

000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 TOWARDS THE GENERATION OF STRUCTURED SCIENTIFIC VECTOR GRAPHICS WITH LARGE LANGUAGE MODELS

Anonymous authors

Paper under double-blind review

ABSTRACT

We address the challenge of automatically visualizing scientific explanations. While prior work has explored large language model (LLM)-based vector graphic generation, existing approaches often overlook structural correctness, a key requirement for valid scientific diagrams. To achieve structurally correct generation, we make three key contributions. First, we introduce SSVG-Bench, a novel benchmark for evaluating the generation of Structured Scientific Vector Graphics. Unlike conventional visual similarity metrics, SSVG-Bench employs task-specific structural analysis for accurate evaluation, and it supports three vector formats: TikZ, SVG, and EPS. Second, we conduct an extensive benchmarking and analysis, revealing key findings such as the crucial role of LLM reasoning in ensuring structural validity. Third, we propose LLM-Oriented Orchestration Prompting (LOOP), a new prompting method that leverages LLMs' reasoning potential by combining familiar subtasks. Experiments demonstrate substantial improvements over existing prompting techniques, suggesting promising directions for scientific diagram generation. We will release our code and benchmark upon acceptance.

1 INTRODUCTION

In this paper, we address the challenge of automatically visualizing scientific explanations. Because scientific explanations are often abstract and complex, they can be difficult to grasp from text alone. To promote intuitive understanding, visualization is widely used in contexts such as science textbooks and research papers. These observations suggest that automated visualization could have a significant impact.

Recent research on scientific diagram generation has investigated vector graphics generation with large language models (LLMs) (Belouadi et al., 2024a;b; Zhang et al., 2025; Belouadi et al., 2025). As vector graphics encode visual content in structured text form, they can be directly produced by LLMs. Given that LLMs are capable of capturing scientific concepts in depth and encoding complex constraints, they are particularly promising for this task.

However, existing methods have overlooked a crucial aspect of scientific diagrams: structural correctness. To illustrate its importance, we present a physics scenario in Figure 1. Here, the visualization must strictly satisfy structural constraints: the object should remain in contact with the inclined plane, and three arrows must be shown, one vertical to the ground, one perpendicular to the plane, and one parallel to the plane. An existing method, namely a fine-tuned LLM for generating vector code (Belouadi et al., 2025), fails to meet these constraints. Although it prioritizes visual plausibility, the lack of structural enforcement ultimately leads to invalid scientific diagrams.

Towards the generation of scientific vector graphics with structural correctness, we make three main contributions. Our first contribution is a new benchmark for the generation of Structured Scientific Vector Graphics, named SSVG-Bench. It targets two foundational domains: plane geometry and molecular structure. The plane geometry task involves generating geometric figures from textual descriptions of theorems or construction methods, while the molecular structure task requires correctly generating a molecule's structure from its IUPAC name, which encodes structural information. These domains are representative of broader applications: the ability to generate plane geometry structures is fundamental for physics illustrations, engineering diagrams, and architec-

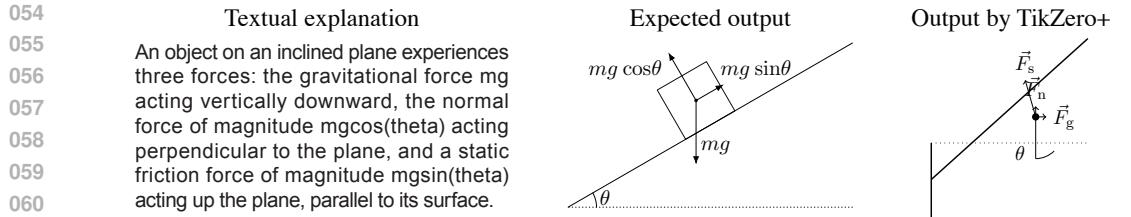


Figure 1: An example highlighting the importance of structural correctness: the object should remain in contact with the inclined plane, and three arrows must be shown, one vertical to the ground, one perpendicular to the plane, and one parallel to the plane. TikZero+ (Belouadi et al., 2025), a fine-tuned LLM for generating vector code, does not meet these structural constraints.

tural blueprints, while generating molecular structures requires correctly producing graph structures, which are important in various fields such as algorithm flowcharts, circuit designs, and biological pathways. The most significant feature of SSVG-Bench is its evaluation method. Conventional visual similarity metrics are insufficient for assessing structural correctness, as even minor visual changes can cause structural inconsistencies. To address this, we provide task-specific Python scripts that analyze the structure of the generated outputs, enabling accurate evaluation of structural correctness. In addition, SSVG-Bench supports three vector formats: TikZ, SVG, and EPS, which allows us to examine performance across formats.

Our second contribution is a comprehensive benchmarking and analysis of existing models using SSVG-Bench, which yields several key findings. First, LLMs fine-tuned on existing vector graphic generation datasets to produce vector code rarely generate structurally valid vector graphics. Second, we show that the reasoning capabilities of LLMs are essential for generating structurally correct vector graphics. Finally, although prior work has mainly focused on the TikZ format, our results demonstrate that the SVG format is better suited for LLM reasoning.

As our third contribution, we propose a new prompting technique, LLM-Oriented Orchestration Prompting (LOOP), to further enhance the reasoning capabilities of LLMs. Recent LLMs are explicitly trained to perform step-by-step reasoning and can solve complex tasks, but it has been reported that they struggle with tasks not encountered during training (Shojaee et al., 2025; Malek et al., 2025). Since LLMs are not explicitly trained to generate vector graphics from scientific explanations, they cannot fully utilize their reasoning potential when the task is presented in its original form. To fully leverage their reasoning abilities, we design a prompt that explicitly guides LLMs to perform LLM-friendly subtasks such as information extraction and relationship extraction, which enables them to generate vector graphics with correct structure. Our experiments demonstrate that LOOP achieves substantially better performance than existing prompting methods.

Our contributions can be summarized as follows.

- **Dataset:** We introduce SSVG-Bench, a new benchmark for structured scientific vector graphics generation, including scripts that verify structural correctness.
- **Benchmarking and analysis:** Our analysis shows that previous fine-tuned models cannot produce structurally correct graphics, that LLM reasoning capabilities are essential for ensuring structural correctness, and that the SVG format is well-suited for such reasoning.
- **Method:** We propose LOOP, a new prompting method that enhances LLM reasoning by solving LLM-friendly sub-problems step by step.

2 RELATED WORKS

Scientific vector graphic generation benchmarks. Vector graphics are gaining attention as an image format well-suited for LLMs, as they are represented in text and can be directly input or output by LLMs without requiring a vision adapter. Several benchmarks have been developed for general vector graphics generation, such as SVGEditBench (Nishina & Matsui, 2024), SVG Taxonomy (Xu & Wall, 2024), and VGBench (Zou et al., 2024). Additionally, several benchmarks for visualizing scientific data have been developed, such as MatPlotBench (Yang et al., 2024), PandasPlotBench (Galimzyanov et al., 2025), and ChartMimic (Yang et al., 2025).

108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
Table 1: Comparison of previous benchmarks with our SSVG-Bench.

Benchmark	Scientific?	# Evaluation data	Evaluation method	Vector format
SVGEEditBench	No	1,366	Visual similarity	SVG
SVG Taxonomy	No	400	Exact match	SVG
VGBench	No	5,845	Visual similarity	TikZ, SVG, Graphviz
DaTikZ v1	Yes	1,000	Visual/code similarity, Human evaluation	TikZ
DaTikZ v2	Yes	1,000	Visual/code similarity, Human evaluation	TikZ
DaTikZ v3	Yes	1,000	Visual/code similarity, Human evaluation	TikZ
ScImage	Yes	404	Human evaluation	TikZ
DiagramGenBench	Yes	470	Visual/code similarity, Human evaluation	TikZ, DOT
SSVG-Bench (ours)	Yes	1,230	Structural analysis scripts	TikZ, SVG, EPS

Motivated by this trend, some benchmarks have also been created specifically for scientific vector graphics generation. DaTikZ v1 (Belouadi et al., 2024a) collects TikZ code and corresponding captions from sources such as arXiv papers. DaTikZ v2 (Belouadi et al., 2024b) collects pairs of hand-drawn sketches and TikZ code to evaluate the performance of sketch-to-TikZ conversion. DaTikZ v3 (Belouadi et al., 2025) further extends DaTikZ v1 and v2. ScImage (Zhang et al., 2025) employs synthetic data to analyze scientific vector generation in terms of attributes, numbers, and spatial dimensions. DiagramGenBenchmark (Wei et al., 2025) provides diagram structures in TikZ as well as graph structures in the DOT language.

However, these benchmarks generally do not focus on the structural correctness of the generated graphics. We present a comparison of these benchmarks with our SSVG-Bench in Table 15. The most significant feature of SSVG-Bench is its evaluation method. Visual similarity-based and code similarity-based metrics are insufficient for determining whether the structure of a generated graphic truly reflects the intended structure. **Human evaluations, while informative, are not scalable and are subject to variability and inconsistency across evaluators.** As a result, it is difficult to increase the number of models evaluated or to use diverse experimental conditions. To address these issues, we provide task-specific Python scripts that analyze the structure of the generated output and determine its correctness. Our evaluation framework offers precise, objective, and consistent assessments of performance. In addition, SSVG-Bench supports three vector formats: TikZ, SVG, and EPS, which allows us to examine performance across formats.

Scientific vector graphic generation methods. Based on the benchmarks, several scientific vector graphic generation methods have been proposed. AutomaTikZ (Belouadi et al., 2024a) is designed for TikZ generation by fine-tuning Llama (Touvron et al., 2023) to output TikZ code from captions. It leverages CLIP features (Radford et al., 2021), derived from captions, to further improve visual alignment. DeTikZify (Belouadi et al., 2024b) converts hand-drawn sketches into TikZ code by combining a vision encoder (SigLIP (Zhai et al., 2023)) with an LLM such as Llama. TikZero (Belouadi et al., 2025) addresses the scarcity of paired caption-TikZ data by leveraging readily available captioned raster images for training. DiagramAgent (Wei et al., 2025) enables the creation of complex diagrams by coordinating multiple agents. Despite their innovations, these models are primarily trained to predict output code and are not designed to guarantee structural correctness.

LLM prompting methods. The reasoning capabilities of LLMs can be elicited through effective prompting. Seminal work on Chain-of-Thought (CoT) demonstrated that allowing models to generate intermediate reasoning steps dramatically improves multi-step reasoning (Wei et al., 2022). Follow-ups revealed that simply appending “Let’s think step by step” can unlock zero-shot reasoning (Kojima et al., 2022), and that sampling multiple reasoning paths and selecting the most consistent answer (“self-consistency”) further boosts accuracy (Wang et al., 2023b). Beyond linear reasoning, researchers decomposed problems via least-to-most prompting (Zhou et al., 2023), introduced plan-first then execute strategies such as Plan-and-Solve (Wang et al., 2023a), and proposed Step-Back prompting, which encourages the model to abstract away from the immediate problem and reason at a higher conceptual level before providing a solution (Zheng et al., 2024). In this paper, we introduce a novel prompting method designed for scientific vector graphic generation.

Plane geometry		
162	Vector graphic	
163		
164	Input text	
165	The exterior angle bisector in A intersects the extended side BC in E, the exterior angle bisector in B intersects the extended side AC in D and the exterior angle bisector in C intersects the extended side AB in F.	
166	The three points of intersection between the exterior angle bisectors and the extended triangle sides D, E, F are collinear, that is they lie on a common line.	
167		
168		
169		
170		
171		
172		
173		
174		
175		
176		
177		
178		
179	To state the theorem, suppose that ABCD and AB'C'D' are two squares with common vertex A. Let E and G be the midpoints of B'D and D'B respectively, and let F and H be the centers of the two squares. Then the theorem states that the quadrilateral EFGH is a square as well.	
180		
181		
182		
183		
184		
185		
186		
187		
188		
189		
190		
191		
192		
193		
194	The Apollonius point of a triangle is defined as follows. Let $\triangle ABC$ be any given triangle. Let the excircles of $\triangle ABC$ opposite to the vertices A, B, C be EA, EB, EC respectively. Let E be the circle which touches the three excircles EA, EB, EC such that the three excircles are within E. Let A', B', C' be the points of contact of the circle E with the three excircles. The lines AA', BB', CC' are concurrent. The point of concurrence is the Apollonius point of $\triangle ABC$.	
195		
196		
197		
198		
199		
200	Vector graphic	
201		
202	Input text	
203	the molecular structure of the compound with the IUPAC name 4-butyl-2,6-dimethylmorpholine	
204		
205	the molecular structure of the compound with the IUPAC name 6-methoxy-4-methyl-5-phenylmethoxyquinolin-8-amine	
206		
207	the molecular structure of the compound with the IUPAC name 4-[[2,4-diamino-5-[(4-carboxyphenyl)diazenyl]phenyl]diazenyl]benzoic acid	
208		
209		
210		
211		
212		
213		
214		
215		

Figure 2: Examples in SSVG-Bench.¹

3 SSVG-BENCH

To evaluate whether LLMs can generate vector graphics with structural correctness, we introduce SSVG-Bench. SSVG-Bench covers two key topics: 1) plane geometry and 2) molecular structure. For each topic, we also develop an automatic evaluation framework. Figure 2 presents some examples, and Table 2 summarizes respective statistics. We provide a detailed explanation of these components in the following sections.

3.1 PLANE GEOMETRY

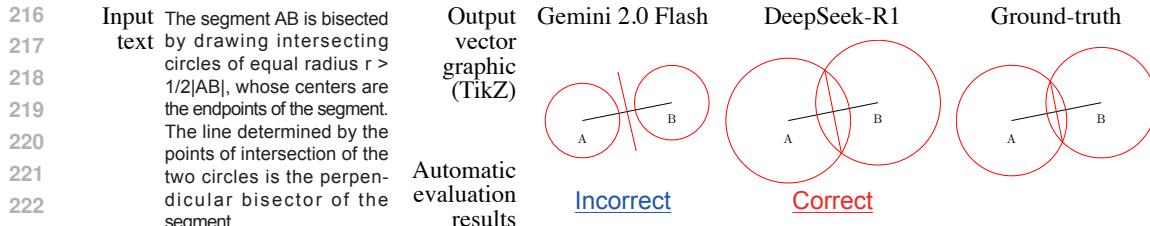
This task involves generating visualizations of plane geometric figures from textual descriptions, translating explanations of theorems or constructions into precise visual representations. To succeed, LLMs must accurately interpret spatial relationships such as “intersection”, “tangent”, and “perpendicular”, as well as uniquely determined constructions like “angle bisectors” and “excircles.” This task can evaluate visualization capabilities that are important in a variety of applications, including physics illustrations, engineering diagrams, and architectural blueprints.

Table 2: Statistics of SSVG-Bench. The number of elements is counted based on the SVG.

Plane geometry	
Total number of input texts	110
Average number of elements per vector graphic	
- line	3.96
- circle	2.77
- ellipse	0.03
- polygon	1.65
- polyline	0.01

Molecular structure	
Total number of input texts	300
Average number of elements per vector graphic	
- line	32.45
- circle	30.38

¹The input texts and vector graphics are from (Wikipedia contributors, 2025b;f; 2024; Kmhkmh, 2019a; 2015; Krishnachandranvn, 2012; National Center for Biotechnology Information, 2025f;l;g)



270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323
 tion of a molecule, and by interpreting it, the molecular structure can be reconstructed. For example, the IUPAC name shown on the left side of Figure 2 is “4-butyl-2,6-dimethylmorpholine,” which indicates that a butyl group is attached to the nitrogen atom at the 4-position (shown in blue) of the morpholine ring (a six-membered ring), and that methyl groups are attached at the 2- and 6-positions. In this way, the molecular structure can be accurately restored from the IUPAC name. It is essential to correctly identify the types and numbers of atoms involved, with particular importance placed on the graph structure formed by atomic bonds. This task can evaluate the graph structure visualization capabilities of LLMs, which are important in various fields such as algorithm flowcharts, circuit design, and biological pathways.

We obtained pairs of IUPAC names and molecular structures from PubChem⁴. The structural data is stored in JSON format, and we developed a Python script to convert this information into vector graphics automatically. Using this script, we generated ground-truth data. We collected 300 molecules in total, with 50 examples each for molecules with fewer than 20 elements (atoms plus bonds), 21-40, 41-60, 61-80, 81-100, and more than 100 elements.

To enable automatic evaluation, we implemented a Python-based evaluation tool. The generated vector graphic is converted into a molecular graph, where nodes represent atoms and edges represent bonds. We then check for “graph isomorphism” between the generated structure and the ground-truth to automatically assess correctness. Graph isomorphism refers to the problem of determining whether two graphs are structurally identical, meaning their nodes and edges can be matched one-to-one while preserving connectivity. We use the NetworkX library to solve this problem. To simplify the task, we do not consider bond order. We present an example of automatic evaluation results in Figure 4. Although the output of GPT-5 appears visually different from the ground-truth, it is considered correct based on graph-theoretic equivalence.

3.3 MULTIPLE VECTOR FORMATS

There are various types of vector graphic formats. To analyze performance differences across formats, we use three types: TikZ, SVG, and EPS. For the plane geometry task, since the vector graphics collected from Wikipedia are in SVG format, we developed Python scripts to automatically convert SVG to TikZ and EPS, thereby generating ground-truth data. For the molecular structure task, we generate vector graphics in each format directly from molecular structure data stored in JSON files, using custom Python scripts to automate the process. When evaluating LLMs, we add instructions to the prompt to generate output in a specific format. This approach allows us to explore which vector format is most suitable for LLMs. If the syntax is incorrect, it will result in a compilation error (for TikZ and EPS) or a parsing error (for SVG). In such cases, the output is considered incorrect. Therefore, LLMs must strictly adhere to the syntax of each format. Our SSVG-Bench dataset consists of two tasks and three vector formats, comprising a total of 1,230 text-vector graphic pairs.

4 BENCHMARKING AND ANALYSIS

Using SSVG-Bench, we evaluate whether recent models can generate structurally correct vector graphics. Our experiments utilize two fine-tuned models: AutomaTikZ (Belouadi et al., 2024a) and TikZero+ (Belouadi et al., 2025). As these models are trained to generate TikZ, we evaluate them exclusively on TikZ. We also evaluate recent general-purpose LLMs, including DeepSeek-V3, R1, V3.1 (DeepSeek, 2025), Claude Opus 4.1 (Anthropic, 2025), Gemini 2.0 Flash, 2.5 Flash, 2.5 Pro (Google, 2025), o4-mini, GPT-4.1, and GPT-5 (OpenAI, 2025). The detailed prompts used for evaluation are provided in the Appendix. The overall performance is shown in Table 4, with a detailed analysis provided below.

Limitations of fine-tuned models. Our benchmarking with SSVG-Bench reveals that models fine-tuned to generate TikZ code (AutomaTikZ and TikZero+) seldom produce structurally valid outputs. This highlights a limitation: simply training to predict TikZ code from captions is insufficient to generate correct scientific figures.

⁴<https://pubchem.ncbi.nlm.nih.gov/>

324 Table 4: Accuracies on SSVG-Bench (%). The fill colors correspond to the values.
325

326 Model	327 Plane geometry			328 Molecular structure			329 Average
	330 TikZ	331 SVG	332 EPS	333 TikZ	334 SVG	335 EPS	
336 Fine-tuned models							
337 AutomaTikZ	338 0.0	339 -	340 -	341 0.0	342 -	343 -	344 0.0
345 TikZero+	346 0.9	347 -	348 -	349 0.0	350 -	351 -	352 0.2
353 Non-reasoning models							
354 DeepSeek-V3	355 10.0	356 5.5	357 7.3	358 5.3	359 3.3	360 3.0	361 4.9
362 DeepSeek-V3.1	363 11.8	364 6.4	365 9.1	366 6.3	367 3.7	368 3.3	369 5.7
370 Claude Opus 4.1	371 14.5	372 12.7	373 20.9	374 24.3	375 26.0	376 16.0	377 20.5
378 Gemini 2.0 Flash	379 7.3	380 5.5	381 1.8	382 6.0	383 3.7	384 0.7	385 3.8
386 Gemini 2.5 Flash	387 12.7	388 9.1	389 5.5	390 22.7	391 11.3	392 14.3	393 14.2
394 GPT-4.1	395 10.9	396 10.0	397 14.5	398 19.0	399 15.0	400 13.7	401 14.8
402 GPT-5 Chat	403 12.7	404 10.0	405 7.3	406 16.0	407 14.3	408 11.0	409 12.8
410 Reasoning models							
411 DeepSeek-R1	412 28.2	413 40.9	414 39.1	415 18.3	416 20.0	417 19.7	418 23.8
419 DeepSeek-V3.1 reasoning	420 23.6	421 39.1	422 27.3	423 31.0	424 7.3	425 20.7	426 22.4
427 Claude Opus 4.1 thinking	428 20.0	429 23.6	430 17.3	431 26.7	432 27.7	433 23.3	434 24.4
435 Gemini 2.5 Flash reasoning	436 30.0	437 55.5	438 41.8	439 32.0	440 39.3	441 34.7	442 37.2
444 Gemini 2.5 Pro	445 50.0	446 62.7	447 56.4	448 41.3	449 63.3	450 57.3	451 54.6
453 o4-mini	454 48.2	455 62.7	456 55.5	457 33.3	458 42.7	459 39.0	460 42.9
463 GPT-5	464 54.5	465 75.5	466 66.4	467 52.3	468 55.7	469 49.7	470 56.0

347 Table 5: Comparison of averages for models with reasoning enabled vs. disabled. DeepSeek-V3.1,
348 Claude Opus 4.1, Gemini 2.5 Flash, and GPT-5 are considered.
349

350 Model	351 Plane geometry			352 Molecular structure			353 Average
	354 TikZ	355 SVG	356 EPS	357 TikZ	358 SVG	359 EPS	
360 Reasoning disabled	361 13.0	362 9.5	363 10.7	364 17.3	365 13.8	366 11.2	367 13.3
368 Reasoning enabled	369 32.0	370 48.4	371 38.2	372 35.5	373 32.5	374 32.1	375 35.0

356 **Importance of reasoning.** To evaluate the effectiveness of reasoning, we consider models where
357 reasoning can be toggled on and off, and we report results for both configurations in Table 4. The
358 models compared are as follows: DeepSeek-V3.1 vs. DeepSeek-V3.1 reasoning, Claude Opus 4.1
359 vs. Claude Opus 4.1 thinking, Gemini 2.5 Flash vs. Gemini 2.5 Flash reasoning, and GPT-5 Chat
360 vs. GPT-5. Table 5 compares the averages of models with reasoning disabled and enabled. Enabling
361 reasoning significantly improves performance. These results demonstrate that enabling reasoning
362 plays a crucial role in generating structured vector graphics.
363

364 **Impact of vector format.** Focusing on the two best-
365

366 Table 3: Google search hits for format-specific keywords (November 2025).

	TikZ	SVG	EPS
Keyword “tikzpicture” “</svg>” “showpage” and “moveto”			
# Hits	272K	524M	20.7K

370 One possible explanation is the scale of resources used for training. Existing research (Zhu et al.,
371 2024) has demonstrated that the reasoning capabilities of LLMs tend to be weaker in low-resource
372 languages (e.g., Bengali or Thai) compared to high-resource languages (e.g., English). Similarly,
373 it is possible that LLMs are not well-suited for reasoning with TikZ and EPS, because they may
374 be considered “low-resource languages.” To test this hypothesis, it would be necessary to examine
375 the training data, but the datasets used to train Gemini 2.5 Pro and GPT-5 remain unspecified. We
376 instead query Google with format-specific keywords and record the number of hits, since much of
377 the training data for LLMs is derived from internet sources. Table 3 presents the number of hits

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

Table 6: Comparison between our LOOP and other zero-shot prompting methods.

Model	Plane geometry			Molecular structure			Average
	TikZ	SVG	EPS	TikZ	SVG	EPS	
Gemini 2.5 Pro	50.0	62.7	56.4	41.3	63.3	57.3	54.6
w/ Zero-shot CoT	39.1	66.4	61.8	47.7	63.0	58.7	56.3
w/ Plan-and-Solve	39.1	69.1	66.4	41.3	58.7	55.0	53.4
w/ Step-Back	33.6	64.5	59.1	40.7	54.3	56.3	51.0
w/ LOOP (ours)	65.5	80.9	62.7	47.7	64.7	67.7	62.6
GPT-5	54.5	75.5	66.4	52.3	55.7	49.7	56.0
w/ Zero-shot CoT	58.2	80.0	75.5	53.0	52.0	49.3	56.7
w/ Plan-and-Solve	61.8	77.3	70.9	52.3	50.3	50.7	56.2
w/ Step-Back	55.5	75.5	72.7	50.7	51.7	48.3	55.0
w/ LOOP (ours)	70.0	80.0	77.3	55.0	57.3	54.3	61.0

obtained from searches using format-specific keywords. Compared to TikZ and EPS, SVG yielded a much higher number of hits, suggesting that SVG constitutes a high-resource format.

5 LLM-ORIENTED ORCHESTRATION PROMPTING (LOOP)

Through our analysis, we find that LLM reasoning plays a crucial role. Building on this finding, we propose a method to enhance their reasoning capabilities. Previous research has shown that carefully crafted prompts can significantly improve LLM reasoning, even in zero-shot settings. For instance, zero-shot CoT prompting (Kojima et al., 2022), which simply instructs the model with “Let’s think step by step,” has been shown to improve performance. In this work, we introduce a novel zero-shot prompting method, termed LOOP. While recent LLMs are explicitly trained for step-by-step reasoning and can solve complex tasks, their performance often degrades on tasks outside their training distribution (Shojaee et al., 2025; Malek et al., 2025). Because LLMs are not inherently trained to generate vector graphics from scientific explanations, their reasoning potential remains under-utilized. The core idea of LOOP is to instruct LLMs to generate vector graphics by orchestrating LLM-familiar tasks. Specifically, we use the following tasks: 1) information extraction, 2) relationship extraction, 3) mathematical reasoning, and 4) code generation. The first three tasks provide the information necessary for visualization, while the final task produces the vector graphics. Information and relationship extraction are long-standing tasks in the field of natural language processing, whereas mathematical reasoning and code generation are areas where recent LLMs have made significant progress. By orchestrating these familiar tasks, LOOP aims to accelerate and enhance LLM reasoning capabilities.

Specifically, for the plane geometry visual task, we use the following prompt:

“Let’s think step by step, following this workflow: 1. Information extraction: describe the necessary elements. 2. Relationship extraction: describe their relationships. 3. Mathematical reasoning: compute the attributes of each element so that they satisfy those relationships. 4. Code generation: generate the TikZ.”

For the molecular structure visualization task, we exclude mathematical reasoning, as it is not necessary. We use the following prompt:

“Let’s think step by step, following this workflow: 1. Information extraction: describe the functional groups and substituents present in the IUPAC name. 2. Relationship extraction: describe how these groups are connected. 3. Code generation: generate the TikZ.”

The word “TikZ” is replaced with the target vector format.

5.1 EVALUATION

We evaluate the performance of LOOP by applying it to the two best-performing models: Gemini 2.5 Pro and GPT-5. For comparison, we use the following three zero-shot prompting techniques:

432	Input text	433 In geometry, the six circles theorem relates to a chain of six circles together with a triangle, such that each circle is tangent to two sides of the triangle and also to the preceding circle in the chain. The chain closes, in the sense that the sixth circle is always tangent to the first circle.	434 Output vector graphic (SVG)	435 Gemini 2.5 Pro	436 Gemini 2.5 Pro w/ LOOP	437 Ground-truth
438						

440 Input text Let O be a circle of radius R .
441 Let O_1, O_2, O_3, O_4 be (in
442 that order) four non-intersecting
443 circles that lie inside O and tangent to it. Denote by t_{ij} the
444 length of the exterior common
445 bitangent of the circles O_i, O_j .
446 Then $t_{12} \cdot t_{34} + t_{14} \cdot t_{23} = t_{13} \cdot t_{24}$.
447

448 Figure 5: Examples demonstrating improvements from our prompting method on the plane geometry SVG generation task. In the bottom example, since there are two possible exterior common bitangents for each pair of circles, either line is considered correct.⁵

452	453 Input text	454 the molecular structure of the compound with the IUPAC name 3-[(4-(4-methylphenyl)piperazin-1-yl)methyl]-5-[(2-methylquinazolin-4-yl)oxymethyl]-1,3,4-oxadiazole-2-thione	455 Output vector graphic (SVG)	456 Gemini 2.5 Pro	457 Gemini 2.5 Pro w/ LOOP	458 Ground-truth
459	460 Input text	461 the molecular structure of the compound with the IUPAC name (6-chloro-4-methyl-2-oxochromen-7-yl) (2R)-2-[(4-methylphenyl)sulfonylamino]pentanoate	462 Output vector graphic (SVG)			
463						

464 Figure 6: Examples demonstrating improvements from our prompting method on the molecular structure SVG generation task.⁶

- 469 • Zero-shot CoT Prompting (Kojima et al., 2022): “*Let’s think step by step.*”
- 470 • Plan-and-Solve Prompting (Wang et al., 2023a): “*Let’s first understand the problem and devise a plan to solve the problem. Then, let’s carry out the plan and solve the problem step by step.*”
- 471 • Step-Back Prompting (Zheng et al., 2024): “*Let’s think step by step, following this workflow: 1. Step back and pose higher-level, abstract questions. 2. Answer those questions. 3. Generate the TikZ.*”

472 The results are shown in Table 6. While some methods degrade performance, the proposed method provides the highest performance gain. This result clearly demonstrates that the proposed method can efficiently leverage the potential of LLMs.

473 We present examples in Figures 5 and 6 showing that LOOP enables the generation of structurally 474 correct vector graphics. Without LOOP, LLMs fail to accurately produce complex structures. LOOP 475 encourages deeper reasoning, which leads to the generation of structurally correct vector graphics.

476
477
478
479
480
481
482
483
484
485
⁵The input texts and the ground-truth vector graphics are from (Wikipedia contributors, 2022; 2025e; Rocchini, 2010; Kmkmh, 2018).

486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
15010
15011
15012
15013
15014
15015
15016
15017
15018
15019
15020
15021
15022
15023
15024
15025
15026
15027
15028
15029
15030
15031
15032
15033
15034
15035
15036
15037
15038
15039
15040
15041
15042
15043
15044
15045
15046
15047
15048
15049
15050
15051
15052
15053
15054
15055
15056
15057
15058
15059
15060
15061
15062
15063
15064
15065
15066
15067
15068
15069
15070
15071
15072
15073
15074
15075
15076
15077
15078
15079
15080
15081
15082
15083
15084
15085
15086
15087
15088
15089
15090
15091
15092
15093
15094
15095
15096
15097
15098
15099
150100
150101
150102
150103
150104
150105
150106
150107
150108
150109
150110
150111
150112
150113
150114
150115
150116
150117
150118
150119
150120
150121
150122
150123
150124
150125
150126
150127
150128
150129
150130
150131
150132
150133
150134
150135
150136
150137
150138
150139
150140
150141
150142
150143
150144
150145
150146
150147
150148
150149
150150
150151
150152
150153
150154
150155
150156
150157
150158
150159
150160
150161
150162
150163
150164
150165
150166
150167
150168
150169
150170
150171
150172
150173
150174
150175
150176
150177
150178
150179
150180
150181
150182
150183
150184
150185
150186
150187
150188
150189
150190
150191
150192
150193
150194
150195
150196
150197
150198
150199
150200
150201
150202
150203
150204
150205
150206
150207
150208
150209
150210
150211
150212
150213
150214
150215
150216
150217
150218
150219
150220
150221
150222
150223
150224
150225
150226
150227
150228
150229
150230
150231
150232
150233
150234
150235
150236
150237
150238
150239
150240
150241
150242
150243
150244
150245
150246
150247
150248
150249
150250
150251
150252
150253
150254
150255
150256
150257
150258
150259
150260
150261
150262
150263
150264
150265
150266
150267
150268
150269
150270
150271
150272
150273
150274
150275
150276
150277
150278
150279
150280
150281
150282
150283
150284
150285
150286
150287
150288
150289
150290
150291
150292
150293
150294
150295
150296
150297
150298
150299
150300
150301
150302
150303
150304
150305
150306
150307
150308
150309
150310
150311
150312
150313
150314
150315
150316
150317
150318
150319
150320
150321
150322
150323
150324
150325
150326
150327
150328
150329
150330
150331
150332
150333
150334
150335
150336
150337
150338
150339
150340
150341
150342
150343
150344
150345
150346
150347
150348
150349
150350
150351
150352
150353
150354
150355
150356
150357
150358
150359
150360
150361
150362
150363
150364
150365
150366
150367
150368
150369
150370
150371
150372
150373
150374
150375
150376
150377
150378
150379
150380
150381
150382
150383
150384
150385
150386
150387
150388
150389
150390
150391
150392
150393
150394
150395
150396
150397
150398
150399
150400
150401
150402
150403
150404
150405
150406
150407
150408
150409
150410
150411
150412
150413
150414
150415
150416
150417
150418
150419
150420
150421
150422
150423
150424
150425
150426
150427
150428
150429
150430
150431
150432
150433
150434
150435
150436
150437
150438
150439
150440
150441
150442
150443
150444
150445
150446
150447
150448
150449
150450
150451
150452
150453
150454
150455
150456
150457
150458
1504

486 6 CONCLUSION
 487

488 In this paper, we tackled the problem of scientific vector graphics generation using LLMs. Specifi-
 489 cally, aiming for structurally correct vector graphics generation, we made three contributions. First,
 490 we introduced a new benchmark that assesses the structural correctness of generated graphics using
 491 structural analysis scripts. Second, we conducted a comprehensive benchmarking study and pro-
 492 vided detailed analyses based on this benchmark. Third, we proposed a novel prompting technique
 493 that accelerates LLM inference and significantly improves performance.
 494

495 REFERENCES
 496

497 Ag2gaeh. Construction of a perpendicular line segment bisector. <https://commons.wikimedia.org/wiki/File:Mittelsenkr-ab-konstr-e.svg>, 2021. Licensed
 498 under CC BY-SA 4.0; modified by the authors.
 499

500 Anthropic. Models overview, 2025. <https://docs.claude.com/en/docs/about-claude/models/overview>.
 501

502 Jonas Belouadi, Anne Lauscher, and Steffen Eger. Automatikz: Text-guided synthesis of sci-
 503 entific vector graphics with tikz. In *International Conference on Learning Representations (ICLR)*,
 504 2024a.

505 Jonas Belouadi, Simone Ponzetto, and Steffen Eger. Detikzify: Synthesizing graphics programs
 506 for scientific figures and sketches with tikz. *Advances in Neural Information Processing Systems*
 507 (*NeurIPS*), 37:85074–85108, 2024b.

508 Jonas Belouadi, Eddy Ilg, Margret Keuper, Hideki Tanaka, Masao Utiyama, Raj Dabre, Steffen
 509 Eger, and Simone Paolo Ponzetto. Tikzero: Zero-shot text-guided graphics program synthesis. In
 510 *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

511 Braindrain0000. Graphic representing medial triangles gfdl. https://commons.wikimedia.org/wiki/File:Medial_Triangle.svg, 2006. Licensed under CC BY/SA 3.0; modi-
 512 fied by the authors.
 513

513 DeepSeek. Models & pricing, 2025. https://api-docs.deepseek.com/quick_start/pricing.
 514

515 Timur Galimzyanov, Sergey Titov, Yaroslav Golubev, and Egor Bogomolov. Drawing pandas: A
 516 benchmark for llms in generating plotting code. In *IEEE/ACM International Conference on Min-
 517 ing Software Repositories (MSR)*, pp. 503–507, 2025.

518 Google. Gemini models, 2025. <https://ai.google.dev/gemini-api/docs/models>.
 519

520 Gustavb. Butterfly theorem. https://commons.wikimedia.org/wiki/File:Butterfly_theorem.svg, 2006. Licensed under CC BY/SA 3.0; modified by the authors.
 521

522 Inductiveload. Diagram to shown the construction of the incenter (blue, i), the incircle (blue), the ex-
 523 centres (orange, ja,jb,jc and the excircles (orange). https://commons.wikimedia.org/wiki/File:Incircle_and_Excircles.svg, 2007a. Released into the public domain
 524 under CC0 1.0; modified by the authors.
 525

525 Inductiveload. Extouch_triangle_and_nagel_point.svg — diagram of the extouch triangle and nagel
 526 point illustrating splitters in a triangle. https://commons.wikimedia.org/wiki/File:Extouch_Triangle_and_Nagel_Point.svg, 2007b. Released into the public
 527 domain under CC0 1.0; modified by the authors.
 528

528 Inductiveload. Diagram to shown the construction of the intouch, or contact, triangle (red) and the
 529 gergonne point (green) of a triangle (black). the blue circle is the incircle, and the blue point, i, is
 530 the incentre of the original triangle. https://commons.wikimedia.org/wiki/File:Intouch_Triangle_and_Gergonne_Point.svg, 2007c. Released into the public
 531 domain under CC0 1.0; modified by the authors.
 532

540 Ixnay. An animated gif of the construction of a bisection (with ruler and compass). https://en.wikipedia.org/wiki/File:Bisection_construction.gif, 2007. Released into
 541 the public domain under CC0 1.0; modified by the authors.
 542

543 Kmhmh. hadwiger-finsler theorem about squares. https://commons.wikimedia.org/wiki/File:Hadwiger_finsler_theorem.svg, 2015. Licensed under CC BY 4.0;
 544 modified by the authors.
 545

546 Kmhmh. Antiparallel symmedian.svg — diagram showing symmedian and antiparallel segments.
 547 https://commons.wikimedia.org/wiki/File:Lemoine_punkt.svg, 2016. Li-
 548 censed under CC BY 4.0; modified by the authors.
 549

549 Kmhmh. Casey new1a.svg. https://commons.wikimedia.org/wiki/File:Casey_new1a.svg, 2018. Licensed under CC BY 4.0; modified by the authors.
 550

550 Kmhmh. exterior angle bisectors of a triangle. <https://commons.wikimedia.org/wiki/File:Aussenwinkelhalbierende2.svg>, 2019a. Licensed under CC BY 4.0; modified
 551 by the authors.
 552

552 Kmhmh. tangential triangle of reference triangle. https://commons.wikimedia.org/wiki/File:Tangential_triangle.svg, 2019b. Licensed under CC BY 4.0; modified
 553 by the authors.
 554

554 Kmhmh. Japanese theorem for cyclic quadrilaterals (correct version) — diagram.
 555 https://commons.wikimedia.org/wiki/File:Japanese_theorem_2_correct_version_a.svg, 2024. Licensed under CC BY 4.0; modified by the authors.
 556

556 Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large
 557 language models are zero-shot reasoners. *Advances in Neural Information Processing Systems*
 558 (*NeurIPS*), 35:22199–22213, 2022.
 559

559 Krishnachandranvn. The construction of the apollonius point. https://commons.wikimedia.org/wiki/File:Apollonius_point.svg, 2012. Released into the pub-
 560 lic domain under CC0 1.0; modified by the authors.
 561

561 Alan Malek, Jiawei Ge, Nevena Lazic, Chi Jin, András György, and Csaba Szepesvári. Frontier llms
 562 still struggle with simple reasoning tasks. *arXiv preprint arXiv:2507.07313*, 2025.
 563

563 National Center for Biotechnology Information. PubChem Compound Summary for
 564 CID 100329, 2,5-dihydroxy-3,6-bis[1-(2-methylbut-3-en-2-yl)indol-3-yl]cyclohexa-2,5-diene-
 565 1,4-dione, 2025a. URL <https://pubchem.ncbi.nlm.nih.gov/compound/100329>.
 566 Retrieved September 10, 2025.
 567

567 National Center for Biotechnology Information. PubChem Compound Summary for
 568 CID 179317, N-[5-chloro-4-(trifluoromethyl)-1,3-thiazol-2-yl]-N-(trideuteriomethyl)-3,5-
 569 bis(trifluoromethyl)benzamide, 2025b. URL <https://pubchem.ncbi.nlm.nih.gov/compound/179317>. Retrieved September 10, 2025.
 570

570 National Center for Biotechnology Information. PubChem Compound Summary for CID 21011,
 571 4-chloro-1-methyl-5-nitroimidazole, 2025c. URL <https://pubchem.ncbi.nlm.nih.gov/compound/21011>. Retrieved September 10, 2025.
 572

572 National Center for Biotechnology Information. PubChem Compound Summary for CID 23521,
 573 N-heptylacridin-9-amine, 2025d. URL <https://pubchem.ncbi.nlm.nih.gov/compound/23521>. Retrieved September 10, 2025.
 574

574 National Center for Biotechnology Information. PubChem Compound Summary for CID 243518,
 575 3-bromobut-3-en-2-amine, 2025e. URL <https://pubchem.ncbi.nlm.nih.gov/compound/243518>. Retrieved September 10, 2025.
 576

576 National Center for Biotechnology Information. PubChem Compound Summary for CID 251192,
 577 4-Butyl-2,6-dimethylmorpholine, 2025f. URL <https://pubchem.ncbi.nlm.nih.gov/compound/251192>. Retrieved September 10, 2025.
 578

594 National Center for Biotechnology Information. PubChem Compound Summary for CID 257753,
 595 4-[[2,4-diamino-5-[(4-carboxyphenyl)diaz恒y]phenyl]diaz恒y]benzoic acid, 2025g. URL
 596 <https://pubchem.ncbi.nlm.nih.gov/compound/257753>. Retrieved September
 597 10, 2025.

598 National Center for Biotechnology Information. PubChem Compound Summary for CID 272448, methyl 4-[3-acetamido-4,5-diacyloxy-6-(acyloxyethyl)oxan-2-yl]oxy-1-(2,4-
 599 dinitrophenyl)pyrrolidine-2-carboxylate, 2025h. URL <https://pubchem.ncbi.nlm.nih.gov/compound/272448>. Retrieved September 10, 2025.

600 National Center for Biotechnology Information. PubChem Compound Summary for CID 319843, ethyl 3-iminobutanoate, 2025i. URL <https://pubchem.ncbi.nlm.nih.gov/compound/319843>. Retrieved September 10, 2025.

601 National Center for Biotechnology Information. PubChem Compound Summary for CID 57105, 3-[[4-(4-methylphenyl)piperazin-1-yl]methyl]-5-[(2-methylquinazolin-4-yl)oxymethyl]-
 602 1,3,4-oxadiazole-2-thione, 2025j. URL <https://pubchem.ncbi.nlm.nih.gov/compound/57105>. Retrieved September 10, 2025.

603 National Center for Biotechnology Information. PubChem Compound Summary for CID 574024, 5-ethylcyclopentene-1-carboxylic acid, 2025k. URL <https://pubchem.ncbi.nlm.nih.gov/compound/574024>. Retrieved September 10, 2025.

604 National Center for Biotechnology Information. PubChem Compound Summary for CID 609005, 6-methoxy-4-methyl-5-phenylmethoxyquinolin-8-amine, 2025l. URL <https://pubchem.ncbi.nlm.nih.gov/compound/609005>. Retrieved September 10, 2025.

605 National Center for Biotechnology Information. PubChem Compound Summary for CID 70260, 1,2,5-trimethylpyrrole, 2025m. URL <https://pubchem.ncbi.nlm.nih.gov/compound/70260>. Retrieved September 10, 2025.

606 National Center for Biotechnology Information. PubChem Compound Summary for CID 784953, N-(2-methylsulfinylethyl)acetamide, 2025n. URL <https://pubchem.ncbi.nlm.nih.gov/compound/784953>. Retrieved September 10, 2025.

607 National Center for Biotechnology Information. PubChem Compound Summary for CID 982106, (6-chloro-4-methyl-2-oxochromen-7-yl) (2R)-2-[(4-methylphenyl)sulfonylamino]pentanoate, 2025o. URL <https://pubchem.ncbi.nlm.nih.gov/compound/982106>. Retrieved September 10, 2025.

608 National Center for Biotechnology Information. PubChem Compound Summary for CID 141986, 2-Tellurophenecarboxylic acid, 2025p. URL <https://pubchem.ncbi.nlm.nih.gov/compound/141986>. Retrieved September 10, 2025.

609 Kunato Nishina and Yusuke Matsui. Sveditbench: A benchmark dataset for quantitative assessment of ILM's SVG editing capabilities. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 8142–8147, 2024.

610 OpenAI. Models, 2025. <https://platform.openai.com/docs/models>.

611 PegasusRoe. image for projection formula in trigonometry. [https://commons.wikimedia.org/wiki/File:Projection_formula_\(3\).png](https://commons.wikimedia.org/wiki/File:Projection_formula_(3).png), 2007. Released into the public domain under CC0 1.0; modified by the authors.

612 Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, pp. 8748–8763, 2021.

613 Claudio Rocchini. Isogonal conjugate.svg — illustration of isogonal conjugacy in a triangle. https://commons.wikimedia.org/wiki/File:Isogonal_Conjugate.svg, 2008. Licensed under CC BY 3.0; modified by the authors.

648 Claudio Rocchini. Six circles theorem: examples of some configuration. https://commons.wikimedia.org/wiki/File:Six_circles_theorem.svg, 2010. Licensed under CC
649 BY-SA 3.0; modified by the authors.
650

651 Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad
652 Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning
653 models via the lens of problem complexity. *arXiv preprint arXiv:2506.06941*, 2025.
654

655 Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée
656 Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and
657 efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
658

659 Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim.
660 Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language
661 models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 2609–
662 2634, 2023a.
663

663 Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdh-
664 ery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models.
665 In *International Conference on Learning Representations (ICLR)*, 2023b.
666

666 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny
667 Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in
668 Neural Information Processing Systems (NeurIPS)*, 35:24824–24837, 2022.
669

670 Jingxuan Wei, Cheng Tan, Qi Chen, Gaowei Wu, Siyuan Li, Zhangyang Gao, Linzhuang Sun, Bihui
671 Yu, and Ruifeng Guo. From words to structured visuals: A benchmark and framework for text-
672 to-diagram generation and editing. In *IEEE/CVF Conference on Computer Vision and Pattern
673 Recognition (CVPR)*, pp. 13315–13325, 2025.
674

674 Wikipedia contributors. Six circles theorem. https://en.wikipedia.org/wiki/Six_circles_theorem, 2022. [Online; accessed 10-September-2025; licensed under CC BY-SA
675 4.0].
676

676 Wikipedia contributors. Apollonius point. https://en.wikipedia.org/wiki/Apollonius_point, 2024. [Online; accessed 10-September-2025; licensed under CC BY-
677 /SA 4.0].
678

678 Wikipedia contributors. Altitude (triangle). [https://en.wikipedia.org/wiki/Altitude_\(triangle\)](https://en.wikipedia.org/wiki/Altitude_(triangle)), 2025a. [Online; accessed 10-September-2025; licensed under CC
679 BY-SA 4.0].
680

680 Wikipedia contributors. Angle bisector theorem. https://en.wikipedia.org/wiki/Angle_bisector_theorem, 2025b. [Online; accessed 10-September-2025; licensed under
681 CC BY-SA 4.0].
682

682 Wikipedia contributors. Bisection. <https://en.wikipedia.org/wiki/Bisection>,
683 2025c. [Online; accessed 10-September-2025; licensed under CC BY-SA 4.0].
684

684 Wikipedia contributors. Butterfly theorem. https://en.wikipedia.org/wiki/Butterfly_theorem, 2025d. [Online; accessed 10-September-2025; licensed under CC BY-
685 /SA 4.0].
686

686 Wikipedia contributors. Casey's theorem. https://en.wikipedia.org/wiki/Casey%27s_theorem, 2025e. [Online; accessed 10-September-2025; licensed under CC BY-SA 4.0].
687

687 Wikipedia contributors. Finsler-hadwiger theorem. https://en.wikipedia.org/wiki/Finsler%25E2%2580%2593Hadwiger_theorem, 2025f. [Online; accessed 10-September-2025;
688 licensed under CC BY-SA 4.0].
689

689 Wikipedia contributors. Incircle and excircles. https://en.wikipedia.org/wiki/Incircle_and_excircles, 2025g. [Online; accessed 10-September-2025; licensed under
690 CC BY-SA 4.0].
691

702 Wikipedia contributors. Isogonal conjugate. [https://en.wikipedia.org/wiki/](https://en.wikipedia.org/wiki/Isogonal_conjugate)
 703 Isogonal_conjugate, 2025h. [Online; accessed 10-September-2025; licensed under CC
 704 BY-SA 4.0].

705 Wikipedia contributors. Japanese theorem for cyclic quadrilaterals. https://en.wikipedia.org/wiki/Japanese_theorem_for_cyclic_quadrilaterals, 2025i. [Online; ac-
 706 cessed 10-September-2025; licensed under CC BY-SA 4.0].

707 Wikipedia contributors. Medial triangle. [https://en.wikipedia.org/wiki/Medial_](https://en.wikipedia.org/wiki/Medial_triangle)
 708 triangle, 2025j. [Online; accessed 10-September-2025; licensed under CC BY-SA 4.0].

709 Wikipedia contributors. Splitter (geometry). [https://en.wikipedia.org/wiki/Splitter_\(geometry\)](https://en.wikipedia.org/wiki/Splitter_(geometry)), 2025k. [Online; accessed 10-September-2025; licensed under
 710 CC BY-SA 4.0].

711 Wikipedia contributors. Symmedian. <https://en.wikipedia.org/wiki/Symmedian>,
 712 2025l. [Online; accessed 10-September-2025; licensed under CC BY-SA 4.0].

713 Wikipedia contributors. Tangential triangle. https://en.wikipedia.org/wiki/Tangential_triangle, 2025m. [Online; accessed 10-September-2025; licensed under CC
 714 BY-SA 4.0].

715 Zhongzheng Xu and Emily Wall. Exploring the capability of llms in performing low-level visual
 716 analytic tasks on svg data visualizations. In *IEEE Visualization and Visual Analytics (VIS)*, pp.
 717 126–130, 2024.

718 Cheng Yang, Chufan Shi, Yixin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran XU, Xinyu Zhu,
 719 Siheng Li, Yuxiang Zhang, et al. Chartmimic: Evaluating lmm’s cross-modal reasoning capability
 720 via chart-to-code generation. In *International Conference on Learning Representations (ICLR)*,
 721 2025.

722 Zhiyu Yang, Zihan Zhou, Shuo Wang, Xin Cong, Xu Han, Yukun Yan, Zhenghao Liu, Zhixing
 723 Tan, Pengyuan Liu, Dong Yu, et al. Matplotagent: Method and evaluation for llm-based agentic
 724 scientific data visualization. In *Findings of the Association for Computational Linguistics (ACL)*,
 725 pp. 11789–11804, 2024.

726 Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language
 727 image pre-training. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp.
 728 11975–11986, 2023.

729 Leixin Zhang, Steffen Eger, Yinjie Cheng, Weihe Zhai, Jonas Belouadi, Christoph Leiter, Si-
 730 mone Paolo Ponzetto, Fahimeh Moafian, and Zhixue Zhao. Scimage: How good are multimodal
 731 large language models at scientific text-to-image generation? In *International Conference on
 732 Learning Representations (ICLR)*, 2025.

733 Huaxiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H Chi, Quoc V Le,
 734 and Denny Zhou. Take a step back: Evoking reasoning via abstraction in large language models.
 735 In *International Conference on Learning Representations (ICLR)*, 2024.

736 Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuur-
 737 mans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex
 738 reasoning in large language models. In *International Conference on Learning Representations
 739 (ICLR)*, 2023.

740 Wenhao Zhu, Shujian Huang, Fei Yuan, Shuaijie She, Jiajun Chen, and Alexandra Birch. Question
 741 translation training for better multilingual reasoning. In *Findings of the Association for Compu-
 742 tational Linguistics (ACL)*, pp. 8411–8423, 2024.

743 Bocheng Zou, Mu Cai, Jianrui Zhang, and Yong Jae Lee. Vgbench: A comprehensive benchmark
 744 of vector graphics understanding and generation for large language models. In *Conference on
 745 Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3647–3659, 2024.

746

756 A THE USE OF LARGE LANGUAGE MODELS (LLMs)
757758 We used ChatGPT when writing our paper, for translation purposes.
759760 B DETAILED PROMPTS USED IN OUR EXPERIMENTS
761762 Figures 7, 8, and 9 show examples of prompts used in the plane geometry task. The
763 [Explanation] section represents the textual description, while the [TikZ], [SVG], and
764 [EPS] sections represent the input elements (the black elements in Figure 2).
765766 Figures 10, 11, and 12 show examples of prompts used in the molecular structure task. We first
767 provide the IUPAC name, followed by a set of instructions. In particular, we include color specifi-
768 cations so that the types of atoms can be identified by their color. We also present an example using
769 chlorobenzene, which is a relatively simple molecule.
770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

```

810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863

```

[Explanation]The exterior angle bisector in A intersects the extended side BC in E, the exterior angle bisector in B intersects the extended side AC in D and the exterior angle bisector in C intersects the extended side AB in F.

The three points of intersection between the exterior angle bisectors and the extended triangle sides D, E, F are collinear, that is they lie on a common line.

```

[TikZ]\documentclass{standalone}
\usepackage{tikz}
\begin{document}
\begin{tikzpicture}[x=1pt,y=1pt]
\draw (38.23,-264.41) -- (65.98,-195.89) -- (32.16,-166.40) -- cycle;
\node at (23.54,-167.59) {A};
\node at (70.26,-197.89) {B};
\node at (28.79,-270.30) {C};
\end{tikzpicture}
\end{document}

```

Please visualize the explanation by adding elements to the TikZ diagram.

- Set the color of any added elements to red.
- Do not modify any elements that are originally present in the TikZ.
- Do not use any animations.

Figure 7: An example prompt from the TikZ generation task on plane geometry.

```

829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863

```

[Explanation]The exterior angle bisector in A intersects the extended side BC in E, the exterior angle bisector in B intersects the extended side AC in D and the exterior angle bisector in C intersects the extended side AB in F.

The three points of intersection between the exterior angle bisectors and the extended triangle sides D, E, F are collinear, that is they lie on a common line.

```

[SVG]<svg xmlns="http://www.w3.org/2000/svg"
xmlns:ev="http://www.w3.org/2001/xml-events" version="1.1"
viewBox="0 0 300 300">
<style>
.input_object {
  fill: none;
  stroke: black;
  stroke-width: 1;
}
.input_text {
  fill: black;
  stroke: none;
  font-size: 12px;
}
.output_object {
  fill: none;
  stroke: red;
  stroke-width: 1;
}
.output_text {
  fill: red;
  stroke: none;
  font-size: 12px;
}
</style>
<polygon class="input_object" points="38.23232323232324
264.4065656565657 65.9848484848485 195.89015151515151
32.15909090909092 166.40151515151516"/>
<text class="input_text" x="23.5416666666666668" y="167.58838383838383" data-cs="A"></text>
<text class="input_text" x="70.2588383838384" y="197.89141414141415" data-cs="B"></text>
<text class="input_text" x="28.7878787878793" y="270.30303030303037" data-cs="C"></text>
</svg>

```

Please visualize the explanation by adding elements to the SVG diagram.

- Assign class="output_text" to any added text elements, and class="output_object" to all other added elements.
- Do not modify any elements that are originally present in the SVG.
- Do not use any animations.

Figure 8: An example prompt from the SVG generation task on plane geometry.

864 [Explanation]The exterior angle bisector in A intersects the extended
 865 side BC in E, the exterior angle bisector in B intersects the extended
 866 side AC in D and the exterior angle bisector in C intersects the
 867 extended side AB in F.
 868 The three points of intersection between the exterior angle bisectors
 869 and the extended triangle sides D, E, F are collinear, that is they lie on
 870 a common line.
 871 [EPS]%!PS-Adobe-3.0 EPSF-3.0
 872 %%BoundingBox: 0 0 300 300
 873 1 setlinewidth
 874 0 0 0 setrgbcolor
 875 newpath
 876 38.232323232324 35.593434343434296 moveto
 877 65.98484848485 104.10984848485 lineto
 878 32.159090909092 133.598484848484 lineto
 879 closepath
 880 stroke
 881 /Helvetica findfont 12 scalefont selfont
 882 0 0 0 setrgbcolor
 883 newpath
 884 23.54166666666668 132.41161616161617 moveto
 885 (A) show
 886 /Helvetica findfont 12 scalefont selfont
 887 0 0 0 setrgbcolor
 888 newpath
 889 70.2588383838384 102.10858585858585 moveto
 890 (B) show
 891 /Helvetica findfont 12 scalefont selfont
 892 0 0 0 setrgbcolor
 893 newpath
 894 28.7878787878793 29.69696969696963 moveto
 895 (C) show
 896 showpage
 897 Please visualize the explanation by adding elements to the EPS
 898 diagram.
 899 – Set the color of any added elements to red.
 900 – Do not modify any elements that are originally present in the EPS.
 901 – Do not use any animations.

Figure 9: An example prompt from the EPS generation task on plane geometry.

893 Please create a TikZ file that visualizes the molecular structure of the
 894 compound with the IUPAC name 4-butyl-2,6-dimethylmorpholine.
 895 Represent each atom as a circle, using colors to indicate atom types.
 896 The color mapping for each atom type is provided below, although not
 897 all listed types may be present in the molecule. Omit hydrogen atoms
 898 from the visualization. Depict bonds between atoms as lines, using a
 899 single line for each bond regardless of bond order.
 900
 901 H: #638c8c, B: #2AD52A, C: #274A4A, N: #0000FF, O: #FF0000,
 902 #D52092, Si: #D59E13, P: #D58600, S: #D5D500, Cl: #2AD52A, Br:
 903 #D58639, Te: #D5CD72, I: #FF00FF, Eu: #00CCD5, Lu: #00CCD5,
 904 Os: #838C8C, U: #00CCD5
 905
 906 As a reference, an example TikZ visualization of the compound
 907 chlorobenzene is provided below.
 908 \documentclass[tikz]{standalone}
 909 \definecolor{274A4A}{HTML}{274A4A}
 910 \definecolor{2AD52A}{HTML}{2AD52A}
 911 \begin{document}
 912 \begin{tikzpicture}[x=1pt,y=1pt]
 913 \draw[line width=1pt] (20.00,6.22) -- (28.66,1.22);
 914 \draw[line width=1pt] (45.98,1.22) -- (45.98,-8.78);
 915 \draw[line width=1pt] (45.98,1.22) -- (37.32,6.22);
 916 \draw[line width=1pt] (45.98,-8.78) -- (37.32,-13.78);
 917 \draw[line width=1pt] (37.32,6.22) -- (28.66,1.22);
 918 \draw[line width=1pt] (37.32,-13.78) -- (28.66,-8.78);
 919 \draw[line width=1pt] (28.66,1.22) -- (28.66,-8.78);
 920 \filldraw[fill=2AD52A, draw=none] (20.00,6.22) circle (1.5pt);
 921 \filldraw[fill=274A4A, draw=none] (45.98,1.22) circle (1.5pt);
 922 \filldraw[fill=274A4A, draw=none] (45.98,-8.78) circle (1.5pt);
 923 \filldraw[fill=274A4A, draw=none] (37.32,6.22) circle (1.5pt);
 924 \filldraw[fill=274A4A, draw=none] (37.32,-13.78) circle (1.5pt);
 925 \filldraw[fill=274A4A, draw=none] (28.66,1.22) circle (1.5pt);
 926 \filldraw[fill=274A4A, draw=none] (28.66,-8.78) circle (1.5pt);
 927 \end{tikzpicture}
 928 \end{document}

Figure 10: An example prompt from the TikZ generation task on molecular structure.

918
 919
 920
 921
 922
 923
 924
 925
 926
 927
 928
 929
 930

931 Please create an SVG file that visualizes the molecular structure of the
 932 compound with the IUPAC name 4-butyl-2,6-dimethylmorpholine.
 933 Represent each atom as a circle, using colors to indicate atom types.
 934 The color mapping for each atom type is provided below, although not
 935 all listed types may be present in the molecule. Omit hydrogen atoms
 936 from the visualization. Depict bonds between atoms as lines, using a
 937 single line for each bond regardless of bond order.
 938

939 H: #638c8c, B: #2AD52A, C: #274A4A, N: #0000FF, O: #FF0000, F:
 940 #D52092, Si: #D59E13, P: #D58600, S: #D5D500, Cl: #2AD52A, Br:
 941 #D58639, Te: #D5CD72, I: #FF00FF, Eu: #00CCD5, Lu: #00CCD5,
 942 Os: #838C8C, U: #00CCD5

943 As a reference, an example SVG visualization of the compound
 944 chlorobenzene is provided below.
 945 <svg xmlns="http://www.w3.org/2000/svg" viewBox="10 -29.976
 946 67.479 59.646">
 947 <line x1="20" y1="6.22399999999999" x2="28.66" y2="1.224"
 948 stroke="black" stroke-width="1" />
 949 <line x1="45.98099999999995" y1="1.224"
 950 x2="45.98099999999995" y2="-8.776" stroke="black"
 951 stroke-width="1" />
 952 <line x1="45.98099999999995" y1="1.224" x2="37.32"
 953 y2="6.22399999999999" stroke="black" stroke-width="1" />
 954 <line x1="45.98099999999995" y1="1.224" x2="37.32" y2="-13.776"
 955 stroke="black" stroke-width="1" />
 956 <line x1="37.32" y1="6.22399999999999" x2="28.66" y2="1.224"
 957 stroke="black" stroke-width="1" />
 958 <line x1="37.32" y1="-13.776" x2="28.66" y2="-8.776" stroke="black"
 959 stroke-width="1" />
 960 <circle cx="20" cy="6.22399999999999" r="1.5" fill="#2AD52A" />
 961 <circle cx="45.98099999999995" cy="1.224" r="1.5" fill="#274A4A" />
 962 </circle>
 963 <circle cx="45.98099999999995" cy="-8.776" r="1.5" fill="#274A4A" />
 964 </circle>
 965 <circle cx="37.32" cy="6.22399999999999" r="1.5" fill="#274A4A" />
 966 <circle cx="37.32" cy="-13.776" r="1.5" fill="#274A4A" />
 967 <circle cx="28.66" cy="1.224" r="1.5" fill="#274A4A" />
 968 <circle cx="28.66" cy="-8.776" r="1.5" fill="#274A4A" />
 969 </circle>
 970 </svg>

971

Figure 11: An example prompt from the SVG generation task on molecular structure.

972
 973
 974
 975
 976
 977
 978
 979
 980
 981

982 Please create an EPS file that visualizes the molecular structure of the
 compound with the IUPAC name 4-butyl-2,6-dimethylmorpholine.
 983 Represent each atom as a circle, using colors to indicate atom types.
 984 The color mapping for each atom type is provided below, although not
 985 all listed types may be present in the molecule. Omit hydrogen atoms
 986 from the visualization. Depict bonds between atoms as lines, using a
 987 single line for each bond regardless of bond order.
 988 H: [0.39, 0.55, 0.55], B: [0.16, 0.84, 0.16], C: [0.15, 0.29, 0.29], N: [0.0,
 989 0.0, 1.0], O: [1.0, 0.0, 0.0], F: [0.84, 0.13, 0.57], Si: [0.84, 0.62, 0.07],
 990 P: [0.84, 0.53, 0.0], S: [0.84, 0.84, 0.0], Cl: [0.16, 0.84, 0.16], Br: [0.84,
 991 0.53, 0.22], Te: [0.84, 0.8, 0.45], I: [1.0, 0.0, 1.0], Eu: [0.0, 0.8, 0.84],
 992 Lu: [0.0, 0.8, 0.84], Os: [0.51, 0.55, 0.55], U: [0.0, 0.8, 0.84]
 993 As a reference, an example EPS visualization of the compound
 994 chlorobenzene is provided below.
 995 %!PS-Adobe-3.0 EPSF-3.0
 996 %%BoundingBox: 10 -29 77 29
 997 %%Creator: EPS Generator
 998 %%EndComments
 999 1 setlinewidth
 1000 0 0 0 setrgbcolor
 1001 newpath
 1002 20 6.22399999999999 moveto
 1003 28.66 1.224 lineto
 1004 stroke
 1005 newpath
 1006 37.32 6.22399999999999 moveto
 1007 28.66 1.224 lineto
 1008 stroke
 1009 newpath
 1010 28.66 -8.776 moveto
 1011 37.32 -13.776 lineto
 1012 stroke
 1013
 1014
 1015

1016 newpath
 1017 37.32 -13.776 moveto
 1018 45.98099999999995 -8.776 lineto
 1019 stroke
 1020 newpath
 1021 45.98099999999995 1.224 lineto
 1022 stroke
 1023 0.16 0.84 0.16 setrgbcolor
 1024 newpath
 1025 20 6.22399999999999 1.5 0 360 arc
 closepath
 fill
 0.15 0.29 0.29 setrgbcolor
 newpath
 45.98099999999995 1.224 1.5 0 360 arc
 closepath
 fill
 0.15 0.29 0.29 setrgbcolor
 newpath
 45.98099999999995 -8.776 1.5 0 360 arc
 closepath
 fill
 0.15 0.29 0.29 setrgbcolor
 newpath
 37.32 6.22399999999999 1.5 0 360 arc
 closepath
 fill
 0.15 0.29 0.29 setrgbcolor
 newpath
 37.32 -13.776 1.5 0 360 arc
 closepath
 fill
 0.15 0.29 0.29 setrgbcolor
 newpath
 28.66 1.224 1.5 0 360 arc
 closepath
 fill
 0.15 0.29 0.29 setrgbcolor
 newpath
 28.66 -8.776 1.5 0 360 arc
 closepath
 fill
 showpage

Figure 12: An example prompt from the EPS generation task on molecular structure.

1026 **C LIMITATIONS OF OUR AUTOMATIC EVALUATION FRAMEWORK**
 1027

1028 Our automatic evaluation framework for the plane geometry visualization task assesses whether the
 1029 necessary elements are present in the output, but it does not penalize the inclusion of unnecessary
 1030 elements. In the example shown in Figure 13, the output from Gemini 2.5 Flash reasoning includes
 1031 an irrelevant straight line, yet it is still considered correct. We do not penalize unnecessary elements
 1032 because it is often non-trivial to determine whether an additional element is truly unnecessary. For
 1033 instance, the output from Gemini 2.5 Flash reasoning in Figure 13 includes circles not anticipated in
 1034 the ground-truth, but these represent intersections and the circle center, and they do not hinder the
 1035 explanation.

1036 Because in practical scenarios it is usually easier for humans to remove unnecessary elements than to
 1037 create necessary ones from scratch, we do not currently view this limitation as a major issue. How-
 1038 ever, enabling the framework to identify and evaluate such extraneous content remains an important
 1039 direction for future work.

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

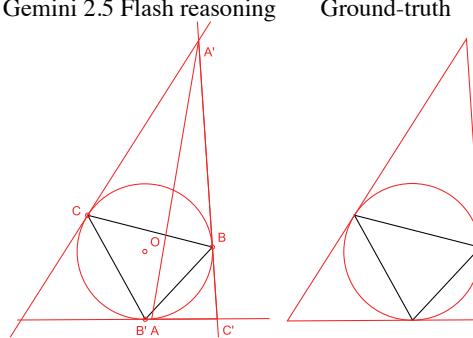
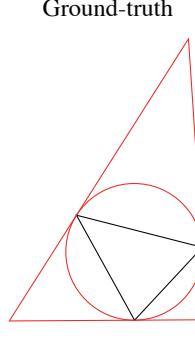
1075

1076

1077

1078

1079

Input	Output	Gemini 2.5 Flash reasoning	Ground-truth
In geometry, the tangential text triangle of a reference triangle (other than a right triangle) is the triangle whose sides are on the tangent lines to the reference triangle's circumcircle at the reference triangle's vertices.	vector graphic (SVG)		

1055 Figure 13: An example judged correct despite an unrelated line.⁷
 1056

1057
 1058
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1079
⁷The input text and the ground-truth vector graphic are from (Wikipedia contributors, 2025m; Kmhkmh, 2019b).

1080 D ADDITIONAL EXAMPLES OF GENERATED VECTOR GRAPHICS
10811082 D.1 EXAMPLES GENERATED BY FINE-TUNED MODELS
1083

1084 Figure 14 shows examples generated by two fine-tuned models, AutomaTikZ (Belouadi et al.,
1085 2024a) and TikZero+ (Belouadi et al., 2025). The top example is the only case where TikZero+
1086 produces a structurally correct vector graphic. However, in all other cases, the models fail to gen-
1087 erate structurally correct vector graphics. In the plane geometry task, they cannot follow simple
1088 instructions such as “Set the color of any added elements to red” or “Do not modify any elements
1089 that are originally present in the TikZ.” In the molecular structure task, the models generate entirely
1090 invalid vector graphics. These results reveal the limitations of fine-tuned models.

1093 Input	1094 In Euclidean geometry, the 1095 text medial triangle or midpoint 1096 triangle of a triangle $\triangle ABC$ is 1097 the triangle with vertices at the 1098 midpoints of the triangle's sides 1099 AB , AC , BC .	1093 Output	1094 vector 1095 graphic 1096 (TikZ)	1093 AutomaTikZ	1094 TikZero+	1095 Ground-truth
1101 Input	1102 The center of an excircle is the 1103 text intersection of the internal bisector 1104 of one angle and the external 1105 bisectors of the other two. Every 1106 triangle has three excircle centers. 1107	1101 Output	1102 vector 1103 graphic 1104 (TikZ)	1101 AutomaTikZ	1102 TikZero+	1103 Ground-truth
1110 Input	1111 the molecular structure of the 1112 compound with the IUPAC name 1113 5-ethylcyclopentene-1-carboxylic 1114 acid	1110 Output	1111 vector 1112 graphic 1113 (TikZ)	1110 AutomaTikZ	1111 TikZero+	1112 Ground-truth
1117 Input	1118 the molecular structure of the 1119 compound with the IUPAC name 1120 N-(2-methylsulfinylethyl)acetamide	1117 Output	1118 vector 1119 graphic 1120 (TikZ)	1117 AutomaTikZ	1118 TikZero+	1119 Ground-truth

1126 Figure 14: Examples generated by fine-tuned models.⁸
1127
1128
1129
1130
1131

1132 ⁸The input text and the ground-truth vector graphic are from (Wikipedia contributors, 2025j;g;
1133 Braindrain0000, 2006; Inductiveload, 2007a). The molecular structure data is from (National Center for
Biotechnology Information, 2025k;n).

1134 D.2 IMPACT OF REASONING
11351136 We show in Figures 15 and 16 that enabling reasoning allows LLMs to output the correct structure.
1137 Without reasoning, they struggle to generate even simple structures.
1138
1139

1140	Input	Output vector graphic (SVG)	DeepSeek-V3.1	DeepSeek-V3.1 reasoning	Ground-truth
1141	In geometry, an altitude of a text triangle is a line segment through a given vertex (called apex) and perpendicular to a line containing the side or edge opposite the apex. The altitude from A intersects the extended base at D (a point outside the triangle).				
1142			Incorrect	Correct	
1143					
1144					
1145					
1146					
1147					
1148					
1149	Input The Gergonne triangle (of text $\triangle ABC$) is defined by the three touchpoints of the incircle on the three sides. The touchpoint opposite A is denoted T_A , etc. The three lines $A\{T_A\}$, $B\{T_B\}$, and $C\{T_C\}$ intersect in a single point called the Gergonne point, denoted as G_e .	Output vector graphic (SVG)	Claude Opus 4.1	Claude Opus 4.1 thinking	Ground-truth
1150					
1151			Incorrect	Correct	
1152					
1153					
1154					
1155					
1156					
1157	Input To bisect an angle with straightedge and compass, one draws a circle whose center is the vertex. The circle meets the angle at two points: one on each leg. Using each of these points as a center, draw two circles of the same size. The intersection of the circles (two points) determines a line that is the angle bisector.	Output vector graphic (SVG)	Gemini 2.5 Flash	Gemini 2.5 Flash reasoning	Ground-truth (Pattern 2)
1158					
1159			Incorrect	Correct	
1160					
1161					
1162					
1163					
1164					
1165					
1166					
1167	Input The butterfly theorem is a classical result in Euclidean geometry, which can be stated as follows: Let M be the midpoint of a chord PQ of a circle, through which two other chords AB and CD are drawn; AD and BC intersect chord PQ at X and Y correspondingly. Then M is the midpoint of XY.	Output vector graphic (SVG)	GPT-5 Chat	GPT-5	Ground-truth (Pattern 2)
1168					
1169			Incorrect	Correct	
1170					
1171					
1172					
1173					
1174					
1175					
1176					
1177	Figure 15: Examples where enabling reasoning allows LLMs to generate the correct structure in the plane geometry SVG geneartion task. ⁹				
1178					

⁹The input texts and the ground-truth vector graphics are from (Wikipedia contributors, 2025a;g;c;d; PegasusRoe, 2007; Inductiveload, 2007c; Ixnay, 2007; Gustavb, 2006).

1242
1243

D.3 IMPACT OF FORMAT

1244
1245
1246

Figures 17 and 18 present examples where LLMs produce correct structures in SVG format but fail in TikZ and EPS formats. Although the input text is identical, the results clearly vary depending on the output format.

1247

1248

1249

Input Let X be any triangle center of text $\triangle ABC$. Draw the lines AX , BX , CX and their reflections in the internal bisectors of the angles at the vertices A , B , C respectively. The reflected lines are concurrent and the point of concurrence is the isogonal conjugate Y of X .

1250

1251

1252

1253

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

1264

1265

1266

1267

1268

1269

1270

1271

1272

1273

1274

Input	Output vector graphic (SVG)	Gemini 2.5 Pro (TikZ)	Gemini 2.5 Pro (SVG)	Ground-truth
Let X be any triangle center of text $\triangle ABC$. Draw the lines AX , BX , CX and their reflections in the internal bisectors of the angles at the vertices A , B , C respectively. The reflected lines are concurrent and the point of concurrence is the isogonal conjugate Y of X .				
Specifically, let $\square ABCD$ be an arbitrary cyclic quadrilateral and let M_1, M_2, M_3, M_4 be the incenters of the triangles $\triangle ABD, \triangle ABC, \triangle BCD, \triangle ACD$. Then the quadrilateral formed by M_1, M_2, M_3, M_4 is a rectangle.				
In geometry, symmedians are three particular lines associated with every triangle. They are constructed by taking a median of the triangle (a line connecting a vertex with the midpoint of the opposite side), and reflecting the line over the corresponding angle bisector (the line through the same vertex that divides the angle there in half).				
In Euclidean geometry, a splitter text is a line segment through one of the vertices of a triangle (that is, a cevian) that bisects the perimeter of the triangle. The opposite endpoint of a splitter to the chosen triangle vertex lies at the point on the triangle's side where one of the excircles of the triangle is tangent to that side. Each triangle has three splitters, one through each vertex.				

1285

1286

Figure 17: Examples where LLMs produce correct structures in SVG format but fail in TikZ and EPS formats in the plane geometry task.¹¹

1287

1288

1289

1290

1291

1292

1293

1294

1295

¹¹The input texts and the ground-truth vector graphics are from (Wikipedia contributors, 2025h;i;l;k; Rocchini, 2008; Kmkmh, 2024; 2016; Inductiveload, 2007b).

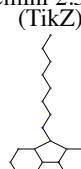
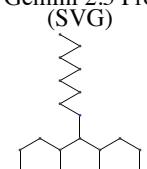
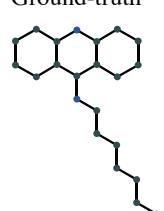
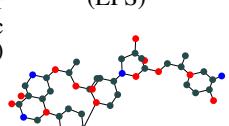
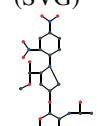
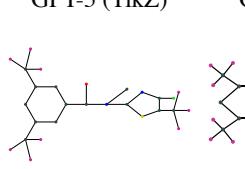
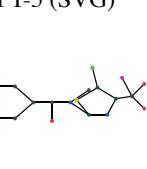
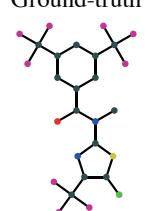
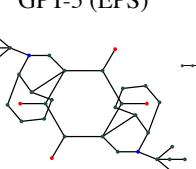
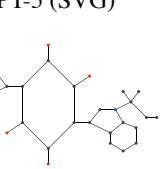
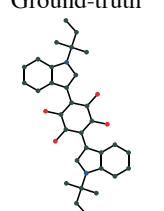
1296	Input the molecular structure of the text compound with the IUPAC name N-heptylacridin-9-amine	Output vector graphic (SVG)	Gemini 2.5 Pro (TikZ)	Gemini 2.5 Pro (SVG)	Ground-truth
1297					
1298			Incorrect	Correct	
1299					
1300					
1301					
1302					
1303					
1304					
1305					
1306					
1307					
1308					
1309					
1310					
1311					
1312					
1313	Input the molecular structure of the text compound with the IUPAC name methyl 4-[3-acetamido-4,5-diacetoxy-6-(acetoxyethyl)oxan-2-yl]oxy-1-(2,4-dinitrophenyl)pyrrolidine-2-carboxylate	Output vector graphic (SVG)	Gemini 2.5 Pro (EPS)	Gemini 2.5 Pro (SVG)	Ground-truth
1314					
1315			Incorrect	Correct	
1316					
1317					
1318					
1319					
1320					
1321	Input the molecular structure of the text compound with the IUPAC name N-[5-chloro-4-(trifluoromethyl)-1,3-thiazol-2-yl]-N-(trideutero methyl)-3,5-bis(trifluoromethyl)benzamide	Output vector graphic (SVG)	GPT-5 (TikZ)	GPT-5 (SVG)	Ground-truth
1322					
1323			Incorrect	Correct	
1324					
1325					
1326					
1327					
1328					
1329	Input the molecular structure of the text compound with the IUPAC name 2,5-dihydroxy-3,6-bis[1-(2-methylbut-3-en-2-yl)indol-3-yl]cyclohexa-2,5-diene-1,4-dione	Output vector graphic (SVG)	GPT-5 (EPS)	GPT-5 (SVG)	Ground-truth
1330					
1331			Incorrect	Correct	
1332					
1333					
1334					
1335					
1336					

Figure 18: Examples where LLMs produce correct structures in SVG format but fail in TikZ and EPS formats in the molecular structure task.¹²

¹²The molecular structure data is from (National Center for Biotechnology Information, 2025d;h;b;a).

1350
1351

D.4 COMPARISON OF ZERO-SHOT PROMPTING METHODS

1352
1353
1354
1355
1356
1357
1358
1359
1360

We present results obtained by applying zero-shot prompting methods to Gemini 2.5 Pro in Figures 19, 20, 21, and 22. When using methods other than Zero-shot CoT, Gemini 2.5 Pro produces an explanation prior to generating vector graphics, and we include that explanation as well. In the case of Plan-and-Solve prompting, the output explanation “C3 will also be in corner B, tangent to C2” is incorrect, revealing that Gemini 2.5 Pro fails to capture the relationships. In the case of Step-Back prompting, the output explanation reveals that Gemini 2.5 Pro captures the necessary elements and relationships, but overlooks the importance of mathematical reasoning. In contrast, by applying our LOOP, Gemini 2.5 Pro captures the necessary elements and relationships and recognizes the importance of mathematical reasoning, enabling it to generate structurally correct vector graphics.

1361
1362
1363
1364
1365
1366

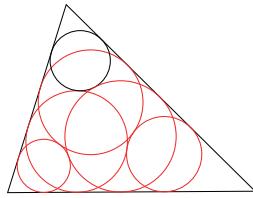
1367

Input text and ground-truth1368
1369
1370

Input text In geometry, the six circles theorem relates to a chain of six circles together with a triangle, such that each circle is tangent to two sides of the triangle and also to the preceding circle in the chain. The chain closes, in the sense that the sixth circle is always tangent to the first circle.

1371
1372
1373
1374
1375
1376
1377

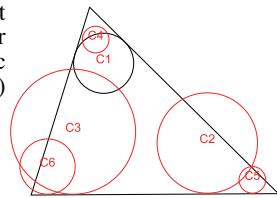
Ground-truth

1378
1379**Gemini 2.5 Pro w/ Zero-shot CoT**1380
1381

Output explanation (There is no output explanation.)

1382
1383
1384
1385
1386
1387
1388
1389

Output vector graphic (SVG)



Incorrect

1390
1391

Figure 19: Example output generated by applying Zero-shot CoT prompting to Gemini 2.5 Pro.¹³

1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402

1403

¹³The input text and the ground-truth vector graphic are from (Wikipedia contributors, 2022; Rocchini, 2010).

1404
 1405
 1406
 1407
 1408
 1409
 1410
 1411
 1412
 1413

Gemini 2.5 Pro w/ Plan-and-Solve

1414
 1415 Output An explanation of the six circles theorem has been provided, and it needs to be visualized by adding elements to
 1416 explanation the given SVG.

1417 ### Plan:
 1418 1. **Deconstruct the Theorem**: The theorem describes a chain of six circles (C1 to C6) associated with a
 1419 triangle.
 1420 * There are three pairs of circles, with each pair nestled into one corner of the triangle.
 1421 * Circles in the same corner are tangent to each other and to the two adjacent sides of the triangle.
 1422 * The chain follows a sequence around the triangle's vertices: A -> B -> C -> A.
 1423 * Let's say the given circle, C1, is in corner A.
 1424 * The next circle, C2, will be in corner B and tangent to C1.
 1425 * C3 will also be in corner B, tangent to C2.
 1426 * C4 will be in corner C, tangent to C3.
 1427 * C5 will also be in corner C, tangent to C4.
 1428 * C6 will be in corner A, tangent to C5.
 1429 * The theorem's conclusion is that C6 will also be tangent to the first circle, C1, thus closing the chain.

1430 2. **Analyze the Input SVG**:
 1431 * We have a triangle. Let's label its vertices A (top), B (bottom-left), and C (bottom-right).
 1432 * We have one circle, which we will label C1. It is located in corner A.

1433 3. **Add Visual Elements**:
 1434 * Add the remaining five circles (C2, C3, C4, C5, C6) with the class 'output_object'.
 1435 * Place them according to the theorem's rules, ensuring they appear tangent to the correct triangle sides and
 1436 preceding circles. The positions and radii will be estimated to create a visually plausible representation of the
 1437 theorem.
 1438 * C1 and C6 will be in corner A.
 1439 * C2 and C3 will be in corner B.
 1440 * C4 and C5 will be in corner C.
 1441 * The crucial part is to draw the chain such that C6, which is tangent to C5, is also tangent to C1.
 1442 * Add text labels "C1" through "C6" for each circle, assigning them the class 'output_text'.

1443 ### Execution:
 1444 The following SVG code adds the five circles and six labels to illustrate the theorem. The positions and sizes of
 1445 the new circles have been chosen to create a clear and accurate visual representation of the tangency
 1446 relationships described.

1447 Output
 1448 vector
 1449 graphic
 1450 (SVG)

Incorrect

1451
 1452
 1453
 1454
 1455
 1456
 1457

Figure 20: Example output generated by applying Plan-and-Solve prompting to Gemini 2.5 Pro.

1458
 1459
 1460
 1461
 1462
 1463
 1464
 1465
 1466
 1467
 1468
 1469
 1470
 1471
 1472
 1473

Gemini 2.5 Pro w/ Step-Back

1474 **Output** An explanation of the six circles theorem has been provided. This theorem describes a closed chain of six
 1475 explanation circles, where each circle is tangent to two sides of a triangle and the preceding circle in the chain. The provided
 1476 SVG shows the triangle and the first circle of such a chain.

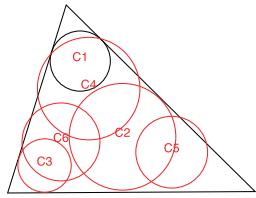
1477
 1478
 1479
 1480
 1481
 1482
 1483
 1484
 1485
 1486
 1487
 1488
 1489

To visualize the theorem, we will add the remaining five circles to complete the chain.

1. ****Circle C1:**** The initial circle is provided, located in the top corner of the triangle.
2. ****Circle C2:**** We add the second circle, C2, in the bottom-right corner, making it tangent to the two sides of the triangle at that corner and also to circle C1.
3. ****Circle C3:**** The third circle, C3, is placed in the bottom-left corner, tangent to the two sides there and to circle C2.
4. ****Circle C4:**** The fourth circle, C4, is placed back in the top corner. It is tangent to the same two sides as C1, and also to circle C3.
5. ****Circle C5:**** The fifth circle, C5, is placed in the bottom-right corner, tangent to the sides and to circle C4.
6. ****Circle C6:**** The sixth and final circle, C6, is placed in the bottom-left corner, tangent to the sides and to circle C5.

The remarkable property of the theorem is that this chain always closes perfectly: circle C6 will not only be tangent to C5 but will also be tangent to the very first circle, C1, completing the loop. The visualization below illustrates this closed chain.

1490 **Output**
 1491 vector
 1492 graphic
 1493 (SVG)



1496 Incorrect

1498 Figure 21: Example output generated by applying Step-Back prompting to Gemini 2.5 Pro.

1499
 1500
 1501
 1502
 1503
 1504
 1505
 1506
 1507
 1508
 1509
 1510
 1511

1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523

Gemini 2.5 Pro w/ LOOP (ours)

1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565

Output explanation An elegant theorem in geometry, the six circles theorem, describes a fascinating property of a chain of circles within a triangle. Starting with a circle tangent to two sides of the triangle, a sequence of five more circles is constructed. Each new circle is tangent to two sides of the triangle and to the previous circle in the chain. The remarkable result is that this chain always forms a closed loop, with the sixth circle being perfectly tangent to the first one.

This visualization completes the chain described by the theorem, starting from the single provided circle.

1. **Information extraction**: The core components are a triangle and a chain of six circles. The initial SVG provides the triangle and the first circle (let's call it C1). C1 is tangent to two sides of the triangle at one vertex. The goal is to construct the remaining five circles, C2 through C6, following the theorem's rules.

2. **Relationship extraction**: The chain proceeds sequentially around the vertices of the triangle.

- * C1 is tangent to sides AB and AC (at vertex A).
- * C2 must be tangent to sides AC and BC (at vertex C) and also to C1.
- * C3 must be tangent to sides BC and AB (at vertex B) and also to C2.
- * C4 must be tangent to sides AB and AC (at vertex A) and also to C3.
- * C5 must be tangent to sides AC and BC (at vertex C) and also to C4.
- * C6 must be tangent to sides BC and AB (at vertex B) and also to C5.

The theorem guarantees that the resulting C6 will also be tangent to the starting circle, C1.

3. **Mathematical reasoning**: The center of any circle tangent to two sides of the triangle must lie on the angle bisector of the corresponding vertex. The condition that the new circle must also be tangent to the previous circle provides a second constraint. Combining these two conditions leads to a solvable quadratic equation for the position of the new circle's center along the angle bisector. This process is applied iteratively to find the center and radius of each subsequent circle in the chain (C2, C3, C4, C5, and C6). The final calculation confirms the theorem: the distance between the centers of C6 and C1 is indeed equal to the sum of their radii, proving their tangency.

4. **Code generation**: Based on the calculated centers and radii, five '`<circle>`' elements are added to the SVG to represent C2 through C6. Additionally, '`<text>`' elements are added to label each circle from C1 to C6, clarifying the sequence of the chain.

Output
vector
graphic
(SVG)

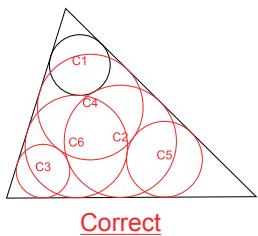


Figure 22: Example output generated by applying our LOOP to Gemini 2.5 Pro.

1566
1567

E ADDITIONAL EXPERIMENTS

1568
1569

E.1 COMPLEMENTARY AUTOMATIC METRICS

1570
1571

Our evaluation in the main paper relies solely on a single binary metric (Accuracy), which provides an overly coarse assessment because near-misses and completely incorrect outputs are both scored as 0. To address this limitation, we employ additional evaluation metrics. Following TikZero+ (Bellouadi et al., 2025), we use five image- and code-similarity metrics: DreamSim (**DSim**), Kernel Inception Distance (**KID**), CLIPScore (**CLIP**), CrystalBLEU (**cBLEU**), and Extended Edit Distance (**EED**). We additionally use **Coverage** as an extra metric, which considers an output correct if it contains the required elements, regardless of their spatial arrangement. As a comprehensive evaluation, we compute an **Average** score, defined as the mean of all metrics after applying min-max normalization. When applying min-max normalization, we normalize indicators where higher values are better such that the minimum becomes 0 and the maximum becomes 100. For indicators where lower values are better, we normalize them such that the minimum becomes 100 and the maximum becomes 0. We present these metrics in Tables 7, 8, 9, 10, 11, and 12.

1572
1573
1574
1575
1576
1577
1578
1579
1580

Focusing on the **Average** scores, we consistently observe that (1) reasoning models outperform non-reasoning models, and (2) our proposed LOOP improves the performance in most cases.

1581
1582
1583

E.2 ADDITIONAL MODELS

1584
1585

To demonstrate the versatility of our LOOP, we apply it to two additional reasoning models (DeepSeek-V3.2 Reasoning and Claude Opus 4.1 Thinking) and two additional non-reasoning models (Gemini 2.5 Flash and GPT-5 Chat). The results are shown in Tables 7, 8, 9, 10, 11, and 12. Our LOOP achieves improved performance in many cases, clearly demonstrating its robustness.

1586

E.3 IMPACT OF RESAMPLING ON FINE-TUNED MODELS

1591
1592

The impact of resampling is considered a possible reason for the low performance of the fine-tuned models. In our experiments, we evaluate each model using only a single generation attempt, and any outputs that resulted in compilation errors are considered incorrect. This evaluation procedure is applied uniformly across all models. However, the fine-tuned models are expected to be used under the assumption that resampling continues until no compilation errors occur. To investigate the impact of resampling, we allow up to 10 resampling attempts for each sample until a compilable output is obtained. As shown in Tables 7 and 10, resampling improves performance, and notably, TikZero+ achieves performance comparable to non-reasoning models.

1593

1594

1595

1596

1597

1598

1599

1600

1601

1602

1603

1604

1605

1606

1607

1608

1609

1610

1611

1612

1613

1614

1615

1616

1617

1618

1619

Table 7: Complementary metrics for plane geometry TikZ generation.

Model	Acc↑	DSim↑	KID↓	CLIP↑	cBLEU↑	EED↓	Coverage↑	Average↑
Fine-tuned models								
AutomaTikZ	0.0	36.0	109.4	3.2	1.1	63.5	3.6	17.3
TikZero+	0.9	49.0	55.5	13.5	5.6	51.4	12.7	32.6
AutomaTikZ resampling	0.0	56.4	12.5	16.1	1.2	61.1	13.6	33.7
DeTikZify resampling	0.9	67.1	5.8	27.2	7.4	49.6	30.0	46.7
Non-reasoning models								
DeepSeek-V3	10.0	56.1	42.4	28.3	11.4	44.0	42.7	48.4
DeepSeek-V3.1	11.8	53.8	49.6	23.8	10.1	45.2	41.8	45.9
Claude Opus 4.1	14.5	60.8	31.4	32.6	9.5	46.1	51.8	51.8
Gemini 2.0 Flash	7.3	71.1	9.9	39.9	19.4	37.4	50.0	62.5
Gemini 2.5 Flash non-reasoning	12.7	48.4	70.7	17.7	6.9	50.7	30.0	37.5
GPT-4.1	10.9	50.9	54.3	25.0	8.3	47.5	39.1	43.1
GPT-5 Chat	12.7	54.9	48.6	28.1	11.4	44.1	38.2	47.5
Rasoning models								
DeepSeek-R1	28.2	74.6	7.6	39.9	21.5	33.8	54.5	69.6
DeepSeek-V3.1 reasoning	23.6	63.5	29.5	31.6	11.3	40.8	40.9	54.4
Claude Opus 4.1 thinking	20.0	66.9	18.8	43.8	12.4	41.8	64.5	61.4
Gemini 2.5 Flash reasoning	30.0	55.7	49.6	25.2	8.5	47.8	44.5	48.7
Gemini 2.5 Pro	50.0	67.1	25.0	34.5	9.7	43.3	56.4	61.6
o4-mini	48.2	74.9	7.7	46.5	17.4	37.4	70.0	73.7
GPT-5	54.5	69.7	13.3	44.4	9.3	43.4	66.4	66.9
Prompting								
Gemini 2.5 Pro	50.0	67.1	25.0	34.5	9.7	43.3	56.4	61.6
w/ Zero-shot CoT	39.1	64.3	28.5	37.4	9.8	43.7	52.7	58.8
w/ Plan-and-Solve	39.1	60.7	37.3	28.2	9.1	45.9	48.2	54.1
w/ Step-Back	33.6	57.2	45.3	28.6	8.1	47.0	42.7	50.3
w/ LOOP (ours)	65.5	77.4	6.4	47.1	14.7	38.1	67.3	75.6
GPT-5	54.5	69.7	13.3	44.4	9.3	43.4	66.4	66.9
w/ Zero-shot CoT	58.2	72.5	12.4	44.0	10.1	43.3	66.4	68.6
w/ Plan-and-Solve	61.8	74.0	8.1	49.5	11.6	41.4	71.8	72.9
w/ Step-Back	55.5	70.0	13.8	41.7	9.6	43.5	68.2	67.0
w/ LOOP (ours)	70.0	80.6	3.1	52.3	13.4	39.0	82.7	79.7
Gemini 2.5 Flash non-reasoning	12.7	48.4	70.7	17.7	6.9	50.7	30.0	37.5
w/ LOOP (ours)	33.6	72.3	11.4	38.6	11.4	42.6	60.9	63.3
GPT-5 Chat	12.7	54.9	48.6	28.1	11.4	44.1	38.2	47.5
w/ LOOP (ours)	15.5	66.8	15.0	47.2	13.5	40.8	58.2	61.3
DeepSeek-V3.2 reasoning	14.5	63.6	26.8	32.8	11.9	42.1	43.6	53.6
w/ LOOP (ours)	22.7	70.9	12.1	37.6	16.0	36.7	61.8	64.7
Claude Opus 4.1 thinking	20.0	66.9	18.8	43.8	12.4	41.8	64.5	61.4
w/ LOOP (ours)	32.7	84.5	0.2	58.3	19.8	34.5	87.3	80.2

Table 8: Complementary metrics for plane geometry SVG generation.

Model	Acc↑	DSim↑	KID↓	CLIP↑	cBLEU↑	EED↓	Coverage↑	Average↑
Non-reasoning models								
DeepSeek-V3								
DeepSeek-V3.1	5.5	80.8	3.5	68.2	21.9	32.8	89.1	77.9
Claude Opus 4.1	6.4	81.1	2.4	68.5	21.0	33.6	86.4	77.2
Gemini 2.0 Flash	12.7	81.8	3.4	74.8	15.0	37.1	96.4	77.2
Gemini 2.5 Flash non-reasoning	5.5	83.1	1.5	59.7	26.9	29.8	79.1	78.7
GPT-4.1	9.1	86.5	1.0	61.4	22.7	31.4	84.5	78.7
GPT-5 Chat	10.0	82.9	4.1	73.2	16.9	38.5	87.3	75.9
	10.0	82.9	4.0	74.6	21.2	33.5	81.8	78.7
Rasoning models								
DeepSeek-R1	40.9	84.1	0.9	59.2	22.9	30.8	62.7	80.4
DeepSeek-V3.1 reasoning	39.1	83.1	1.1	65.2	20.8	32.4	63.6	79.7
Claude Opus 4.1 thinking	23.6	81.6	3.8	77.8	14.5	36.9	97.3	79.5
Gemini 2.5 Flash reasoning	55.5	87.8	0.7	65.2	19.7	36.1	84.5	85.4
Gemini 2.5 Pro	62.7	88.4	0.7	72.5	20.7	31.8	90.0	90.3
o4-mini	62.7	88.7	1.5	64.0	20.6	32.9	80.9	87.1
GPT-5	62.7	83.5	4.4	76.3	17.4	35.5	94.5	90.1
Prompting								
Gemini 2.5 Pro	62.7	88.4	0.7	72.5	20.7	31.8	90.0	90.3
w/ Zero-shot CoT	66.4	81.4	2.3	64.6	17.7	33.2	90.0	86.1
w/ Plan-and-Solve	69.1	83.9	1.3	69.0	17.8	32.5	91.8	88.4
w/ Step-Back	64.5	81.7	2.4	66.3	17.0	33.4	91.8	86.0
w/ LOOP (ours)	80.9	84.6	1.1	62.6	19.2	31.6	93.6	90.6
GPT-5	75.5	83.5	4.4	76.3	17.4	35.5	94.5	90.1
w/ Zero-shot CoT	80.0	84.6	3.9	74.4	16.0	36.3	96.4	90.2
w/ Plan-and-Solve	77.3	82.1	5.6	76.7	15.0	37.4	95.5	88.6
w/ Step-Back	75.5	82.9	4.7	75.3	16.6	35.8	94.5	89.3
w/ LOOP (ours)	80.0	84.8	3.0	74.3	17.2	35.3	94.5	90.9
Gemini 2.5 Flash non-reasoning	9.1	86.5	1.0	61.4	22.7	31.4	84.5	78.7
w/ LOOP (ours)	45.5	86.7	0.8	64.8	21.1	33.6	67.3	82.0
GPT-5 Chat	10.0	82.9	4.0	74.6	21.2	33.5	81.8	78.7
w/ LOOP (ours)	21.8	83.7	3.5	73.4	18.9	34.5	86.4	80.0
DeepSeek-V3.2 reasoning	25.5	84.1	1.3	63.3	25.0	31.3	73.6	81.0
w/ LOOP (ours)	48.2	87.1	0.8	55.0	27.5	28.8	68.2	85.2
Claude Opus 4.1 thinking	23.6	81.6	3.8	77.8	14.5	36.9	97.3	79.5
w/ LOOP (ours)	30.9	83.3	2.1	71.5	15.0	36.1	96.4	80.4

1728

1729

1730

1731

1732

1733

1734

1735

1736

Table 9: Complementary metrics for plane geometry EPS generation.

Model	Acc↑	DSim↑	KID↓	CLIP↑	cBLEU↑	EED↓	Coverage↑	Average↑
Non-reasoning models								
DeepSeek-V3								
DeepSeek-V3.1	7.3	65.7	19.0	37.5	12.9	40.9	50.9	56.2
Claude Opus 4.1	9.1	66.9	16.9	35.5	8.6	43.8	55.5	54.4
Gemini 2.0 Flash	20.9	80.8	2.1	54.1	10.1	43.4	84.5	69.1
Gemini 2.5 Flash non-reasoning	1.8	63.4	23.1	34.2	19.3	36.6	43.6	56.9
GPT-4.1	5.5	60.4	29.3	29.9	11.3	44.0	39.1	49.3
GPT-5 Chat	14.5	69.2	11.8	44.9	13.1	46.3	62.7	60.8
	7.3	72.5	7.6	45.9	16.6	39.7	60.9	63.7
Rasoning models								
DeepSeek-R1	39.1	85.2	1.7	45.9	18.4	32.7	59.1	74.6
DeepSeek-V3.1 reasoning	27.3	77.5	4.4	38.4	14.0	36.0	51.8	65.3
Claude Opus 4.1 thinking	17.3	81.6	1.5	58.2	17.7	38.7	79.1	73.6
Gemini 2.5 Flash reasoning	41.8	79.9	2.8	48.3	14.7	45.8	65.5	70.5
Gemini 2.5 Pro	56.4	83.4	0.7	51.2	17.2	38.2	76.4	79.1
o4-mini	55.5	86.1	0.3	52.9	19.6	35.9	76.4	81.5
GPT-5	66.4	87.0	-0.6	57.6	19.3	37.9	78.2	84.2
Prompting								
Gemini 2.5 Pro	56.4	83.4	0.7	51.2	17.2	38.2	76.4	79.1
w/ Zero-shot CoT	61.8	86.6	-0.4	53.9	17.1	39.5	78.2	81.2
w/ Plan-and-Solve	66.4	84.3	0.8	52.9	17.3	38.4	75.5	81.2
w/ Step-Back	59.1	84.1	0.1	50.1	16.2	39.0	73.6	78.5
w/ LOOP (ours)	62.7	85.9	-0.3	50.2	18.4	36.1	74.5	81.4
GPT-5	66.4	87.0	-0.6	57.6	19.3	37.9	78.2	84.2
w/ Zero-shot CoT	75.5	87.6	-0.8	56.2	18.6	38.5	80.0	85.5
w/ Plan-and-Solve	70.9	86.2	-0.3	56.9	15.2	42.1	85.5	82.8
w/ Step-Back	72.7	88.4	-0.6	57.4	18.1	38.7	82.7	85.5
w/ LOOP (ours)	77.3	89.4	0.0	59.1	19.9	37.6	86.4	88.5
Gemini 2.5 Flash non-reasoning	5.5	60.4	29.3	29.9	11.3	44.0	39.1	49.3
w/ LOOP (ours)	34.5	67.8	15.8	35.4	14.3	40.2	51.8	62.3
GPT-5 Chat	7.3	72.5	7.6	45.9	16.6	39.7	60.9	63.7
w/ LOOP (ours)	24.5	81.7	1.9	51.7	16.3	37.3	73.6	72.4
DeepSeek-V3.2 reasoning	18.2	79.8	4.3	44.5	15.5	38.7	65.5	67.5
w/ LOOP (ours)	17.3	78.9	3.6	46.3	17.5	35.7	56.4	67.8
Claude Opus 4.1 thinking	17.3	81.6	1.5	58.2	17.7	38.7	79.1	73.6
w/ LOOP (ours)	22.7	86.8	1.4	60.3	19.8	36.2	88.2	78.9

1774

1775

1776

1777

1778

1779

1780

1781

Table 10: Complementary metrics for molecular structure TikZ generation.

Model	Acc↑	DSim↑	KID↓	CLIP↑	cBLEU↑	EED↓	Coverage↑	Average↑
Fine-tuned models								
AutomaTikZ								
AutomaTikZ	0.0	24.0	177.3	0.00	0.00	96.8	0.0	0.7
TikZero+	0.0	26.1	155.5	0.00	0.00	69.9	0.0	8.5
AutomaTikZ resampling	0.0	27.7	122.7	0.06	0.00	90.8	0.0	7.1
DeTikZify resampling	0.0	43.4	34.8	0.12	0.01	54.3	0.7	25.2
Non-reasoning models								
DeepSeek-V3	5.3	72.9	15.9	3.70	0.49	39.0	15.3	40.08
DeepSeek-V3.1	6.3	75.4	12.5	4.91	0.61	38.2	19.0	42.04
Claude Opus 4.1	24.3	80.9	6.0	8.26	0.69	38.7	52.0	52.27
Gemini 2.0 Flash	6.0	68.9	13.0	5.14	0.31	51.3	18.7	37.68
Gemini 2.5 Flash non-reasoning	22.7	64.4	21.5	3.25	0.25	53.9	34.3	40.38
GPT-4.1	19.0	71.6	18.1	3.14	0.34	44.6	33.7	43.41
GPT-5 Chat	16.0	66.0	19.9	1.88	0.06	51.1	24.7	38.47
Rasoning models								
DeepSeek-R1	18.3	69.9	16.5	2.64	0.29	44.2	25.7	41.82
DeepSeek-V3.1 reasoning	31.0	71.0	12.8	3.14	0.54	39.5	46.3	48.85
Claude Opus 4.1 thinking	26.7	83.6	4.5	8.97	0.71	37.1	54.7	54.27
Gemini 2.5 Flash reasoning	32.0	60.0	31.3	3.06	0.15	53.1	38.7	41.03
Gemini 2.5 Pro	41.3	78.6	8.1	5.39	0.62	34.4	72.3	57.96
o4-mini	33.3	69.5	15.4	4.39	0.28	41.1	43.7	48.09
GPT-5	52.3	71.3	13.6	3.01	0.81	42.7	74.7	56.20
Prompting								
Gemini 2.5 Pro	41.3	78.6	8.1	5.39	0.62	34.4	72.3	57.96
w/ Zero-shot CoT	47.7	79.7	6.8	5.13	0.77	33.7	70.3	59.32
w/ Plan-and-Solve	41.3	77.6	8.2	4.63	0.63	33.8	70.3	57.44
w/ Step-Back	40.7	79.4	6.3	5.56	0.68	33.3	65.7	57.49
w/ LOOP (ours)	47.7	80.2	6.3	7.03	0.68	33.5	71.0	59.90
GPT-5	52.3	71.3	13.6	3.01	0.81	42.7	74.7	56.20
w/ Zero-shot CoT	53.0	70.0	14.8	2.79	0.76	42.4	73.7	55.82
w/ Plan-and-Solve	52.3	71.5	13.4	3.03	0.70	39.6	73.3	56.66
w/ Step-Back	50.7	70.8	13.7	3.05	0.69	41.5	72.3	55.67
w/ LOOP (ours)	55.0	76.7	9.4	4.72	0.78	36.7	79.3	60.40
Gemini 2.5 Flash non-reasoning	22.7	64.4	21.5	3.25	0.25	53.9	34.3	40.38
w/ LOOP (ours)	21.3	60.5	25.4	4.19	0.16	56.2	26.0	37.44
GPT-5 Chat	16.0	66.0	19.9	1.88	0.06	51.1	24.7	38.47
w/ LOOP (ours)	19.0	69.7	17.0	2.61	0.13	48.5	22.3	40.38
DeepSeek-V3.2 reasoning	14.0	70.0	14.6	3.65	0.66	40.5	28.3	42.78
w/ LOOP (ours)	16.0	75.2	10.4	4.76	0.75	38.3	33.7	46.06
Claude Opus 4.1 thinking	26.7	83.6	4.5	8.97	0.71	37.1	54.7	54.27
w/ LOOP (ours)	30.7	83.7	4.9	9.43	0.70	37.0	54.7	55.05

1836
1837
1838
1839
1840
1841
1842
1843
1844

Table 11: Complementary metrics for molecular structure SVG generation.

Model	Acc↑	DSim↑	KID↓	CLIP↑	cBLEU↑	EED↓	Coverage↑	Average↑
Non-reasoning models								
DeepSeek-V3	3.3	20.77	171.88	11.09	0.47	45.71	10.00	15.5
DeepSeek-V3.1	3.7	22.79	161.72	11.75	0.33	49.49	24.84	18.3
Claude Opus 4.1	26.0	30.04	152.34	7.81	0.58	48.64	66.00	30.1
Gemini 2.0 Flash	3.7	39.08	96.48	4.47	0.53	47.65	22.36	25.7
Gemini 2.5 Flash non-reasoning	11.3	39.68	103.13	3.70	0.44	56.45	27.16	25.3
GPT-4.1	15.0	28.96	145.31	5.55	0.40	48.00	24.67	22.0
GPT-5 Chat	14.3	30.30	146.09	3.20	0.35	48.81	23.00	21.3
Reasoning models								
DeepSeek-R1	20.0	27.48	143.75	4.48	0.52	47.01	29.67	23.5
DeepSeek-V3.1 reasoning	7.3	22.73	164.84	9.79	0.49	48.47	31.89	19.6
Claude Opus 4.1 thinking	27.7	30.34	153.13	6.75	0.68	44.91	62.33	30.5
Gemini 2.5 Flash reasoning	39.3	51.54	66.80	2.81	0.37	52.59	45.63	38.9
Gemini 2.5 Pro	63.3	36.66	125.78	6.78	0.53	42.48	73.09	42.3
o4-mini	42.7	28.43	155.47	6.09	0.38	46.60	56.62	31.1
GPT-5	55.7	37.97	121.09	4.07	0.60	51.98	76.00	39.6
Prompting								
Gemini 2.5 Pro	63.3	36.66	125.78	6.78	0.53	42.48	73.09	42.3
w/ Zero-shot CoT	63.0	36.85	122.66	5.81	0.61	42.61	72.85	42.3
w/ Plan-and-Solve	58.7	35.59	125.00	5.61	0.57	42.44	70.53	40.8
w/ Step-Back	54.3	41.46	104.69	4.55	0.52	42.99	64.69	41.6
w/ LOOP (ours)	64.7	33.94	135.94	7.98	0.65	42.53	75.33	41.8
GPT-5	55.7	37.97	121.09	4.07	0.60	51.98	76.00	39.6
w/ Zero-shot CoT	52.0	42.00	107.03	3.26	0.45	54.45	75.00	40.0
w/ Plan-and-Solve	50.3	41.66	107.81	3.75	0.53	52.63	72.00	39.6
w/ Step-Back	51.7	37.52	122.66	4.29	0.54	51.40	74.33	38.5
w/ LOOP (ours)	57.3	31.55	150.78	5.12	0.59	47.66	81.00	38.0
Gemini 2.5 Flash non-reasoning	11.3	39.68	103.13	3.70	0.44	56.45	27.16	25.3
w/ LOOP (ours)	30.0	46.39	82.81	3.21	0.34	57.12	39.68	33.1
GPT-5 Chat	14.3	30.30	146.09	3.20	0.35	48.81	23.00	21.3
w/ LOOP (ours)	19.7	27.16	159.38	5.27	0.59	45.16	28.00	22.5
DeepSeek-V3.2 reasoning	10.7	27.40	142.97	7.41	0.68	49.65	25.00	21.3
w/ LOOP (ours)	27.7	28.11	146.09	7.01	0.74	47.89	38.00	26.5
Claude Opus 4.1 thinking	27.7	30.34	153.13	6.75	0.68	44.91	62.33	30.5
w/ LOOP (ours)	27.0	27.25	166.41	7.64	0.82	45.15	59.00	28.3

1882
1883
1884
1885
1886
1887
1888
1889

1890

1891

1892

1893

1894

1895

1896

1897

1898

Table 12: Complementary metrics for molecular structure EPS generation.

Model	Acc↑	DSim↑	KID↓	CLIP↑	cBLEU↑	EED↓	Coverage↑	Average↑
Non-reasoning models								
DeepSeek-V3	3.0	61.38	31.25	2.61	0.03	49.28	9.33	32.6
DeepSeek-V3.1	3.3	64.77	24.90	1.77	0.03	49.10	13.33	34.3
Claude Opus 4.1	16.0	77.47	11.13	5.14	0.07	49.79	58.33	47.4
Gemini 2.0 Flash	0.7	29.54	140.63	0.03	0.01	56.48	0.67	13.5
Gemini 2.5 Flash non-reasoning	14.3	58.86	34.57	2.23	0.04	57.81	31.33	35.1
GPT-4.1	13.7	66.67	22.75	3.33	0.05	51.73	24.33	38.1
GPT-5 Chat	11.0	52.22	49.80	0.46	0.01	54.50	16.00	30.1
Reasoning models								
DeepSeek-R1	19.7	67.07	24.22	1.34	0.05	48.31	27.33	39.9
DeepSeek-V3.1 reasoning	20.7	65.49	22.66	1.48	0.04	49.62	33.00	40.4
Claude Opus 4.1 thinking	23.3	84.57	6.74	7.93	0.11	47.26	54.00	50.9
Gemini 2.5 Flash reasoning	34.7	57.88	38.67	2.63	0.03	54.74	43.33	40.7
Gemini 2.5 Pro	57.3	79.00	3.98	9.83	0.08	45.14	76.67	60.1
o4-mini	39.0	71.66	14.84	4.94	0.04	49.13	51.67	49.1
GPT-5	49.7	69.79	18.26	3.13	0.05	57.29	73.33	51.4
Prompting								
Gemini 2.5 Pro	57.3	79.00	3.98	9.83	0.08	45.14	76.67	60.1
w/ Zero-shot CoT	58.7	79.59	3.91	8.83	0.10	45.61	79.33	60.6
w/ Plan-and-Solve	55.0	79.22	3.88	9.25	0.07	45.44	76.00	59.5
w/ Step-Back	56.3	77.40	4.83	7.94	0.07	45.62	73.67	58.6
w/ LOOP (ours)	67.7	80.65	3.81	9.14	0.09	44.97	79.67	62.7
GPT-5	49.7	69.79	18.26	3.13	0.05	57.29	73.33	51.4
w/ Zero-shot CoT	49.3	69.82	17.19	3.57	0.05	56.81	70.00	51.1
w/ Plan-and-Solve	50.7	70.57	15.92	4.05	0.04	55.41	70.33	52.1
w/ Step-Back	48.3	68.53	20.31	3.12	0.04	56.33	68.67	50.3
w/ LOOP (ours)	54.3	76.43	9.67	4.73	0.07	50.75	79.00	56.8
Gemini 2.5 Flash non-reasoning	14.3	58.86	34.57	2.23	0.04	57.81	31.33	35.1
w/ LOOP (ours)	22.3	54.08	46.48	2.50	0.04	59.45	34.33	34.8
GPT-5 Chat	11.0	52.22	49.80	0.46	0.01	54.50	16.00	30.1
w/ LOOP (ours)	18.7	70.96	20.70	2.19	0.06	48.52	22.33	40.2
DeepSeek-V3.2 reasoning	6.7	68.01	19.24	1.86	0.04	51.89	18.67	36.2
w/ LOOP (ours)	9.0	73.88	14.65	3.59	0.08	51.96	23.33	39.3
Claude Opus 4.1 thinking	23.3	84.57	6.74	7.93	0.11	47.26	54.00	50.9
w/ LOOP (ours)	22.3	83.46	6.59	7.45	0.10	47.23	51.67	50.1

1936

1937

1938

1939

1940

1941

1942

1943

1944
1945

E.4 EVALUATION ON THE DATIKZ v3 DATASET

1946
1947
1948
1949
1950

We evaluate the robustness of the proposed LOOP using the DaTikZ v3 dataset. The same prompt from the plane geometry task is employed. For comparison, we also include the fine-tuned models in the evaluation. Because DaTikZ v3 publishes only a subset of its test set, the fine-tuned models are re-evaluated. The results are shown in Figure 13. LOOP improves performance even on DaTikZ v3, demonstrating its robustness.

1951
1952
1953

Table 13: Evaluation on the DaTikZ v3 Dataset.

Model	DSim↑	KID↓	CLIP↑	cBLEU↑	EED↓	Average↑
AutomaTikZ resampling	46.2	26.4	10.4	1.4	58.4	20.3
TikZero+ resampling	47.1	22.1	10.0	2.5	59.1	40.6
Gemini 2.5 Pro	46.6	14.2	28.4	1.7	60.7	30.7
w/ LOOP (ours)	50.8	8.3	36.8	1.9	59.1	70.9
GPT-5	53.1	6.1	42.3	2.2	59.5	84.7
w/ LOOP (ours)	53.5	6.1	39.5	2.5	59.6	87.5

1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997

1998
1999

E.5 ABLATIONS ON LOOP

2000
2001
2002

To identify the contribution of each component of LOOP to the overall performance, we perform an ablation study. The target task is plane geometry SVG generation using Gemini 2.5 Pro. The experiments are performed under the following three settings.

2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017

- w/o information extraction: “*Let’s think step by step, following this workflow: 1. Relationship extraction: describe the relationships among the elements. 2. Mathematical reasoning: compute the attributes of each element so that they satisfy those relationships. 3. Code generation: generate the TikZ.*”
- w/o relationship extraction: “*Let’s think step by step, following this workflow: 1. Information extraction: describe the necessary elements. 2. Mathematical reasoning: compute the attributes of each element. 3. Code generation: generate the TikZ.*”
- w/o mathematical reasoning: “*Let’s think step by step, following this workflow: 1. Information extraction: describe the necessary elements. 2. Relationship extraction: describe their relationships. 3. Code generation: generate the TikZ.*”
- w/o code generation: “*Let’s think step by step, following this workflow: 1. Information extraction: describe the necessary elements. 2. Relationship extraction: describe their relationships. 3. Mathematical reasoning: compute the attributes of each element so that they satisfy those relationships.*”

2018
2019
2020

We show the results of applying these prompts in Figure 14. The original prompt achieves the best performance, clearly demonstrating that each component contributes to the improvement.

2021
2022
2023

Table 14: The performance of Gemini 2.5 Pro with different prompts in the plane-geometry SVG generation task.

Method	Accuracy
LOOP	80.9
w/o information extraction	72.7
w/o relationship extraction	72.7
w/o mathematical reasoning	70.0
w/o code generation	72.7

2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051

2052 F DETAILS OF THE EVALUATION CODE

2053
 2054 We describe below the evaluation code used in our SSVG-Bench, specifically for the SVG format.
 2055 For TikZ and EPS, since we first compile them to PDF and then convert them to SVG using the
 2056 `pdf2svg` command before processing, the code is nearly identical to that used for SVG.
 2057

2058 F.1 PLANE GEOMETRY

2059 F.1.1 PARSING CODE

2060 We first present the parsing code for extracting straight lines, circles, and ellipses from vector data
 2061 (in this case, SVG), as shown in Figures 23, 24, 25, 26, and 27. This enables subsequent structural
 2062 analysis.
 2063

2064 **Overview.** The code uses Python’s built-in XML parser (`xml.etree.ElementTree`) to traverse
 2065 the SVG document tree. For selected shapes, it extracts the coordinates of geometric primitives in a
 2066 uniform form:
 2067

- 2068 • Line-like shapes are decomposed into straight line segments of the form
 2069 $((x_1, y_1), (x_2, y_2))$.
 2070
- 2071 • Circles are returned as $((cx, cy), r)$.
 2072
- 2073 • Ellipses are represented as $((cx, cy), R_1, R_2, \theta)$, where R_1, R_2 are the principal semi-axes
 (radii) and θ is the rotation of the major axis from the x -axis.
 2074

2075 All extracted data can optionally be filtered by CSS class.
 2076

2077 **Extracting Line Segments.** The function `parse_segments()` scans the SVG tree and detects
 2078 elements whose geometry can be expressed as connected straight lines:
 2079

- 2080 • `<line>` elements directly provide two endpoints.
 2081
- 2082 • `<polygon>` and `<polyline>` elements contain a sequence of vertex coordinates, where
 2083 each consecutive pair defines a segment (polygons also include the closing edge).
 2084
- 2085 • `<rect>` elements are decomposed into four boundary segments.
 2086
- 2087 • `<path>` elements are partially supported for straight motions (M, L). Each L instruction
 2088 yields a segment from the previous position.
 2089

2090 For all these shapes, the function returns a list of straight-line segments.
 2091

2092 **Extracting Circles.** The function `parse_circles()` searches for `<circle>` elements and
 2093 extracts their center (cx, cy) and radius r . When a filter is specified, only circles matching the
 2094 given CSS class are processed.
 2095

2096 **Extracting Ellipses with Affine Transforms.** The function `parse_ellipses()` identifies
 2097 `<ellipse>` elements as well as transformed `<circle>` elements. It supports general SVG affine
 2098 transformations including: `matrix`, `translate`, `rotate`, `scale`, `skewX`, `skewY`. The trans-
 2099 formation is processed as follows:
 2100

- 2101 1. Construct the affine matrix from the SVG `transform` attribute.
 2102
- 2103 2. Apply the affine transformation to the ellipse center.
 2104
- 2105 3. Decompose the linear part of the affine transform applied to the ellipse radii. The principal
 2106 radii R_1, R_2 and orientation θ are obtained from eigenvalue analysis of the matrix AA^T .
 2107

2108 **Result.** The function `parse_svg_file()` loads the SVG data, calls the three extraction routines,
 2109 and returns: `segs`, `circs`, `ells`, where:
 2110

- 2111 • `segs` : list of line segments $(x_1, y_1), (x_2, y_2)$.
 2112
- 2113 • `circs` : list of circles $((cx, cy), r)$.
 2114
- 2115 • `ells` : list of ellipses $((cx, cy), R_1, R_2, \theta)$.
 2116

```

2106
2107
2108     1 #!/usr/bin/env python3
2109     2 # -*- coding: utf-8 -*-
2110
2111     3
2112     4 import re
2113     5 import math
2114     6 import xml.etree.ElementTree as ET
2115
2116     7
2117     8
2118     9 def tag(elem):
2119    10     """Return the tag name without namespace."""
2120    11     return elem.tag.split('}')[ -1]
2121
2122    12
2123    13
2124    14 def parse_segments(root, filter_class=None):
2125    15     """
2126    16         Decompose line, polygon, polyline, path, and rect elements into line segments.
2127    17         If filter_class is specified, only elements that have that class attribute are used.
2128    18         Returns: [ ((x1, y1), (x2, y2)), ... ]
2129    19     """
2130    20     segs = []
2131
2132    21
2133    22     def ok_class(elem):
2134    23         if filter_class is None:
2135    24             return True
2136    25         return filter_class in elem.get('class', '').split()
2137
2138    26
2139    27     for elem in root.iter():
2140    28         t = tag(elem)
2141
2142    29
2143    30     def safe_float(value):
2144    31         try:
2145    32             return float(value)
2146    33         except (TypeError, ValueError):
2147    34             return 0.0
2148
2149    35
2150    36     if t == 'line' and ok_class(elem):
2151    37         x1 = safe_float(elem.get('x1', 0))
2152    38         y1 = safe_float(elem.get('y1', 0))
2153    39         x2 = safe_float(elem.get('x2', 0))
2154    40         y2 = safe_float(elem.get('y2', 0))
2155    41         segs.append(((x1, y1), (x2, y2)))
2156
2157    42
2158    43     elif t == 'polygon' and ok_class(elem):
2159    44         pts = re.split(r'[, \s]+', elem.get('points', '').strip())
2160    45         coords = [float(v) for v in pts if v]
2161    46         pts_list = list(zip(coords[0::2], coords[1::2]))
2162    47         for i in range(len(pts_list)):
2163    48             segs.append((pts_list[i], pts_list[(i + 1) % len(pts_list)]))
2164
2165    49
2166    50     elif t == 'polyline' and ok_class(elem):
2167    51         pts = re.split(r'[, \s]+', elem.get('points', '').strip())
2168    52         coords = [float(v) for v in pts if v]
2169    53         pts_list = list(zip(coords[0::2], coords[1::2]))
2170    54         for i in range(len(pts_list) - 1):
2171    55             segs.append((pts_list[i], pts_list[i + 1]))
2172
2173    56
2174    57     elif t == 'path' and ok_class(elem):
2175    58         d = elem.get('d', '')
2176    59         tokens = re.findall(r'[ML] | [-+]? \d* \. ? \d+ (?: \. \d+)?', d)
2177    60         cur = None
2178    61         i = 0
2179    62         while i < len(tokens):
2180    63             tok = tokens[i]
2181    64             if tok == 'M':
2182                 cur = (float(tokens[i + 1]), float(tokens[i + 2]))

```

Figure 23: SVG parsing code (1/5).

```

2160
2161
2162
2163
2164     i += 3
2165     elif tok == 'L' and cur is not None:
2166         nxt = (float(tokens[i + 1]), float(tokens[i + 2]))
2167         segs.append((cur, nxt))
2168         cur = nxt
2169         i += 3
2170     else:
2171         i += 1
2172
2173
2174     elif t == 'rect' and ok_class(elem):
2175         x = float(elem.get('x', '0'))
2176         y = float(elem.get('y', '0'))
2177         w = float(elem.get('width', '0'))
2178         h = float(elem.get('height', '0'))
2179         p1 = (x, y)
2180         p2 = (x + w, y)
2181         p3 = (x + w, y + h)
2182         p4 = (x, y + h)
2183         segs.extend([(p1, p2), (p2, p3), (p3, p4), (p4, p1)])
2184
2185
2186     return segs
2187
2188
2189
2190
2191 def parse_circles(root, filter_class=None):
2192     """
2193     Extract circle elements. If filter_class is specified, filter by that class.
2194     Returns: [ ((cx, cy), r), ... ]
2195     """
2196
2197     circs = []
2198
2199     for elem in root.iter():
2200         if tag(elem) == 'circle':
2201             cls = elem.get('class', '').split()
2202             if filter_class is None or filter_class in cls:
2203                 cx = float(elem.get('cx', 0))
2204                 cy = float(elem.get('cy', 0))
2205                 r = float(elem.get('r', 0))
2206                 circs.append(((cx, cy), r))
2207
2208     return circs
2209
2210
2211 def parse_ellipses(root, filter_class=None):
2212     """
2213     Extract ellipse elements. Supports translate / rotate / scale / matrix in the transform attribute.
2214     Returns: [ ((cx, cy), rx, ry, angle_deg), ... ]
2215     - angle_deg: angle in degrees, counter-clockwise from the x-axis
2216     - Even with non-uniform scale or arbitrary matrices, approximates major/minor axis lengths and
2217     orientation.
2218     """
2219
2220     ellipses = []
2221
2222     # Tokenizer for transform attribute
2223     t_re = re.compile(r'(matrix|translate|rotate|scale|skewX|skewY)\s*\(([^\)]+)\)\)', re.I)
2224
2225
2226     def mult(A, B):
2227         """
2228         3x3 matrix multiplication A @ B.
2229         """
2230         return [
2231             [
2232                 A[0][0] * B[0][0] + A[0][1] * B[1][0] + A[0][2] * B[2][0],
2233                 A[0][0] * B[0][1] + A[0][1] * B[1][1] + A[0][2] * B[2][1],
2234                 A[0][0] * B[0][2] + A[0][1] * B[1][2] + A[0][2] * B[2][2],
2235             ],
2236             [
2237                 A[1][0] * B[0][0] + A[1][1] * B[1][0] + A[1][2] * B[2][0],
2238                 A[1][0] * B[0][1] + A[1][1] * B[1][1] + A[1][2] * B[2][1],
2239                 A[1][0] * B[0][2] + A[1][1] * B[1][2] + A[1][2] * B[2][2],
2240             ],
2241             [0, 0, 1],
2242         ]
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3297
3298
3299
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3397
3398
3399
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3497
3498
3499
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3528
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3597
3598
3599
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697
3697
3698
3699
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3738
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3768
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3797
3798
3799
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3818
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3828
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3838
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3848
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3858
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3868
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3878
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3888
3889
3890
3891
3892
3893
3894
3895
3896
3897
3897
3898
3899
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3908
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3918
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3928
3929
3930
3931
3932
3933
3934
3935
3936
3937
3938
3938
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3948
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3988
3989
3990
3991
3992
3993
3994
3995
3996
3997
3997
3998
3999
3999
4000
4001
4002
4003
4004
4005
4006
4007
4008
4008
4009
4010
4011
4012
4013
4014
4015
4016
4017
4018
4018
4019
4020
4021
4022
4023
4024
4025
4026
4027
4028
4028
4029
4030
4031
4032
4033
4034
4035
4036
4037
4038
4038
4039
4040
4041
4042
4043
4044
4045
4046
4047
4048
4048
4049
4050
4051
4052
4053
4054
4055
4056
4057
4058
4058
4059
4060
4061
4062
4063
4064
4065
4066
4067
4068
4068
4069
4070
4071
4072
4073
4074
4075
4076
4077
4078
4078
4079
4080
4081
4082
4083
4084
4085
4086
4087
4088
4088
4089
4090
4091
4092
4093
4094
4095
4096
4097
4097
4098
4099
4099
4100
4101
4102
4103
4104
4105
4106
4107
4108
4109
4110
4111
4112
4113
4114
4115
4116
4117
4118
4119
4120
4121
4122
4123
4124
4125
4126
4127
4128
4129
4130
4131
4132
4133
4134
4135
4136
4137
4138
4139
4140
4141
4142
4143
4144
4145
4146
4147
4148
4149
4149
4150
4151
4152
4153
4154
4155
4156
4157
4158
4159
4159
4160
4161
4162
4163
4164
4165
4166
4167
4168
4169
4169
4170
4171
4172
4173
4174
4175
4176
4177
4178
4178
4179
4180
4181
4182
4183
4184
4185
4186
4187
4188
4188
4189
4190
4191
4192
4193
4194
4195
4196
4197
4198
4199
4199
4200
4201
4202
4203
4204
4205
4206
4207
4208
4208
4209
4210
4211
4212
4213
4214
4215
4216
4217
4218
4219
4219
4220
4221
4222
4223
4224
4225
4226
4227
4228
4229
4229
4230
4231
4232
4233
4234
4235
4236
4237
4238
4239
4239
4240
4241
4242
4243
4244
4245
4246
4247
4248
4248
4249
4250
4251
4252
4253
4254
4255
4256
4257
4258
4259
4259
4260
4261
4262
4263
4264
4265
4266
4267
4268
4268
4269
4270
4271
4272
4273
4274
4275
4276
4277
4278
4278
4279
4280
4281
4282
4283
4284
4285
4286
4287
4288
4288
4289
4290
4291
4292
4293
4294
4295
4296
4297
4297
4298
4299
4299
4300
4301
4302
4303
4304
4305
4306
4307
4308
4308
4309
4310
4311
4312
4313
4314
4315
4316
4317
4318
4318
4319
4320
4321
4322
4323
4324
4325
4326
4327
4328
4329
4329
4330
4331
4332
4333
4334
4335
4336
4337
4338
4339
4339
4340
4341
4342
4343
4344
4345
4346
4347
4348
4348
4349
4350
4351
4352
4353
4354
4355
4356
```

Figure 24: SVG parsing code (2/5).

```

2214
2215
2216
2217
2218     132 |     def mat_translate(tx, ty=0.0):
2219     133 |         return [[1, 0, tx], [0, 1, ty], [0, 0, 1]]
2220     134 |
2221     135 |
2222     136 |     def mat_rotate(angle_deg, cx=0.0, cy=0.0):
2223     137 |         a = math.radians(angle_deg)
2224     138 |         c = math.cos(a)
2225     139 |         s = math.sin(a)
2226     140 |         # Affine transform including rotation around center (cx, cy)
2227     141 |         return mult(
2228     142 |             mult(mat_translate(cx, cy), [[c, -s, 0], [s, c, 0], [0, 0, 1]]),
2229     143 |             mat_translate(-cx, -cy),
2230     144 |         )
2231     145 |
2232     146 |     def mat_scale(sx, sy=None):
2233     147 |         if sy is None:
2234     148 |             sy = sx
2235     149 |         return [[sx, 0, 0], [0, sy, 0], [0, 0, 1]]
2236     150 |
2237     151 |     def mat_skewx(a_deg):
2238     152 |         t = math.tan(math.radians(a_deg))
2239     153 |         # Note: SVG skewX corresponds to  $x' = x + \tan(ax) * y$ 
2240     154 |         return [[1, math.tan(0), 0], [t, 1, 0], [0, 0, 1]]
2241     155 |
2242     156 |     def mat_skewy(a_deg):
2243     157 |         t = math.tan(math.radians(a_deg))
2244     158 |         # Note: SVG skewY corresponds to  $y' = y + \tan(ay) * x$ 
2245     159 |         return [[1, t, 0], [math.tan(0), 1, 0], [0, 0, 1]]
2246     160 |
2247     161 |     def mat_matrix(a, b, c, d, e, f):
2248     162 |         # SVG:  $[x', y'] = [a \ c \ e; b \ d \ f; 0 \ 0 \ 1] [x, y, 1]^T$ 
2249     163 |         return [[a, c, e], [b, d, f], [0, 0, 1]]
2250     164 |
2251     165 |     def parse_transform(txt):
2252     166 |         """Compose transform string from left to right (SVG applies transforms in that order)."""
2253     167 |         M = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
2254     168 |         for m in t_re.finditer(txt or ''):
2255     169 |             name = m.group(1).lower()
2256     170 |             parts = [float(v) for v in re.split(r'[, \s]+', m.group(2).strip()) if v]
2257     171 |             if name == 'matrix' and len(parts) == 6:
2258     172 |                 Mi = mat_matrix(*parts)
2259     173 |             elif name == 'translate':
2260     174 |                 Mi = mat_translate(parts[0], parts[1] if len(parts) > 1 else 0.0)
2261     175 |             elif name == 'rotate':
2262     176 |                 if len(parts) >= 3:
2263     177 |                     Mi = mat_rotate(parts[0], parts[1], parts[2])
2264     178 |                 else:
2265     179 |                     Mi = mat_rotate(parts[0])
2266     180 |             elif name == 'scale':
2267     181 |                 Mi = mat_scale(parts[0], parts[1] if len(parts) > 1 else None)
2268     182 |             elif name == 'skewx':
2269     183 |                 Mi = mat_skewx(parts[0])
2270     184 |             elif name == 'skewy':
2271     185 |                 Mi = mat_skewy(parts[0])
2272     186 |             else:
2273     187 |                 continue
2274     188 |             M = mult(M, Mi)
2275     189 |
2276     190 |         return M
2277     191 |
2278     192 |     def apply_affine(M, x, y):
2279     193 |         """Apply 3x3 affine matrix to point (x, y)."""
2280     194 |         return (
2281     195 |             M[0][0] * x + M[0][1] * y + M[0][2],
2282     196 |             M[1][0] * x + M[1][1] * y + M[1][2],
2283     197 |         )
2284     198 |
2285     199 |     def decompose_axes(Mlin, rx, ry):
2286     200 |
2287     201 |
2288     202 |
2289     203 |
2290     204 |
2291     205 |
2292     206 |
2293     207 |
2294     208 |
2295     209 |
2296     210 |
2297     211 |
2298     212 |
2299     213 |
2300     214 |
2301     215 |
2302     216 |
2303     217 |
2304     218 |
2305     219 |
2306     220 |
2307     221 |
2308     222 |
2309     223 |
2310     224 |
2311     225 |
2312     226 |
2313     227 |
2314     228 |
2315     229 |
2316     230 |
2317     231 |
2318     232 |
2319     233 |
2320     234 |
2321     235 |
2322     236 |
2323     237 |
2324     238 |
2325     239 |
2326     240 |
2327     241 |
2328     242 |
2329     243 |
2330     244 |
2331     245 |
2332     246 |
2333     247 |
2334     248 |
2335     249 |
2336     250 |
2337     251 |
2338     252 |
2339     253 |
2340     254 |
2341     255 |
2342     256 |
2343     257 |
2344     258 |
2345     259 |
2346     260 |
2347     261 |
2348     262 |
2349     263 |
2350     264 |
2351     265 |
2352     266 |
2353     267 |
2354     268 |
2355     269 |
2356     270 |
2357     271 |
2358     272 |
2359     273 |
2360     274 |
2361     275 |
2362     276 |
2363     277 |
2364     278 |
2365     279 |
2366     280 |
2367     281 |
2368     282 |
2369     283 |
2370     284 |
2371     285 |
2372     286 |
2373     287 |
2374     288 |
2375     289 |
2376     290 |
2377     291 |
2378     292 |
2379     293 |
2380     294 |
2381     295 |
2382     296 |
2383     297 |
2384     298 |
2385     299 |
2386     300 |
2387     301 |
2388     302 |
2389     303 |
2390     304 |
2391     305 |
2392     306 |
2393     307 |
2394     308 |
2395     309 |
2396     310 |
2397     311 |
2398     312 |
2399     313 |
2400     314 |
2401     315 |
2402     316 |
2403     317 |
2404     318 |
2405     319 |
2406     320 |
2407     321 |
2408     322 |
2409     323 |
2410     324 |
2411     325 |
2412     326 |
2413     327 |
2414     328 |
2415     329 |
2416     330 |
2417     331 |
2418     332 |
2419     333 |
2420     334 |
2421     335 |
2422     336 |
2423     337 |
2424     338 |
2425     339 |
2426     340 |
2427     341 |
2428     342 |
2429     343 |
2430     344 |
2431     345 |
2432     346 |
2433     347 |
2434     348 |
2435     349 |
2436     350 |
2437     351 |
2438     352 |
2439     353 |
2440     354 |
2441     355 |
2442     356 |
2443     357 |
2444     358 |
2445     359 |
2446     360 |
2447     361 |
2448     362 |
2449     363 |
2450     364 |
2451     365 |
2452     366 |
2453     367 |
2454     368 |
2455     369 |
2456     370 |
2457     371 |
2458     372 |
2459     373 |
2460     374 |
2461     375 |
2462     376 |
2463     377 |
2464     378 |
2465     379 |
2466     380 |
2467     381 |
2468     382 |
2469     383 |
2470     384 |
2471     385 |
2472     386 |
2473     387 |
2474     388 |
2475     389 |
2476     390 |
2477     391 |
2478     392 |
2479     393 |
2480     394 |
2481     395 |
2482     396 |
2483     397 |
2484     398 |
2485     399 |
2486     400 |
2487     401 |
2488     402 |
2489     403 |
2490     404 |
2491     405 |
2492     406 |
2493     407 |
2494     408 |
2495     409 |
2496     410 |
2497     411 |
2498     412 |
2499     413 |
2500     414 |
2501     415 |
2502     416 |
2503     417 |
2504     418 |
2505     419 |
2506     420 |
2507     421 |
2508     422 |
2509     423 |
2510     424 |
2511     425 |
2512     426 |
2513     427 |
2514     428 |
2515     429 |
2516     430 |
2517     431 |
2518     432 |
2519     433 |
2520     434 |
2521     435 |
2522     436 |
2523     437 |
2524     438 |
2525     439 |
2526     440 |
2527     441 |
2528     442 |
2529     443 |
2530     444 |
2531     445 |
2532     446 |
2533     447 |
2534     448 |
2535     449 |
2536     450 |
2537     451 |
2538     452 |
2539     453 |
2540     454 |
2541     455 |
2542     456 |
2543     457 |
2544     458 |
2545     459 |
2546     460 |
2547     461 |
2548     462 |
2549     463 |
2550     464 |
2551     465 |
2552     466 |
2553     467 |
2554     468 |
2555     469 |
2556     470 |
2557     471 |
2558     472 |
2559     473 |
2560     474 |
2561     475 |
2562     476 |
2563     477 |
2564     478 |
2565     479 |
2566     480 |
2567     481 |
2568     482 |
2569     483 |
2570     484 |
2571     485 |
2572     486 |
2573     487 |
2574     488 |
2575     489 |
2576     490 |
2577     491 |
2578     492 |
2579     493 |
2580     494 |
2581     495 |
2582     496 |
2583     497 |
2584     498 |
2585     499 |
2586     500 |
2587     501 |
2588     502 |
2589     503 |
2590     504 |
2591     505 |
2592     506 |
2593     507 |
2594     508 |
2595     509 |
2596     510 |
2597     511 |
2598     512 |
2599     513 |
2600     514 |
2601     515 |
2602     516 |
2603     517 |
2604     518 |
2605     519 |
2606     520 |
2607     521 |
2608     522 |
2609     523 |
2610     524 |
2611     525 |
2612     526 |
2613     527 |
2614     528 |
2615     529 |
2616     530 |
2617     531 |
2618     532 |
2619     533 |
2620     534 |
2621     535 |
2622     536 |
2623     537 |
2624     538 |
2625     539 |
2626     540 |
2627     541 |
2628     542 |
2629     543 |
2630     544 |
2631     545 |
2632     546 |
2633     547 |
2634     548 |
2635     549 |
2636     550 |
2637     551 |
2638     552 |
2639     553 |
2640     554 |
2641     555 |
2642     556 |
2643     557 |
2644     558 |
2645     559 |
2646     560 |
2647     561 |
2648     562 |
2649     563 |
2650     564 |
2651     565 |
2652     566 |
2653     567 |
2654     568 |
2655     569 |
2656     570 |
2657     571 |
2658     572 |
2659     573 |
2660     574 |
2661     575 |
2662     576 |
2663     577 |
2664     578 |
2665     579 |
2666     580 |
2667     581 |
2668     582 |
2669     583 |
2670     584 |
2671     585 |
2672     586 |
2673     587 |
2674     588 |
2675     589 |
2676     590 |
2677     591 |
2678     592 |
2679     593 |
2680     594 |
2681     595 |
2682     596 |
2683     597 |
2684     598 |
2685     599 |
2686     600 |
2687     601 |
2688     602 |
2689     603 |
2690     604 |
2691     605 |
2692     606 |
2693     607 |
2694     608 |
2695     609 |
2696     610 |
2697     611 |
2698     612 |
2699     613 |
2700     614 |
2701     615 |
2702     616 |
2703     617 |
2704     618 |
2705     619 |
2706     620 |
2707     621 |
2708     622 |
2709     623 |
2710     624 |
2711     625 |
2712     626 |
2713     627 |
2714     628 |
2715     629 |
2716     630 |
2717     631 |
2718     632 |
2719     633 |
2720     634 |
2721     635 |
2722     636 |
2723     637 |
2724     638 |
2725     639 |
2726     640 |
2727     641 |
2728     642 |
2729     643 |
2730     644 |
2731     645 |
2732     646 |
2733     647 |
2734     648 |
2735     649 |
2736     650 |
2737     651 |
2738     652 |
2739     653 |
2740     654 |
2741     655 |
2742     656 |
2743     657 |
2744     658 |
2745     659 |
2746     660 |
2747     661 |
2748     662 |
2749     663 |
2750     664 |
2751     665 |
2752     666 |
2753     667 |
2754     668 |
2755     669 |
2756     670 |
2757     671 |
2758     672 |
2759     673 |
2760     674 |
2761     675 |
2762     676 |
2763     677 |
2764     678 |
2765     679 |
2766     680 |
2767     681 |
2768     682 |
2769     683 |
2770     684 |
2771     685 |
2772     686 |
2773     687 |
2774     688 |
2775     689 |
2776     690 |
2777     691 |
2778     692 |
2779     693 |
2780     694 |
2781     695 |
2782     696 |
2783     697 |
2784     698 |
2785     699 |
2786     700 |
2787     701 |
2788     702 |
2789     703 |
2790     704 |
2791     705 |
2792     706 |
2793     707 |
2794     708 |
2795     709 |
2796     710 |
2797     711 |
2798     712 |
2799     713 |
2800     714 |
2801     715 |
2802     716 |
2803     717 |
2804     718 |
2805     719 |
2806     720 |
2807     721 |
2808     722 |
2809     723 |
2810     724 |
2811     725 |
2812     726 |
2813     727 |
2814     728 |
2815     729 |
2816     730 |
2817     731 |
2818     732 |
2819     733 |
2820     734 |
2821     735 |
2822     736 |
2823     737 |
2824     738 |
2825     739 |
2826     740 |
2827     741 |
2828     742 |
2829     743 |
2830     744 |
2831     745 |
2832     746 |
2833     747 |
2834     748 |
2835     749 |
2836     750 |
2837     751 |
2838     752 |
2839     753 |
2840     754 |
2841     755 |
2842     756 |
2843     757 |
2844     758 |
2845     759 |
2846     760 |
2847     761 |
2848     762 |
2849     763 |
2850     764 |
2851     765 |
2852     766 |
2853     767 |
2854     768 |
2855     769 |
2856     770 |
2857     771 |
2858     772 |
2859     773 |
2860     774 |
2861     775 |
2862     776 |
2863     777 |
2864     778 |
2865     779 |
2866     780 |
2867     781 |
2868     782 |
2869     783 |
2870     784 |
2871     785 |
2872     786 |
2873     787 |
2874     788 |
2875     789 |
2876     790 |
2877     791 |
2878     792 |
2879     793 |
2880     794 |
2881     795 |
2882     796 |
2883     797 |
2884     798 |
2885     799 |
2886     800 |
2887     801 |
2888     802 |
2889     803 |
2890     804 |
2891     805 |
2892     806 |
2893     807 |
2894     808 |
2895     809 |
2896     810 |
2897     811 |
2898     812 |
2899     813 |
2900     814 |
2901     815 |
2902     816 |
2903     817 |
2904     818 |
2905     819 |
2906     820 |
2907     821 |
2908     822 |
2909     823 |
2910     824 |
2911     825 |
2912     826 |
2913     827 |
2914     828 |
2915     829 |
2916     830 |
2917     831 |
2918     832 |
2919     833 |
2920     834 |
2921     835 |
2922     836 |
2923     837 |
2924     838 |
2925     839 |
2926     840 |
2927     841 |
2928     842 |
2929     843 |
2930     844 |
2931     845 |
2932     846 |
2933     847 |
2934     848 |
2935     849 |
2936     850 |
2937     851 |
2938     852 |
2939     853 |
2940     854 |
2941     855 |
2942     856 |
2943     857 |
2944     858 |
2945     859 |
2946     860 |
2947     861 |
2948     862 |
2949     863 |
2950     864 |
2951     865 |
2952     866 |
2953     867 |
2954     868 |
2955     869 |
2956     870 |
2957     871 |
2958     872 |
2959     873 |
2960     874 |
2961     875 |
2962     876 |
2963     877 |
2964     878 |
2965     879 |
2966     880 |
2967     881 |
2968     882 |
2969     883 |
2970     884 |
2971     885 |
2972     886 |
2973     887 |
2974     888 |
2975     889 |
2976     890 |
2977     891 |
2978     892 |
2979     893 |
2980     894 |
2981     895 |
2982     896 |
2983     897 |
2984     898 |
2985     899 |
2986     900 |
2987     901 |
2988     902 |
2989     903 |
2990     904 |
2991     905 |
2992     906 |
2993     907 |
2994     908 |
2995     909 |
2996     910 |
2997     911 |
2998     912 |
2999     913 |
2999     914 |
2999     915 |
2999     916 |
2999     917 |
2999     918 |
2999     919 |
2999     920 |
2999     921 |
2999     922 |
2999     923 |
2999     924 |
2999     925 |
2999     926 |
2999     927 |
2999     928 |
2999     929 |
2999     930 |
2999     931 |
2999     932 |
2999     933 |
2999     934 |
2999     935 |
2999     936 |
2999     937 |
2999     938 |
2999     939 |
2999     940 |
2999     941 |
2999     942 |
2999     943 |
2999     944 |
2999     945 |
2999     946 |
2999     947 |
2999     948 |
2999     949 |
2999     950 |
2999     951 |
2999     952 |
2999     953 |
2999     954 |
2999     955 |
2999     956 |
2999     957 |
2999     958 |
2999     959 |
2999     960 |
2999     961 |
2999     962 |
2999     963 |
2999     964 |
2999     965 |
2999     966 |
2999     967 |
2999     968 |
2999     969 |
2999     970 |
2999     971 |
2999     972 |
2999     973 |
2999     974 |
2999     975 |
2999     976 |
2999     977 |
2999     978 |
2999     979 |
2999     980 |
2999     981 |
2999     982 |
2999     983 |
2999     984 |
2999     985 |
2999     986 |
2999     987 |
2999     988 |
2999     989 |
2999     990 |
2999     991 |
2999     992 |
2999     993 |
2999     994 |
2999     995 |
2999     996 |
2999     997 |
2999     998 |
2999     999 |
2999     1000 |
2999     1001 |
2999     1002 |
2999     1003 |
2999     1004 |
2999     1005 |
2999     1006 |
2999     1007 |
2999     1008 |
2999     1009 |
2999     1010 |
2999     1011 |
2999     1012 |
2999     1013 |
2999     1014 |
2999     1015 |
2999     1016 |
2999     1017 |
2999     1018 |
2999     1019 |
2999     1020 |
2999     1021 |
2999     1022 |
2999     1023 |
2999     1024 |
2999     1025 |
2999     1026 |
2999     1027 |
2999     1028 |
2999     1029 |
2999     1030 |
2999     1031 |
2999     1032 |
2999     1033 |
2999     1034 |
2999     1035 |
2999     1036 |
2999     1037 |
2999     1038 |
2999     1039 |
2999     1040 |
2999     1041 |
2999     1042 |
2999     1043 |
2999     1044 |
2999     1045 |
2999     1046 |
2999     1047 |
2999     1048 |
2999     1049 |
2999     1050 |
2999     1051 |
2999     1052 |
2999     1053 |
2999     1054 |
2999     1055 |
2999     1056 |
2999     1057 |
2999     1058 |
2999     1059 |
2999     1060 |
2999     1061 |
2999     1062 |
2999     1063 |
2999     1064 |
2999     1065 |
2999     1066 |
2999     1067 |
2999     1068 |
2999     1069 |
2999     1070 |
2999     1071 |
2999     1072 |
2999     1073 |
2999     1074 |
2999     1075 |
2999     1076 |
2999     1077 |
2999     1078 |
2999     1079 |
2999     1080 |
2999     1081 |
2999     1082 |
2999     1083 |
2999     1084 |
2999     1085 |
2999     1086 |
2999     1087 |
2999     1088 |
2999     1089 |
2999     1090 |
2999     1091 |
2999     1092 |
2999     1093 |
2999     1094 |
2999     1095 |
2999     1096 |
2999     1097 |
2999     1098 |
2999     1099 |
2999     1100 |
2999     1101 |
2999     1102 |
2999     1103 |
2999     1104 |
2999     1105 |
2999     1106 |
2999     1107 |
2999     1108 |
2999     1109 |
2999     1110 |
2999     1111 |
2999     1112 |
2999     1113 |
2999     1114 |
2999     1115 |
2999     1116 |
2999     1117 |
2999     1118 |
2999     1119 |
2999     1120 |
2999     1121 |
2999     1122 |
2999     1123 |
2999     1124 |
2999     1125 |
2999     1126 |
2999     1127 |
2999     1128 |
2999     1129 |
2999     1130 |
2999     1131 |
2999     1132 |
2999     1133 |
2999     1134 |
2999     1135 |
2999     1136 |
2999     1137 |
2999     1138 |
2999     1139 |
2999     1140 |
2999     1141 |
2999     1142 |
2999     1143 |
2999     1144 |
2999     1145 |
2999     1146 |
2999     1147 |
2999     1148 |
2999     1149 |
2999     1150 |
2999     1151 |
2999     1152 |
2999     1153 |
2999     1154 |
2999     1155 |
2999     1156 |
2999     1157 |
2999     1158 |
2999     1159 |
2999     1160 |
2999     1161 |
2999     11
```

```

2268
2269
2270
2271
2272     199 |     """
2273     200 |     Linear part of ellipse transform: A = Mlin @ diag(rx, ry)
2274     201 |     Major/minor axis lengths = sqrt(eigenvalues of A A^T),
2275     202 |     orientation angle = angle of the principal eigenvector.
2276     203 |     """
2277     204 |     a, c = Mlin[0][0], Mlin[0][1]
2278     205 |     b, d = Mlin[1][0], Mlin[1][1]
2279     206 |     # A = [[a*rx, c*ry],
2280     207 |     #     [b*rx, d*ry]]
2281     208 |     arx, cry = a * rx, c * ry
2282     209 |     brx, dry = b * rx, d * ry
2283     210 |     # B = A A^T = [[p, r], [r, q]]
2284     211 |     p = arx * arx + cry * cry
2285     212 |     q = brx * brx + dry * dry
2286     213 |     r = arx * brx + cry * dry
2287     214 |     # Eigenvalues (>= 0)
2288     215 |     trace = p + q
2289     216 |     diff = p - q
2290     217 |     disc = math.hypot(diff, 2 * r) # sqrt(diff^2 + (2r)^2)
2291     218 |     lam1 = 0.5 * (trace + disc) # larger eigenvalue
2292     219 |     lam2 = 0.5 * (trace - disc) # smaller eigenvalue
2293     220 |     # Radii (>= 0)
2294     221 |     R1 = math.sqrt(max(lam1, 0.0))
2295     222 |     R2 = math.sqrt(max(lam2, 0.0))
2296     223 |     # Angle (orientation of major axis): 0.5 * atan2(2r, p - q)
2297     224 |     angle = 0.5 * math.atan2(2 * r, diff) if (abs(r) + abs(diff)) > 0 else 0.0
2298     225 |     return R1, R2, math.degrees(angle)
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999

```

Figure 26: SVG parsing code (4/5).

```

2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340      266      M = parse_transform(elem.get('transform', ''))
2341      267
2342      268      # Transformed center
2343      269      cx_t, cy_t = apply_affine(M, cx, cy)
2344      270
2345      271      # Linear part (2x2)
2346      272      Mlin = [[M[0][0], M[0][1]], [M[1][0], M[1][1]]]
2347      273
2348      274      # Decompose radii and angle
2349      275      if rx == 0.0 and ry == 0.0:
2350      276          R1 = R2 = 0.0
2351      277          angle_deg = 0.0
2352      278      else:
2353      279          R1, R2, angle_deg = decompose_axes(Mlin, rx, ry)
2354      280
2355      281          ellipses.append((cx_t, cy_t), R1, R2, angle_deg))
2356      282
2357      283
2358  def parse_svg_file(path, filter_class=None):
2359      284
2360      285      root = ET.parse(path).getroot()
2361      286      segs = parse_segments(root, filter_class)
2362      287      circs = parse_circles(root, filter_class)
2363      288      ells = parse_ellipses(root, filter_class)
2364      289
2365      290      return segs, circs, ells
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375

```

Figure 27: SVG parsing code (5/5).

2376 F.1.2 EVALUATION CODE FOR PATTERN 1
2377

2378 We now present the evaluation code for Pattern 1, where the correct objects are not uniquely de-
2379 termined. This code checks whether the SVG produced by the LLM correctly reproduces the
2380 ground-truth primitives (straight segments, circles, and ellipses), as shown in Figures 28, 29, and
2381 30. The core of this evaluation is implemented by the three functions `match_segments()`,
2382 `match_circles()`, and `match_ellipses()`.

2383 **Overall Evaluation Procedure.** Given two SVG files,

2384 • `ground_truth`: the reference SVG, and
2385 • `llm_output`: the SVG generated by the model,

2388 the script first calls `parse_svg_file()` to extract:

2389 • line segments `gt_segs`, `out_segs`,
2390 • circles `gt_circs`, `out_circs`,
2392 • ellipses `gt_ells`, `out_ells`.

2394 The output segments are then augmented by `merge_collinear_segments()`, which attempts
2395 to merge nearly collinear segments that share an endpoint, so that broken polylines can still match a
2396 single ground-truth segment.

2397 The evaluation is decomposed into three independent stages:

2398 `stage1 = match_segments(gt_segs, out_segs),`
2399 `stage2 = match_circles(gt_circs, out_circs),`
2400 `stage3 = match_ellipses(gt_ells, out_ells).`

2402 If and only if all three stages succeed, the script prints "1"; otherwise it prints "0".

2403 **Step 1: Matching Line Segments.** The function `match_segments(gt_segs, out_segs, tol)` verifies that every ground-truth segment is represented in the LLM output within a geometric
2405 tolerance. Each segment is represented by its two endpoints:

$$2407 \quad \text{gt_segs} = \{(p_1, p_2)\}, \quad \text{out_segs} = \{(q_1, q_2)\},$$

2409 where $p_1, p_2, q_1, q_2 \in \mathbb{R}^2$ denote 2D coordinates.

2410 • For each ground-truth segment (p_1, p_2) , the function searches over all output segments
2411 (q_1, q_2) .
2412 • The helper `dist_point_to_segment(pt, a, b)` computes the Euclidean distance
2413 from a point pt to the finite segment \overline{ab} by projecting pt onto the segment and clamping
2414 the projection parameter to $[0, 1]$.
2415 • A ground-truth segment is considered *matched* if there exists an output segment such that
2416 both endpoints p_1 and p_2 are within distance tol of the segment $\overline{q_1q_2}$:

$$2418 \quad \text{dist}(p_1, \overline{q_1q_2}) \leq \text{tol}, \quad \text{dist}(p_2, \overline{q_1q_2}) \leq \text{tol}.$$

2420 If all ground-truth segments find such a matching segment in `out_segs`, the function returns `True`;
2421 otherwise it returns `False`. This ensures that every reference straight line is geometrically repro-
2422 duced in the LLM output.

2423 **Step 2: Matching Circles.** The function `match_circles(gt_circs, out_circs, tol_center, tol_r)` checks that all ground-truth circles are present in the output with simi-
2424 lar centers and radii. Each circle is represented as $((c_x, c_y), r)$, where $c = (c_x, c_y)$ is the center and
2425 r is the radius.

2426 • The outer loop iterates over ground-truth circles $(c_{\text{gt}}, r_{\text{gt}})$. A working copy of output circles
2427 is stored in `unmatched` so that once an output circle is assigned to a ground-truth circle,
2428 it is removed and cannot be reused.

2430
 2431 • For each ground-truth circle, the function selects the output circle with the maximum
 2432 intersection-over-union (IoU), computed by `circle_iou(c1, r1, c2, r2)`. The
 2433 IoU is defined between the two disks D_1, D_2 as:

2434
$$\text{IoU}(D_1, D_2) = \frac{\text{area}(D_1 \cap D_2)}{\text{area}(D_1 \cup D_2)}.$$

 2435

2436 This IoU is used only to choose the *best* candidate in `unmatched`.

2437
 2438 • After selecting the circle with maximum IoU, the function checks strict geometric toler-
 2439 ances on center and radius:

2440
$$\|c_{\text{gt}} - c_{\text{out}}\| \leq \text{tol_center}, \quad |r_{\text{gt}} - r_{\text{out}}| \leq \text{tol_r}.$$

 2441

2442 If both conditions are satisfied, the output circle is removed from `unmatched` and the
 2443 ground-truth circle is deemed matched. Otherwise, the entire matching fails and the func-
 2444 tion returns `False`.

2445 If all ground-truth circles are successfully matched in this way, `match_circles()` returns `True`.
 2446 Thus, every reference circle must appear in the LLM output with nearly the same center and radius.

2447 **Step 3: Matching Ellipses.** The function `match_ellipses(gt_ells, out_ells, iou_thresh)` evaluates whether each ground-truth ellipse has a corresponding ellipse in the out-
 2448 put that overlaps sufficiently in area. An ellipse is represented as:

2449
$$((c_x, c_y), R_x, R_y, \theta),$$

 2450

2451 where (c_x, c_y) is the center, R_x, R_y are the radii along the principal axes, and θ is the rotation angle.

2452
 2453 • As in circle matching, a list `unmatched` stores the remaining output ellipses that have not
 2454 yet been assigned.

2455
 2456 • For each ground-truth ellipse e_{gt} , the function searches for the output ellipse e_{out} that max-
 2457 imizes the IoU, computed by `ellipse_iou(e1, e2)`.

2458 **Approximate IoU for Rotated Ellipses.** The helper `ellipse_iou(e1, e2, samples)` esti-
 2459 mates IoU by Monte Carlo sampling:

2460 1. For each ellipse, an axis-aligned bounding box is computed that tightly encloses the rotated
 2461 ellipse.

2462 2. A joint bounding box that covers both ellipses is obtained by taking the min/max of the two
 2463 boxes.

2464 3. A number of random points (given by `samples`) are uniformly sampled in this joint
 2465 bounding box.

2466 4. For each point, the function tests membership in each ellipse by transforming the point into
 2467 the ellipse-aligned coordinate system and checking

2468
$$\frac{x^2}{R_x^2} + \frac{y^2}{R_y^2} \leq 1.$$

 2469

2470 5. From these samples, the areas of intersection and union are estimated, and the IoU is ap-
 2471 proximated as:

2472
$$\text{IoU}(e_1, e_2) \approx \frac{\text{area}(e_1 \cap e_2)}{\text{area}(e_1 \cup e_2)}.$$

 2473

2474 **IoU-based Acceptance Criterion.** Back in `match_ellipses()`, after computing IoU values
 2475 between a ground-truth ellipse and all remaining output ellipses:

2476
 2477 • The output ellipse with maximum IoU is selected as the candidate match.

2478
 2479 • If this maximum IoU is greater than or equal to the threshold `iou_thresh` (default 0.95),
 2480 the candidate is accepted and removed from `unmatched`.

2484 • Otherwise, the function returns `False`, indicating that no sufficiently overlapping ellipse
2485 was found.
2486

2487 If every ground-truth ellipse is matched with IoU at least `iou_thresh`, the function returns `True`.

2488 **Result.** Combining the three matching functions, the script provides a binary evaluation:

2489 • The LLM output is considered *correct* if and only if all ground-truth line segments, circles,
2490 and ellipses are geometrically reproduced within the specified tolerances and IoU thresh-
2491 olds.
2493 • In that case, the program prints "`1`"; otherwise it prints "`0`".
2494

2495 This evaluation criterion enforces a strict structural fidelity of the generated SVG against the refer-
2496 ence vector graphics.

2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537

```

2538
2539
2540
2541
2542     1 #!/usr/bin/env python3
2543     2 # -*- coding: utf-8 -*-
2544
2545     4 import argparse
2546     5 import xml.etree.ElementTree as ET
2547     6 import re
2548     7 import math
2549     8 import sys
2550     9 import random
2551    10 from parse_svg_file import parse_svg_file
2552
2553
2554    13 def dist_point_to_segment(pt, a, b):
2555        """Distance between point `pt` and segment `ab`"""
2556        x, y = pt; x1 = a; x2, y2 = b
2557        dx, dy = x2-x1, y2-y1
2558        if dx == 0 and dy == 0:
2559            return math.hypot(x-x1, y-y1)
2560        t = ((x-x1)*dx + (y-y1)*dy) / (dx*dx + dy*dy)
2561        t = max(0.0, min(1.0, t))
2562        proj = (x1 + t*dx, y1 + t*dy)
2563        return math.hypot(x-proj[0], y-proj[1])
2564
2565
2566    25 def match_segments(gt_segs, out_segs, tol=1e1):
2567        """Check whether each segment in ground truth matches any segment in output"""
2568        for p1, p2 in gt_segs:
2569            ok = False
2570            for q1, q2 in out_segs:
2571                if dist_point_to_segment(p1, q1, q2) <= tol and \
2572                    dist_point_to_segment(p2, q1, q2) <= tol:
2573                    ok = True
2574                    break
2575            if not ok:
2576                return False
2577        return True
2578
2579
2580    39 def circle_iou(c1, r1, c2, r2):
2581        """Calculate IoU (Intersection over Union) of two circles"""
2582        d = math.hypot(c1[0]-c2[0], c1[1]-c2[1])
2583        if d >= r1 + r2:
2584            return 0.0
2585        if d <= abs(r1 - r2):
2586            return (min(r1, r2)**2) / (max(r1, r2)**2)
2587        r1_sq, r2_sq = r1**2, r2**2
2588        alpha = math.acos((d*d + r1_sq - r2_sq) / (2*d*r1))
2589        beta = math.acos((d*d + r2_sq - r1_sq) / (2*d*r2))
2590        inter = (r1_sq*alpha + r2_sq*beta -
2591            0.5*math.sqrt((-d+r1+r2)*(d+r1-r2)*(d-r1+r2)*(d+r1+r2)))
2592        union = math.pi*(r1_sq + r2_sq) - inter
2593        return inter / union
2594
2595
2596    55 def match_circles(gt_circs, out_circs, tol_center=1e1, tol_r=1e1):
2597        """
2598        Match each ground-truth circle by maximizing IoU and
2599        filter by center distance and radius difference
2600        """
2601        unmatched = list(out_circs)
2602        for c_gt, r_gt in gt_circs:
2603            if not unmatched:
2604                return False
2605            # find best IoU
2606            best_i, best_iou = 0, circle_iou(c_gt, r_gt, unmatched[0][0], unmatched[0][1])
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691

```

Figure 28: Evaluation code for Pattern 1 (1/3).

```

2592
2593
2594
2595
2596     66     for i, (c_out, r_out) in enumerate(unmatched[1:], start=1):
2597         67         iou = circle_iou(c_gt, r_gt, c_out, r_out)
2598         68         if iou > best_iou:
2599             69             best_iou, best_i = iou, i
2600             70             c_out, r_out = unmatched[best_i]
2601             71             dc = math.hypot(c_gt[0]-c_out[0], c_gt[1]-c_out[1])
2602             72             if dc <= tol_center and abs(r_gt-r_out) <= tol_r:
2603                 73                 unmatched.pop(best_i)
2604             74             else:
2605                 75                 return False
2606             76             return True
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
66     for i, (c_out, r_out) in enumerate(unmatched[1:], start=1):
67         iou = circle_iou(c_gt, r_gt, c_out, r_out)
68         if iou > best_iou:
69             best_iou, best_i = iou, i
70             c_out, r_out = unmatched[best_i]
71             dc = math.hypot(c_gt[0]-c_out[0], c_gt[1]-c_out[1])
72             if dc <= tol_center and abs(r_gt-r_out) <= tol_r:
73                 unmatched.pop(best_i)
74             else:
75                 return False
76             return True
77
78     def ellipse_iou(e1, e2, samples=2000):
79         """Compute approximate IoU for two rotated ellipses by Monte Carlo sampling"""
80         def inside(pt, ell):
81             (cx, cy), rx, ry, angle = ell
82             rad = math.radians(-angle)
83             cosA = math.cos(rad); sinA = math.sin(rad)
84             dx, dy = pt[0]-cx, pt[1]-cy
85             x = dx*cosA - dy*sinA
86             y = dx*sinA + dy*cosA
87             return x*x/(rx*rx) + y*y/(ry*ry) <= 1
88
89         def bbox(ell):
90             (cx, cy), rx, ry, angle = ell
91             rad = math.radians(angle)
92             cosA = math.cos(rad); sinA = math.sin(rad)
93             w = abs(rx*cosA) + abs(ry*sinA)
94             h = abs(rx*sinA) + abs(ry*cosA)
95             return (cx-w, cy-h, cx+w, cy+h)
96
97         b1 = bbox(e1); b2 = bbox(e2)
98         minx = min(b1[0], b2[0]); miny = min(b1[1], b2[1])
99         maxx = max(b1[2], b2[2]); maxy = max(b1[3], b2[3])
100        area_box = (maxx-minx)*(maxy-miny)
101        cnt1 = cnt2 = cnt_both = 0
102        for _ in range(samples):
103            x = random.uniform(minx, maxx)
104            y = random.uniform(miny, maxy)
105            in1 = inside((x, y), e1)
106            in2 = inside((x, y), e2)
107            if in1: cnt1 += 1
108            if in2: cnt2 += 1
109            if in1 and in2: cnt_both += 1
110        if cnt1+cnt2-cnt_both == 0:
111            return 0.0
112        area_inter = cnt_both/samples * area_box
113        area_union = (cnt1+cnt2-cnt_both)/samples * area_box
114        return area_inter / area_union
115
116
117
118     def match_ellipses(gt_ells, out_ells, iou_thresh=0.95):
119         """Match each ground-truth ellipse by maximizing IoU; pass if above threshold"""
120         unmatched = list(out_ells)
121         for gt in gt_ells:
122             if not unmatched:
123                 return False
124             best_i, best_iou = 0, ellipse_iou(gt, unmatched[0])
125             for i, out in enumerate(unmatched[1:], start=1):
126                 iou = ellipse_iou(gt, out)
127                 if iou > best_iou:
128                     best_iou, best_i = iou, i
129             if best_iou >= iou_thresh:
130                 unmatched.pop(best_i)
131             else:
132                 return False

```

Figure 29: Evaluation code for Pattern 1 (2/3).

```

2646
2647
2648
2649
2650
2651
2652
2653     133 |     return True
2654     134 |
2655     135 |
2656     136 | def merge_collinear_segments(segments, angle_tol=math.radians(5)):
2657     137 |     """Merge segments that share an endpoint and are almost collinear"""
2658     138 |     merged = []
2659     139 |     thresh = -math.cos(angle_tol)
2660     140 |     n = len(segments)
2661     141 |     for i in range(n):
2662     142 |         u, v = segments[i]
2663     143 |         for j in range(i+1, n):
2664     144 |             w, x = segments[j]
2665     145 |             shared = None
2666     146 |             if u == w:
2667     147 |                 shared, p1, p2 = u, v, x
2668     148 |             elif u == x:
2669     149 |                 shared, p1, p2 = u, v, w
2670     150 |             elif v == w:
2671     151 |                 shared, p1, p2 = v, u, x
2672     152 |             elif v == x:
2673     153 |                 shared, p1, p2 = v, u, w
2674     154 |             else:
2675     155 |                 continue
2676     156 |             vec1 = (p1[0]-shared[0], p1[1]-shared[1])
2677     157 |             vec2 = (p2[0]-shared[0], p2[1]-shared[1])
2678     158 |             n1 = math.hypot(*vec1); n2 = math.hypot(*vec2)
2679     159 |             if n1 == 0 or n2 == 0:
2680     160 |                 continue
2681     161 |             cos_ang = (vec1[0]*vec2[0] + vec1[1]*vec2[1]) / (n1*n2)
2682     162 |             if cos_ang <= thresh:
2683     163 |                 merged.append((p1, p2))
2684     164 |
2685     165 |
2686     166 |
2687     167 | def main():
2688     168 |     p = argparse.ArgumentParser(description="SVG Output Evaluation Script")
2689     169 |     p.add_argument('ground_truth', help="File path of ground-truth SVG")
2690     170 |     p.add_argument('llm_output', help="File path of LLM-generated SVG")
2691     171 |     args = p.parse_args()
2692     172 |
2693     173 |     try:
2694     174 |         gt_segs, gt_circs, gt_ells = parse_svg_file(args.ground_truth, filter_class='output_object')
2695     175 |         out_segs, out_circs, out_ells = parse_svg_file(args.llm_output)
2696     176 |     except Exception:
2697     177 |         print("0")
2698     178 |         sys.exit(0)
2699     179 |     out_segs.extend(merge_collinear_segments(out_segs))
2700     180 |
2701     181 |     # Matching
2702     182 |     stage1 = match_segments(gt_segs, out_segs)
2703     183 |     stage2 = match_circles(gt_circs, out_circs)
2704     184 |     stage3 = match_ellipses(gt_ells, out_ells)
2705     185 |
2706     186 |     print("1" if (stage1 and stage2 and stage3) else "0")
2707     187 |
2708     188 |     if __name__ == '__main__':
2709     189 |         main()

```

Figure 30: Evaluation code for Pattern 1 (3/3).

```

2693
2694
2695
2696
2697
2698
2699

```

2700 F.1.3 EVALUATION CODE FOR PATTERN 2
2701

2702 In Pattern 2, it is not possible to uniquely determine the correct objects. To address this, we im-
2703 plemented case-specific Python logic that evaluates correctness based on textual input, allowing for
2704 variations in valid outputs. As examples, we present the evaluation code for the case in Figure 3 and
2705 for the bottom case in Figure 5.

2706 The evaluation code corresponding to Figure 3 is shown in Figures 31 and 32.

2707 **Overall Evaluation Procedure.** The script compares two SVG files:

2709 • the ground-truth construction (annotated with the class `input_object`), and
2710 • the LLM-generated output (annotated with the class `output_object`).

2712 Both are parsed by `parse_svg_file()`, which returns:

2713 $(gt_segs, gt_circs, gt_ells)$, $(out_segs, out_circs, out_ells)$

2715 for the input and output respectively. If parsing fails at any point, the script prints 0 and terminates,
2716 indicating an incorrect solution.

2717 **Geometric Helper Functions.** Two helper routines implement basic Euclidean geometry needed
2718 for the evaluation:

2720 • `dist_point_to_segment(pt, a, b)`: given a point pt and a segment with end-
2721 points a and b , this function computes the shortest distance from pt to the segment. It
2722 projects pt onto the supporting line of ab , clamps the projection parameter to the interval
2723 $[0, 1]$, and returns the Euclidean distance from pt to the resulting closest point.

2724 • `circle_intersections(c1, r1, c2, r2)`: given two circles with centers c_1, c_2
2725 and radii r_1, r_2 , this function returns their intersection points. It first computes the center
2726 distance d ; if the circles are too far apart, nested, or numerically degenerate, it returns an
2727 empty list. Otherwise, it computes the base point on the line between the centers and the
2728 perpendicular offset, yielding either one point (tangency) or two intersection points.

2729 **Tolerance Parameters.** Because the LLM-generated SVG may differ slightly due to numerical
2730 precision or stylistic variation, the script uses small tolerances:

2732 • `tol_center`: allowed deviation when matching circle centers to line endpoints.
2733 • `tol_r`: allowed difference between radii.
2734 • `tol_line`: allowed distance when checking whether intersection points lie on a candidate
2735 line segment.

2737 These tolerances make the evaluation robust to minor floating-point noise.

2738 **Step 1: Extracting the Reference Segment.** From the ground-truth data, the script takes the first
2739 input segment:

$$(p_1, p_2) = gt_segs[0].$$

2741 This segment represents the original line in Figure 3 on which the construction is based. If no such
2742 segment exists, the evaluation immediately fails.

2744 **Step 2: Matching Circles at the Endpoints.** The first condition requires that the output contain
2745 two circles whose centers coincide with the endpoints p_1 and p_2 of the input segment:

2746 1. The script scans `out_circs` to find a circle with center within `tol_center` of p_1 . This
2747 circle is stored as (c_1, r_1) .
2748 2. It then scans again to find a *different* circle with center within `tol_center` of p_2 , stored
2749 as (c_2, r_2) .

2751 If either endpoint does not have a corresponding circle center, the script prints 0 (incorrect).

2752 **Step 3: Checking the Radii.** The second condition enforces both equality and sufficient size of the
2753 radii:

2754 • The radii must be equal up to tolerance: $|r_1 - r_2| \leq \text{tol_r}$.
 2755 • Each radius must be strictly larger than half the length of the original segment:
 2756

$$r_1 > \frac{\|p_2 - p_1\|}{2} - \text{tol_r}.$$

2759 Intuitively, this ensures that the two circles intersect in two distinct points above and below the
 2760 segment, as in the geometric construction. If either inequality is violated, the output is rejected.
 2761

2762 **Step 4: Computing Circle Intersections.** The third condition uses the intersection points of the
 2763 two circles:

2764 1. The script calls `circle_intersections(c1, r1, c2, r2)` to compute the inter-
 2765 section points.
 2766 2. If fewer than two intersection points are found (no intersection or tangency), the configu-
 2767 ration cannot reproduce the intended construction, and the script outputs 0.

2769 When two intersections exist, denote them by I_1 and I_2 .
 2770

2771 **Step 5: Verifying a Line Through Both Intersections.** Finally, the script checks whether the LLM
 2772 has drawn a line that passes through both intersection points:

2773 • It iterates over each output segment $(q_1, q_2) \in \text{out_segs}$.
 2774 • For each segment, it computes the distance from I_1 and I_2 to the segment using
 2775 `dist_point_to_segment`.
 2776 • If both distances are within `tol_line`, then (q_1, q_2) is considered to pass through both
 2777 circle intersections.

2779 If such a segment is found, the configuration is deemed correct and the script prints 1. Otherwise, it
 2780 prints 0.
 2781

2782
 2783
 2784
 2785
 2786
 2787
 2788
 2789
 2790
 2791
 2792
 2793
 2794
 2795
 2796
 2797
 2798
 2799
 2800
 2801
 2802
 2803
 2804
 2805
 2806
 2807

```

2808
2809
2810
2811
2812
2813     1 #!/usr/bin/env python3
2814     2 # -*- coding: utf-8 -*-
2815     3
2816     4 import argparse
2817     5 import xml.etree.ElementTree as ET
2818     6 import re
2819     7 import math
2820     8 import sys
2821     9 from parse_svg_file import parse_svg_file
2822    10
2823    11 # ----- Tolerances (adjust if necessary) -----
2824    12 tol_center = 1e-1 # Tolerance for matching a circle center with a line endpoint
2825    13 tol_r = 1e-1 # Tolerance for comparing radii
2826    14 tol_line = 1e-1 # Tolerance for checking if intersection points lie on a line segment
2827    15
2828    16 def dist_point_to_segment(pt, a, b):
2829    17     """
2830    18     Return the distance between a point pt and a line segment ab.
2831    19     """
2832    20     x,y = pt
2833    21     x1,y1 = a
2834    22     x2,y2 = b
2835    23     dx = x2 - x1
2836    24     dy = y2 - y1
2837    25     if dx==0 and dy==0:
2838    26         return math.hypot(x-x1, y-y1)
2839    27     t = ((x-x1)*dx + (y-y1)*dy) / (dx*dx + dy*dy)
2840    28     t = max(0.0, min(1.0, t))
2841    29     proj_x = x1 + t*dx
2842    30     proj_y = y1 + t*dy
2843    31     return math.hypot(x - proj_x, y - proj_y)
2844    32
2845    33 def circle_intersections(c1, r1, c2, r2, tol=1e-6):
2846    34     """
2847    35     Return the intersection points of two circles.
2848    36     If the circles intersect at two points, return a list of length 2.
2849    37     If they are tangent, return a list of length 1.
2850    38     If they do not intersect, return an empty list.
2851    39     """
2852    40     x0,y0 = c1
2853    41     x1,y1 = c2
2854    42     dx = x1 - x0
2855    43     dy = y1 - y0
2856    44     d = math.hypot(dx,dy)
2857    45     # Same center or too far apart
2858    46     if d < tol or d > r1 + r2 + tol or d < abs(r1 - r2) - tol:
2859    47         return []
2860    48     # Distance a and height h of intersection
2861    49     a = (r1*r1 - r2*r2 + d*d) / (2*d)
2862    50     h2 = r1*r1 - a*a
2863    51     if h2 < 0 and abs(h2) <= tol:
2864    52         h = 0.0
2865    53     elif h2 < 0:
2866    54         return []
2867    55     else:
2868    56         h = math.sqrt(h2)
2869    57         xm = x0 + a * dx / d
2870    58         ym = y0 + a * dy / d
2871    59         rx = -dy * (h / d)
2872    60         ry = dx * (h / d)
2873    61         p1 = (xm + rx, ym + ry)
2874    62         p2 = (xm - rx, ym - ry)
2875    63         return [p1] if h == 0 else [p1, p2]

```

Figure 31: Evaluation code for the case in Figure 3 (1/2).

2858
2859
2860
2861

```

2862
2863
2864
2865     66 def main():
2866     67     p = argparse.ArgumentParser(description="SVG Output Evaluation Script")
2867     68     p.add_argument('ground_truth', help="File path of correct SVG")
2868     69     p.add_argument('llm_output', help="File path of SVG output from LLM")
2869     70     args = p.parse_args()
2870
2871     71     try:
2872         72         gt_segs, gt_circles, gt_ells = parse_svg_file(args.ground_truth, filter_class='input_object')
2873         73         out_segs, out_circles, out_ells = parse_svg_file(args.llm_output, filter_class='output_object')
2874     74     except Exception:
2875         75         print("0")
2876         76         sys.exit(0)
2877
2878         77     input_line = gt_segs[0]
2879         78     if input_line is None:
2880             79         print("0"); sys.exit(0)
2881
2882         80     p1, p2 = input_line
2883
2884         81     # Find circles that match p1 and p2
2885         82     c1 = r1 = c2 = r2 = None
2886
2887         83     # ----- Condition 1: The output must contain exactly two circles whose centers match the endpoints
2888         84     # of the input segment -----
2889         85     # Search for the circle centered at p1
2890         86     for center, r in out_circles:
2891             87         if math.hypot(center[0]-p1[0], center[1]-p1[1]) <= tol_center:
2892             88             c1, r1 = center, r
2893             89             break
2894         90     if c1 is None:
2895             91         print("0"); sys.exit(0)
2896
2897         92     # Search for the circle centered at p2 (make sure it's not the same one as c1)
2898         93     for center, r in out_circles:
2899             94         # Avoid matching the same circle by excluding identical centers
2900             95         if (abs(center[0]-c1[0]) > tol_center or abs(center[1]-c1[1]) > tol_center) \
2901             96             and math.hypot(center[0]-p2[0], center[1]-p2[1]) <= tol_center:
2902             97             c2, r2 = center, r
2903             98             break
2904         99     if c2 is None:
2905             100         print("0"); sys.exit(0)
2906
2907         101     # --- Condition 2: Radii must match and must be greater than half the segment length ---
2908         102     line_len = math.hypot(p2[0]-p1[0], p2[1]-p1[1])
2909         103     if abs(r1 - r2) > tol_r or r1 <= line_len/2 - tol_r:
2910             104         print("0"); sys.exit(0)
2911
2912         105     # --- Condition 3: There must exist a line segment passing through the intersection points of the
2913         106     # circles ---
2914         107     inters = circle_intersections(c1, r1, c2, r2, tol=tol_r)
2915         108     if len(inters) < 2:
2916         109         print("0"); sys.exit(0)
2917
2918         110     found_line = False
2919     111     for q1, q2 in out_segs:
2920         112         if dist_point_to_segment(inters[0], q1, q2) <= tol_line \
2921         113             and dist_point_to_segment(inters[1], q1, q2) <= tol_line:
2922             114                 found_line = True
2923                 break
2924
2925         115     if not found_line:
2926             116         print("0"); sys.exit(0)
2927
2928         117     # All conditions satisfied
2929         118     print("1")
2930
2931     119     if __name__ == '__main__':
2932         120         main()

```

Figure 32: Evaluation code for the case in Figure 3 (2/2).

2916 The evaluation code corresponding to the bottom case in Figure 5 is shown in Figures 33 and 34.
 2917
 2918 **Overall Evaluation Procedure.** The script evaluates whether an LLM-generated SVG correctly
 2919 draws the external tangents between four small circles found in a ground-truth SVG. It parses both
 2920 SVG files using `parse_svg_file()`:

2921 `(gt_segs, gt_circs, gt_ells), (out_segs, out_circs, out_ells)`

2922 Only graphical items marked with the class names `input_object` (ground truth) and
 2923 `output_object` (LLM output) are retrieved. If parsing fails, the script immediately prints 0.
 2924

2925 **Geometric Helper Functions.** Two main routines support the evaluation:

2926

- 2927 • `dist_point_to_segment(pt, a, b)` computes the Euclidean distance from a point
 2928 `pt` to a segment with endpoints `a, b`. Using a projection onto line `ab`, the value is clamped
 2929 to the segment and the closest-point distance is returned.
- 2930 • `external_tangents(c1, r1, c2, r2)` returns pairs of tangent contact points be-
 2931 tween two circles with centers `c1, c2` and radii `r1, r2`. If the distance between the centers is
 2932 too small (one circle inside another or touching internally), no external tangents exist and
 2933 an empty list is returned.

2934 **Tolerance Handling.** A fixed distance tolerance

2935 $\text{tol} = 10$

2936 is used when determining whether a drawn segment approximates a target tangent. Small variations
 2937 due to SVG scaling or floating precision are therefore accepted.
 2938

2939 **Reference Circles in the Ground Truth.** The script extracts all circles from the ground truth
 2940 drawing and sorts them by increasing radius. Only

2941 $\text{the four circles with the smallest radii}$

2942 are used as the construction targets. If fewer than five ground-truth circles exist, the problem is
 2943 considered invalid and the output score is 0.
 2944

2945 **Expected Geometry: External Tangents.** For every pair among the four selected circles (six pairs
 2946 in total), the script computes the two possible external tangent segments:

2947 $((p_{11}, p_{12}), (p_{21}, p_{22}))$

2948 Each tangent is a pair of points, one on each circle. At least one tangent for each circle pair must
 2949 appear in the LLM’s output as a drawn segment.
 2950

2951 **Matching Tangents in the LLM Output.** For each correct tangent candidate (A, B) , the script
 2952 checks whether any output segment (Q_1, Q_2) matches it using:

2953 $\text{match_segments}() \iff (\text{dist}(A, Q_1 Q_2) \leq \text{tol} \wedge \text{dist}(B, Q_1 Q_2) \leq \text{tol}).$

2954 If no output segment matches at least one tangent for a given circle pair, evaluation halts and returns
 2955 0.
 2956

2957 **Final Decision.** If all six circle pairs have at least one tangent segment successfully approximated
 2958 by the LLM-generated SVG, the script prints: 1 otherwise it prints: 0.
 2959

2960

2961

2962

2963

2964

2965

2966

2967

2968

2969

```

2970
2971
2972
2973
2974     1 #!/usr/bin/env python3
2975     2 # -*- coding: utf-8 -*-
2976
2977     4 import argparse
2978     5 import xml.etree.ElementTree as ET
2979     6 import re
2980     7 import math
2981     8 import sys
2982     9 import itertools
2983    10 from parse_svg_file import parse_svg_file
2984
2985
2986    13 def dist_point_to_segment(pt, a, b):
2987        14     x, y = pt; x1, y1 = a; x2, y2 = b
2988        15     dx = x2 - x1; dy = y2 - y1
2989        16     if dx==0 and dy==0:
2990        17         return math.hypot(x1, y1)
2991        18     t = ((x-x1)*dx + (y-y1)*dy) / (dx*dx + dy*dy)
2992        19     t = max(0.0, min(1.0, t))
2993        20     proj_x = x1 + t*dx; proj_y = y1 + t*dy
2994        21     return math.hypot(x - proj_x, y - proj_y)
2995
2996
2997    24 def match_segments(gt_segs, out_segs, tol=1e-1):
2998        25         """Check whether each segment in gt is contained in any segment in out"""
2999        26         for (p1,p2) in gt_segs:
3000        27             ok = False
3001        28             for (q1,q2) in out_segs:
3002        29                 if dist_point_to_segment(p1, q1, q2) <= tol and dist_point_to_segment(p2, q1, q2) <= tol:
3003        30                     ok = True; break
3004        31             if not ok:
3005        32                 return False
3006        33         return True
3007
3008
3009    36 def external_tangents(c1, r1, c2, r2):
3010        37         """Return pairs of tangent points for the external common tangents of two circles"""
3011        38         (x1,y1), (x2,y2) = c1, c2
3012        39         dx = x2 - x1; dy = y2 - y1
3013        40         d2 = dx*dx + dy*dy
3014        41         if d2 <= (r1 - r2)**2:
3015        42             return []
3016        43         d = math.sqrt(d2)
3017        44         a = (r1 - r2) / d
3018        45         a = max(-1.0, min(1.0, a))
3019        46         b = math.sqrt(max(0.0, 1 - a*a))
3020        47         ux = dx / d; uy = dy / d
3021        48         vx = -dy / d; vy = dx / d
3022        49         n1 = (a*ux + b*vx, a*uy + b*vy)
3023        50         n2 = (a*ux - b*vx, a*uy - b*vy)
3024        51         p11 = (x1 + r1 * n1[0], y1 + r1 * n1[1])
3025        52         p12 = (x2 + r2 * n1[0], y2 + r2 * n1[1])
3026        53         p21 = (x1 + r1 * n2[0], y1 + r1 * n2[1])
3027        54         p22 = (x2 + r2 * n2[0], y2 + r2 * n2[1])
3028        55         return [(p11, p12), (p21, p22)]
3029
3030
3031    58 def main():
3032        59         p = argparse.ArgumentParser(description="SVG output evaluation script")
3033        60         p.add_argument('ground_truth', help="Path to the ground truth SVG file")
3034        61         p.add_argument('llm_output', help="Path to the LLM output SVG file")
3035        62         args = p.parse_args()
3036
3037
3038        64     try:
3039        65         gt_segs, gt_circs, gt_ells = parse_svg_file(args.ground_truth, filter_class='input_object')

```

Figure 33: Evaluation code for the bottom case in Figure 5 (1/2).

```

3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043     66     out_segs, out_circs, out_ells = parse_svg_file(args.llm_output, filter_class='output_object')
3044  except Exception:
3045     67     print("0")
3046     68     sys.exit(0)
3047
3048     69
3049
3050     70
3051     71     # Get circles from the ground truth and select the top 4 with the smallest radius
3052     72     if len(gt_circs) < 5:
3053     73         print("0"); sys.exit(0)
3054     74     gt_circs_sorted = sorted(gt_circs, key=lambda x: x[1])[:4]
3055
3056     75     tol = 1e1
3057     76     ok_all = True
3058     77     for (c1, r1), (c2, r2) in itertools.combinations(gt_circs_sorted, 2):
3059     78         tangents = external_tangents(c1, r1, c2, r2)
3060
3061     79         if not tangents:
3062             80             ok_all = False; break
3063         # It is OK if at least one of the two tangent pairs exists in the output
3064         81             if not any(match_segments([seg], out_segs, tol) for seg in tangents):
3065             82                 ok_all = False; break
3066
3067         83             print("1" if ok_all else "0")
3068
3069     84     if __name__ == '__main__':
3070         85         main()
3071
3072
3073
3074
3075
3076
3077

```

Figure 34: Evaluation code for the bottom case in Figure 5 (2/2).

3078
3079

F.2 MOLECULAR STRUCTURE

3080

F.2.1 PARSING CODE

3081
3082
3083

We present the parsing code for converting vector data into graphs representing molecular structures as shown in Figure 35.

3084
3085
3086
3087
3088
3089
3090

The `parse_svg_file` function reads an SVG file and constructs a network graph using `networkx`. Nodes are extracted from `<circle>` elements, where each circle's coordinates and fill color are stored as node attributes. Line endpoints (`<line>` elements) are matched with the nearest circle positions to determine which nodes are connected. When both endpoints correspond to valid circles, an edge is created between the associated nodes. The function therefore produces an undirected graph whose topology reproduces the molecular connectivity encoded in the SVG drawing.

3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import math
5  import xml.etree.ElementTree as ET
6  import networkx as nx
7
8
9  def parse_svg_file(path, tol=1e-1):
10     """
11         A function that builds a graph structure from an SVG file.
12         Nodes are extracted from <circle> elements, and edges are extracted by associating
13         endpoints of <line> elements with those nodes.
14     """
15     root = ET.parse(path).getroot()
16
17     G = nx.Graph()
18     # Specify the SVG namespace
19     ns = {'svg': 'http://www.w3.org/2000/svg'}
20
21     circles = []
22     # Retrieve circle elements and register each node
23     for i, circle in enumerate(root.findall("./svg:circle", ns)):
24         cx = float(circle.attrib['cx'])
25         cy = float(circle.attrib['cy'])
26         fill = circle.attrib.get('fill', '')
27         circles.append((cx, cy, fill, i))
28         G.add_node(i, fill=fill, pos=(cx, cy))
29
30     # Find the closest circle (node) to the given coordinates
31     def find_circle(x, y):
32         for cx, cy, fill, node_id in circles:
33             if math.hypot(cx - x, cy - y) < tol:
34                 return node_id
35         return None
36
37     # Add edges based on line elements
38     for line in root.findall("./svg:line", ns):
39         x1 = float(line.attrib['x1'])
40         y1 = float(line.attrib['y1'])
41         x2 = float(line.attrib['x2'])
42         y2 = float(line.attrib['y2'])
43         n1 = find_circle(x1, y1)
44         n2 = find_circle(x2, y2)
45         # Add edge only if both endpoints correspond to circles (nodes)
46         if n1 is not None and n2 is not None:
47             G.add_edge(n1, n2)
48
49     return G

```

Figure 35: SVG parsing code.

3132
3133

F.2.2 EVALUATION CODE

3134
3135
3136

The script in Figure 36 evaluates whether two SVG files represent the same molecular graph. It takes two SVG file paths as input: a ground-truth structure and an SVG produced by a language model. Both files are parsed into graphs using the previously defined `parse_svg_file` function.

3137
3138
3139
3140
3141

After parsing, the script compares the two graphs using `networkx.is_isomorphic`, where nodes are matched based on their `fill` attribute, ensuring that atoms of the same type correspond between graphs. If the two SVG-derived graphs are isomorphic under this constraint, the script outputs 1; otherwise, it outputs 0. This enables automated validation of molecular structure predictions generated from vector graphics.

3142

```

3143 1 #!/usr/bin/env python3
3144 2 # -*- coding: utf-8 -*-
3145 3
3146 4 import argparse
3147 5 import xml.etree.ElementTree as ET
3148 6 from parse_svg_file import parse_svg_file
3149 7 import networkx as nx
3150 8
3151 9
3152 10 def main():
3153 11     p = argparse.ArgumentParser(description="SVG output evaluation script")
3154 12     p.add_argument('ground_truth', help="File path of the ground truth SVG")
3155 13     p.add_argument('llm_output', help="File path of the LLM-generated SVG")
3156 14     args = p.parse_args()
3157 15
3158 16     gt_G = parse_svg_file(args.ground_truth)
3159 17     out_G = parse_svg_file(args.llm_output)
3160 18
3161 19     node_match = lambda n1, n2: n1['fill'] == n2['fill']
3162 20
3163 21     # Graph isomorphism check
3164 22     iso = nx.is_isomorphic(gt_G, out_G, node_match=node_match)
3165 23
3166 24     print("1" if iso else "0")
3167 25
3168 26 if __name__ == '__main__':
3169 27     main()

```

Figure 36: Evaluation code.

3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185

3186 G VALIDATION OF HUMAN-SYSTEM AGREEMENT

3188 We verify the validity of our automated evaluation code by assessing how closely human evaluations
 3189 align with the code’s judgments. Two human evaluators judge whether LLM outputs are correct or
 3190 incorrect. One is a master’s student and the other is an undergraduate student. They both major in
 3191 engineering. They were not involved in our research and had no prior knowledge of the project. For
 3192 each description, we present a ground-truth vector graphic along with an LLM-generated output,
 3193 and ask them to evaluate its correctness. For the plane geometry task, we use all 110 instances.
 3194 For the molecular structure task, highly complex structures can increase the likelihood of errors by
 3195 human evaluators; therefore, we use a relatively simple set of 50 instances. We randomly select LLM
 3196 outputs so that the data labeled as correct and incorrect by our code are evenly balanced. Therefore,
 3197 the instances we used consist of 50% judged correct by our code and 50% judged incorrect. We
 3198 include all output formats: TikZ, SVG, and EPS.

3199 Table 15 shows the percentage of agreement and Cohen’s Kappa scores between the human evaluators
 3200 and our code. These results demonstrate very high agreement rates, indicating the reliability of
 3201 our evaluation approach.

3203 Table 15: The percentage of agreement and Cohen’s Kappa scores between the human evaluators
 3204 and our code.

3205 3206 3207 3208 3209 3210 3211 3212 3213 3214 3215 3216 3217 3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229 3230 3231 3232 3233 3234 3235 3236 3237 3238 3239	3203 Plane geometry			3203 Molecular structure		
	3203 TikZ	3203 SVG	3203 EPS	3203 TikZ	3203 SVG	3203 EPS
3203 Percentage of agreement	3203 97.3%	3203 95.5%	3203 95.5%	3203 98.0%	3203 98.0%	3203 98.0%
3203 Cohen’s Kappa	3203 0.946	3203 0.909	3203 0.909	3203 0.960	3203 0.960	3203 0.960
3203 3204 3205 3206 3207 3208 3209 3210 3211 3212 3213 3214	3203 Plane geometry			3203 Molecular structure		
	3203 TikZ	3203 SVG	3203 EPS	3203 TikZ	3203 SVG	3203 EPS
3203 Percentage of agreement	3203 96.4%	3203 99.1%	3203 95.5%	3203 98.0%	3203 100.0%	3203 96.0%
3203 Cohen’s Kappa	3203 0.927	3203 0.982	3203 0.909	3203 0.960	3203 1.000	3203 0.920