FOCUSTIKZ: QUALITY-AWARE FOCUSED TOKEN OPTIMIZATION FOR TIKZ CODE GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Generating TikZ code from textual descriptions requires models to produce executable code while preserving the intended semantics, spatial organization, and visual appearance of the resulting graphics. Current approaches primarily learn from paired descriptions and reference code, lacking explicit supervision of rendered diagram quality and assigning equal importance to all token positions during optimization. In this paper, we introduce **FocusTikZ**, a quality-aware approach for focused token optimization in TikZ code generation. FocusTikZ assesses generated candidates through compilation and code quality signals, together with three specialized assessment models focused on logic, layout, and style. We further propose Difference-and-Entropy Focused GRPO (DE-GRPO), which identifies informative update positions using reference differences and generation uncertainty. Across multiple benchmarks, FocusTikZ consistently improves both code-level performance and visual fidelity relative to general-purpose and task-specific baselines, with the same trend further supported by human evaluation.

1 INTRODUCTION

Transforming textual descriptions into structured diagrams, referred to as text-to-diagram, is an important yet challenging generation task. Unlike natural images, diagrams are structured visual artifacts composed of discrete elements, explicit relations, precise spatial arrangements, and consistent visual styles. A small local error, such as a misplaced node, a reversed arrow, or an inconsistent alignment pattern, may damage the semantics and readability of the whole diagram. Therefore, text-to-diagram generation is not only a visually grounded generation problem, but also a structure-sensitive reasoning and rendering problem. Accurate text-to-diagram generation holds great potential to improve productivity in domains such as education, scientific research, and industry (Cao et al., 2024; Cui et al., 2025; Wang et al., 2025a; Zou et al., 2024).

Recent progress in text-to-image generation has significantly advanced visual content synthesis (Saharia et al., 2022; Esser et al., 2024). However, these methods are primarily designed for natural images and often struggle to generate structured diagrams that require precise topology, explicit symbolic relations, and consistent spatial organization (Lin et al., 2024; Wei et al., 2025; Belouadi et al., 2025). Compared with natural-image generation, diagram generation demands stronger structural fidelity. For this reason, an appealing alternative is to formulate diagram generation as a text-to-code task, where the model outputs an executable representation rather than pixels (Allamanis et al., 2018; Kumar et al., 2023; Belouadi et al., 2024b).

In this work, we formulate the task as mapping textual descriptions to executable TikZ code. Representing structured diagrams in TikZ enables fine-grained control over graphical elements while supporting convenient modification and repeatable rendering. At the same time, it also makes the task challenging: the model must jointly capture the semantic content of the description, infer the logical structure of the diagram, determine an appropriate spatial layout, and produce compilable code with a consistent visual style. As shown in Figure 1, the goal is to produce a diagram that not only looks plausible, but also compiles successfully, preserves the intended structure, and faithfully reflects the input semantics.

Most text-to-TikZ methods are trained by imitating target codes paired with descriptions (Belouadi et al., 2024a; Wei et al., 2025). Although this paradigm provides a useful foundation for code generation, optimizing generation primarily against a reference program does not fully align the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

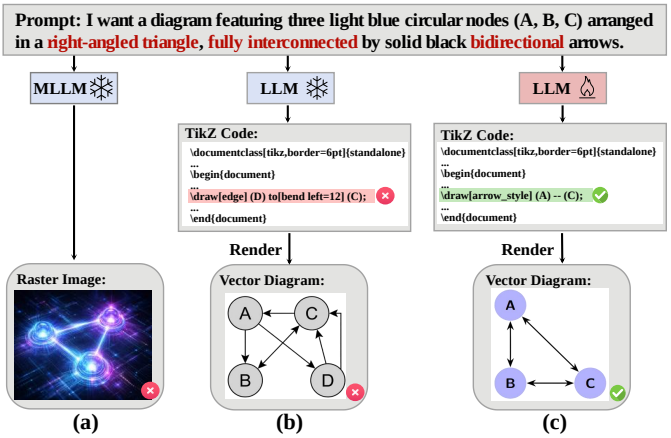


Figure 1: Illustration of the text-to-diagram task. Given a textual description, (a) multimodal large language models (MLLMs) often generate non-editable raster images; (b) general large language models (LLMs) may generate executable but incorrect TikZ code, leading to rendered diagrams with structural or semantic errors; and (c) our trained model can generate accurate vector diagrams.

actual objective of diagram synthesis. The same visual target may be realized by different command sequences, while even minor changes can substantially affect the resulting output. As a result, lexical similarity to the reference code does not necessarily indicate successful compilation or faithful visual realization. This limitation motivates the use of reinforcement learning, which allows the model to optimize generation quality beyond a single reference implementation.

Among reinforcement learning methods, Group Relative Policy Optimization (GRPO) (Shao et al., 2024) is particularly appealing because it improves generation quality through relative preference optimization without relying on a separately trained critic, making it effective and stable for large language model post-training. However, directly applying GRPO to text-to-TikZ remains suboptimal. Standard sequence-level optimization equally treats all tokens, without considering that some tokens are more important than others for the final diagram. Consequently, the optimization process may waste learning capacity on less important tokens, while failing to adequately correct the key decisions that keep relations, layouts, and visual consistency.

To address these limitations, we introduce **FocusTikZ**, a quality-aware framework that concentrates optimization on informative tokens during TikZ code generation. We construct the reward from complementary signals, including compilation success, token-level similarity, and three diagram-quality assessments targeting logic, layout, and style. To make optimization more effective for this task, we further propose Difference-and-Entropy Focused GRPO (DE-GRPO), which selectively updates only the key tokens that are different from the reference and with high uncertainty, instead of uniformly optimizing the entire sequence. In this way, our framework aligns policy learning with key tokens that substantially affect the rendered diagram.

We use SketikZ (Saito et al., 2025) as the main benchmark and test cross-dataset robustness across three external test sets, including DaTikZ-v1 (Belouadi et al., 2024a), DaTikZ-v4 (Greisinger & Eger, 2026), and DiagramGenBenchmark (Wei et al., 2025). Together, these datasets cover a broader range of diagram categories than SketikZ. The resulting comparisons consistently favor our approach in both code-level accuracy and rendered-diagram fidelity against general models and task-specific baselines, with the same trend also reflected in human judgments. Component-wise ablations further quantify the contribution of each reward term and validate the role of the focused RL optimization strategy.

2 Preliminaries

In the *reinforcement learning* (RL) paradigm for LLM alignment in text-to-code tasks, the generative model serves as a policy $\pi_\theta(c|x)$ that generates a code c from a natural language description x. A *reward function* $R(c)$ is designed to evaluate the quality of the generated code, providing a scalar

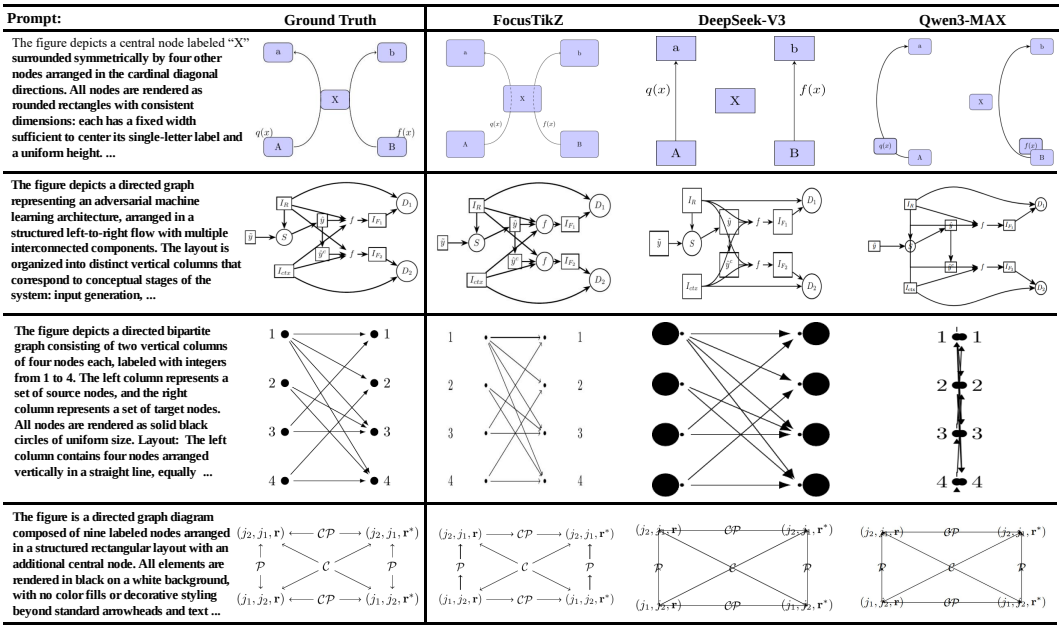


Figure 2: Examples of TikZ diagrams generated by our method and two strong baselines.

feedback signal to guide policy optimization. The core goal of RL in this setting is to update the model parameters θ to maximize the expected reward of the generated code. For text-to-code generation, the reward function should effectively reflect the quality of code.

GRPO (Shao et al., 2024) is a lightweight policy-only reinforcement learning algorithm. Specifically, for each input x, GRPO samples a group of candidate code sequences $\{c_i\}$ from the old policy π_{old}, and then optimizes the policy model π_θ by maximizing the following objective:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}\Big[\min\Big(r_i\hat{A}_i,\ \text{clip}(r_i, 1-\epsilon, 1+\epsilon)\hat{A}_i\Big) - \beta D_{\text{KL}}(\pi_\theta \parallel \pi_{\text{ref}})\Big],$$

where $r_i = \pi_\theta(c_i|x)/\pi_{\text{old}}(c_i|x)$ is the importance sampling ratio, ϵ is the clipping hyperparameter, β is the KL divergence penalty coefficient, and $\hat{A}_i$ is the relative advantage, obtained by normalizing $R(c_i)$ within the sampled group.

3 METHODOLOGY

3.1 OVERVIEW

Given a training dataset $\mathcal{D} = \{(x, c^{\text{ref}})\}$, we adopt the RL paradigm for training a language model as the generative model π_θ to generate TikZ codes c from an input description x. The model output, i.e., the code c, is then compiled to render a diagram. In this paper, we focus on the reward mechanism and training algorithm for π_θ. For the text-to-diagram task, we propose to reward the generative model not only based on the reference code but also according to the diagram quality. Furthermore, we propose a task-specific GRPO with selective token updates for training the generative model.

Figure 3 illustrates the training framework of the code generation model π_θ, which is first trained by the standard *supervised fine-tuning* (SFT) as the warm-up. Next, the model performs group sampling to generate a set of candidate codes that are then scored by our reward mechanism. Besides the token similarity and compilation success, we design three *assessment models* (AM) based on contrastive learning to evaluate the *logic*, *layout*, *style* capability of the generated code for compiling the diagram. After rewarding each candidate code, we propose a difference-and-entropy focused GRPO which selects a subset of key tokens for optimization.

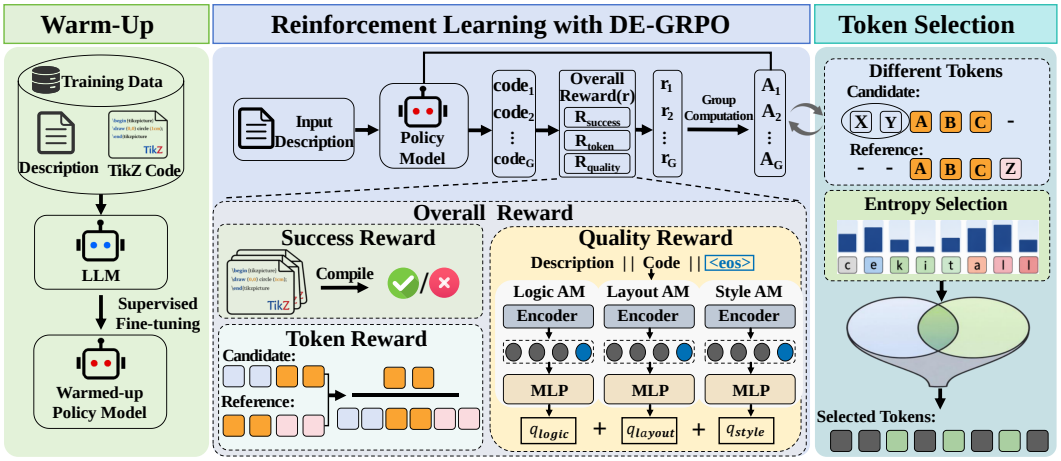


Figure 3: FocusTikZ first uses SFT warm-up to initialize the code generation model, then performs multi-reward reinforcement learning, and finally applies DE-GRPO to optimize the model with selective token updates.

3.2 Reward Design

The generated TikZ code c needs to be successfully compiled to render a diagram. So the reward function includes a *rendering success* reward R_{success}. It is also expected that the generated code follows its reference code, which motivates us to define a *token-level similarity* reward R_{token}. We note that the quality of the rendered diagram is the ultimate goal. Even if the generated code can be compiled and resembles the reference code, it may lead to some quality issues of the diagram, such as reversed arrows. To link diagram quality to generated code, we design three assessment models to compute a *quality* reward R_{quality}.

Our reward function consists of a rendering success gate and two quality-related rewards:

$$R_{\text{overall}} = R_{\text{success}} \cdot (R_{\text{token}} + R_{\text{quality}}) .$$

The rendering success reward R_{success} serves as a binary gate based on whether the generated code can be successfully compiled for diagram rendering. Specifically, $R_{\text{success}} = 0$ if the generated code c fails to compile, and $R_{\text{success}} = 1$ otherwise.

Although we expect a generated code c can follow its reference code c^{ref}, such an expectation may be neither practical nor necessary. It is not practical due to few training data and imperfect training method. We note that a single diagram can admit multiple valid TikZ implementations. So it is not necessary for code-level sequential resemblance. Instead, we propose token-level similarity for R_{token} as follows. For the trained model π_θ, we feed the c and c^{ref} respectively into its tokenizer to obtain the corresponding token set $\mathcal{S}$ and $\mathcal{S}^{\text{ref}}$. Then R_{token} is computed as the Jaccard similarity between the two sets

$$R_{\text{token}} = |\mathcal{S} \cap \mathcal{S}^{\text{ref}}| / |\mathcal{S} \cup \mathcal{S}^{\text{ref}}|.$$

For quality reward, we design three assessment models to evaluate the generated code from different aspects. Let $\phi_{\text{logic}}, \phi_{\text{layout}}, \phi_{\text{style}}$ denote the logic, layout and style assessment model, respectively. Each assessment model is designed as a neural network but trained independently. Take the logic AM ϕ_{logic} for an introduction, which consists of an LLM-based text encoder and a multi-layer perceptron (MLP). We construct an *assessment prompt* $\mathbb{T}$ containing the input description x, generated code c and a special token <eos> as follows:

$$\mathbb{T}(x, c) = x \,\|\, c \,\|\, \texttt{<eos>}$$

The text encoder of ϕ_{logic} first encodes the input prompt $\mathbb{T}$, and the hidden state of <eos> $\mathbf{h}_{\text{logic}}$ is then input into the MLP to generate the logic assessment score q_{logic}. Similarly, we use ϕ_{layout} to obtain the layout score q_{layout}, and ϕ_{style} to obtain the style score q_{style}. The quality reward is then computed by

$$R_{\text{quality}} = q_{\text{logic}} + q_{\text{layout}} + q_{\text{style}}.$$

3.3 DIFFERENCE-AND-ENTROPY FOCUSED GRPO

We note that the generated TikZ code c contains many constant procedural tokens, such as `begin{tikzpicture}`, while only a subset of tokens determines the quality of the rendered diagram. This imbalance means that the standard GRPO equally treats and updates all tokens of the generated code may not be a wise choice in the text-to-diagram task. To address this, we propose a **D**ifference-and-**E**ntropy focused **GRPO** (DE-GRPO) to select and update only key tokens in a generated code. Specifically, given a description x and the reference code c^{ref}, the current policy generates a set of candidate codes $\{c_i\}$. For each candidate c_i, we construct a *focused token set* $\mathcal{T}_i$ as follows.

Firstly, we feed a candidate code c_i and the reference code c^{ref} into the tokenizer of the current policy model π_θ to obtain their respective token sequences. Since the two token sequences often differ in length, we align their longest common subsequence to serve as an anchor. The remaining tokens are aligned based on this anchor to construct a strict token-to-token mapping. We then compare the two aligned token sequences and extract the tokens in the c_i token sequence that differ from the reference token sequence to form the difference token set $\mathcal{T}_i^{\text{diff}}$. For example, when aligning a generated token sequence `X Y A B C` against a reference token sequence `A B C Z`, we anchor the shared block `A B C` and pad the sequences into `X Y A B C -` and `- - A B C Z`, respectively. Consequently, the mismatched tokens from the generated token sequence are extracted, yielding {`X`, `Y` } as $\mathcal{T}_i^{\text{diff}}$.

Secondly, we compute the entropy of each token in the generated code c_i under the current policy. Specifically, for each generation step t in generating c_i , given the input description x and the prefix $y_{<t}$ of c, the policy model generates a probability distribution over the vocabulary $\mathcal{V}$ for the next token. The entropy H_t of this distribution is defined as:

$$H_t = -\sum_{v \in \mathcal{V}} P(v \mid y_{<t}, x) \log P(v \mid y_{<t}, x),$$

where $P(v \mid y_{<t}, x)$ is the probability assigned by the current policy in selecting token v at the step t. The tokens with the top 20% highest entropy values are selected to construct $\mathcal{T}_i^{\text{ent}}$.

Finally, we take the intersection of the two sets to obtain the final focused token set $\mathcal{T}_i$, i.e., $\mathcal{T}_i = \mathcal{T}_i^{\text{diff}} \cap \mathcal{T}_i^{\text{ent}}$. In the model training phase, we restrict to update only the tokens in $\mathcal{T}_i$ in the GRPO training step.

3.4 TRAINING

Our training consists of two sequential parts: the three assessment models $\phi_{\text{logic}}, \phi_{\text{layout}}, \phi_{\text{style}}$ are independently trained, and after their training, the generative model π_θ is trained.

Assessment Model Training. Given a description x and its reference code c^{ref}, we construct three types of negative codes $\{c_{\text{logic}}^{\text{neg}}, c_{\text{layout}}^{\text{neg}}, c_{\text{style}}^{\text{neg}}\}$ (Section 4). Take the ϕ_{logic} training for an introduction. We construct two prompts $\mathbb{T}(x, c^{\text{ref}})$ and $\mathbb{T}(x, c_{\text{logic}}^{\text{neg}})$, and feed them separately into ϕ_{logic} to obtain their corresponding scores $q_{\text{logic}}^{\text{ref}}$ and $q_{\text{logic}}^{\text{neg}}$. Based on the two scores, we adopt a Bradley-Terry style pairwise ranking objective to train the assessment model ϕ_{logic}

$$\mathcal{L}_{\text{logic}} = -\log \sigma(q_{\text{logic}}^{\text{ref}} - q_{\text{logic}}^{\text{neg}}),$$

where $\sigma(\cdot)$ is the sigmoid function.

Generative Model Training. The training of generative model π_θ is divided into two stages. In the first stage, we warm up π_θ via supervised fine-tuning on each training pair (x, c^{ref}) to provide a stable initialization for subsequent reinforcement learning. The loss function is defined as follows:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(x, c^{\text{ref}}) \sim \mathcal{D}} \Big[\sum_{t=1}^{|c^{\text{ref}}|} \log \pi_\theta \Big(c_t^{\text{ref}} \mid x, c_{<t}^{\text{ref}} \Big) \Big],$$

where c_t^{ref} is the t-th token in c^{ref}.

In the second stage, we further train π_θ using reinforcement learning based on the proposed DE-GRPO. For each input description x, we sample a set of G candidate codes from the current policy,

and compute each candidate c_i a reward R_{overall}. The rewards are normalized within group to obtain the advantage $\hat{A}_i$. Based on the normalized advantages, the policy model is updated according to the DE-GRPO objective:

$$\mathcal{J}_{\text{DE-GRPO}}(\theta) = \mathbb{E}_{(x,c^{\text{ref}})\sim\mathcal{D}}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|\mathcal{T}_i|}\sum_{t\in\mathcal{T}_i}\min\Big(r_{i,t}\hat{A}_i,\ \text{clip}(r_{i,t},1-\epsilon,1+\epsilon)\hat{A}_i\Big) - \beta D_{\text{KL}}(\pi_\theta \parallel \pi_{\text{ref}})\right],$$

where $\mathcal{T}_i$ denotes the selected token set for candidate code c_i, $|\mathcal{T}_i|$ is its size, and $r_{i,t} = \pi_\theta(y_t|y_{<t},x)/\pi_{\text{old}}(y_t|y_{<t},x)$.

4 Negative Construction

We construct three types of negative samples for training the three assessment models via contrastive learning. Our work is based on the SketikZ dataset, which contains ground-truth samples of $\{(c^{\text{ref}}, d)\}$ pairs (c^{ref} the reference TikZ code and d the rendered diagram). For each ground-truth sample, we design an LLM-based multi-agent pipeline to generate its textual description x and three negative codes $c^{\text{neg}}_{\text{logic}}$, $c^{\text{neg}}_{\text{layout}}$ and $c^{\text{neg}}_{\text{style}}$ from the diagram `logic`, `layout`, and `style` aspect, respectively. In what follows, we take the construction of $c^{\text{neg}}_{\text{logic}}$ for an introduction. The multi-agent pipeline and related LLM prompts are provided in Appendix C.

Step 1. Describe Agent: This agent prompts an LLM to generate structured description x for a ground-truth sample, which covers four facets, including logic (key elements and their semantic relations), layout (element relative positions and alignment), style (visual attributes of elements, such as color, line width, font, etc.), and labels (diagram texts).

Step 2. Plan Agent: Given x and the aspect `logic`, this agent prompts an LLM to generate an aspect-specific change plan Δ_{logic}. The Δ_{logic} contains a few perturbations only within the `logic` aspect. The Δ_{logic} is formatted as an ordered list of executable edit directives (e.g., the edit location and the corresponding modification) for subsequent code editing.

Step 3. Edit Agent: Given the TikZ reference code c^{ref} and the change plan Δ_{neg}, the Edit Agent prompts an LLM to produce the corresponding modified code c^{neg}_{logic}.

Step 4. Quality control: The generated negative code $c^{\text{neg}}_{\text{logic}}$ is required to pass the compilation for rendering a diagram. We recruit graduate students to manually inspect the rendered diagram to ensure that the intended perturbation is confined only to the `logic` aspect.

Data Statistics: We use the SigLIP (Zhai et al., 2023) to classify and analyze the reference codes, with example descriptions and charts provided in Appendix B.1. The dataset contains 2585 training samples and 323 test samples. The negative samples include: 2111 logic negatives, 2264 layout negatives, and 2376 style negatives.

5 Experiments

5.1 Evaluation Setup

Datasets. We train the model on SketikZ (Saito et al., 2025) and use its test set as the main benchmark for evaluation. We further conduct additional evaluation on DaTikZ-V1, DaTikZ-V4 and DiagramGenBenchmark, which include diagram types different from those in SketikZ. Detailed statistics are provided in Appendix B. Besides, we use SigLIP (Zhai et al., 2023) to categorize examples in DaTikZ and DiagramGenBenchmark by diagram type. Detailed results regarding in-distribution and out-of-distribution diagram types are provided in Appendix E.5.

Metrics. We evaluate code quality using ROUGE-L, Edit Distance, chrF, and CrystalBLEU (C-BLEU), code executability using Compilation Success Rate (CSR), and rendered diagram quality using $\text{CLIPScore}_{\text{IMG}}$, LPIPS, PSNR, and MS-SSIM. Metric definitions are provided in Appendix E.7.

Implementation Details. All experiments are based on Qwen2.5-Coder-7B. During training, we set the policy model learning rate to 1×10^{-6} and use sampling with a temperature of 1.0. For inference, we adopt the vLLM framework for acceleration and set the maximum input length to 8192 tokens. More implementation details are provided in Appendix E.2.

Table 1: Main results on the SKETIKZ dataset. The full results are reported in Table 15.

Model	Size	ROUGE-L↑	EditDist.↓	chrF↑	C-BLEU↑	CLIPScore↑	LPIPS↓	PSNR↑	MS-SSIM↑	CSR↑
Coder Large Language Models										
Qwen2.5-Coder	7B	39.438	67.608	44.042	24.062	0.816	0.361	20.269	0.676	0.807
Code-Llama	34B	38.592	68.509	44.328	24.913	0.805	0.412	17.562	0.612	0.370
StarCoder2	15B	40.543	67.357	43.802	23.647	0.812	0.379	19.368	0.620	0.582
Chat Large Language Models										
Baichuan2-Chat	13B	33.702	71.176	36.297	16.375	0.780	0.407	17.421	0.586	0.535
InternLM2.5-Chat	20B	38.196	69.164	45.521	23.835	0.804	0.324	21.087	0.683	0.766
Yi-1.5	34B	34.683	71.087	43.168	22.450	0.804	0.406	18.967	0.627	0.454
Commercial Large Language Models										
GLM-4.7	355B	38.157	68.651	48.843	25.709	0.819	0.423	18.085	0.603	0.849
DeepSeek-V3	671B	42.384	66.305	47.740	29.698	0.810	0.428	17.390	0.574	0.929
Qwen3-Max	–	39.214	67.410	49.793	25.832	0.829	0.431	17.815	0.579	0.816
GPT-5.4	–	43.490	63.362	49.609	26.374	0.837	0.375	20.815	0.641	0.951
TikZ-specific Large Language Models										
CLiMA-7B	7B	35.165	69.827	42.482	18.078	0.709	0.512	19.192	0.519	0.323
CLiMA-13B	13B	37.953	69.587	41.204	20.982	0.847	0.376	19.805	0.570	0.471
DiagramCodeAgent	7B	38.682	68.482	46.279	21.027	0.842	0.337	21.417	0.684	0.753
TikZilla	8B	43.847	63.951	48.670	27.545	0.862	0.266	25.309	0.817	0.910
FocusTikZ	**7B**	**45.553**	**62.985**	**50.144**	**29.986**	**0.872**	**0.189**	**26.499**	**0.828**	**0.963**

Table 2: Generalization results on DATIKZ-V1 and DATIKZ-V4. The best results are in bold.

Model	DATIKZ-V1				DATIKZ-V4			
	ROUGE-L↑	chrF↑	CLIPScore↑	PSNR↑	EditDist.↓	C-BLEU↑	LPIPS↓	MS-SSIM↑
Qwen2.5-Coder	37.143	39.705	0.834	20.559	69.742	11.679	0.415	0.683
GLM-4.7	37.163	44.973	0.812	17.855	68.251	21.672	0.405	0.676
DeepSeek-V3	40.495	44.101	0.812	17.727	69.506	16.713	0.416	0.630
Qwen3-Max	36.465	44.357	0.824	18.212	70.795	19.078	0.307	0.731
GPT-5.4	41.346	45.210	0.863	21.682	68.762	**24.956**	0.274	0.826
CLiMA-7B	36.328	40.460	0.795	20.780	72.635	9.738	0.492	0.519
CLiMA-13B	39.635	42.710	0.817	20.039	70.153	12.341	0.462	0.589
DiagramCodeAgent	38.387	42.630	0.837	21.876	69.698	13.075	0.379	0.768
TikZilla	41.847	43.270	0.786	26.519	67.713	19.081	0.276	0.819
FocusTikZ	**43.285**	**46.161**	**0.896**	**27.285**	**66.358**	19.154	**0.203**	**0.857**

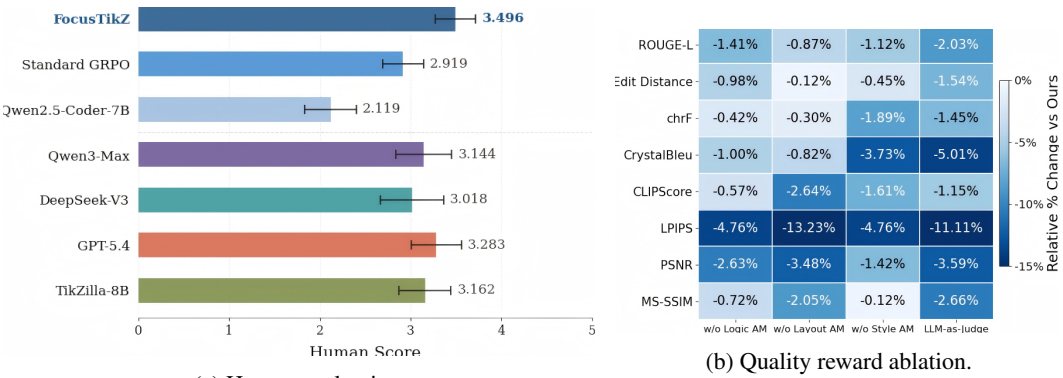
5.2 MAIN RESULTS

Table 1 presents the main results on the SKETIKZ dataset. Tables 2 and 3 report the results on DATIKZ-V1, DATIKZ-V4 and DIAGRAMGENBENCHMARK, to evaluate the generalization capability of our FocusTikZ framework. Our FocusTikZ framework achieves the most competitive results on all datasets. We attribute this superiority to our reinforcement learning paradigm. Specifically, the multi-reward mechanism guides the LLM to capture the intrinsic logic, layout, and style of the diagrams, while the focused update strategy concentrates the learning capacity on the key tokens that determine the quality of the rendered diagrams.

We can observe that LLMs in zero-shot learning, including both coder and chat models, struggle to generate accurate TikZ code. Notably, this performance gap persists across varying parameter sizes, which indicates that Text-to-TikZ is still a challenging task for LLMs due to the strict requirements for syntactic correctness, structural consistency, and visual fidelity. Besides, it is interesting to note that our FocusTikZ also outperforms the supervised baseline DiagramCodeAgent. We argue that supervised methods suffer from the paradigm of matching ground-truth codes, which ignores the fact that a single diagram can be validly rendered by diverse TikZ implementations. While the FocusTikZ employs a reinforcement learning framework with multi-reward, aligning the generated diagram with the description across logic, layout, and style to accommodate diverse code implementations.

Table 3: Generalization results on DIAGRAMGENBENCHMARK. Full results are reported in Table 16.

Model	Size	ROUGE-L↑	EditDist.↓	chrF↑	C-BLEU↑	CLIPScore↑	LPIPS↓	PSNR↑	MS-SSIM↑	CSR↑
Qwen2.5-Coder	7B	36.903	69.725	43.148	25.265	0.830	0.418	17.939	0.593	0.776
GLM-4.7	355B	35.138	71.397	48.665	24.099	0.849	0.420	16.650	0.561	0.835
DeepSeek-V3	671B	39.176	68.684	48.312	27.269	0.844	0.442	16.231	0.544	0.933
Qwen3-Max	–	35.278	70.834	48.651	21.810	**0.857**	0.433	16.238	0.542	0.830
GPT-5.4	–	40.571	66.831	**49.418**	27.594	0.853	0.383	18.458	0.722	0.947
CLiMA-7B	7B	34.372	70.236	41.869	18.826	0.679	0.472	16.795	0.572	0.340
CLiMA-13B	13B	37.458	69.785	44.497	17.582	0.791	0.458	17.815	0.596	0.395
DiagramCodeAgent	7B	38.721	68.372	47.673	22.843	0.845	0.379	18.800	0.620	0.813
TikZilla	8B	39.751	68.144	46.490	24.072	0.758	0.327	20.070	0.633	0.895
FocusTikZ	**7B**	**41.887**	**66.522**	48.063	**28.938**	0.850	**0.300**	**22.617**	**0.727**	**0.955**



(a) Human evaluation.

(b) Quality reward ablation.

Figure 4: (a) Human evaluation results of different models with 95% confidence intervals. (b) Relative performance changes under different quality reward variants.

5.3 HUMAN EVALUATION

To complement the automated metrics and better assess the quality of the generated diagrams, we conducted a comprehensive human evaluation. We randomly sampled 50 generated diagrams per evaluated model. A panel of 15 independent annotators, all possessing bachelor-level education or higher, was recruited to review the outputs. The annotators were blind to the model identities and were asked to rate each generated diagram on a 1–5 scale based on its overall similarity and visual fidelity to the ground-truth reference diagram, where a score of 1 indicates complete dissimilarity and minimal quality, whereas a score of 5 represents a perfect visual match and maximum quality.

As illustrated in Figure 4(a), FocusTikZ achieves the highest average human evaluation score among all evaluated models. In particular, FocusTikZ substantially improves over its base model, as well as the standard GRPO baseline (2.919). FocusTikZ also surpasses the specialized diagram generation model TikZilla-8B (3.162) and strong general models such as DeepSeek-V3 (3.018), Qwen3-Max (3.144) and GPT-5.4 (3.283). These results indicate that the improvements observed under automated metrics are also reflected in human judgments, providing further evidence that FocusTikZ generates diagrams with better overall similarity and visual fidelity to the reference diagrams.

5.4 ABLATION STUDIES AND ANALYSIS

Ablation of FocusTikZ. To evaluate the contribution of each training stage, we conduct an ablation study where removing SFT refers to directly applying RL to the model without an SFT warm-up, and removing RL means training the model using only SFT. Table 4 provides the ablation results of the FocusTikZ framework. We can observe that removing either training stage leads to a decline in overall performance. We argue that SFT provides a necessary and stable initialization, enabling the framework to generate valid TikZ syntax. RL further optimizes the generation process, focusing on the quality of rendered diagram.

Table 4: Ablation studies of FocusTikZ. Gray superscripts denote changes relative to the full model.

Model	ROUGE -L↑	Edit Dist.↓	chrF↑	Crystal BLEU↑	CLIP Score↑	LPIPS↓	PSNR↑	MS SSIM↑	CSR↑
FocusTikZ	**45.553**	**62.985**	**50.144**	**29.986**	**0.872**	**0.189**	**26.499**	**0.828**	**0.963**
Ablation on Training Stages									
w/o SFT	39.453$^{(-6.10)}$	67.787$^{(+4.80)}$	43.869$^{(-6.28)}$	24.171$^{(-5.82)}$	0.814$^{(-0.06)}$	0.368$^{(+0.18)}$	19.589$^{(-6.91)}$	0.647$^{(-0.18)}$	0.847$^{(-0.12)}$
w/o RL	38.336$^{(-7.22)}$	68.571$^{(+5.59)}$	46.916$^{(-3.23)}$	21.706$^{(-8.28)}$	0.849$^{(-0.02)}$	0.230$^{(+0.04)}$	25.054$^{(-1.45)}$	0.789$^{(-0.04)}$	0.819$^{(-0.14)}$
Ablation on Reward Components									
w/o R_{success}	42.483$^{(-3.07)}$	65.251$^{(+2.27)}$	45.302$^{(-4.84)}$	27.005$^{(-2.98)}$	0.832$^{(-0.04)}$	0.224$^{(+0.04)}$	25.303$^{(-1.20)}$	0.798$^{(-0.03)}$	0.877$^{(-0.09)}$
w/o R_{token}	44.693$^{(-0.86)}$	63.617$^{(+0.63)}$	48.875$^{(-1.27)}$	28.747$^{(-1.24)}$	0.852$^{(-0.02)}$	0.204$^{(+0.02)}$	26.126$^{(-0.37)}$	0.814$^{(-0.01)}$	0.932$^{(-0.03)}$
Ablation on Token Selection Strategies									
$\mathcal{T}^{\text{diff}} \cup \mathcal{T}^{\text{ent}}$	44.362$^{(-1.19)}$	64.646$^{(+1.66)}$	48.595$^{(-1.55)}$	29.107$^{(-0.88)}$	0.857$^{(-0.02)}$	0.195$^{(+0.01)}$	26.129$^{(-0.37)}$	0.804$^{(-0.02)}$	0.942$^{(-0.02)}$
w/o $\mathcal{T}^{\text{ent}}$	44.595$^{(-0.96)}$	63.960$^{(+0.98)}$	49.504$^{(-0.64)}$	28.123$^{(-1.86)}$	0.856$^{(-0.02)}$	0.202$^{(+0.01)}$	26.276$^{(-0.22)}$	0.813$^{(-0.02)}$	0.948$^{(-0.02)}$
w/o $\mathcal{T}^{\text{diff}}$	44.698$^{(-0.86)}$	63.569$^{(+0.58)}$	48.989$^{(-1.16)}$	26.830$^{(-3.16)}$	0.854$^{(-0.02)}$	0.207$^{(+0.02)}$	25.648$^{(-0.85)}$	0.816$^{(-0.01)}$	0.950$^{(-0.01)}$
w/o DE-GRPO	43.684$^{(-1.87)}$	64.699$^{(+1.71)}$	48.661$^{(-1.48)}$	25.484$^{(-4.50)}$	0.847$^{(-0.03)}$	0.212$^{(+0.02)}$	25.173$^{(-1.33)}$	0.792$^{(-0.04)}$	0.921$^{(-0.04)}$

Ablation of Quality Assessment Models. To evaluate the effectiveness of each assessment model, we conduct the ablation experiment by removing the logic, layout, and style assessment models individually. As shown in Figure 4(b), each assessment model contributes to the performance of the quality reward. Specifically, removing the Logic AM leads to the degradation of sequence-level matching metrics. We argue that the Logic AM focuses on the topology of the rendered diagram, which affects the global alignment of the code sequence. The largest image-level performance drop occurs when removing the Layout AM, indicating that the layout AM plays an important role in the spatial structure of the diagrams. Besides, the style AM evaluates the correctness of stylistic configurations (e.g., color, size, and font), which are crucial for visual details. These comparisons validate that quality assessment models provide complementary learning signals for RL optimization.

Ablation on Update Strategies. We conduct ablation studies on our update strategy to evaluate the contribution of DE-GRPO, with experiments performed under three settings: (1) updating solely the difference token set, (2) updating solely the highest entropy token set, (3) replacing DE-GRPO entirely with standard GRPO. Table 4 reports the corresponding absolute metric values.

The performance drops in the first two settings demonstrate that both token sets are essential. Specifically, the update strategy on the difference token set focuses on correcting token mismatches; the update strategy on the high-entropy token set focuses on capturing contextual uncertainty. We argue that the entropy of tokens reflects the generation confidence based on the existing sequence, optimizing these tokens contributes to overall sequence consistency. Besides, replacing our DE-GRPO with standard GRPO yields the lowest performance, indicating that treating all tokens equally during policy optimization is not a wise choice. Conversely, our DE-GRPO effectively prioritizes the key tokens, which better aligns with the text-to-TikZ task.

Ablation of Reward Components. We further examine the contribution of the two foundational rewards, R_{success} and R_{token}, while keeping the quality reward unchanged. As shown in Table 4, removing R_{success} causes the largest degradation in code-level metrics, including substantial drops in ROUGE-L, chrF, and CrystalBLEU. This result highlights the importance of directly penalizing failed compilation, which provides a basic executability constraint for policy optimization. Removing R_{token} also consistently reduces performance across both code-level and diagram-level metrics. The token similarity reward therefore serves as a useful code-level anchor that regularizes policy updates while allowing the quality reward to optimize the rendered diagram beyond exact reference matching.

6 Conclusion

This paper has proposed the FocusTikZ, a reinforcement learning framework for text-to-diagram generation task. The FocusTikZ designs a multi-reward to assess the generated code across different aspects, including rendering success, token-level similarity, as well as the logic, layout, and style quality. The DE-GRPO is then introduced to optimize the policy model over key tokens in the generated code by multi-reward. Experiments show that FocusTikZ outperforms the baselines in both code accuracy and rendered diagram fidelity. We hope this work will inspire future research on reinforcement learning for high-quality diagram generation.

REFERENCES

01.AI. Meet yi-coder: A small but mighty llm for code, 2024.

Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu, and Charles Sutton. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)*, 51(4):1–37, 2018.

Jonas Belouadi, Anne Lauscher, and Steffen Eger. Automatikz: Text-guided synthesis of scientific vector graphics with tikz. In *International Conference on Learning Representations*, volume 2024, pp. 55917–55943, 2024a.

Jonas Belouadi, Simone Paolo Ponzetto, and Steffen Eger. Detikzify: Synthesizing graphics programs for scientific figures and sketches with tikz. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024b. URL http://papers.nips.cc/paper_files/paper/2024/hash/9a8d52eb05eb7b13f54b3d9eada667b7-Abstract-Conference.html.

Jonas Belouadi, Eddy Ilg, Margret Keuper, Hideki Tanaka, Masao Utiyama, Raj Dabre, Steffen Eger, and Simone Paolo Ponzetto. Tikzero: Zero-shot text-guided graphics program synthesis. In *IEEE/CVF International Conference on Computer Vision, ICCV 2025, Honolulu, HI, USA, October 19-25, 2025*, pp. 17793–17806. IEEE, 2025. doi: 10.1109/ICCV51701.2025.01653. URL https://doi.org/10.1109/ICCV51701.2025.01653.

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, Xiaoyi Dong, Haodong Duan, Qi Fan, Zhaoye Fei, Yang Gao, Jiaye Ge, Chenya Gu, Yuzhe Gu, Tao Gui, Aijia Guo, Qipeng Guo, Conghui He, Yingfan Hu, Ting Huang, Tao Jiang, Penglong Jiao, Zhenjiang Jin, Zhikai Lei, Jiaxing Li, Jingwen Li, Linyang Li, Shuaibin Li, Wei Li, Yining Li, Hongwei Liu, Jiangning Liu, Jiawei Hong, Kaiwen Liu, Kuikun Liu, Xiaoran Liu, Chengqi Lv, Haijun Lv, Kai Lv, Li Ma, Runyuan Ma, Zerun Ma, Wenchang Ning, Linke Ouyang, Jiantao Qiu, Yuan Qu, Fukai Shang, Yunfan Shao, Demin Song, Zifan Song, Zhihao Sui, Peng Sun, Yu Sun, Huanze Tang, Bin Wang, Guoteng Wang, Jiaqi Wang, Jiayu Wang, Rui Wang, Yudong Wang, Ziyi Wang, Xingjian Wei, Qizhen Weng, Fan Wu, Yingtong Xiong, Xiaomeng Zhao, and et al. Internlm2 technical report. *CoRR*, abs/2403.17297, 2024. doi: 10.48550/ARXIV.2403.17297. URL https://doi.org/10.48550/arXiv.2403.17297.

Hanqun Cao, Cheng Tan, Zhangyang Gao, Yilun Xu, Guangyong Chen, Pheng-Ann Heng, and Stan Z Li. A survey on generative diffusion models. *IEEE transactions on knowledge and data engineering*, 36(7):2814–2830, 2024.

Zhiqing Cui, Jiahao Yuan, Hanqing Wang, Yanshu Li, Chenxu Du, and Zhenglong Ding. Draw with thought: Unleashing multimodal reasoning for scientific diagram generation. In Cathal Gurrin, Klaus Schoeffmann, Min Zhang, Luca Rossetto, Stevan Rudinac, Duc-Tien Dang-Nguyen, Wen-Huang Cheng, Phoebe Chen, and Jenny Benois-Pineau (eds.), *Proceedings of the 33rd ACM International Conference on Multimedia, MM 2025, Dublin, Ireland, October 27-31, 2025*, pp. 5050–5059. ACM, 2025. doi: 10.1145/3746027.3755731. URL https://doi.org/10.1145/3746027.3755731.

DeepSeek-AI. Deepseek-v3 technical report. *CoRR*, abs/2412.19437, 2024. doi: 10.48550/ARXIV.2412.19437. URL https://doi.org/10.48550/arXiv.2412.19437.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12606–12633. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/esser24a.html.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. Codebert: A pre-trained model for programming and natural languages. In Trevor Cohn, Yulan He, and Yang Liu (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pp. 1536–1547. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.FINDINGS-EMNLP.139. URL https://doi.org/10.18653/v1/2020.findings-emnlp.139.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Christian Greisinger and Steffen Eger. Tikzilla: Scaling text-to-tikz with high-quality data and reinforcement learning. *arXiv preprint arXiv:2603.03072*, 2026.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. Deepseek-coder: When the large language model meets programming - the rise of code intelligence. *CoRR*, abs/2401.14196, 2024. doi: 10.48550/ARXIV.2401.14196. URL https://doi.org/10.48550/arXiv.2401.14196.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, An Yang, Rui Men, Fei Huang, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-coder technical report. *CoRR*, abs/2409.12186, 2024. doi: 10.48550/ARXIV.2409.12186. URL https://doi.org/10.48550/arXiv.2409.12186.

Angela H. Jiang, Daniel L.-K. Wong, Giulio Zhou, David G. Andersen, Jeffrey Dean, Gregory R. Ganger, Gauri Joshi, Michael Kaminsky, Michael Kozuch, Zachary C. Lipton, and Padmanabhan Pillai. Accelerating deep learning by focusing on the biggest losers. *CoRR*, abs/1910.00762, 2019. URL http://arxiv.org/abs/1910.00762.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4401–4410, 2019.

Satyam Kumar, Dayima Musharaf, Seerat Musharaf, and Anil Kumar Sagar. A comprehensive review of the latest advancements in large generative ai models. In *International conference on advanced communication and intelligent systems*, pp. 90–103. Springer, 2023.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and Sushant Prakash. RLAIF vs. RLHF: scaling reinforcement learning from human feedback with AI feedback. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 26874–26901. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/lee24t.html.

Zhiqiu Lin, Deepak Pathak, Baiqi Li, Jiayao Li, Xide Xia, Graham Neubig, Pengchuan Zhang, and Deva Ramanan. Evaluating text-to-visual generation with image-to-text generation. In *European Conference on Computer Vision*, pp. 366–384. Springer, 2024.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian J. McAuley, Han Hu,

Torsten Scholak, Sébastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, and et al. Starcoder 2 and the stack v2: The next generation. *CoRR*, abs/2402.19173, 2024. doi: 10.48550/ARXIV.2402.19173. URL https://doi.org/10.48550/arXiv.2402.19173.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*, 2023.

OpenAI. Gpt-4o system card. *CoRR*, abs/2410.21276, 2024. doi: 10.48550/ARXIV.2410.21276. URL https://doi.org/10.48550/arXiv.2410.21276.

OpenAI. Introducing GPT-5.4. OpenAI, March 2026. URL https://openai.com/index/introducing-gpt-5-4/.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International conference on machine learning*, pp. 8821–8831. Pmlr, 2021.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950, 2023. doi: 10.48550/ARXIV.2308.12950. URL https://doi.org/10.48550/arXiv.2308.12950.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/ec795aeadae0b7d230fa35cbaf04c041-Abstract-Conference.html.

Itsumi Saito, Haruto Yoshida, and Keisuke Sakaguchi. Sketch2diagram: Generating vector diagrams from hand-drawn sketches. In *13th International Conference on Learning Representations, ICLR 2025*, pp. 52825–52847. International Conference on Learning Representations, ICLR, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL https://doi.org/10.48550/arXiv.2402.03300.

Llama Team. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/ARXIV.2407.21783. URL https://doi.org/10.48550/arXiv.2407.21783.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Junxiao Wang, Ting Zhang, Heng Yu, Jingdong Wang, and Hua Huang. Magicgeo: Training-free text-guided geometric diagram generation. *CoRR*, abs/2502.13855, 2025a. doi: 10.48550/ARXIV.2502.13855. URL https://doi.org/10.48550/arXiv.2502.13855.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xiong-Hui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for LLM reasoning. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025b. URL http://papers.nips.cc/paper_files/paper/2025/hash/a797c2d2e0c1fdabf4d1ab8cd0b465c6-Abstract-Conference.html.

Jingxuan Wei, Cheng Tan, Qi Chen, Gaowei Wu, Siyuan Li, Zhangyang Gao, Linzhuang Sun, Bihui Yu, and Ruifeng Guo. From words to structured visuals: A benchmark and framework for text-to-diagram generation and editing. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 13315–13325, 2025.

Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, Fan Yang, Fei Deng, Feng Wang, Feng Liu, Guangwei Ai, Guosheng Dong, Haizhou Zhao, Hang Xu, Haoze Sun, Hongda Zhang, Hui Liu, Jiaming Ji, Jian Xie, Juntao Dai, Kun Fang, Lei Su, Liang Song, Lifeng Liu, Liyun Ru, Luyao Ma, Mang Wang, Mickel Liu, MingAn Lin, Nuolan Nie, Peidong Guo, Ruiyang Sun, Tao Zhang, Tianpeng Li, Tianyu Li, Wei Cheng, Weipeng Chen, Xiangrong Zeng, Xiaochuan Wang, Xiaoxi Chen, Xin Men, Xin Yu, Xuehai Pan, Yanjun Shen, Yiding Wang, Yiyu Li, Youxin Jiang, Yuchen Gao, Yupeng Zhang, Zenan Zhou, and Zhiying Wu. Baichuan 2: Open large-scale language models. *CoRR*, abs/2309.10305, 2023. doi: 10.48550/ARXIV.2309.10305. URL https://doi.org/10.48550/arXiv.2309.10305.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report. *CoRR*, abs/2407.10671, 2024a. doi: 10.48550/ARXIV.2407.10671. URL https://doi.org/10.48550/arXiv.2407.10671.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *CoRR*, abs/2412.15115, 2024b. doi: 10.48550/ARXIV.2412.15115. URL https://doi.org/10.48550/arXiv.2412.15115.

Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, Kaidong Yu, Peng Liu, Qiang Liu, Shawn Yue, Senbin Yang, Shiming Yang, Tao Yu, Wen Xie, Wenhao Huang, Xiaohui Hu, Xiaoyi Ren, Xinyao Niu, Pengcheng Nie, Yuchi Xu, Yudong Liu, Yue Wang, Yuxuan Cai, Zhenyu Gu, Zhiyuan Liu, and Zonghong Dai. Yi: Open foundation models by 01.ai. *CoRR*, abs/2403.04652, 2024. doi: 10.48550/ARXIV.2403.04652. URL https://doi.org/10.48550/arXiv.2403.04652.

Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. Chatglm: A family of large language models from GLM-130B to GLM-4 all tools. *CoRR*, abs/2406.12793, 2024. doi: 10.48550/ARXIV.2406.12793. URL https://doi.org/10.48550/arXiv.2406.12793.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 11975–11986, 2023.

Honglei Zhuang, Zhen Qin, Kai Hui, Junru Wu, Le Yan, Xuanhui Wang, and Michael Bendersky. Beyond yes and no: Improving zero-shot llm rankers via scoring fine-grained relevance labels. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pp. 358–370, 2024.

Bocheng Zou, Mu Cai, Jianrui Zhang, and Yong Jae Lee. Vgbench: Evaluating large language models on vector graphics understanding and generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 3647–3659, 2024.

AI USE STATEMENT

In this work, generative AI tools were used as part of the research pipeline to construct synthetic data for training and evaluation. We did not use generative AI tools to formulate mathematical claims or proofs, or to determine the final interpretation and conclusions of the experimental results. Additionally, generative AI tools were used to polish the manuscript for readability. All AI-assisted content was reviewed and revised by the authors. We take responsibility for the final content of this work, including text, claims, data, and artifacts produced with the aid of generative AI.

APPENDIX CONTENTS

A Related Work **15**

A.1 Text-to-Image Generation 15

A.2 Text-to-Code Generation 16

A.3 Reinforcement Learning for LLMs 16

A.4 Concurrent Work 16

B Dataset Details **17**

B.1 Positive Dataset 17

B.2 Negative Dataset 18

C Multi-Agent Prompts for Negative Sample Construction **20**

D Analysis and Statistics of Highly Uncertain Tokens **20**

E Additional Experiments **22**

E.1 Prompt Details 22

E.2 Implementation Details 22

E.3 Evaluation and Analysis of Quality Assessment Models 25

E.4 LLM-as-Judge Score Statistics 26

E.5 Results by Diagram Type 27

E.5.1 Evaluation Setup 27

E.5.2 Analysis 27

E.6 Qualitative Examples of Selected Key Tokens 28

E.7 Metrics Details 29

F Generalization to Other Diagramming Languages **31**

A RELATED WORK

A.1 TEXT-TO-IMAGE GENERATION

Text-to-image generation has rapidly evolved and can be summarized in three stages: GAN-based methods, autoregressive models, and diffusion models. Early progress was driven by generative adversarial networks (Goodfellow et al., 2020), with representative improvements in style-based generators such as StyleGAN (Karras et al., 2019). Autoregressive text-to-image models later formulated images as discrete token sequences and generated them via Transformer decoding,

exemplified by DALL–E (Ramesh et al., 2021). More recently, diffusion models have become the dominant paradigm for high-fidelity synthesis, including DDPM (Ho et al., 2020) and text-conditioned variants such as latent diffusion (Rombach et al., 2022) and Imagen (Saharia et al., 2022). Despite their success in photorealistic image synthesis, these approaches are less suited for diagram-oriented generation that requires explicit logical structure, precise relations, and fine-grained editability.

A.2 Text-to-Code Generation

Text-to-code generation aims to convert natural language descriptions into executable programs, and has been substantially advanced by large language models trained on code corpora. Code-specialized open models such as Qwen2.5-Coder (Hui et al., 2024), DeepSeek-Coder (Guo et al., 2024), Code Llama (Rozière et al., 2023), and WizardCoder (Luo et al., 2023) demonstrate strong capabilities in synthesizing complex code with coherent program structure. General-purpose open LLMs including Llama 3 (Team, 2024) and Qwen2 (Yang et al., 2024a) also show competitive coding performance, while proprietary systems such as GPT-4o (OpenAI, 2024) further improve reliability on instruction-following code generation. These advances have inspired research on synthesizing executable graphics programs for diagrams. Automatikz (Belouadi et al., 2024a) treats TikZ as a compilable vector-graphics language for scientific figures and constructs paired data for text-guided TikZ synthesis. Recent benchmark and framework efforts for text-to-diagram generation and editing, including DiagramAgent (Wei et al., 2025), emphasize the need for planning and verification to produce editable structured visuals. However, applying generic text-to-code objectives to text-to-TikZ remains challenging because token-level overlap and compilation success do not reliably reflect diagram-level semantic correctness.

A.3 Reinforcement Learning for LLMs

Reinforcement learning has become a central technique for improving LLM generations beyond supervised fine-tuning. A representative line is reinforcement learning from human feedback, which trains reward models from preference data and optimizes the policy model with PPO-style objectives (Ouyang et al., 2022; Schulman et al., 2017). Preference-based alternatives such as DPO learn directly from preference pairs (Rafailov et al., 2023), and RLAIF scales preference collection using model-generated supervision (Lee et al., 2024). More recently, group-based variants have been explored to stabilize optimization and reduce memory overhead. Group Relative Policy Optimization estimates baselines using grouped samples and has been applied to improve LLM reasoning (Shao et al., 2024). In addition, efficiency-motivated strategies that focus learning on a small subset of high-impact signals, such as selective backpropagation (Jiang et al., 2019), align with the strong token imbalance in program-like outputs. For text-to-TikZ, standard sequence-level RL can over-optimize deterministic syntax tokens and weaken credit assignment for decision-critical edits, motivating task-aware RL designs that concentrate updates on tokens most responsible for diagram-level faithfulness.

A.4 Concurrent Work

A concurrent manuscript (Anonymous authors, personal correspondence, 2026) investigates the same general text-to-TikZ setting, but addresses a different learning problem. It primarily adopts a counterfactual reward attribution paradigm for fine-grained reinforcement learning. Specifically, it decomposes TikZ code into syntactic segments and estimates the reward contribution of each segment from the change in overall reward caused by removing that segment. It also introduces a TikZ-specific Abstract Syntax Tree (AST) analysis module that parses TikZ code into structured representations to capture abstract structural relations among commands, nodes, paths, and other diagram components. Its multi-view reward is constructed from compilability, AST-based structural alignment, textual consistency, and VLM-based visual similarity. FocusTikZ, instead, addresses how to identify and optimize the most informative token positions under diagram-quality supervision. It trains dedicated logic, layout, and style assessment models and proposes DE-GRPO, which selects tokens jointly according to reference differences and generation uncertainty. Therefore, the two methods follow distinct fine-grained RL paradigms: the concurrent work attributes rewards to code segments through counterfactual interventions and explicit structural analysis, whereas FocusTikZ identifies and selectively optimizes informative token positions using diagram-quality assessment and generation uncertainty.

Table 5: Key statistics of the positive dataset (left) and negative dataset (right).

Positive Dataset

Statistic	Number	Proportion
Total Samples	2908	100%
Training Samples	2585	88.9%
Testing Samples	323	11.1%
Diagram Type		
Graph	1050	36.1%
Neural Networks	877	30.2%
Tree	867	29.8%
Venn Diagram	83	2.8%
Architecture Diagram	31	1.1%
Description Length		
Minimum	1597	–
Maximum	10410	–
Average	4317	–
Code Length		
Minimum	346	–
Maximum	5834	–
Average	1601	–

Negative Dataset

Statistic	Number	Proportion
Total Samples	6751	100%
Logic Negative Samples	2111	31.3%
Layout Negative Samples	2264	33.5%
Style Negative Samples	2376	35.2%
Description Length		
Minimum	1833	–
Maximum	9363	–
Average	4238	–
Code Length		
Minimum	419	–
Maximum	4669	–
Average	1538	–

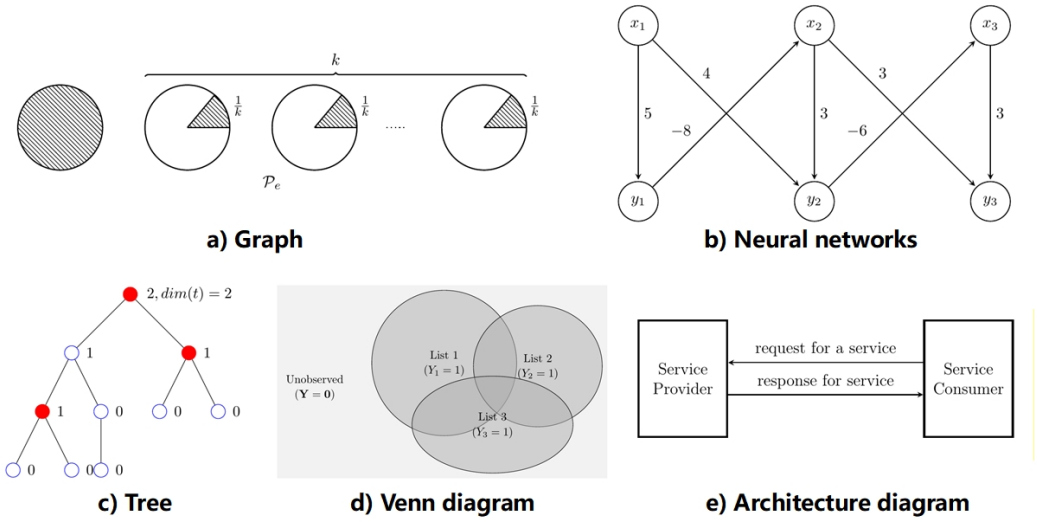


Figure 5: Examples of estimated image labels and their diagrams.

B Dataset Details

B.1 Positive Dataset

Dataset Statistics. Table 5 reports the statistics of the positive dataset. The dataset contains 2,585 training samples and 323 test samples. The average description length is 4,317 characters, and the average code length is 1,601 characters. The large gaps between the maximum and minimum lengths for both descriptions and code indicate the complexity of the task.

Dataset classification. We use SigLIP (Zhai et al., 2023)[1] as a zero-shot image–text matching model to classify diagram images in the positive dataset. Specifically, we load each image and convert it to RGB format. For each candidate class, we construct multiple text prompts. Each prompt contains a general template and an additional fine-grained description for confusing classes, as illustrated in Figure 6 and Figure 7. We then use SigLIPProcessor to encode the image together with all prompts

[1] `google/siglip-so400m-patch14-384`

General Templates

- a figure of {}.
- an academic paper figure showing {}.
- a plot/diagram of {}.
- a visualization that is {}.

Figure 6: General Prompt Templates.

Fine-grained description

```
Architecture Diagram: [
    "a system architecture diagram with multiple components/modules connected by arrows.",
    "a block diagram of a software/hardware system showing data flow between services.",
    "boxes labeled as modules/services with arrows indicating communication or pipeline stages."],
Graph: [
    "a node-link graph: vertices (nodes) connected by edges/lines, not an x-y chart.",
    "a network diagram with circles and connecting lines representing relationships.",
    "a graph theory style visualization with nodes, edges, possibly weighted/directed arrows."],
Neural networks: [
    "a neural network diagram with layers of nodes (neurons) connected between layers.",
    "a deep learning architecture figure (CNN/RNN/Transformer) with blocks like embedding/attention/conv.",
    "a model diagram showing input -> layers -> output, often with arrows and layer names."],
Tree: [
    "a tree diagram with a root node and hierarchical branches to child nodes.",
    "an organizational or hierarchical tree structure with parent-child relationships.",
    "a branching structure (not linguistic parse specifically) showing hierarchy of categories/nodes."],
Venn diagram: [
    "a Venn diagram with overlapping circles showing set intersections.",
    "two or three overlapping shapes with labeled regions for unions/intersections.",
    "a set diagram highlighting overlaps and unique regions."],
...
```

Figure 7: Fine-Grained Prompt Templates.

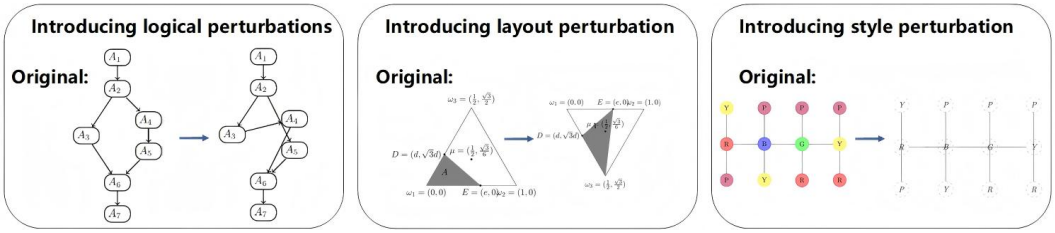


Figure 8: Examples of multi-view perturbations.

and feed them into the SigLIP model to obtain similarity scores between the image and each prompt. The detailed classification results are reported in Table 5 (left) and Figure 5.

B.2 Negative Dataset

Dataset classification. Table 5 (right) reports the statistics of the negative dataset, and Figure 8 shows example negative samples for each view. The negative dataset contains 6,751 samples in

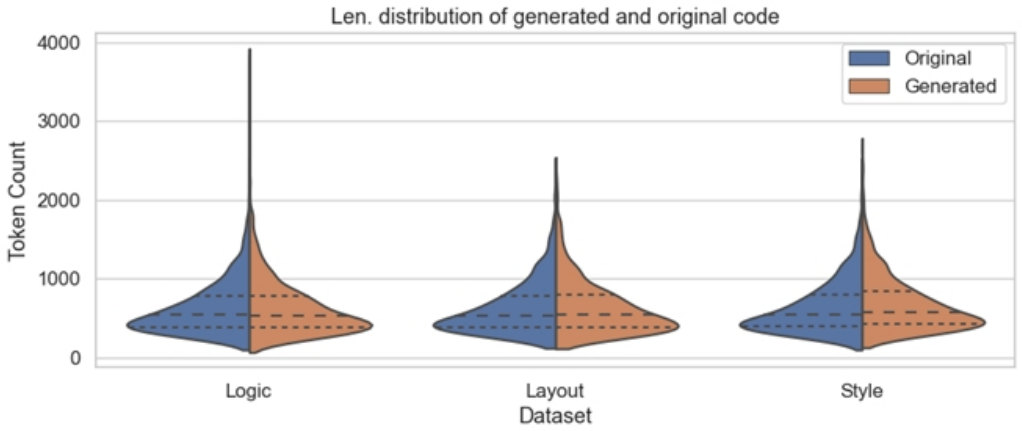


Figure 9: Comparison of length distribution between positive and negative datasets.

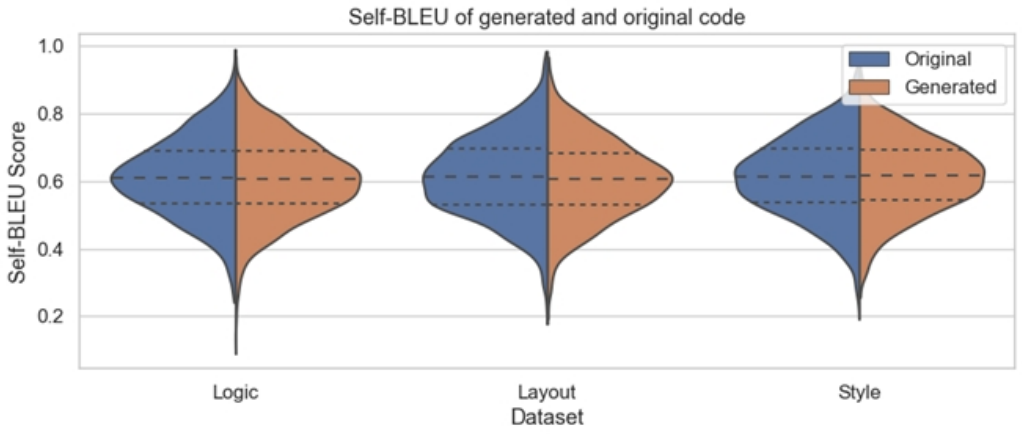


Figure 10: Comparison of Self-BLEU distribution between positive and negative datasets.

total, including 2,111 logic negatives, 2,264 layout negatives, and 2,376 style negatives. The average description length is 4,238 characters, and the average code length is 1,538 characters.

Characterizing. We further analyze the relationship between the positive and negative datasets from three aspects.

Length. As shown in Figure 9, the length distributions for each view are approximately Gaussian and closely match those of the original data. This similarity suggests that our construction procedure preserves the original length distribution, which improves the reliability of the negative dataset.

Diversity. As shown in Figure 10, the Self-BLEU distribution of the negative dataset is highly similar to that of the positive dataset, indicating comparable diversity.

Semantic Embedding. We use CodeBERT (Feng et al., 2020) to compute semantic embeddings for code in both datasets and compare their distributions. As shown in Figure 11, the negative dataset exhibits a distribution that is largely consistent with the original data. This observation indicates that the negative samples are semantically consistent with the original dataset.

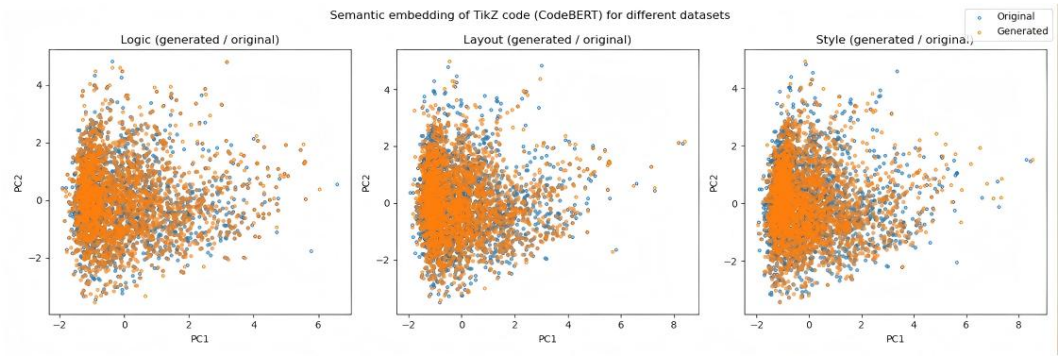


Figure 11: Semantic embedding of TikZ code for positive and negative datasets.

User Prompt:

{code}

- **Based on the above code, generate a detailed and specific textual description of the plotted figure from the perspectives of logic, layout, style, and labels/values.**
- **The description should be as thorough as possible and must include all key details necessary to recreate the figure from scratch.**
- **Do not disclose or reference the plotting code I provided; only return the detailed textual description.**

Figure 12: Prompt for Generating Description.

C Multi-Agent Prompts for Negative Sample Construction

This appendix provides a comprehensive description of the prompt designs used by our Multi-Agent system, including the Describe Agent, Plan Agent, and Edit Agent. These prompts support key tasks such as extracting structured descriptions, generating edit plans, and executing edits, ensuring that each module contributes to accurate construction of the negative dataset. The corresponding prompt designs for these agents are illustrated in Figures 12, 13, 14, and 15.

D Analysis and Statistics of Highly Uncertain Tokens

This threshold is motivated by recent evidence that effective RL updates are concentrated on a small minority of high-entropy tokens, rather than being evenly distributed over the entire sequence. In particular, Wang et al. (2025b) show that the top 20% highest-entropy tokens capture most of the useful optimization signal: they serve as critical branching points in generation, and restricting policy updates to this subset can achieve performance comparable to full-token RL, while optimizing only the low-entropy majority leads to substantial degradation. We therefore set the entropy threshold to 20% to focus learning on the most informative and uncertain tokens. This design is especially suitable for text-to-diagram generation, where a large portion of TikZ tokens are routine syntactic scaffolding, whereas a relatively small subset governs key semantic and structural decisions such as node relations, layout coordinates, and style attributes. Combined with our difference-based filtering, this threshold allows DE-GRPO to concentrate updates on uncertain positions that are more likely to influence diagram semantics or structure.

We analyze generation uncertainty of Qwen2.5-Coder-7B on the SketikZ test set and summarize token-level entropy statistics. Specifically, we feed the test set in JSONL format and build inputs using the same instruction template as in training. During inference, we compute token entropy at

User Prompt:

You are an image-annotation assistant. Below is a graphical description (PROMPT).

Please extract and summarize the **logical structure, components, layout, visual style, and numerical labels from it, and output them in strict JSON format.**
The top-level JSON must contain the following five fields:
****logic, components, layout, style, value_labels****.

Requirements:
1. Output **only JSON, in English, with **no explanatory text**.**
2. The JSON must be valid UTF-8 text.
3. If information for any field is missing, fill it with an empty object {{}} or an empty array [], but the field name must still exist.
4. All mathematical symbols (e.g., \mathbb{{R}}, \pi, 0.\overline{{3}}) must be preserved exactly as they appear in the string.
5. Numerical values must have corresponding information, e.g., node a = 1, node b = 2.

Below is the input graphic description PROMPT:
{prompt}

Figure 13: Prompt for Generating Structured Description.

System Prompt:
You are an IR (Intermediate Representation) editing assistant for graphical structures.

User Prompt:
Your task: Based on the given error_type, read and understand the JSON, think about how to introduce a clear modification in the corresponding section, and express in one or a few concise English sentences what changes you intend to make.

Note: At this step, you only need to produce a verbal description of the modification plan — do NOT output the modified JSON.

Detailed requirements:
1. For each error_type:
You must introduce clear and noticeable modifications in the corresponding section. In your output, use natural language to describe "what it originally was" and "what you intend to change it to."

2. Special constraints for different types:
If error_type is "logic_error": Only modify the existing logic description (e.g., ordering of hierarchy, inclusion structure). Do NOT change logic by adding or removing components. Do NOT change labels or numerical values to alter the logic. Do NOT change layout or visual style to alter the logic.

If error_type is "component_error_add": Add new components to the components field (such as a new ellipse, set label, or graphical element). You may briefly describe the type, hierarchy, color, or purpose of the new component. Do NOT modify the set definitions or relationships in "logic".

If error_type is "component_error_del": Remove an existing component from "components" and specify which one. Do NOT modify the definitions or relationships in "logic".

If error_type is "layout_error": Modify only layout or geometric arrangement, such as orientation, relative position, alignment, spacing, etc. Do NOT add/remove components or change color to achieve the modification.

If error_type is "style_error": Modify only colors, fonts, line thickness, line style (dashed/solid), transparency, etc. Do NOT change geometric layout or logical content. If error_type is "value_label_error": Modify only numerical/symbolic example values in "value_labels". For example: change label names, change sample numbers to incorrect ones, etc. Do NOT change the set structure or components.

3. Output format:
Output only a paragraph of English text, without JSON. Recommended examples: "Change the original logic XXX to YYY." "Add a new ellipse component representing the complex set $\mathbb{C}$ with color …" "Remove the ellipse **component representing** $\mathbb{Q}$ and its label from components." "Change the arrangement of ellipses from diagonal alignment to horizontal alignment." "Modify the sample values of $\mathbb{Q}$ to … which include irrational numbers."

error_type: "{error_type}"

Below is the original IR JSON. Based on error_type, design a modification plan and describe it in English:
{structured description}

Figure 14: Prompt for Generating Change Plan.

each step using the entropy definition in Section 3.3, and record both the occurrence count and the average entropy for each token. Based on these statistics, we select representative tokens from a high-frequency token set. Figures 16(a) and 16(b) show word clouds constructed from the 180 tokens with the highest and lowest average entropy within this set, respectively. In each word cloud, the font size of a token is proportional to its average entropy, where a larger font indicates higher average uncertainty during generation.

System Prompt:

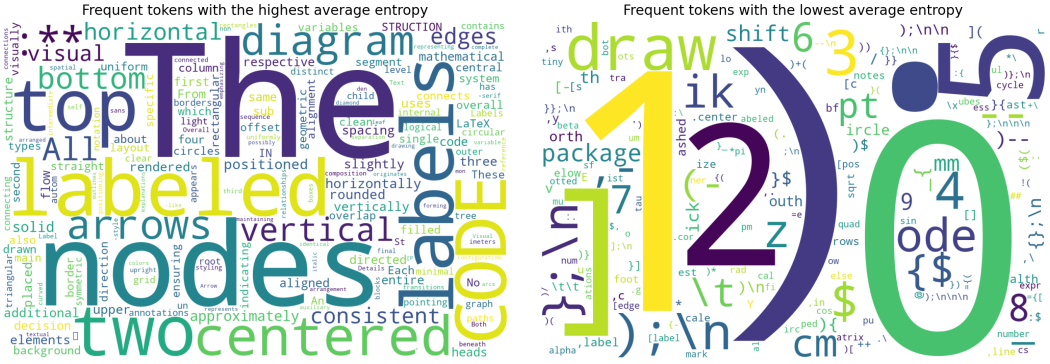
You are a LaTeX/TikZ code-editing assistant.

User Prompt:

Below is a piece of original TikZ code:
{code}
The modification instruction is as follows:
{instruction}

Please modify the above code according to the instruction and output the complete **modified LaTeX code directly.**
Please output **only the code, with no explanatory text.**

Figure 15: Prompt for Generating Corresponding Negative Code.



(a) Highest average entropy.

(b) Lowest average entropy.

Figure 16: Frequent tokens with the highest and lowest average entropy. (a) Tokens with the highest average entropy. (b) Tokens with the lowest average entropy.

E Additional Experiments

E.1 Prompt Details

This appendix provides a detailed description of the prompts used in our experiments, including those for the assessment models, model inference, and the LLM-as-Judge strategy, as illustrated in Figures 17, 18, and 19.

E.2 Implementation Details

Both our assessment models and the policy model are built on Qwen2.5-Coder-7B. For assessment models training, in addition to the input text description, the corresponding positive code, and perspective-specific negative code, we further expand the training data by treating positive code from other samples as additional negatives. In practice, we set this number to 3. We train the assessment models for 10 epochs with a learning rate of 1.5×10^{-4} and a weight decay of 0.01.

For the computation of rewards, we normalize the token-level reward R_{token} and the three quality assessment scores, q_{logic}, q_{layout} and q_{style}, before aggregation. Since R_{token} is computed using Jaccard similarity, it is inherently bounded between 0 and 1. For the three quality assessment scores, we apply a sigmoid transformation to map each score into the same 0-to-1 range. This ensures that different reward components are on a comparable scale and avoids any component dominating the overall reward due to differences in numerical magnitude. The normalized logic, layout, and style scores are subsequently summed to obtain R_{quality}, which is then combined with R_{token} according to the reward formulation in Section 3.2.

System Prompt:

You are a strict code judge.

User Prompt:

Only judge whether there are {view} errors in the code. Ignore all other types of errors.
[PROMPT]
{prompt}
[/PROMPT]

[CODE]
{code}
[/CODE]

[SPECIAL_Token]

Figure 17: Multi-view Assessment Models Prompt Design.

System Prompt:

You are a LaTeX/TikZ code generator.

User Prompt:

Output only the complete drawing code, with no explanations or additional text:

[PROMPT]
{prompt}
[/PROMPT]

Figure 18: Reasoning Prompt Design.

The policy model is trained with 10 epochs of supervised fine-tuning followed by 3 epochs of reinforcement learning. During SFT, we set the learning rate to 1×10^{-4}, the weight decay to 0.01, and the batch size to 8. During RL training, we set the policy model learning rate to 1×10^{-6}, use sampling with a temperature of 1.0, with 16 sampled candidates per prompt for GRPO group construction.

For inference, we use the vLLM framework for acceleration and run the model in FP16. We set the sampling temperature to 0.2, top-p to 0.9, top-k to 50, and the maximum input length to 8192 tokens. In addition, for all baselines and our method, we report the average results over five runs with different random seeds after excluding samples that fail to compile. We further report the detailed repeated-run results of FocusTikZ and Standard GRPO in Table 6, where FocusTikZ outperforms Standard GRPO on nearly all code-level and diagram-level metrics on both SKETIKZ and DATIKZ-V1, while exhibiting relatively small standard deviations across runs.

Baselines. We compare our trained policy model against 15 mainstream language models and several TikZ-specific generation baselines. The code-specialized models include Qwen2.5-Coder (Hui et al., 2024), Code-Llama (Rozière et al., 2023), Qwen3-Coder-Plus (Team, 2025), StarCoder2 (Lozhkov et al., 2024), WizardCoder (Luo et al., 2023), and Yi-Coder (01.AI, 2024). The general-purpose open-source models include Baichuan2-Chat (Yang et al., 2023), InternLM2.5-Chat (Cai et al., 2024), Llama-3.1-Instruct (Team, 2024), Qwen2.5-Instruct (Yang et al., 2024b), Yi-1.5 (Young et al., 2024), GLM-4.7 (Zeng et al., 2024), and DeepSeek-V3 (DeepSeek-AI, 2024), while the closed-source models include Qwen3-Max (Team, 2025) and GPT-5.4 (OpenAI, 2026). For task-specific TikZ generation methods, we compare against the CLiMA variants introduced in AutomaTikZ (Belouadi et al., 2024a), DiagramCodeAgent (Wei et al., 2025), and TikZilla (Greisinger & Eger, 2026). DiagramCodeAgent is built on Qwen2.5-Coder-7B (Hui et al., 2024), while TikZilla is based on Qwen3-8B (Team, 2025).

System Prompt:

You are a strict code judge.

User Prompt:

Given a PROMPT and CODE, output a single digit score in {0,1,2,3,4} that measures how well the CODE satisfies the PROMPT under the error criteria.

Scoring rubric:
4: Fully satisfies the prompt; no relevant errors; meets all key constraints.
3: Mostly satisfies the prompt; minor deviations that do not affect the main goal.
2: Partially satisfies the prompt; clear issues or missing key constraints.
1: Barely satisfies the prompt; only a small portion is relevant.
0: Essentially unrelated to the prompt or clearly does not satisfy it.

[PROMPT]
{prompt}
[/PROMPT]

Anchor example for the SAME PROMPT:
Example (Score=4):
[CODE]
{ground_truth_code}
[/CODE]
[OUTPUT]
4

Now judge the following for the SAME PROMPT. Output only one digit.
[CODE]
{code}
[/CODE]

Figure 19: LLM-as-Judge Prompt Design.

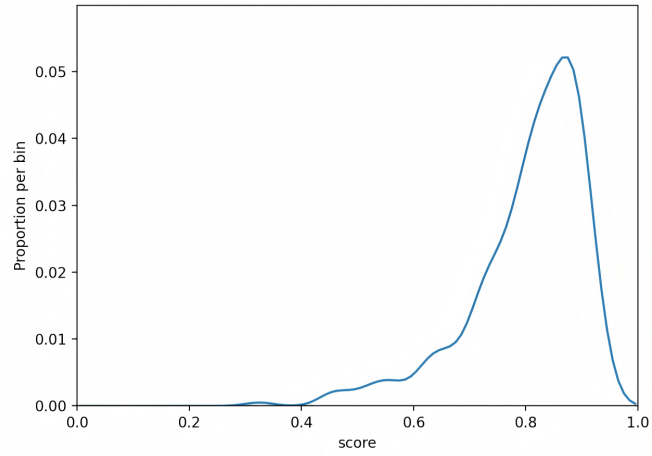


Figure 20: Probability Density of Normalized LLM-as-Judge Scores.

Computational Cost. To provide a comprehensive overview of the resource requirements, we summarize the computational cost for each training stage in Table 7. In our implementation, all trainable modules are optimized using parameter-efficient fine-tuning.

Example of TikZ code

```
\node[input] (input) {Input};
\node[decision, below=2cm of input] (decision) {text?};
\node[process, left=2cm of decision] (process) {text};
\node[output, below=3cm of input] (output) {Output};

\path (input) edge[->] node {no} (decision)
    (decision) edge[->] node {yes} (process)
    (process) edge[->, red] (decision|-input);

\path (decision) edge[->] node {no} (output);
```

Figure 21: Representative TikZ code used to illustrate the tokens selected by DE-GRPO.

Table 6: Repeated-run comparison between FocusTikZ and Standard GRPO on SKETIKZ and DATIKZ-V1.

Model	ROUGE-L↑	EditDist.↓	chrF↑	CrystalBLEU↑	CLIPScore↑	LPIPS↓	PSNR↑	MS-SSIM↑
SKETIKZ								
FocusTikZ	$\mathbf{45.553 \pm 0.638}$	$\mathbf{62.985 \pm 0.720}$	$\mathbf{50.144 \pm 0.471}$	$\mathbf{29.986 \pm 0.453}$	$\mathbf{0.872 \pm 0.007}$	$\mathbf{0.189 \pm 0.005}$	$\mathbf{26.499 \pm 0.171}$	$\mathbf{0.828 \pm 0.005}$
Standard GRPO	43.684 ± 0.596	64.699 ± 0.825	48.661 ± 0.464	25.484 ± 0.483	0.847 ± 0.008	0.212 ± 0.004	25.173 ± 0.212	0.792 ± 0.007
DATIKZ-V1								
FocusTikZ	$\mathbf{43.285 \pm 0.641}$	$\mathbf{65.639 \pm 0.644}$	$\mathbf{46.161 \pm 0.627}$	$\mathbf{17.874 \pm 0.334}$	$\mathbf{0.896 \pm 0.014}$	$\mathbf{0.154 \pm 0.007}$	$\mathbf{27.285 \pm 0.525}$	$\mathbf{0.872 \pm 0.013}$
Standard GRPO	42.507 ± 0.671	66.784 ± 0.663	44.639 ± 0.504	16.605 ± 0.349	0.856 ± 0.017	0.215 ± 0.012	26.159 ± 0.305	0.718 ± 0.012

DE-GRPO Token Selection Details. For each sampled candidate c_i, DE-GRPO constructs the focused token set by intersecting the reference-difference token set T_i^{diff} and the high-entropy token set T_i^{ent}, i.e., $T_i = T_i^{\text{diff}} \cap T_i^{\text{ent}}$. As described in Section 3.3, T_i^{diff} identifies positions where the generated implementation differs from the reference code, while T_i^{ent} captures tokens with high generation uncertainty under the current policy. Their intersection therefore focuses policy updates on positions that are both implementation-sensitive and uncertain.

A boundary case may arise when T_i^{diff} and T_i^{ent} have no overlap. Since the DE-GRPO objective is normalized by $|T_i|$, an empty focused token set would make the objective undefined. To handle this case, our implementation falls back to the union $T_i^{\text{diff}} \cup T_i^{\text{ent}}$, ensuring that the update set remains non-empty.

We use the intersection as the default because it provides a more selective update signal: it filters out reference-different tokens with low uncertainty as well as high-entropy tokens unrelated to meaningful implementation differences. Using the union by default would include a broader set of tokens and may dilute the focused-update effect, which is also consistent with the ablation results in Table 4.

E.3 EVALUATION AND ANALYSIS OF QUALITY ASSESSMENT MODELS

We evaluate the effectiveness of our quality assessment models on two datasets, AugCode and GenCode, containing logic, layout, and style perturbations. Specifically, AugCode contains 300 test samples in total, with 100 samples for each view, while GenCode contains 357 test samples, including 154 logic samples, 162 layout samples, and 41 style samples. To ensure a robust evaluation of our assessment models, we intentionally introduced diversity in the negative-sample construction pipelines. In the pipeline for constructing negative samples for training, the Describe Agent is powered by Qwen2.5-32B, while the Plan and Edit Agents utilize Qwen3-Max. Conversely, when constructing the AugCode evaluation set, we replaced all constituent agent models with GPT-4o (OpenAI, 2024). This heterogeneous design serves to minimize potential model-family and pipeline-specific artifacts, ensuring that the negative samples in our evaluation benchmark are distinct from, and less correlated with, the patterns encountered during the training of our assessment models.

Table 7: Computational cost across different training stages.

Stage	Optimizer / Hyperparameters	GPU Setting	Time
Assessment Models	AdamW, lr = 1.5×10^{-4}	1× RTX PRO 6000 96GB	∼12 hours
SFT Initialization	AdamW, lr = 1.0×10^{-4}, wd = 0.01	1× RTX PRO 6000 96GB	∼6 hours
RL Training	AdamW, lr = 1.0×10^{-6}, temp = 1.0	8× A100 80GB	∼48 hours

Table 8: Accuracy (%) of the base model and our assessment models on *AugCode* and *GenCode*.

Assessment Model	*AugCode*		*GenCode*	
	Base	Ours	Base	Ours
Logic AM	70.00%	**91.00%**	55.19%	**100.00%**
Layout AM	57.00%	**94.00%**	60.49%	**100.00%**
Style AM	53.00%	**100.00%**	46.34%	**100.00%**

Given a sample and its corresponding textual description, we use each assessment model to compute a score, and do the same for the corresponding reference code paired with the same description. If the sample receives a lower score than the reference code, we regard the error as successfully detected. We then compute the error-detection accuracy for each view based on this criterion. The results are shown in Table 8. Our assessment models outperform the base model.

To further analyze the experimental results, we measure the similarity between samples in AugCode and GenCode and their corresponding reference codes, as shown in Table 9. The results indicate that AugCode, generated by our multi-agent augmentation pipeline, is much more similar to the reference code, while GenCode is relatively less similar. This indicates that the errors in AugCode are more subtle, making accurate code-based error detection more challenging for the model.

Furthermore, since the assessment models are trained using synthetic negative samples, it is crucial to verify whether their assigned scores authentically align with actual human judgments, rather than merely capturing artifacts from the synthetic perturbation process. To address this, we conducted an analysis comparing the reward scores against human evaluations. Table 10(a) presents the human scores alongside the corresponding average rewards assigned by our assessment models for outputs generated by various baselines and our FocusTikZ framework.

To further examine whether the learned assessment models align with human-perceived diagram quality, we compute sample-level correlations between human ratings and reward-model scores. As detailed in Table 10(b), the overall reward score exhibits a strong positive correlation with human ratings, achieving a Pearson $r = 0.950$ and a Spearman $\rho = 0.700$. Similarly, the individual assessment model scores also demonstrate consistently positive correlations with human judgments. These findings provide evidence that the learned assessment models capture quality signals that are meaningfully aligned with human-perceived diagram quality, alleviating the concern that they merely exploit artifacts of the synthetic negative construction pipeline and further supporting their use as reward signals during reinforcement learning.

E.4 LLM-AS-JUDGE SCORE STATISTICS

To evaluate the impact of the quality reward, we replace our quality assessment model with an LLM-as-Judge strategy using Qwen2.5-Coder. Following the Beyond Yes and No strategy (Zhuang et al., 2024), the judge uses the ground-truth code as a 4-point anchor to assign each sampled code a discrete score from 0 to 4, where 0 denotes complete irrelevance and 4 denotes a perfect match.

As shown in Figure 4(b), this replacement leads to a clear performance drop. To further understand the cause of the degraded RL performance under the LLM-as-Judge setting, we analyze the distribution of normalized judge scores on Qwen2.5-Coder generations over the SKETIKZ test set. As shown in Figure 20, the score distribution is sharply concentrated around high values. The normalized scores have a mean of 0.804 and a variance of 0.010, with the maximum and minimum being 0.948 and 0.327, respectively. Such a narrow distribution suggests that the judge provides weakly discriminative

Table 9: Code similarity to reference codes on *AugCode* and *GenCode*.

Dataset	ROUGE-L↑	EditDist.↓	chrF↑	CrystalBLEU↑
AugCode	81.222	23.963	86.683	68.415
GenCode	37.566	69.177	42.651	22.622

Table 10: Human evaluation and alignment of assessment model rewards. (a) Comparison of human evaluation scores and assessment model rewards across different models. (b) Correlation between assessment model rewards and human judgments.

(a) Human Scores and Assessment Model Rewards

Model	Human Score ↑	Overall Reward ↑	Logic AM ↑	Layout AM ↑	Style AM ↑
FocusTikZ	3.496 ± 0.225	3.610	0.991	0.879	0.896
Standard GRPO	2.919 ± 0.223	2.669	0.788	0.741	0.672
DeepSeek-V3	3.018 ± 0.348	3.121	0.808	0.815	0.850
Qwen3-Max	3.144 ± 0.309	3.025	0.773	0.796	0.845
Qwen2.5-Coder-7B	2.119 ± 0.282	1.103	0.274	0.221	0.167

(b) Reward–Human Alignment

Reward Type	Pearson r ↑	Spearman ρ ↑
Overall Reward	0.950	0.700
Logic AM	0.908	0.400
Layout AM	0.944	0.700
Style AM	0.972	0.700

reward signals, which limits effective RL optimization and helps explain the inferior performance of LLM-as-Judge.

E.5 Results by Diagram Type

E.5.1 Evaluation Setup

Using the data categorization method described in Appendix B.1, we categorize examples in DaTikZ-V1 and DiagramGenBenchmark according to their diagram types. To construct the textual inputs, we follow protocols similar to those used in DiagramGenBenchmark (Wei et al., 2025) and DaTikZ-V4 (Greisinger & Eger, 2026): a separate description-generation model converts each reference diagram representation into a detailed natural-language description. Importantly, the evaluated models receive only these descriptions and are never exposed to the reference TikZ code used during description construction. This setup provides well-specified textual inputs while preventing direct access to the target programs, allowing us to evaluate diagram generation and cross-type generalization under a controlled setting.

We further divide the examples according to whether their diagram types overlap with those in the SketikZ test set. The corresponding results are reported in Tables 11 and 12 for DaTikZ-V1, and in Tables 13 and 14 for DiagramGenBenchmark.

E.5.2 Analysis

Across DaTikZ-V1 and DiagramGenBenchmark, grouping test cases by whether their diagram types match SketikZ shows a clear trend: same-type cases are easier, while cross-type cases are harder and better reflect out-of-distribution generalization. For our method, switching from cross-type to same-type consistently improves visual fidelity. On DaTikZ-V1, CLIPScore rises from 0.892 to 0.911, LPIPS drops from 0.163 to 0.117, PSNR increases from 26.970 to 28.594, and MS-SSIM increases from 0.866 to 0.896; on DiagramGenBenchmark, CLIPScore rises from 0.845 to 0.869, LPIPS drops from 0.307 to 0.270, and PSNR increases from 22.227 to 24.063. These results indicate that type alignment leads to better visual reconstruction quality. Importantly, our method remains strong under cross-type shifts: on DaTikZ-V1 in the cross-type setting, it achieves the best overall performance, with top ROUGE-L (43.765), lowest EditDist (65.313), highest chrF (46.660), best CLIPScore (0.892), lowest LPIPS (0.163), highest PSNR (26.970), and highest MS-SSIM (0.866). On DiagramGenBenchmark in the cross-type setting, it also leads most metrics: ROUGE-L (41.936), EditDist (66.627), CrystalBLEU (29.398), LPIPS (0.307), PSNR (22.227), MS-SSIM (0.725). Overall, the experiments confirm that our approach provides consistently better perceptual and reconstruction quality, and is robust to diagram-type transfer, demonstrating strong cross-type generalization while further benefiting from in-type alignment.

Table 11: Results on DATIKZ-V1 for examples with diagram types the same as those in SKETIKZ.

Model	Size	ROUGE-L↑	EditDist.↓	chrF↑	CrystalBLEU↑	CLIPScore↑	LPIPS↓	PSNR↑	MS-SSIM↑
Qwen2.5-Coder	7B	35.985	70.487	38.600	5.661	0.820	0.374	18.933	0.655
Code-Llama	34B	36.190	69.998	39.778	12.671	0.813	0.372	17.913	0.657
Qwen3-Coder-Plus	–	36.053	70.251	43.562	17.358	0.802	0.423	17.781	0.604
StarCoder2	15B	36.687	69.473	38.181	8.160	0.821	0.377	18.589	0.647
WizardCoder	15B	31.447	73.581	35.774	12.776	0.875	0.149	25.747	0.875
Yi-Coder	9B	34.351	71.698	37.325	8.197	0.858	0.235	23.200	0.786
Baichuan2-Chat	13B	31.272	74.004	32.118	5.424	0.800	0.394	18.099	0.653
InternLM2.5-Chat	20B	33.602	71.518	38.059	5.923	0.837	0.273	23.822	0.755
Llama-3.1-Instruct	8B	35.594	70.522	39.066	14.335	0.892	0.145	27.259	0.875
Qwen2.5-Instruct	72B	40.931	67.493	41.958	10.323	0.800	0.434	16.586	0.590
Yi-1.5	34B	31.246	73.319	37.068	7.497	0.804	0.348	19.701	0.685
GLM-4.7	355B	34.496	72.391	42.151	13.660	0.797	0.419	17.978	0.614
DeepSeek-V3	671B	38.760	69.156	41.574	10.901	0.807	0.424	17.670	0.608
Qwen3-Max	–	33.739	72.133	41.765	11.698	0.807	0.438	17.992	0.604
DiagramCodeAgent	7B	37.307	69.411	40.955	12.191	0.831	0.302	21.319	0.723
FocusTikZ	7B	**41.236**	**67.044**	**43.983**	**18.499**	**0.911**	**0.117**	**28.594**	**0.896**

Table 12: Results on DATIKZ-V1 for examples with diagram types different from those in SKETIKZ.

Model	Size	ROUGE-L↑	EditDist.↓	chrF↑	CrystalBLEU↑	CLIPScore↑	LPIPS↓	PSNR↑	MS-SSIM↑
Qwen2.5-Coder	7B	37.436	69.935	39.984	10.009	0.837	0.331	20.971	0.711
Code-Llama	34B	37.469	69.934	40.144	19.866	0.821	0.382	18.915	0.653
Qwen3-Coder-Plus	–	38.593	69.555	45.942	18.353	0.810	0.439	17.558	0.600
StarCoder2	15B	37.165	69.889	39.241	9.759	0.829	0.358	19.514	0.670
WizardCoder	15B	31.019	73.643	37.911	5.812	0.855	0.189	24.932	0.844
Yi-Coder	9B	33.079	71.969	36.765	11.547	0.835	0.270	22.989	0.763
Baichuan2-Chat	13B	32.184	73.296	33.960	10.364	0.793	0.403	17.921	0.643
InternLM2.5-Chat	20B	37.444	70.597	41.857	16.727	0.851	0.243	24.053	0.793
Llama-3.1-Instruct	8B	36.309	70.592	38.692	11.365	0.870	0.188	26.059	0.849
Qwen2.5-Instruct	72B	41.910	67.135	43.976	19.610	0.814	0.431	17.737	0.608
Yi-1.5	34B	31.683	73.418	38.347	7.976	0.804	0.382	19.356	0.655
GLM-4.7	355B	37.782	69.962	45.649	**25.736**	0.815	0.418	17.829	0.617
DeepSeek-V3	671B	40.937	68.196	44.744	14.291	0.813	0.437	17.742	0.603
Qwen3-Max	–	37.099	70.469	44.959	21.409	0.828	0.436	18.263	0.609
DiagramCodeAgent	7B	38.687	69.278	43.079	13.619	0.838	0.294	22.016	0.736
FocusTikZ	7B	**43.765**	**65.313**	**46.660**	17.588	**0.892**	**0.163**	**26.970**	**0.866**

E.6 QUALITATIVE EXAMPLES OF SELECTED KEY TOKENS

To provide an intuitive understanding of the positions selected by DE-GRPO, we inspect a representative flowchart generated during training. As shown in Figure 21, the generated TikZ code contains four nodes representing an input, a decision, a process, and an output, together with directed branches connecting these components.

For this example, DE-GRPO selects tokens such as `input`, `decision`, `process`, `output`, `below=2cm`, `left=2cm`, `edge[->]`, `yes`, `no`, and `red`. These selected tokens span several aspects that influence the structure and appearance of the rendered diagram. The tokens `input`, `decision`, `process`, and `output` specify the semantic roles and node types of the flowchart components. Spatial attributes such as `below=2cm` and `left=2cm` determine their relative layout. The token `edge[->]` specifies directed connectivity, while `yes` and `no` encode the branch semantics associated with the decision node. The token `red` controls a visual attribute of one transition. Changes to these tokens can therefore alter the semantics, topology, spatial organization, or visual appearance of the rendered diagram. In contrast, routine procedural tokens, such as `\node`, `\path`, braces, and semicolons, are not selected in this example. Although these tokens are required for valid TikZ syntax, many of them follow common and predictable code patterns and provide limited information for distinguishing alternative diagram structures.

This example illustrates why DE-GRPO uses the intersection of difference and high-entropy tokens. Difference tokens identify candidate positions where the generated implementation deviates from the reference, while high-entropy tokens capture uncertain decisions. Their intersection filters out

Table 13: Results on DIAGRAMGENBENCHMARK for examples with diagram types the same as those in SKETIKZ.

Model	Size	ROUGE-L↑	EditDist.↓	chrF↑	CrystalBLEU↑	CLIPScore↑	LPIPS↓	PSNR↑	MS-SSIM↑
Qwen2.5-Coder	7B	37.184	68.927	42.683	21.977	0.839	0.386	19.011	0.597
Code-Llama	34B	35.946	69.335	44.783	22.345	0.834	0.448	15.956	0.517
Qwen3-Coder-Plus	–	35.940	68.932	48.753	21.774	0.830	0.448	15.289	0.508
StarCoder2	15B	38.303	67.560	44.514	22.398	0.840	0.420	17.622	0.558
WizardCoder	15B	29.069	73.701	38.923	16.559	0.789	0.357	20.474	0.674
Yi-Coder	9B	33.200	71.450	38.990	15.297	0.828	0.345	20.430	0.660
Baichuan2-Chat	13B	30.724	72.046	36.199	12.920	0.767	0.454	15.609	0.548
InternLM2.5-Chat	20B	35.778	69.462	43.482	20.847	0.834	0.344	20.788	0.646
Llama-3.1-Instruct	8B	35.240	70.350	41.570	19.371	0.857	0.290	23.919	0.725
Qwen2.5-Instruct	72B	**42.793**	66.719	48.692	**27.320**	0.830	0.452	15.781	0.513
Yi-1.5	34B	32.460	71.880	40.490	18.138	0.812	0.421	17.298	0.553
GLM-4.7	355B	36.352	70.415	**49.321**	24.091	0.821	0.431	15.767	0.536
DeepSeek-V3	671B	39.596	68.125	48.206	24.219	0.824	0.447	15.577	0.520
Qwen3-Max	–	33.230	70.630	47.061	19.273	0.851	0.463	15.222	0.474
DiagramCodeAgent	7B	39.237	68.151	47.512	21.225	0.837	0.415	17.483	0.575
FocusTikZ	7B	41.705	**66.134**	48.285	27.261	**0.869**	**0.270**	**24.063**	**0.735**

Table 14: Results on DIAGRAMGENBENCHMARK for examples with diagram types different from those in SKETIKZ.

Model	Size	ROUGE-L↑	EditDist.↓	chrF↑	CrystalBLEU↑	CLIPScore↑	LPIPS↓	PSNR↑	MS-SSIM↑
Qwen2.5-Coder	7B	36.817	69.968	43.290	26.321	0.827	0.428	17.611	0.592
Code-Llama	34B	36.998	70.330	43.705	22.338	0.828	0.437	16.782	0.565
Qwen3-Coder-Plus	–	36.560	70.836	49.213	24.630	**0.863**	0.428	16.326	0.561
StarCoder2	15B	36.499	69.679	43.369	25.401	0.839	0.410	17.942	0.597
WizardCoder	15B	31.136	73.345	39.064	20.567	0.803	0.336	21.290	0.709
Yi-Coder	9B	32.088	72.972	40.131	19.725	0.793	0.410	18.033	0.615
Baichuan2-Chat	13B	30.972	73.827	37.362	15.994	0.792	0.461	16.774	0.562
InternLM2.5-Chat	20B	34.943	71.290	44.471	22.186	0.823	0.376	19.901	0.658
Llama-3.1-Instruct	8B	32.897	72.687	40.473	19.499	0.818	0.320	21.832	0.718
Qwen2.5-Instruct	72B	40.354	67.708	**49.251**	28.394	0.853	0.432	15.982	0.555
Yi-1.5	34B	33.414	71.046	42.831	21.277	0.822	0.452	16.872	0.558
GLM-4.7	355B	34.780	71.686	48.471	24.100	0.857	0.416	16.910	0.569
DeepSeek-V3	671B	39.050	68.852	48.343	28.342	0.850	0.440	16.428	0.551
Qwen3-Max	–	35.801	70.885	49.057	22.618	0.858	0.425	16.497	0.559
DiagramCodeAgent	7B	38.577	68.433	47.719	23.356	0.847	0.370	19.169	0.633
FocusTikZ	7B	**41.936**	**66.627**	48.003	**29.398**	0.845	**0.307**	**22.227**	**0.725**

many routine code tokens and focuses policy updates on informative positions that affect component semantics, layout, connectivity, branch labels, and visual attributes.

Importantly, reference-different tokens should not be interpreted as erroneous tokens. T_i^{diff} only identifies candidate decision positions where the generated implementation deviates from the reference implementation; it does not determine the correctness or the update direction of these deviations. Whether a selected token is reinforced or suppressed is determined by the trajectory-level advantage derived from R_{overall}. Therefore, a valid alternative TikZ implementation that differs from the reference can still receive a positive advantage, in which case its selected tokens receive a positive optimization signal rather than being penalized for differing from the reference. In this sense, the reference serves as a localization prior for focused credit assignment rather than a token-level correctness target.

E.7 METRICS DETAILS

We report both text-level and image-level metrics. For text metrics, we compare the reference TikZ code with the generated code. For image metrics, we first render both codes into images and then compare the rendered outputs.

Table 15: Full results on the SKETIKZ dataset.

Model	Size	ROUGE-L↑	EditDist.↓	chrF↑	C-BLEU↑	CLIPScore↑	LPIPS↓	PSNR↑	MS-SSIM↑	CSR↑
Coder Large Language Models										
Qwen2.5-Coder	7B	39.438	67.608	44.042	24.062	0.816	0.361	20.269	0.676	0.807
Code-Llama	34B	38.592	68.509	44.328	24.913	0.805	0.412	17.562	0.612	0.370
StarCoder2	15B	40.543	67.357	43.802	23.647	0.812	0.379	19.368	0.620	0.582
WizardCoder	15B	31.540	72.911	40.408	18.318	0.823	0.232	24.508	0.796	0.487
Yi-Coder	9B	35.893	70.201	41.995	19.938	0.816	0.308	21.912	0.706	0.205
Chat Large Language Models										
Baichuan2-Chat	13B	33.702	71.176	36.297	16.375	0.780	0.407	17.421	0.586	0.535
InternLM2.5-Chat	20B	38.196	69.164	45.521	23.835	0.804	0.324	21.087	0.683	0.766
Yi-1.5	34B	34.683	71.087	43.168	22.450	0.804	0.406	18.967	0.627	0.454
Llama-3.1-Instruct	8B	34.152	70.902	41.244	19.418	0.856	0.211	25.976	0.814	0.656
Qwen2.5-Instruct	72B	43.662	65.385	47.959	29.458	0.810	0.432	17.046	0.573	0.924
Commercial Large Language Models										
GLM-4.7	355B	38.157	68.651	48.843	25.709	0.819	0.423	18.085	0.603	0.849
DeepSeek-V3	671B	42.384	66.305	47.740	29.698	0.810	0.428	17.390	0.574	0.929
Qwen3-Max	–	39.214	67.410	49.793	25.832	0.829	0.431	17.815	0.579	0.816
GPT-5.4	–	43.490	63.362	49.609	26.374	0.837	0.375	20.815	0.641	0.951
TikZ-specific Large Language Models										
CLiMA-7B	7B	35.165	69.827	42.482	18.078	0.709	0.512	19.192	0.519	0.323
CLiMA-13B	13B	37.953	69.587	41.204	20.982	0.847	0.376	19.805	0.570	0.471
DiagramCodeAgent	7B	38.682	68.482	46.279	21.027	0.842	0.337	21.417	0.684	0.753
TikZilla	8B	43.847	63.951	48.670	27.545	0.862	0.266	25.309	0.817	0.910
FocusTikZ	**7B**	**45.553**	**62.985**	**50.144**	**29.986**	**0.872**	**0.189**	**26.499**	**0.828**	**0.963**

Table 16: Full results on the DIAGRAMGENBENCHMARK dataset.

Model	Size	ROUGE-L↑	EditDist.↓	chrF↑	C-BLEU↑	CLIPScore↑	LPIPS↓	PSNR↑	MS-SSIM↑	CSR↑
Coder Large Language Models										
Qwen2.5-Coder	7B	36.903	69.725	43.148	25.265	0.830	0.418	17.939	0.593	0.776
Code-Llama	34B	36.734	70.080	43.976	22.342	0.830	0.440	16.574	0.553	0.364
Qwen3-Coder-Plus	–	36.421	70.411	49.110	23.936	0.856	0.432	16.094	0.550	0.481
StarCoder2	15B	36.910	69.196	43.629	24.639	0.840	0.412	17.869	0.588	0.569
WizardCoder	15B	30.508	73.453	39.021	19.325	0.799	0.342	21.042	0.699	0.379
Yi-Coder	9B	32.344	72.621	39.869	18.737	0.801	0.395	18.585	0.625	0.373
Chat Large Language Models										
Baichuan2-Chat	13B	30.913	73.401	37.084	15.220	0.786	0.459	16.495	0.559	0.455
InternLM2.5-Chat	20B	35.121	70.901	44.260	22.635	0.825	0.369	20.090	0.656	0.710
Llama-3.1-Instruct	8B	33.453	72.132	40.733	19.469	0.827	0.313	22.327	0.720	0.702
Qwen2.5-Instruct	72B	40.884	67.493	49.129	28.151	0.848	0.436	15.938	0.545	0.946
Yi-1.5	34B	33.183	71.249	42.262	20.886	0.819	0.444	16.975	0.557	0.304
Commercial Large Language Models										
GLM-4.7	355B	35.138	71.397	48.665	24.099	0.849	0.420	16.650	0.561	0.835
DeepSeek-V3	671B	39.176	68.684	48.312	27.269	0.844	0.442	16.231	0.544	0.933
Qwen3-Max	–	35.278	70.834	48.651	21.810	**0.857**	0.433	16.238	0.542	0.830
GPT-5.4	–	40.571	66.831	**49.418**	27.594	0.853	0.383	18.458	0.722	0.947
TikZ-specific Large Language Models										
CLiMA-7B	7B	34.372	70.236	41.869	18.826	0.679	0.472	16.795	0.572	0.340
CLiMA-13B	13B	37.458	69.785	44.497	17.582	0.791	0.458	17.815	0.596	0.395
DiagramCodeAgent	7B	38.721	68.372	47.673	22.843	0.845	0.379	18.800	0.620	0.813
TikZilla	8B	39.751	68.144	46.490	24.072	0.758	0.327	20.070	0.633	0.895
FocusTikZ	**7B**	**41.887**	**66.522**	48.063	**28.938**	0.850	**0.300**	**22.617**	**0.727**	**0.955**

ROUGE-L. ROUGE-L measures sequence-level overlap based on the longest common subsequence between the generated code and the reference code. We report the average ROUGE-L F1 score, scaled to 0–100.

Edit Distance. Edit Distance measures the minimum number of character-level insertions, deletions, and substitutions required to transform the generated code into the reference code. We compute the Levenshtein distance, normalize it by the length of the longer string, and report the average value on a 0–100 scale.

chrF. chrF measures character n-gram overlap between the generated code and the reference code. We use SacreBLEU's sentence-level implementation, average the scores across samples, and scale the result to 0–100.

CrystalBLEU. CrystalBLEU measures n-gram overlap while discounting trivial and overly frequent patterns. We compute it on the core TikZ content after removing comments and extracting the content inside the `tikzpicture` environment (or the document body when applicable). Following the standard setup, we build an ignoring list from the training set using the top 1000 most frequent n-grams up to 4-grams, and ignore these n-grams during evaluation. The final score is reported on a 0–100 scale.

CLIPScore$_{\text{IMG}}$. CLIPScore$_{\text{IMG}}$ measures semantic similarity between the rendered generated image and the rendered reference image using CLIP image embeddings. We resize images to 224×224, apply CLIP preprocessing, compute the cosine similarity between normalized image embeddings, and report the average score across samples.

LPIPS. LPIPS measures perceptual dissimilarity between the rendered generated image and the rendered reference image. We use the AlexNet backbone and report the average LPIPS score across samples.

PSNR. PSNR measures pixel-level reconstruction fidelity between the rendered generated image and the rendered reference image, with higher values indicating smaller distortion. We compute PSNR on images with pixel values in the [0,1] range and report the average score across samples.

MS-SSIM. MS-SSIM measures structural similarity between the rendered generated image and the rendered reference image across multiple scales. We report the average MS-SSIM score across samples.

Compilation Success Rate. Compilation Success Rate (CSR) measures the executability of the generated TikZ code. It is computed as the proportion of sampled generations that can be successfully compiled, where the numerator is the number of successfully compiled samples and the denominator is the total number of sampled generations.

F Generalization to Other Diagramming Languages

While our empirical evaluation primarily focuses on TikZ, we recognize that other diagramming formats, such as SVG, Mermaid, and Graphviz, are more prevalent in general industry. A natural question arises: can the proposed FocusTikZ framework be adapted to these more widely used languages?

We argue that although our current implementation is built upon the text-to-TikZ setting, the core methodologies of FocusTikZ, namely multi-reward supervision and token-focused policy updates, are independent of specific languages. The logic, layout, and style assessment models take the natural language description and the raw generated code as inputs to produce task-oriented quality scores. In principle, this framework can be re-instantiated for other diagram languages by constructing corresponding positive examples and language-specific negative code pairs. Our Difference-and-Entropy Focused GRPO identifies key update positions through two universal probabilistic signals: reference-difference tokens and high-entropy tokens. Therefore, the focused-update mechanism is applicable to autoregressive diagram-code generation tasks.

Despite the conceptual generality, we acknowledge that adapting FocusTikZ to SVG, Mermaid, or Graphviz still necessitates substantial language-specific engineering. We leave the extension of FocusTikZ to these broader diagramming languages as an exciting avenue for future work.