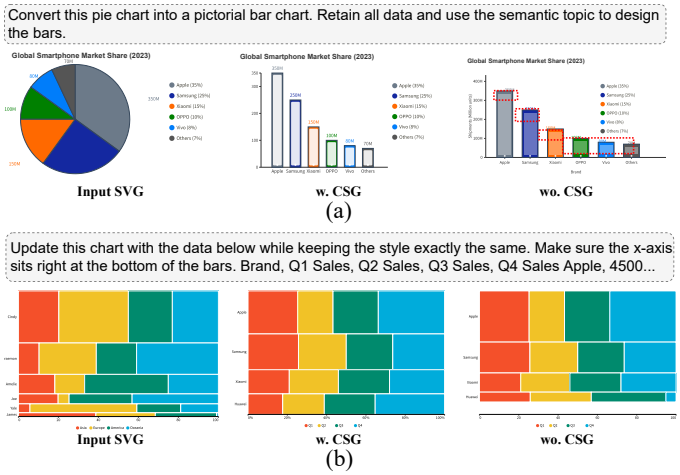

Fig. 6: Chart repurposing (a) and layout reuse (b). (a) CSG enables pie-to-pictorial-bar conversion; baseline hallucinates data (red). (b) CSG preserves x-axis alignment under new data; baseline fails.

adapting the visual form; without the CSG, the LLM reverts to generic geometries and hallucinates data (red).

Layout Reuse. Existing chart layouts can also be reused to visualize new datasets while preserving the original design. Fig. 6(b) shows a normalized stacked bar chart adapted to new sales data.The CSG enforces key spatial constraints, such as aligning the x-axis with the bar bases, ensuring structural consistency. Without it, the LLM introduces gaps that break the layout and degrade visual fdelity.

7 Discussion and Conclusion

We introduced Chart2SVG, a framework that transforms static raster charts into semantically structured SVG that enable programmatic and language-guided editing. By combining specialized training objectives with a rendering-aware post-training phase, Chart2SVG produces charts that are visually accurate and structurally organized for direct manipulation. Furthermore, by constructing and integrating a chart structure graph, we make latent visual dependencies explicit, enabling complex tasks such as interactive data exploration and automated repurposing.

Generalizability. While Chart2SVG performs well on common chart types, its capabilities reflect the Beagle+ dataset, which is dominated by foundational marks such as rectangles, lines, and circles, with complex marks accounting for only ≈5% of the corpus. The training set also excludes the D3 subset of Beagle and SVGs beyond the 8192-token limit, leaving high-density or highly customized charts underrepresented. Consequently, generalization to irregular layouts, degraded screenshots, and unconventional mark types remains limited. Future work will augment training with synthetic data covering complex marks.

Data Fidelity. Chart2SVG prioritizes visual fidelity and structural editability over explicit recovery of the underlying data table. Consequently, precise data binding—such as axis-scale inference, category-to-mark alignment, and value-to-geometry mapping—remains an important direction for future work. Integrating OCR, chart-to-table extraction, and scale-fitting modules with the reconstructed SVG could further improve data-level correctness while preserving editability.

Broader Applications. Like any deep learning model, Chart2SVG cannot guarantee perfect accuracy on degraded or custom charts, making human-in-the-loop refinement essential. Future work could integrate Chart2SVG with formal chart grammars for rule-based refinement, or connect the queryable CSG with autonomous AI agents for complex analytical queries and multi-chart synthesis.

Acknowledgment

We sincerely thank Xuan Hua and Hewen Zhang for their experiments and data generation support. This work was supported by the grants of NSFC (No.U2436209), the Beijing Natural Science Foundation (L247027), the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University of China.

REFERENCES

[1] S. Bai, Y. Cai, R. Chen, K. Chen, X. Chen, Z. Cheng et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025. doi: 10.48550/arXiv.2511.21631 2, 4, 6

[2] H. K. Bako, X. Liu, L. Battle, and Z. Liu. Understanding how designers find and use data visualization examples. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):1048–1058, 2023. doi: 10.1109/TVCG.2022.3209490 1, 2

[3] L. Battle, P. Duan, Z. Miranda, D. Mukusheva, R. Chang, and M. Stonebraker. Beagle: Automated extraction and interpretation of visualizations from the web. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. doi: 10.1145/3173574.3174168 3

[4] A. Bigelow, S. Drucker, D. Fisher, and M. Meyer. Iterating between tools to create and edit visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 2017. doi: 10.1109/TVCG.2016.2598609 1, 2

[5] A. Carlier, M. Danelljan, A. Alahi, and R. Timofte. Deepsvg: A hierarchical generative network for vector graphics animation. In *Advances in Neural Information Processing Systems*, vol. 33, 2020. 2

[6] C. Chen, H. K. Bako, P. Yu, J. Hooker, J. Joyal, S. C. Wang et al. Visanatomy: An svg chart corpus with fine-grained semantic labels. *IEEE Transactions on Visualization and Computer Graphics*, 32(1):560–570, 2026. doi: 10.1109/TVCG.2025.3634263 3, 4, 7

[7] C. Chen, B. Lee, Y. Wang, Y. Chang, and Z. Liu. Mystique: Deconstructing svg charts for layout reuse. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):447–457, 2024. doi: 10.1109/TVCG.2023.3327354 1, 2, 3, 5

[8] J. Chen, L. Kong, H. Wei, C. Liu, Z. Ge, L. Zhao et al. Onechart: Purify the chart structural extraction via one auxiliary token. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 147–155, 2024. doi: 10.1145/3664647.3681167 1, 2

[9] Z.-Q. Cheng, Q. Dai, and A. G. Hauptmann. Chartreader: A unified framework for chart derendering and comprehension without heuristic rules. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 22145–22156, 2023. doi: 10.1109/ICCV51070.2023.02029 2

[10] J. Choi, S. Jung, D. G. Park, J. Choo, and N. Elmqvist. Visualizing for the non-visual: Enabling the visually impaired to use visualization. *Computer graphics forum*, 38(3):249–260, 2019. doi: 10.1111/cgf.13686 2, 3

[11] W. Cui, J. Wang, H. Huang, Y. Wang, C.-Y. Lin, H. Zhang et al. A mixed-initiative approach to reusing infographic charts. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):173–183, 2022. doi: 10.1109/TVCG.2021.3114856 1, 3

[12] J. Harder. *Creating Infographics with Adobe Illustrator: Volume 2*. 1

[13] J. Harper and M. Agrawala. Converting basic d3 charts into reusable style templates. *IEEE Transactions on Visualization and Computer Graphics*, 24(3):1274–1286, 2018. doi: 10.1109/TVCG.2017.2659744 3

[14] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. 4

[15] D. Jung, W. Kim, H. Song, J.-i. Hwang, B. Lee, B. Kim et al. Chartsense: Interactive data extraction from chart images. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pp. 6706–6717, 2017. doi: 10.1145/3025453.3025957 1, 2

[16] D. Li, H. Mei, Y. Shen, S. Su, W. Zhang, J. Wang et al. Echarts: a declarative framework for rapid construction of web-based visualization. *Visual Informatics*, 2(2):136–146, 2018. doi: 10.1016/j.visinf.2018.04.011 3

[17] T.-M. Li, M. Lukáč, M. Gharbi, and J. Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020. doi: 10.1145/3414685.3417871 2

[18] J. Luo, Z. Li, J. Wang, and C.-Y. Lin. Chartocr: Data extraction from charts images via a deep hybrid framework. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1916–1924, 2021. doi: 10.1109/WACV48630.2021.00196 1, 2

[19] X. Ma, Y. Zhou, X. Xu, B. Sun, V. Filev, N. Orlov et al. Towards layer-wise image vectorization. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16293–16302, 2022. doi: 10.1109/CVPR52688.2022.01583 1, 2

[20] D. Masson, S. Malacria, G. Casiez, and D. Vogel. Directgpt: A direct manipulation interface to interact with large language models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pp. 1–16, 2024. doi: 10.1145/3613904.3642462 5

[21] A. M. McNutt and R. Chugh. Integrated visualization editing via parameterized declarative templates. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–14, 2021. doi: 10.1145/3411764.3445356 3

[22] G. G. Méndez, M. A. Nacenta, and S. Vandenheste. ivolver: Interactive visual language for visualization extraction and reconstruction. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pp. 4073–4085, 2016. doi: 10.1145/2858036.2858435 3

[23] S. V. P, M. Yusuf Hassan, and M. Singh. Lineex: Data extraction from scientific line charts. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 6202–6210, 2023. doi: 10.1109/WACV56688.2023.00615 2

[24] J. Poco and J. Heer. Reverse-engineering visualizations: Recovering visual encodings from chart images. *Computer Graphics Forum*, 36(3):353–363, 2017. doi: 10.1111/cgf.13193 1, 2, 3

[25] P. Reddy, M. Gharbi, M. Lukac, and N. J. Mitra. Im2vec: Synthesizing vector graphics without vector supervision. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7338–7347, 2021. doi: 10.1109/CVPR46437.2021.00726 1, 2

[26] J. A. Rodriguez, A. Puri, S. Agarwal, I. H. Laradji, P. Rodriguez, S. Rajeswar et al. Starvector: Generating scalable vector graphics code from images and text. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16175–16186, 2025. doi: 10.1109/CVPR52734.2025.01508 1, 2, 6, 7, 13

[27] J. A. Rodriguez, H. Zhang, A. Puri, R. Pramanik, A. Feizi, P. Wichmann et al. Rendering-aware reinforcement learning for vector graphics generation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. 2, 4

[28] M. Savva, N. Kong, A. Chhajta, L. Fei-Fei, M. Agrawala, and J. Heer. Revision: Automated classification, analysis and redesign of chart images. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, 2011. doi: 10.1145/2047196.2047247 1, 2

[29] P. Selinger. Potrace: a polygon-based tracing algorithm, 2003. 2

[30] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. 2024. doi: 10.48550/arXiv.2402.03300 2, 4, 5

[31] L. S. Snyder and J. Heer. Divi: Dynamically interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):403–413, 2023. doi: 10.1109/TVCG.2023.3327172 1, 2, 3, 5

[32] Svgo: Node.js tool for optimizing svg files. `https://github.com/svg/svgo`. Accessed: 2026-03-04. 3

[33] B. Wang, F. Liu, C. Zhang, J. Chen, Y. Wu, S. Zhou et al. Llm4dsr: Leveraging large language model for denoising sequential recommendation. *ACM Transactions on Information Systems*. doi: 10.1145/3762182 2

[34] C. Wang, Y. Huo, Y. Gan, Q. He, Q. Meng, B. Li et al. Msrl: Scaling generative multimodal reward modeling via multi-stage reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 29410–29420, 2026. 13

[35] C. Wu, Z. Liang, Y. Ge, Q. Guo, Z. Lu, J. Wang et al. Plot2code: A comprehensive benchmark for evaluating multi-modal large language models in code generation from scientific plots. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 3006–3028, 2025. doi: 10.18653/v1/2025.findings-naacl.164 2

[36] R. Xia, H. Ye, X. Yan, Q. Liu, H. Zhou, Z. Chen et al. Chartx and chartvlm: A versatile benchmark and foundation model for complicated chart reasoning. *IEEE Transactions on Image Processing*, 34:7436–7447, 2025. doi: 10.1109/TIP.2025.3607618 2

[37] L. Xie, Y. Lin, C. Liu, H. Qu, and X. Shu. Datawink: Reusing and adapting svg-based visualization examples with large multimodal models. *IEEE Transactions on Visualization and Computer Graphics*, 32(1):824–834, 2026. doi: 10.1109/TVCG.2025.3634635 3

[38] X. Xing, Y. Guan, J. Zhang, D. Xu, and Q. Yu. Reason-svg: Hybrid reward rl for aha-moments in vector graphics generation. *arXiv preprint arXiv:2505.24499*, 4, 2025. 13

[39] X. Xing, J. Hu, G. Liang, J. Zhang, D. Xu, and Q. Yu. Empowering llms to understand and generate complex vector graphics. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19487–19497, 2025. doi: 10.1109/CVPR52734.2025.01815 2

[40] Y. Yang, W. Cheng, S. Chen, X. Zeng, F. Yin, J. Zhang et al. Omnisvg: A unified scalable vector graphics generation model. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. 6, 7, 13

[41] S. Yin, C. Fu, S. Zhao, K. Li, X. Sun, T. Xu et al. A survey on multimodal large language models. *National Science Review*, 11(12):nwae403, 2024. doi: 10.1093/nsr/nwae403 1

[42] X. Zhao, X. Luo, Q. Shi, C. Chen, S. Wang, Z. Liu et al. Chartcoder: Advancing multimodal large language model for chart-to-code generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7333–7348, 2025. doi: 10.18653/v1/2025.acl-long.363 1, 2, 6, 7, 14