BREAKING THE SFT PLATEAU: MULTIMODAL STRUCTURED REINFORCEMENT LEARNING FOR CHART-TO-CODE GENERATION

Lei Chen **Xuanle Zhao** **Zhixiong Zeng**[†]
Jing Huang **Liming Zheng** **Yufeng Zhong** **Lin Ma**[*]
Meituan
zengzhixiong@meituan.com forest.linma@gmail.com
Project: https://github.com/DocTron-hub/MSRL

ABSTRACT

While reinforcement learning (RL) has proven highly effective for general reasoning in vision-language models, its application to tasks requiring deep understanding of information-rich images and structured output generation remains underexplored. Chart-to-code generation exemplifies this challenge, demanding complex reasoning over visual charts to produce structured code. Supervised fine-tuning (SFT) alone is often insufficient, highlighting the need for effective RL strategies tailored to structured outputs. In this paper, we systematically investigate the performance plateau of SFT through large-scale experiments and propose Multimodal Structured Reinforcement Learning (MSRL) for chart-to-code generation. We construct the largest training corpus to date, with 3 million chart-code pairs curated from real-world tables in arXiv papers, addressing the limitations of previous synthetic datasets. Despite achieving state-of-the-art performance, our experiments show that simply increasing SFT data eventually leads to diminishing improvements. To break this plateau, MSRL employs a multi-granularity reward system that integrates both textual and visual feedback. At the textual level, rule-based rewards validate fine-grained code details, while at the visual level, a model-based reward assesses the structural similarity between rendered code and ground-truth charts. We implement a two-stage curriculum training strategy, first optimizing the model with textual rewards and then incorporating visual signals for further enhancement. Experimental results demonstrate that MSRL substantially breaks the SFT plateau, improving high-level metrics by 6.2% and 9.9% on ChartMimic and ReachQA benchmarks, respectively. Notably, our method outperforms all existing approaches in the chart domain and achieves competitive results with advanced closed-source models.

1 INTRODUCTION

Large language models (LLMs) have demonstrated impressive reasoning capabilities on complex textual problems, including code generation and mathematical problem-solving OpenAI (2025b); Guo et al. (2025). Pivotal to this success is the implementation of reinforcement learning (RL) to optimize the capacities of LLM to generate step-by-step reasoning Wei et al. (2022) and verify final answers Shao et al. (2024). While multimodal large language models (MLLMs) have achieved impressive performance on established visual reasoning tasks like visual question answering Zhang et al. (2024a); Huang et al. (2025), their core reasoning capabilities remain limited, especially when tasks require processing information-dense images like charts Xu et al. (2024); Chen et al. (2025a), and generating structured outputs like code Xia et al. (2024); Zhao et al. (2025b).

Among these challenges, chart-to-code generation Yang et al. (2024a); Zhang et al. (2024c) stands out as a task of both high complexity and significant practical value. Charts are widely used to communicate scientific findings, business insights, and statistical data, yet their information is often

[†] Project leader. [*] Corresponding author.

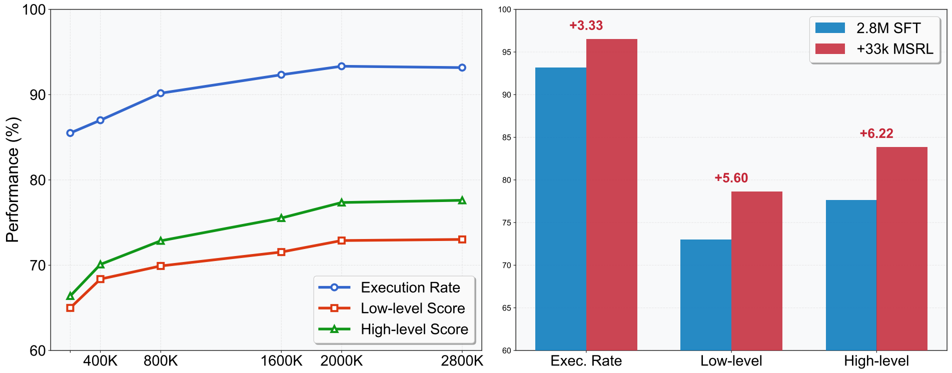


Figure 1: The SFT plateau and RL performance gain from our experiments. The left figure illustrates that scaling SFT data from 200k to 2.8M leads to a performance plateau after exceeding 2M data points. The right figure denotes the performance gain from our proposed MSRL training strategy.

locked in visual formats that are difficult for automated systems to interpret or repurpose. Enabling MLLMs to accurately convert charts into executable code not only advances their visual understanding and structured reasoning capabilities, but also opens up impactful applications, such as assisting researchers in automating the reproduction of scientific visualizations. Although previous works have attempted to solve this task by using methods like Supervised Fine-Tuning (SFT) Zhang et al. (2024b); Zhao et al. (2025b) or Direct Preference Optimization (DPO) Rafailov et al. (2023); Zhang et al. (2025) on synthetic data, their performance remains limited due to the simplistic patterns of the corpus and ineffective generalization strategies. Therefore, a promising direction is to leverage more realistic data and robust strategies to enhance model performance.

In this work, we first systematically investigate the SFT performance plateau through experiments at various data scales. To enable this analysis, we construct the largest training corpus to date for chart-to-code generation, covering 24 distinct chart types and comprising 3 million chart-code pairs. Our two-stage data pipeline curates high-quality tables from arXiv papers and then utilizes LLMs to generate diverse and complex plot codes, ensuring realistic distributions and avoiding monotonous trends. Unlike prior works Ni et al. (2025); Chen et al. (2025b) that apply RL directly after SFT, we establish saturated SFT baselines to isolate the true effectiveness of our proposed multimodal structured reinforcement learning (MSRL) strategy. Through experiments with varying SFT data scales, we demonstrate that simply increasing the volume of SFT data leads to a performance plateau that cannot be overcome by data quantity alone. To break through this plateau, we propose the MSRL training strategy, which incorporates multi-granularity reward functions from both textual and visual aspects. For textual rewards, we standardize code formats in the training corpus and design a comprehensive reward function that verifies code correctness from five key perspectives. Recognizing that text-based rewards alone may overlook overall chart structure, we introduce a visual reward mechanism: generated code is rendered into images and an MLLM evaluates their similarity to the original input chart. By jointly leveraging these multi-granularity rewards, MSRL optimizes both global chart context and fine-grained code details. We evaluate models on various chart-to-code benchmarks Yang et al. (2024a); He et al. (2024). The results demonstrate that it establishes a new state-of-the-art, as it outperforms all previous open-source models and rivals advanced proprietary MLLMs. In summary, the main contributions of this work are as follows:

- We propose an MSRL training strategy that employs a multi-level reward function, which combines a rule-based component for assessing textual code correctness with a model-based evaluator for evaluating the visual fidelity of the rendered chart. Benefits from the MSRL training strategy, our proposed model achieves state-of-the-art (SOTA) performance compared to all other open-source MLLMs.
- We construct the first large-scale chart-to-code training corpus, generated using real-world tables as a data source. After applying several filtering methods, we curate a final dataset of 2.8 million samples for SFT and 33 thousand for RL.
- Through SFT experiments at various data scales, we identify the performance bottleneck, confirming that merely increasing SFT data quantity is insufficient to improve performance.

2 Related Work

2.1 Chart MLLMs

Chart understanding is a crucial research area encompassing both low-level tasks, such as data extraction Liu et al. (2023), and high-level tasks like question answering Masry et al. (2022) and summarization Kantharaj et al. (2022). Recent work has focused on training MLLMs on extensive, chart-specific datasets to enhance their understanding capabilities Xia et al. (2024); Zhang et al. (2024b); Xia et al. (2023), leading to superior performance across various chart-related tasks. For example, ChartMoE Xu et al. (2024) proposes a mixture-of-experts (MoE) structure for multi-task pretraining. More recently, leveraging reinforcement learning Chen et al. (2025a); Masry et al. (2025); Rodriguez et al. (2025b) to enhance reasoning capacities has garnered significant interest. Both Chart-R1 and BigCharts-R1 optimize MLLMs using Chain-of-Thought (CoT) Wei et al. (2022) reasoning data and reinforcement learning with verified rewards (RLVR) Shao et al. (2024).

2.2 Reasoning MLLMs

The success of large reasoning models (LRMs) like DeepSeek-R1 Guo et al. (2025) spurs significant research into enhancing LLM reasoning through reinforcement learning with rule-based rewards Shao et al. (2024). This paradigm extends to the vision-language domain, where numerous works aim to improve the long-chain reasoning capabilities of MLLMs Shen et al. (2025); Wang et al. (2025b). Following this trend, initial efforts such as Vision-R1 Huang et al. (2025) and R1-OneVision Yang et al. (2025b) adapt the Group Relative Policy Optimization (GRPO) algorithm with multimodal reasoning data to enable long-form reasoning in VLMs. Subsequent works, including MMEureka Meng et al. (2025) and R1-Zero Liu et al. (2025), further advance this area by introducing improved reinforcement learning strategies for visual long-term reasoning. Recently, approaches like Point-RFT Ni et al. (2025) utilize task-specific thinking frameworks to enhance reasoning capabilities further.

2.3 MLLMs For Code

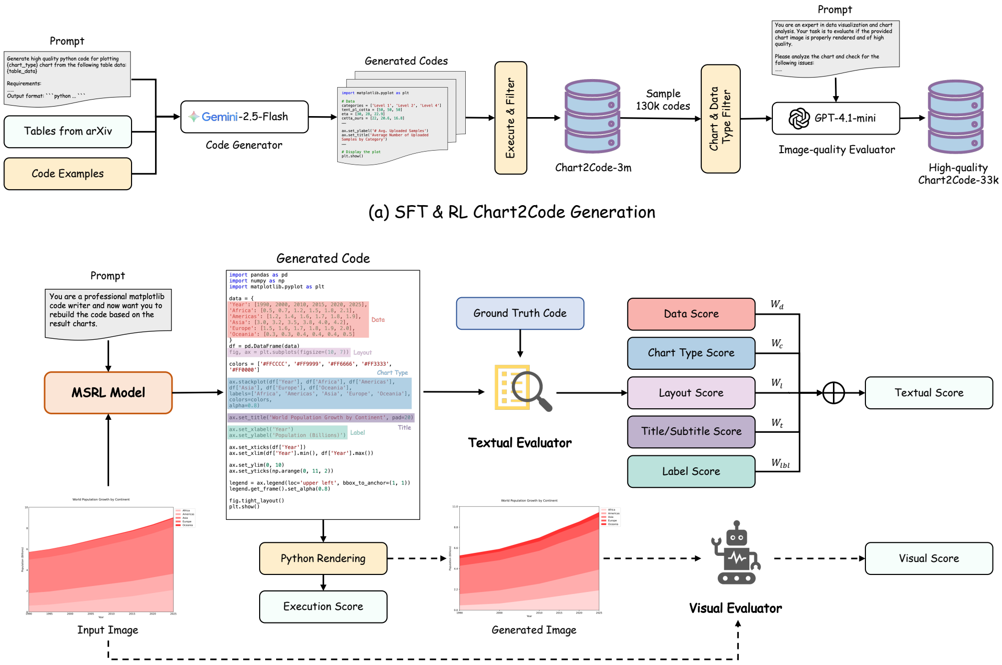
In the rapidly advancing research area of multimodal code generation, foundational work has focused on establishing benchmarks to assess the capabilities of MLLMs. These benchmarks evaluate performance on tasks such as generating execution code for solving visual problems Li et al. (2024); Wang et al. (2025a); Rodriguez et al. (2025a; 2024); Yang et al. (2024b) and HTML code from webpage screenshots Si et al. (2024); Yun et al. (2024); Xiao et al. (2024); Awal et al. (2025). Among these emerging tasks, chart-to-code generation has attracted significant interest due to the complexity of its visual inputs. This task challenges an MLLM to generate plotting code that accurately reproduces the chart image. Recently, several benchmarks have been proposed to evaluate MLLMs in this context, assessing a range of capabilities. For instance, benchmarks such as ChartMimic Yang et al. (2024a), Plot2Code Wu et al. (2025), and ChartX Xia et al. (2024) assess chart-to-code generation capabilities, while others like ChartEdit Zhao et al. (2025a) and ChartM^3 Yang et al. (2025a) evaluate the editing functionalities.

3 Methods

3.1 Training Corpus Construction

While several chart-to-code generation datasets have been proposed, they are generally constrained by a reliance on purely synthetic data and limited scale, resulting in charts that exhibit simplistic trends and lack diversity. To overcome these challenges, we construct our dataset by crawling real-world tables from arXiv papers and then leveraging LLMs to generate the corresponding plotting code. To ensure separation from the ChartMimic benchmark, we exclusively use papers published in 2023 and earlier as our corpus. We ensure data diversity and quality through several strategies, such as using high-quality seed code and filtering out non-executable code.

Through this process, we construct the largest training corpus to date, comprising 3 million chart-code pairs, to investigate the performance limits of SFT. However, previous research Chen et al.



Generate high quality python code for plotting {chart_type} chart from the following table data: {table_data}

Requirements:
......
Output format: ```python ... ```

```
import matplotlib.pyplot as plt

# Data
categories = ['Level 1', 'Level 2', 'Level 4']
tent_pl_cotta = [50, 50, 50]
eta = [30, 28, 22.9]
cetta_ours = [22, 20.6, 16.8]
---

ax.set_ylabel('# Avg. Uploaded Samples')
ax.set_title('Average Number of Uploaded
Samples by Category')
---

# Display the plot
plt.show()
```

You are an expert in data visualization and chart analysis. Your task is to evaluate if the provided chart image is properly rendered and of high quality.

Please analyze the chart and check for the following issues:
......

(a) SFT & RL Chart2Code Generation

You are a professional matplotlib code writer and now want you to rebuild the code based on the result charts.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = {
'Year': [1990, 2000, 2010, 2015, 2020, 2025],
'Africa': [0.5, 0.7, 1.2, 1.5, 1.8, 2.1],
'Americas': [1.2, 1.4, 1.6, 1.7, 1.8, 1.9],
'Asia': [3.0, 3.2, 3.5, 3.8, 4.0, 4.2],
'Europe': [1.5, 1.6, 1.7, 1.8, 1.9, 2.0],
'Oceania': [0.3, 0.3, 0.4, 0.4, 0.4, 0.5]
}
df = pd.DataFrame(data)
fig, ax = plt.subplots(figsize=(10, 7))

colors = ['#FFCCCC', '#FF9999', '#FF6666', '#FF3333',
'#FF0000']

ax.stackplot(df['Year'], df['Africa'], df['Americas'],
df['Asia'], df['Europe'], df['Oceania'],
labels=['Africa', 'Americas', 'Asia', 'Europe', 'Oceania'],
colors=colors,
alpha=0.8)

ax.set_title('World Population Growth by Continent', pad=20)

ax.set_xlabel('Year')
ax.set_ylabel('Population (Billions)')

ax.set_xticks(df['Year'])
ax.set_xlim(df['Year'].min(), df['Year'].max())

ax.set_ylim(0, 10)
ax.set_yticks(np.arange(0, 11, 2))

legend = ax.legend(loc='upper left', bbox_to_anchor=(1, 1))
legend.get_frame().set_alpha(0.8)

fig.tight_layout()
plt.show()
```

(b) Our MSRL Framework

Figure 2: The data generation pipeline and our proposed MSRL framework. (a) Our pipeline prompts Gemini-2.5-Flash with tables from arXiv papers and example codes to generate plotting code. After execution and filtering, the chart-to-code dataset of 3M pairs is obtained. We then sample 130k of these pairs and apply three-stage filters (chart, data, vision) to curate a final, high-quality dataset of 33k examples for RL. (b) The framework of our proposed MSRL strategy. The textual reward is derived from a rule-based evaluation of the generated code across five distinct dimensions. An MLLM quantifies the visual reward based on the rendered image.

(2025a); Huang et al. (2025) shows that separating the SFT and RL training corpora prevents overfitting to the SFT data format and enhances the exploration capacity of RL. To curate a high-quality dataset for RL, we begin with a candidate corpus of 130 thousand samples from the overall training corpus. These candidate data samples are then subjected to a two-stage filtering strategy. The first filtering stage focuses on the code content, filtering based on chart and data type to ensure code diversity and that the output code is suitable for data extraction. Specifically, the chart types are filtered according to plot functions. To handle complex charts containing multiple chart types, we employ a tree-structured parsing method to identify each type present progressively. The data values are filtered according to the data definition format, retaining only data structured as one-dimensional arrays and non-nested dictionaries to facilitate extraction during the RL stage. This step narrows the candidate corpus to 45 thousand samples. The second stage addresses visual quality, employing GPT-4.1-mini OpenAI (2025a) as a visual LLM judge to filter for high-fidelity chart images. To mitigate potential bias from the MLLM judger, we manually inspect 100 samples that are assigned a score of 0 by the model, and observe over 90% consistency with human judgment. This process yields the final RL training corpus of 33 thousand high-quality samples. The remaining samples are utilized for SFT. The comparisons of our dataset with relevant chart-to-code datasets are listed in Table 1. Our dataset features a completely realistic data source, more challenging multi-chart content, and richer visual diversity enabled by more API types.

3.2 Supervised Fine-tuning

Firstly, we perform SFT on Qwen2.5-VL as the base model using our generated chart-to-code dataset. To explore the effectiveness of data scaling and identify the SFT plateau, we fine-tune our model on six distinct training corpus sizes, created by partitioning our main dataset into subsets of 200k, 400k, 800k, 1.6M, 2M and 2.8M samples. The model is trained by minimizing the standard autoregressive objective, which is the negative log-likelihood of the target sequence. As

Table 1: Comparison of existing chart-to-code generation datasets. ✔✗ indicates that ChartMoE uses both data from real CSV tables and data generated by LLMs. ”API types” refers to the number of different Matplotlib API types covered by the dataset.

Dataset	**Realistic data source**	**Multi-chart**	**Chart types**	**Data samples**	**API types**
ChartLlama	✗	✗	10	11k	83
ChartMoE	✔✗	✗	<20	800k	-
Chart2Code	✗	✔	15	3k	168
ChartCoder	✗	✗	27	115k	187
Ours	✔	✔	24	3M	1,555

demonstrated in Section 4.2 and Figure 1, merely scaling the SFT data volume ultimately results in a performance plateau. Increasing the SFT dataset from 2M to 2.8M samples results in no further performance gains.

3.3 Multimodal Structured Reinforcement Learning

Although SFT can significantly enhance the chart-to-code capabilities of MLLMs, our experiments reveal that this approach ultimately reaches a performance plateau, likely due to the limitation that SFT restricts the model's capacity to restore detailed information. This limitation stems from the core drawback of the SFT objective that it treats every token in the target sequence with uniform importance. Therefore, SFT is ineffective in optimizing plotting code generation, as the plotting code comprises largely of boilerplate snippets such as `plt.plot`, with critical information like specific data values or styling parameters appearing infrequently. Consequently, employing RL with a reward function designed to prioritize the accuracy of such critical content presents a promising approach for enhancing code generation capacity. To this end, we propose a two-stage RL training strategy utilizing multi-granularity reward functions that combine both textual and visual feedback.

Textual Reward While the RLVR method provides a powerful mechanism to guide models by measuring the correctness of detailed information, its practical application is challenged by the stylistic diversity of generated code, particularly in data definitions and function calls. This stylistic heterogeneity leads to difficulty in comprehensively extracting and parsing the critical information for reward computation. To mitigate this, we introduce a code normalization step that maps each generated output to a canonical representation before calculating the reward. This process ensures the textual reward function is invariant to syntactic variations, providing a more suitable method for RLVR. Specifically, the textual reward is a granular rule-based accuracy score, which is a weighted average derived from evaluating key aspects of the generated code. We assess data values using soft value matching, chart types with hard string matching, layout with hard value matching and elements like titles and labels via edit distance. The soft value matching allows a relative error tolerance of $\pm 5\%$. Separately, we compute an execution reward, which is a binary score indicating whether the generated code can be executed successfully. These two rewards ensure that both the fidelity of critical information and the executability of the code are considered during RL optimization.

Visual Reward However, our analysis reveals that purely textual rewards generally focus on fine-grained details, while the overall structure of the rendered chart image is not considered. The chart-to-code task requires generating code that replicates the overall style of charts. To this end, we propose visual reward feedback Gu et al. (2025). Specifically, the generated code is executed to render a chart image, and we then utilize MLLMs to score the visual similarity between the generated chart and the input. This score is then normalized to serve as the visual reward. For the visual reward, we employ Qwen2.5-VL-72B Bai et al. (2025) as the evaluation model, which compares the chart image rendered by the generated code against the ground-truth chart across six key aspects: chart type, layout, text content, data, style, and clarity. The resulting scores are then normalized to form the final reward. Crucially, code that fails to render an image receives a reward of 0. The framework of our proposed MSRL is denoted in Figure 2.

Two-stage RL Obtaining visual rewards through model evaluation requires substantial computational time. To ensure training efficiency, our two-stage RL strategy first optimizes the model using only a textual reward. In the second stage, we introduce a hybrid reward, which combines the textual reward with a visual one, to fine-tune the model for visual fidelity using a reduced number of samples. This approach allows us to balance computational cost and model performance, achiev-

Table 2: Evaluation results of various models on the ChartMimic Direct Mimic and ReachQA benchmarks. ‡ denotes the updated result with code_pass constraints reported in its repository.

Model	**Params**	**ChartMimic**			**ReachQA**		
		Exec.Rate	Low-Level	High-Level	Exec.Rate	Low-Level	High-Level
Proprietary							
GeminiProVision	-	68.2	53.8	53.3	74.0	67.0	67.8
Claude-3-opus	-	83.3	60.5	60.1	89.0	51.7	61.1
GPT-4V	-	91.2	76.4	78.9	88.0	69.5	78.6
GPT-4o	-	93.2	**79.0**	83.5	92.8	81.8	84.0
Open-Source General-Domain							
Qwen2-VL-7B	7B	67.0	32.9	35.0	55.4	22.6	29.3
Qwen2.5-VL-7B	7B	73.2	44.6	41.6	62.2	36.9	37.6
InternVL2-8B	8B	61.8	34.4	38.9	50.8	24.1	24.2
InternVL2-26B	26B	69.3	41.4	47.4	55.4	29.0	28.8
Qwen2-VL-72B	72B	73.3	54.4	50.9	77.2	50.0	48.1
Open-Source Chart-Domain							
ChartLlama	13B	57.5	24.8	28.1	54.8	11.1	8.1
TinyChart	3B	42.5	26.3	25.9	34.4	11.6	11.2
ChartVLM-L	14B	19.5	15.8	13.9	8.2	2.1	3.9
Chart2Code	7B	62.1	42.9	33.3	63.6	52.3	49.7
ChartCoder	7B	91.4	72.5‡	74.0	83.8	67.9	69.4
MSRL-SFT	7B	93.2	73.0	77.6	92.2	78.6	80.0
MSRL	7B	**96.5**	78.6	**83.8**	**98.2**	**86.1**	**89.9**

ing strong visual quality without excessive resource consumption. To properly balance the different training objectives, the total reward R for a completed trajectory is calculated as a weighted sum of the following three components:

$$R = w_t R_{\text{text}} + w_v R_{\text{vis}} + w_e R_{\text{exec}} \tag{1}$$

where w_t, w_v, and w_e are hyperparameters that balance the contribution of each component. We adopt the Group Relative Policy Optimization (GRPO) Shao et al. (2024) algorithm for RL. GRPO optimizes policy by leveraging group-wise relative advantages among sampled responses, which is well-suited for our customized reward designs of charts.

4 Experiment

4.1 Baselines and Benchmarks

We evaluate MSRL against three categories of state-of-the-art models. The first category comprises general-domain, open-source Multimodal Large Language Models (MLLMs): InternVL2 (8B, 26B) Chen et al. (2024b), Qwen2-VL (7B, 72B) Wang et al. (2024), and Qwen2.5-VL-7B Bai et al. (2025). The second includes proprietary models: GeminiProVision Team et al. (2023), Claude-3-opus Anthropic (2024), GPT-4V OpenAI (2023), and GPT-4o OpenAI (2024). The third category consists of chart-specific MLLMs: ChartLlama Han et al. (2023), Tinychart Zhang et al. (2024b), ChartVLM Xia et al. (2024), Chart2Code Zhang et al. (2025) and ChartCoder Zhao et al. (2025b).

All models were evaluated in a zero-shot setting on two distinct benchmarks: ChartMimic Yang et al. (2024a) and ReachQA He et al. (2024). For ReachQA, we adopt the evaluation setting of Chart2Code Zhang et al. (2025), using the 500 plotting scripts from its test set. For ChartMimic, we utilize the 600 examples from the latest version of the Direct Mimic task.

4.2 Main Results

Our proposed MSRL establishes a new state-of-the-art among all open-source MLLMs, as detailed in Table 2. On the ChartMimic benchmark Yang et al. (2024a), it achieves an execution rate of

Table 3: Detailed results of low-level scores on the ChartMimic Direct Mimic and ReachQA benchmarks. The ChartCoder performance metrics have been corrected due to a flawed evaluation setup.

Model	Params	ChartMimic				ReachQA			
		Text	Layout	Type	Color	Text	Layout	Type	Color
GPT-4o	-	**81.5**	89.8	**77.3**	67.2	84.4	91.2	81.3	70.5
Qwen2-VL-72B	72B	43.2	80.5	54.6	39.4	41.0	52.2	59.1	47.8
Qwen2.5-VL-7B	7B	37.7	65.6	42.6	32.4	29.6	40.9	44.2	32.8
ChartCoder‡	7B	65.9	85.4	72.3	66.4	58.2	80.0	70.4	63.2
MSRL	7B	78.0	**93.0**	75.3	**68.2**	**87.5**	**96.6**	**83.1**	**77.2**

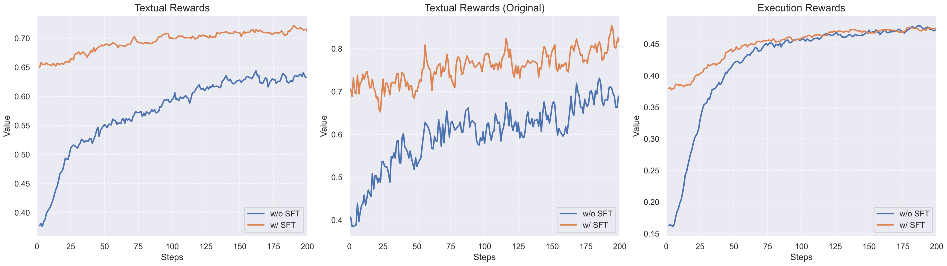


Figure 3: Comparison of textual reward and execution rate changes between baseline and SFT models during the RL stage.

96.5, a low-level score of 78.6, and a high-level score of 83.8. Notably, its performance is not only superior to its open-source models but also comparable to that of GPT-4o, a significantly larger proprietary model. On the ChartMimic benchmark, MSRL outperforms ChartCoder, the previous leading Chart-domain MLLM specifically designed for chart-to-code generation. Unlike other works that yield suboptimal performances from fine-tuning open-source models, our initial MSRL-SFT model already establishes a new state-of-the-art among open-source models. However, MSRL-SFT still falls short of GPT-4o on both low-level and high-level performance metrics. Building upon this strong foundation, the MSRL strategy further boosts the performance, especially in high-level scores, demonstrating the effectiveness of our integrated approach. The performance gap between SFT and RL models is emphasized in Figure 1.

We also conduct a detailed evaluation of low-level performance metrics, with the results presented in Table 3. Notably, MSRL outperforms all open-source models across low-level evaluation metrics and surpasses GPT-4o on the majority of evaluation metrics, which shows the strong capacity of MSRL in chart-to-code generation. As the GPT-4o is much larger than MSRL, the results demonstrate the effectiveness of our proposed methods.

SFT Plateau To investigate and demonstrate the effectiveness and efficiency of RL, we utilize various data scales for SFT to explore the upper limit of it. As illustrated in Figure 1, the results reveal a clear trend, whereby model performance improves sharply as the data size increases to 400k samples, followed by a period of slower growth up to the 1.6M samples. Beyond this data scale, performance saturates and reaches a distinct plateau, showing negligible growth with any further increase in data scale. Based on this trend, even if the data is multiplied, the benefits brought by SFT will be minimal. Applying our proposed MSRL method yields significant performance gains across all metrics. On the ChartMimic benchmark, we observe an average improvement of 6%, with notable increases in execution rate as well as in both low-level and high-level scores.

RL Plateau To investigate whether RL training exhibits a performance plateau, we present the performance curves of the two-stage RL training with the increase of data size, as shown in Figure 4. In the first stage using only textual rewards, the model breaks through the SFT plateau and achieves a significant performance improvement of approximately 5%, converging at a maximum of 22k training samples. In the second stage with multimodal rewards, the model's performance is further improved by about 1.5%, and almost converges at a maximum of 11k training samples. Our experiments effectively capture the performance plateau and improvements within a practical data size range, providing strong evidence for the effectiveness of our two-stage RL strategy.

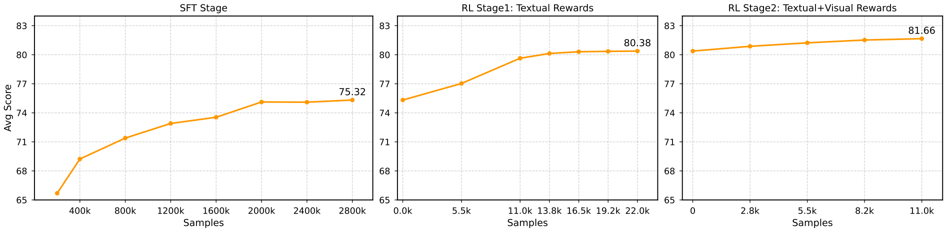


Figure 4: Performance curve of MSRL with increasing data scale. From left to right, the curves correspond to the SFT stage, RL stage 1, and RL stage 2. Avg Score denotes the average of the Low- and High-level scores in ChartMimic.

4.3 Ablation Study

SFT/RL Ablation To understand the impact of different training settings, we conduct ablation studies on both the SFT and RL training stages. In this ablation study, we use only textual rewards to ensure the results are not influenced by our improved RL algorithm, allowing us to demonstrate the interplay between SFT and RL. Using Qwen2.5-VL-7B-Instruct as the baseline model, the results are presented in Table 4. As the RL-only model underperforms on low-level and high-level scores compared to the SFT-only approach, only the execution rate is slightly higher due to the inclusion of an execution-based reward. Furthermore, the significantly larger performance gain from RL on a suboptimal baseline, compared to an SFT model, indicates that this improvement is partly attributable to the unsaturated state of baseline models. The performance gain on a saturated SFT model demonstrates that our proposed multi-granularity reward function can surpass the upper limit of SFT, achieving significantly superior performance. Figure 3 compares the reward and execution rate for the baseline and SFT models during the RL training, showing that applying RL to the SFT model achieves a much higher textual reward.

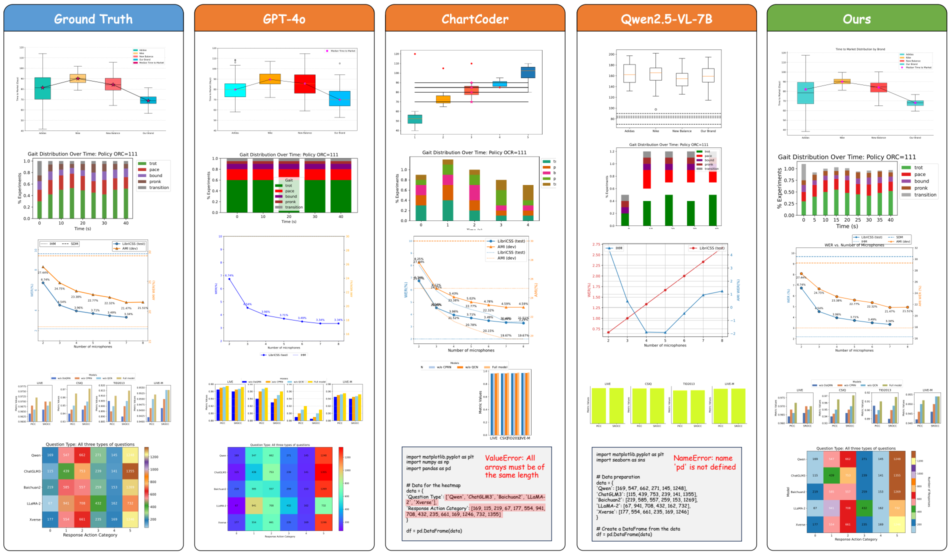
Table 4: Ablation studies on SFT and RL stages. For two RL experiments, only a textual reward is used.

SFT	RL	ChartMimic		
		Exec.Rate	Low-Level	High-Level
		73.16	44.58	41.55
✓		93.17	73.02	77.61
	✓	93.83	65.60	62.29
✓	✓	**97.00**	**78.06**	**82.69**

Table 5: Comparison of single-stage and two-stage RL training with different reward strategies. "T" and "V" represent textual and visual reward, respectively. All training is conducted on H800 GPUs.

Rewards	Samples	ChartMimic			GPU Hours
		Exec.Rate	Low-Level	High-Level	
-	-	93.17	73.02	77.61	-
(T, -)	(22k, -)	97.00	78.06	82.69	240
(V, -)	(22k, -)	97.37	79.13	84.01	1344
(T, T + V)	(22k, 11k)	97.50	79.50	83.81	912
(T, T + V)	(22k, 5.5k)	96.50	78.62	83.83	576

Reward and RL Training Strategy Ablation To thoroughly evaluate the effects of reward design and RL training strategies, we conduct ablation experiments comparing textual rewards, visual rewards, and their combinations. As shown in Table 5, both reward types improve model performance, but differ notably in resource consumption and efficiency. Visual rewards provide notable enhancements in code details and visual fidelity, but require more computational resources due to the overhead of image rendering and model evaluation. In contrast, textual rewards achieve competitive performance with much lower resource requirements. To balance efficiency and effectiveness, we adopt a two-stage RL strategy. The ablation results yield several important insights: 1) Both textual and visual rewards lead to significant improvements over the SFT model, with visual rewards being more effective for visual fidelity. 2) The high resource consumption of visual rewards highlights the importance of balancing performance and efficiency. 3) The two-stage RL strategy achieves the trade-off by first using low-cost textual rewards, followed by high-cost visual rewards with a limited number of samples. Based on these findings, we empirically report the two-stage training strategy with 50% of the training samples in our experiments, as it offers a better trade-off between performance and consumption efficiency.



```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Data for the heatmap
data = {
'Question Type': ['Qwen', 'ChatGLM3', 'Baichuan2', 'LLaMA-2', 'Xverse'],
'Response Action Category': [169, 115, 219, 67, 177, 554, 941, 708, 432, 235, 661, 169, 1246, 732, 1355]
}

df = pd.DataFrame(data)
```

ValueError: All arrays must be of the same length

```
import matplotlib.pyplot as plt
import seaborn as sns

# Data preparation
data = {
'Qwen': [169, 547, 662, 271, 145, 1248],
'ChatGLM3': [115, 439, 753, 239, 141, 1355],
'Baichuan2': [219, 585, 557, 259, 153, 1269],
'LLaMA-2': [67, 941, 708, 432, 162, 732],
'Xverse': [177, 554, 661, 235, 169, 1246]
}

# Create a DataFrame from the data
df = pd.DataFrame(data)
```

NameError: name 'pd' is not defined

Figure 5: Showcasing charts generated by MSRL compared to proprietary and open-source MLLMs. The first column shows the ground-truth chart images. Columns 2–4 display charts rendered from the code generated by GPT-4o, ChartCoder, and Qwen2.5-VL-7B, while the last column shows charts rendered by our MSRL model.

Generalization to Unseen Plotting Libraries To evaluate whether MSRL can transfer its capabilities to other plotting libraries, we collect 107 Seaborn and 150 Plotly images from their official websites. Since we only use Matplotlib-style code to generate charts, the new test data are out-of-domain. We perform inference with both Qwen2.5-VL-7B and MSRL on these two small test sets, calculating execution rates and high-level scores according to the ChartMimic protocol. As shown in Table 6, the results demonstrate that MSRL exhibits generalization to unseen Seaborn- and Plotly-style images.

Table 6: Performance of Qwen2.5-VL-7B and MSRL on Seaborn and Plotly test sets.

Model	**Seaborn**		**Plotly**	
	Exec.Rate	High-Level	Exec.Rate	High-Level
Qwen2.5-VL-7B	69.2	25.7	62.7	22.6
MSRL	**85.1**	**30.5**	**90.0**	**35.9**

4.4 Visualization

To provide a qualitative perspective on performance, we showcase several illustrative examples. As visually demonstrated in Figure 5, the charts generated by MSRL exhibit a markedly higher fidelity to the ground truth when compared with leading open-source MLLMs, even GPT-4o. In particular, the data values and layouts of charts generated by MSRL are significantly more precise.

5 Conclusion

In this work, we investigate the performance plateau of Supervised Fine-Tuning (SFT) in chart-to-code generation and propose a Multimodal Structured Reinforcement Learning (MSRL) strategy to overcome it. We begin by constructing a 3M-pair training corpus from real-world tables to systematically demonstrate that merely increasing SFT data leads to diminishing returns. To address this, we introduce the MSRL, which optimizes the saturated SFT model using a two-stage training process guided by a multi-granularity reward function that assesses both textual and visual correctness. Our approach achieves new state-of-the-art performance on standard benchmarks, surpassing all open-source models and rivaling powerful proprietary ones. We make significant contributions to the field, including the curation of a large-scale dataset for a definitive empirical analysis of the SFT performance bottleneck, and the development of the MSRL framework.

6 Reproducibility Statement

For datasets, we provide a detailed description of the data generation process in Section 3.1 of the paper, along with dataset statistics and examples in Appendix A, and the exact prompts used for dataset construction in Appendix E. For code implementation, we include the SFT and RL training code for MSRL in the supplementary materials, which contains complete training scripts and implementations of the multimodal reward functions.

References

Anthropic. Introducing the next generation of claude, 2024. URL `https://www.anthropic.com/news/claude-3-family`.

Rabiul Awal, Mahsa Massoud, Aarash Feizi, Zichao Li, Suyuchen Wang, Christopher Pal, Aishwarya Agrawal, David Vazquez, Siva Reddy, Juan A Rodriguez, et al. Webmmu: A benchmark for multimodal multilingual website understanding and code generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25129–25156, 2025.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Lei Chen, Xuanle Zhao, Zhixiong Zeng, Jing Huang, Yufeng Zhong, and Lin Ma. Chart-r1: Chain-of-thought supervision and reinforcement for advanced chart reasoner. *arXiv preprint arXiv:2507.15509*, 2025a.

Yang Chen, Yufan Shen, Wenxuan Huang, Shen Zhou, Qunshu Lin, Xinyu Cai, Zhi Yu, Botian Shi, and Yu Qiao. Learning only with images: Visual reinforcement learning with reasoning, rendering, and visual feedback. *arXiv preprint arXiv:2507.20766*, 2025b.

Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, et al. Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling. *arXiv preprint arXiv:2412.05271*, 2024a.

Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24185–24198, 2024b.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. A survey on llm-as-a-judge, 2025. URL `https://arxiv.org/abs/2411.15594`.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Yucheng Han, Chi Zhang, Xin Chen, Xu Yang, Zhibin Wang, Gang Yu, Bin Fu, and Hanwang Zhang. Chartllama: A multimodal llm for chart understanding and generation. *arXiv preprint arXiv:2311.16483*, 2023.

Wei He, Zhiheng Xi, Wanxu Zhao, Xiaoran Fan, Yiwen Ding, Zifei Shan, Tao Gui, Qi Zhang, and Xuanjing Huang. Distill visual chart reasoning ability from llms to mllms. *arXiv preprint arXiv:2410.18798*, 2024.

Wenxuan Huang, Bohan Jia, Zijie Zhai, Shaosheng Cao, Zheyu Ye, Fei Zhao, Zhe Xu, Yao Hu, and Shaohui Lin. Vision-r1: Incentivizing reasoning capability in multimodal large language models. *arXiv preprint arXiv:2503.06749*, 2025.

Shankar Kantharaj, Rixie Tiffany Leong, Xiang Lin, Ahmed Masry, Megh Thakkar, Enamul Hoque, and Shafiq Joty. Chart-to-text: A large-scale benchmark for chart summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4005–4023, 2022.

Kaixin Li, Yuchen Tian, Qisheng Hu, Ziyang Luo, Zhiyong Huang, and Jing Ma. Mmcode: Benchmarking multimodal large language models for code generation with visually rich programming problems. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 736–783, 2024.

Fangyu Liu, Julian Eisenschlos, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Wenhu Chen, Nigel Collier, and Yasemin Altun. Deplot: One-shot visual language reasoning by plot-to-table translation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 10381–10399, 2023.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Ahmed Masry, Xuan Long Do, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the association for computational linguistics: ACL 2022*, pp. 2263–2279, 2022.

Ahmed Masry, Abhay Puri, Masoud Hashemi, Juan A. Rodriguez, Megh Thakkar, Khyati Mahajan, Vikas Yadav, Sathwik Tejaswi Madhusudhan, Alexandre Piche, Dzmitry Bahdanau, Christopher Pal, David Vazquez, Enamul Hoque Prince, Perouz Taslakian, Sai Rajeswar Mudumba, and Spandana Gella. Bigcharts-r1: Enhanced chart reasoning with visual reinforcement finetuning. In *Conference on Language Modeling (COLM)*, 2025.

Fanqing Meng, Lingxiao Du, Zongkai Liu, Zhixiang Zhou, Quanfeng Lu, Daocheng Fu, Tiancheng Han, Botian Shi, Wenhai Wang, Junjun He, Kaipeng Zhang, Ping Luo, Yu Qiao, Qiaosheng Zhang, and Wenqi Shao. Mm-eureka: Exploring the frontiers of multimodal reasoning with rule-based reinforcement learning. *arXiv preprint arXiv:2503.07365*, 2025.

Minheng Ni, Zhengyuan Yang, Linjie Li, Chung-Ching Lin, Kevin Lin, Wangmeng Zuo, and Lijuan Wang. Point-rft: Improving multimodal reasoning with visually grounded reinforcement finetuning. *arXiv preprint arXiv:2505.19702*, 2025.

OpenAI. Gpt-4v(ision) system card, 2023. URL `https://openai.com/index/gpt-4v-system-card/`.

OpenAI. Gpt-4o, 2024. URL `https://openai.com/index/hello-gpt-4o`. Accessed: 2024-05-13.

OpenAI. Gpt-4.1. `https://openai.com/index/gpt-4-1/`, 2025a.

OpenAI. Introducing openai o3 and o4-mini. `https://openai.com/index/introducing-o3-and-o4-mini/`, April 2025b. Accessed: 2025-07-14.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

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.

Juan A Rodriguez, Xiangru Jian, Siba Smarak Panigrahi, Tianyu Zhang, Aarash Feizi, Abhay Puri, Akshay Kalkunte Suresh, François Savard, Ahmed Masry, Shravan Nayak, et al. Bigdocs: An open dataset for training multimodal models on document and code tasks. In *The Thirteenth International Conference on Learning Representations*, 2024.

Juan A Rodriguez, Abhay Puri, Shubham Agarwal, Issam H Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. Starvector: Generating scalable vector graphics code from images and text. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 16175–16186, 2025a.

Juan A Rodriguez, Haotian Zhang, Abhay Puri, Aarash Feizi, Rishav Pramanik, Pascal Wichmann, Arnab Mondal, Mohammad Reza Samsami, Rabiul Awal, Perouz Taslakian, et al. Rendering-aware reinforcement learning for vector graphics generation. In *Advances in Neural Information Processing Systems (NeurIPS 2025)*, 2025b.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Haozhan Shen, Peng Liu, Jingcheng Li, Chunxin Fang, Yibo Ma, Jiajia Liao, Qiaoli Shen, Zilun Zhang, Kangjia Zhao, Qianqian Zhang, et al. Vlm-r1: A stable and generalizable r1-style large vision-language model. *arXiv preprint arXiv:2504.07615*, 2025.

Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: How far are we from automating front-end engineering? *arXiv preprint arXiv:2403.03163*, 2024.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Ke Wang, Junting Pan, Linda Wei, Aojun Zhou, Weikang Shi, Zimu Lu, Han Xiao, Yunqiao Yang, Houxing Ren, Mingjie Zhan, et al. Mathcoder-vl: Bridging vision and code for enhanced multimodal mathematical reasoning. *arXiv preprint arXiv:2505.10557*, 2025a.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024.

Yaoting Wang, Shengqiong Wu, Yuecheng Zhang, Shuicheng Yan, Ziwei Liu, Jiebo Luo, and Hao Fei. Multimodal chain-of-thought reasoning: A comprehensive survey. *arXiv preprint arXiv:2503.12605*, 2025b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Chengyue Wu, Zhixuan Liang, Yixiao Ge, Qiushan Guo, Zeyu Lu, Jiahao Wang, Ying Shan, and Ping Luo. Plot2code: A comprehensive benchmark for evaluating multi-modal large language models in code generation from scientific plots. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 3006–3028, 2025.

Renqiu Xia, Bo Zhang, Haoyang Peng, Hancheng Ye, Xiangchao Yan, Peng Ye, Botian Shi, Yu Qiao, and Junchi Yan. Structchart: Perception, structuring, reasoning for visual chart understanding. *arXiv preprint arXiv:2309.11268*, 2023.

Renqiu Xia, Bo Zhang, Hancheng Ye, Xiangchao Yan, Qi Liu, Hongbin Zhou, Zijun Chen, Min Dou, Botian Shi, Junchi Yan, et al. Chartx & chartvlm: A versatile benchmark and foundation model for complicated chart reasoning. *arXiv preprint arXiv:2402.12185*, 2024.

Jingyu Xiao, Yuxuan Wan, Yintong Huo, Zixin Wang, Xinyi Xu, Wenxuan Wang, Zhiyao Xu, Yuhang Wang, and Michael R Lyu. Interaction2code: Benchmarking mllm-based interactive webpage code generation from interactive prototyping. *arXiv preprint arXiv:2411.03292*, 2024.

Zhengzhuo Xu, Bowen Qu, Yiyan Qi, SiNan Du, Chengjin Xu, Chun Yuan, and Jian Guo. Chartmoe: Mixture of diversely aligned expert connector for chart understanding. In *The Thirteenth International Conference on Learning Representations*, 2024.

Cheng Yang, Chufan Shi, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran XU, Xinyu Zhu, Siheng Li, Yuxiang Zhang, et al. Chartmimic: Evaluating lmm's cross-modal reasoning capability via chart-to-code generation. In *The Thirteenth International Conference on Learning Representations*, 2024a.

Donglu Yang, Liang Zhang, Zihao Yue, Liangyu Chen, Yichen Xu, Wenxuan Wang, and Qin Jin. Chartm3: Benchmarking chart editing with multimodal instructions. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 5001–5009, 2025a.

Yi Yang, Xiaoxuan He, Hongkun Pan, Xiyan Jiang, Yan Deng, Xingtao Yang, Haoyu Lu, Dacheng Yin, Fengyun Rao, Minfeng Zhu, et al. R1-onevision: Advancing generalized multimodal reasoning through cross-modal formalization. *arXiv preprint arXiv:2503.10615*, 2025b.

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

Sukmin Yun, Rusiru Thushara, Mohammad Bhat, Yongxin Wang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, Haonan Li, Preslav Nakov, et al. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *Advances in neural information processing systems*, 37:112134–112157, 2024.

Duzhen Zhang, Yahan Yu, Jiahua Dong, Chenxing Li, Dan Su, Chenhui Chu, and Dong Yu. Mm-llms: Recent advances in multimodal large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, pp. 12401–12430, 2024a.

Liang Zhang, Anwen Hu, Haiyang Xu, Ming Yan, Yichen Xu, Qin Jin, Ji Zhang, and Fei Huang. Tinychart: Efficient chart understanding with visual token merging and program-of-thoughts learning. *arXiv preprint arXiv:2404.16635*, 2024b.

Zhehao Zhang, Weicheng Ma, and Soroush Vosoughi. Is gpt-4v (ision) all you need for automating academic data visualization? exploring vision-language models' capability in reproducing academic charts. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 8271–8288, 2024c.

Zhihan Zhang, Yixin Cao, and Lizi Liao. Enhancing chart-to-code generation in multimodal large language models via iterative dual preference learning. *arXiv preprint arXiv:2504.02906*, 2025.

Xuanle Zhao, Xuexin Liu, Yang Haoyue, Xianzhen Luo, Fanhu Zeng, Jianling Li, Qi Shi, and Chi Chen. Chartedit: How far are mllms from automating chart analysis? evaluating mllms' capability via chart editing. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 3616–3630, 2025a.

Xuanle Zhao, Xianzhen Luo, Qi Shi, Chi Chen, Shuo Wang, Zhiyuan Liu, and Maosong Sun. Chartcoder: Advancing multimodal large language model for chart-to-code generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025b.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguistics. URL `http://arxiv.org/abs/2403.13372`.

A CHART2CODE DETAILS

We count the quantity and distribution of 24 chart types used in the SFT and RL stages respectively, as detailed in Table A. The distribution among the various types is relatively balanced. For the RL data, we filtered out two chart types, Graph and Funnel, which are not suitable for textual reward calculation. Furthermore, we present three visualization examples from the Chart2Code dataset, with increasing data density and complexity, as shown in Figures G, H and I.

Table A: The detailed chart types and corresponding quantities used in the SFT and RL phases.

Split	Bar	Line	ErrorBar	Heatmap	Box	Scatter	Histogram	Radar	3D
SFT	131,623	131,596	130,818	123,795	125,541	127,612	132,180	124,160	128,744
RL	2,669	2,743	1,067	1,984	756	2,639	1,510	776	1,699
Split	Pie	ErrorPoint	Violin	Area	Bubble	Multi-axes	Ring	Rose	Treemap
SFT	125,812	123,774	120,180	120,038	127,207	116,438	105,912	118,047	119,176
RL	1,446	2,277	900	1,558	1,093	1,691	960	674	518
Split	Bar_num	Contour	Density	Quiver	Graph	Funnel	Total		
SFT	128,333	123,461	130,769	128,987	108,150	1,152	2,853,505		
RL	2,757	1,040	1,899	1,700	-	-	34,356		

B Implementation Details

We utilize 2.8M chart2code data for SFT and 33k curated high-quality chart2code data for RL. MSRL-SFT employs Qwen2.5VL-7B-Instruct as the baseline model and LLaMA-Factory Zheng et al. (2024) for SFT. The SFT stage is trained for one epoch with a learning rate of 1e-5 and a batch size of 32. The training process takes 60 hours on 16 H800 GPUs. Built upon the SFT model, MSRL undergoes a two-stage RL process using the MM-EUREKA framework Meng et al. (2025). MSRL consists of two RL stages. Stage one utilizes 22k data pairs and a textual-only reward ($w_t = 1$, $w_v = 0$, $w_e = 0.5$). Stage two uses the remaining 11k pairs with a hybrid reward ($w_t = 0.5$, $w_v = 0.5$, $w_e = 0.5$). The textual reward components are weighted as $w_d = 0.4$, $w_c = 0.3$, $w_l = 0.1$, $w_t = 0.1$, $w_{lbl} = 0.1$. We deploy Qwen2.5-VL-72B-Instruct as the evaluation model with max_model_len set to 12,288 for calculating visual rewards. For both RL stages, we train for one episode with a learning rate of 1e-6 and a batch size of 128. We set the generation temperature to 1, generate 8 rollouts per sample, and omit the KL divergence from the loss computation. The first and second stages required 10 and 24 hours of training, respectively, on 24 H800 GPUs.

C More Ablation Studies

The Use of Visual Evaluator To verify that the improvement in performance results from effective visual reward design rather than overfitting to the preferences of the VLLM judger, we conduct an ablation experiment to observe the distribution of sample scores when the visual evaluator changes. Specifically, we randomly sample 200 instances from the RL data and use the MSRL-SFT model to perform chart-to-code prediction, rendering the resulting Python code into images. Following the ChartMimic scoring guidelines, human ratings of rendered samples serve as the gold scores. Next, we use GPT-4.1, Qwen2.5-VL-72B, Qwen2.5-VL-7B, and InternVL3-38B to score the images. Additionally, we compute the visual similarity between the ground-truth and rendered images using DINO-L. We calculate the Pearson correlation coefficients between the scores from these models and the human scores, and plot the distributions of model scores versus human scores. As shown in the Figure A, GPT-4.1 achieves the highest correlation with human judgment and has the most similar score distribution, followed by Qwen2.5-VL-72B, while Qwen2.5-VL-7B performs the worst. Considering both performance and budget, we select Qwen2.5-VL-72B as the visual evaluator, as it fairly reflects model capability and prevents reward hacking.

In addition, we introduce two vision models, InternViT Chen et al. (2024a) and DINOv2-L Oquab et al. (2023), as evaluators for visual rewards, calculating the visual similarity between ground-truth and generated image features. As shown in Table B, these vision models yield improved performance, but there remains a substantial gap compared to models employing VLMs as visual evaluators. This indicates that vision models trained on image-text matching assess similarity at a coarse level and do not adequately capture fine-grained differences in chart elements. Compared to Qwen2.5-VL-72B, smaller vision models such as DINO-L exhibit greater discrepancies in score distributions relative to human judgment.

GRPO vs DPO Intuitively, GRPO is naturally more suitable for chart-to-code tasks than DPO. The degree to which code restores a chart can be quantitatively measured by a set of well-defined rules, making it ideal for designing the reward function in GRPO. In contrast, DPO learns from preference

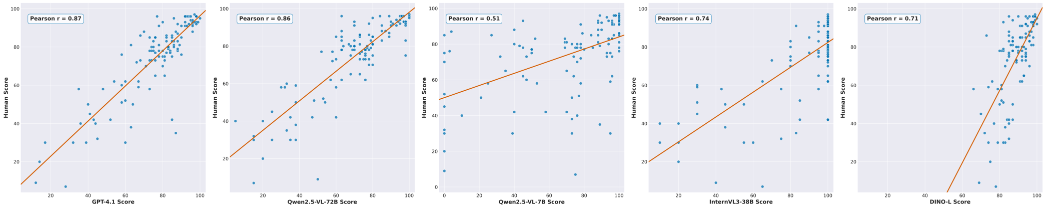


Figure A: Scatter plots of model scores versus human gold scores for five visual evaluators (GPT-4.1, Qwen2.5-VL-72B, Qwen2.5-VL-7B, InternVL3-38B, and DINO-L). Pearson r denotes the Pearson correlation coefficient between model and human scores.

Table B: The ablation study of visual evaluators. ”-” denotes the SFT model. All experiments exclude textual rewards.

Visual Evaluator	**ChartMimic**	
	Low-Level	High-Level
-	73.0	77.6
InternViT	75.1	78.1
DINOv2-L	75.5	79.8
Qwen2.5-VL-72B	**77.8**	**81.9**

pairs, and it is challenging to clearly define what constitutes ”bad” code in chart-to-code tasks, which can lead to ambiguity in the model's optimization direction. To ensure a thorough evaluation, we conduct an ablation study comparing GRPO and DPO. Since DPO requires the construction of positive and negative sample pairs, we follow the Chart2Code Zhang et al. (2025) and randomly perturb 2-3 attributes of the chart code to generate negative samples using GPT-4.1, resulting in an RL dataset of comparable size to GRPO for DPO training on the SFT model. As shown in Table C, the performance improvement of DPO is significantly less than that of GRPO. This indicates both the superiority of the GRPO algorithm and the effectiveness of our multimodal structured reward design and two-stage curriculum learning.

Table C: The ablation study of RL optimizers. ”-” denotes the SFT model.

RL Optimizer	**ChartMimic**	
	Low-Level	High-Level
-	73.0	77.6
DPO	74.0	78.9
GRPO	**78.6**	**83.8**

Comparison with Existing Datasets To highlight the value of our synthetic data, we compare the results of using our data and open-source data (i.e., ChartCoder) in both the SFT and RL stages, as shown in Table D. The results show that models trained with our data achieve better performance in both stages. On the one hand, our sufficient data enable SFT to reach its full potential. On the other hand, our charts are generated from real tabular data, featuring more challenging multi-chart content and greater visual diversity, which are key factors contributing to the effectiveness of our dataset.

D Qualitative Analysis

Figures E and F show two examples of generated code from different models and their corresponding rendered images. Compared to the initial Qwen2.5-VL-7B model, both MSRL-SFT and MSRL show significant improvements in chart type recognition, numerical extraction, and color identification. Compared to MSRL-SFT, MSRL performs better in chart type recognition and clarity. In Figure E, MSRL-SFT incorrectly identifies the line chart as a scatter plot, displays incorrect legends, and has overlapping issues with image boundaries. In Figure F, MSRL-SFT has layout syntax errors, where only the last subplot is successfully displayed out of three subplots. MSRL's results are more consistent with the original images, though it still has some minor issues in detail. For

Table D: Performance comparison with the ChartCoder dataset. We use Qwen2.5-VL-7B and MSRL-SFT as base models for SFT and RL for fine-tuning on the two datasets, respectively.

Stage	**Dataset**	**ChartMimic**	
		Low-Level	High-Level
SFT	ChartCoder	68.5	68.0
	Ours	**73.0**	**77.6**
RL	ChartCoder	76.5	81.2
	Ours	**78.6**	**83.8**

example, in Figure E, MSRL arbitrarily adds a title and scatter data points to the subcategory "Ours (LoSA)". In Figure F, while MSRL successfully renders three subplots, the layout recognition is not sufficiently accurate, leaving room for further improvement.

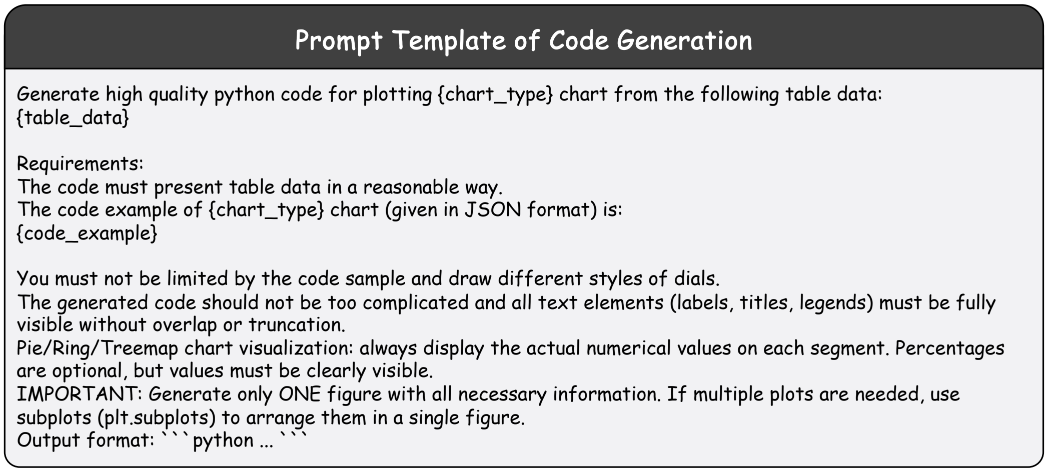
E Prompt Template

To enhance transparency and reproducibility, we provide the exact prompts used for dataset generation and visual reward feedback.

Figure B shows the prompt used for code generation. We use real table data as input, select one chart type from 24 predefined chart types, and sample code examples of the selected chart type to generate plotting code. Specifically, for chart types Pie, Ring, and Treemap, we require the model to display actual values on the image, as these chart types typically only visualize percentage values in their code, making it impossible to read the true values from the image.

Figure C shows the prompt used for evaluating image quality. When processing RL data, we use this prompt to filter out visually low-quality samples, such as those with overlapping text or elements.

Figure D shows the prompt used to calculate visual rewards. Following ChartMimic Yang et al. (2024a), we assess the match between ground truth and model predictions across 6 dimensions, with this score normalized as the visual score and incorporated into the overall reward computation.



Prompt Template of Code Generation

Generate high quality python code for plotting {chart_type} chart from the following table data:
{table_data}

Requirements:
The code must present table data in a reasonable way.
The code example of {chart_type} chart (given in JSON format) is:
{code_example}

You must not be limited by the code sample and draw different styles of dials.
The generated code should not be too complicated and all text elements (labels, titles, legends) must be fully visible without overlap or truncation.
Pie/Ring/Treemap chart visualization: always display the actual numerical values on each segment. Percentages are optional, but values must be clearly visible.
IMPORTANT: Generate only ONE figure with all necessary information. If multiple plots are needed, use subplots (plt.subplots) to arrange them in a single figure.
Output format: ```python ... ```

Figure B: Prompt template for Chart2Code code generation.

F The Use of Large Language Models (LLMs)

In this work, we utilized LLMs primarily as writing assistance tools to polish language and improve clarity. They helped refine sentence structures and enhance readability throughout the manuscript. The core research ideas, experimental design, implementation, and analysis were all conducted by the authors, with LLMs serving only as editorial aids.

Prompt Template of Image-quality Evaluation

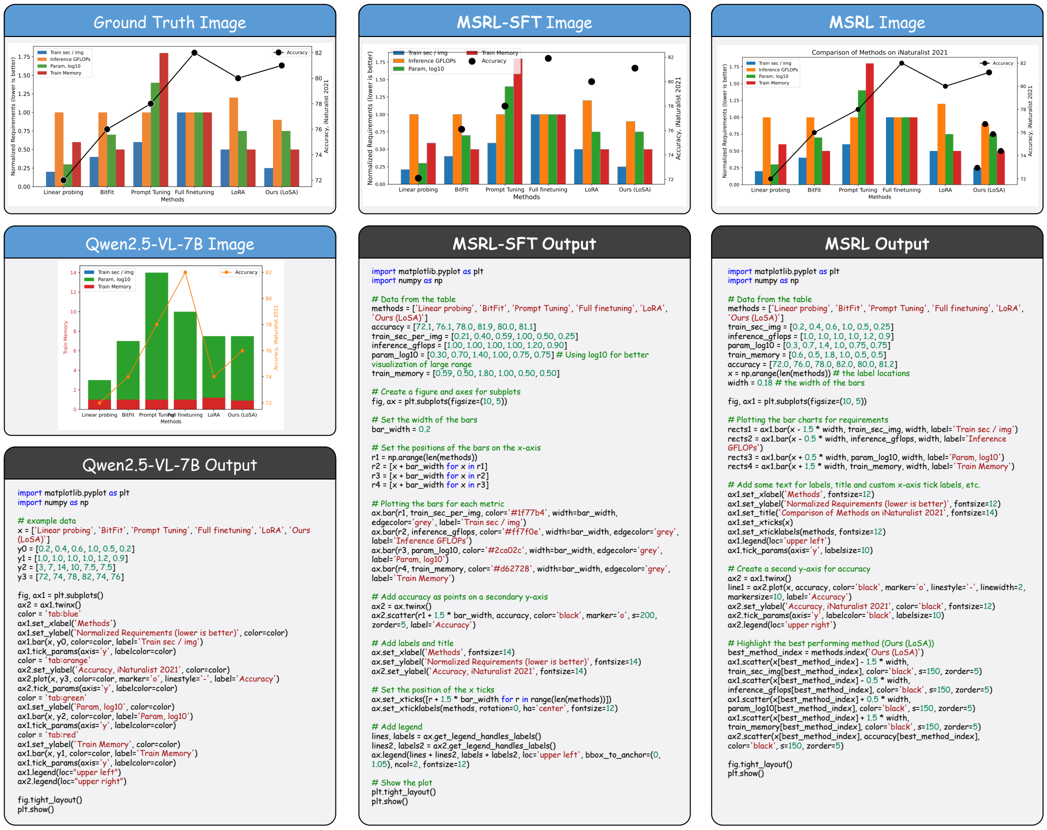
```
You are an expert in data visualization and chart analysis. Your task is to evaluate if the provided chart image is
properly rendered and of high quality.

Please analyze the chart and check for the following issues:
1. Rendering problems (missing elements, incorrect display)
2. Data anomalies (outliers that don't make sense, inconsistent scales)
3. Truncated or cut-off text or elements
4. Overlapping text or elements that reduce readability
5. Poor color choices that make the chart hard to read
6. Missing labels or legends that are necessary for understanding
7. Any other quality issues that affect the chart's usefulness

After your analysis, provide a single score:
- Score 1: The chart is properly rendered, data appears normal, and there are no significant quality issues.
- Score 0: The chart has one or more significant quality issues that affect its usefulness.

Respond with ONLY the score (1 or 0) and nothing else.
```

Figure C: Prompt template for image-quality evaluation during RL data filtering.

Prompt Template of Visual Reward Feedback

```
You are an excellent judge at evaluating visualization chart plots. The first image (reference image) is created
using ground truth matplotlib code, and the second image (AI-generated image) is created using matplotlib
code generated by an AI assistant. Your task is to score how well the AI-generated plot matches the ground
truth plot.

### Scoring Methodology:
The AI-generated image's score is based on the following criteria, totaling a score out of 100 points:

1. **Chart Types (20 points)**
   Does the AI-generated image include all chart types present in the reference image
   (e.g., line charts, bar charts, etc.)?
2. **Layout (10 points)**
   Does the arrangement of subplots in the AI-generated image match the reference image
   (e.g., number of rows and columns)?
3. **Text Content (20 points)**
   Does the AI-generated image include all text from the reference image
   (e.g., titles, annotations, axis labels), excluding axis tick labels?
4. **Data (20 points)**
   How accurately do the data trends in the AI-generated image resemble those in the original image
   and is the number of data groups the same as in the reference image?
5. **Style (20 points)**
   Does the AI-generated image match the original in terms of colors (line colors, fill colors, etc.),
   marker types (point shapes, line styles, etc.), legends, grids, and other stylistic details?
6. **Clarity (10 points)**
   Is the AI-generated image clear and free of overlapping elements?

### Evaluation:
Compare the two images head to head and provide a detailed assessment. Use the following format for your
response:

---
Comments:
- Chart Types: ${your comment and subscore}
- Layout: ${your comment and subscore}
- Text Content: ${your comment and subscore}
- Data: ${your comment and subscore}
- Style: ${your comment and subscore}
- Clarity: ${your comment and subscore}

Score: ${your final score out of 100}
---

Please use the above format to ensure the evaluation is clear and comprehensive.
```

Figure D: Prompt template for visual reward feedback during the RL training.



Qwen2.5-VL-7B Output

```
import matplotlib.pyplot as plt
import numpy as np

# example data
x = ['Linear probing', 'BitFit', 'Prompt Tuning', 'Full finetuning', 'LoRA', 'Ours
(LoSA)']
y0 = [0.2, 0.4, 0.6, 1.0, 0.5, 0.2]
y1 = [1.0, 1.0, 1.0, 1.0, 1.2, 0.9]
y2 = [3, 7, 14, 10, 7.5, 7.5]
y3 = [72, 74, 78, 82, 74, 76]

fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
color = 'tab:blue'
ax1.set_xlabel('Methods')
ax1.set_ylabel('Normalized Requirements (lower is better)', color=color)
ax1.bar(x, y0, color=color, label='Train sec / img')
ax1.tick_params(axis='y', labelcolor=color)
color = 'tab:orange'
ax2.set_ylabel('Accuracy, iNaturalist 2021', color=color)
ax2.plot(x, y3, color=color, marker='o', linestyle='-', label='Accuracy')
ax2.tick_params(axis='y', labelcolor=color)
color = 'tab:green'
ax1.set_ylabel('Param, log10', color=color)
ax1.bar(x, y2, color=color, label='Param, log10')
ax1.tick_params(axis='y', labelcolor=color)
color = 'tab:red'
ax1.set_ylabel('Train Memory', color=color)
ax1.bar(x, y1, color=color, label='Train Memory')
ax1.tick_params(axis='y', labelcolor=color)
ax1.legend(loc="upper left")
ax2.legend(loc="upper right")

fig.tight_layout()
plt.show()
```

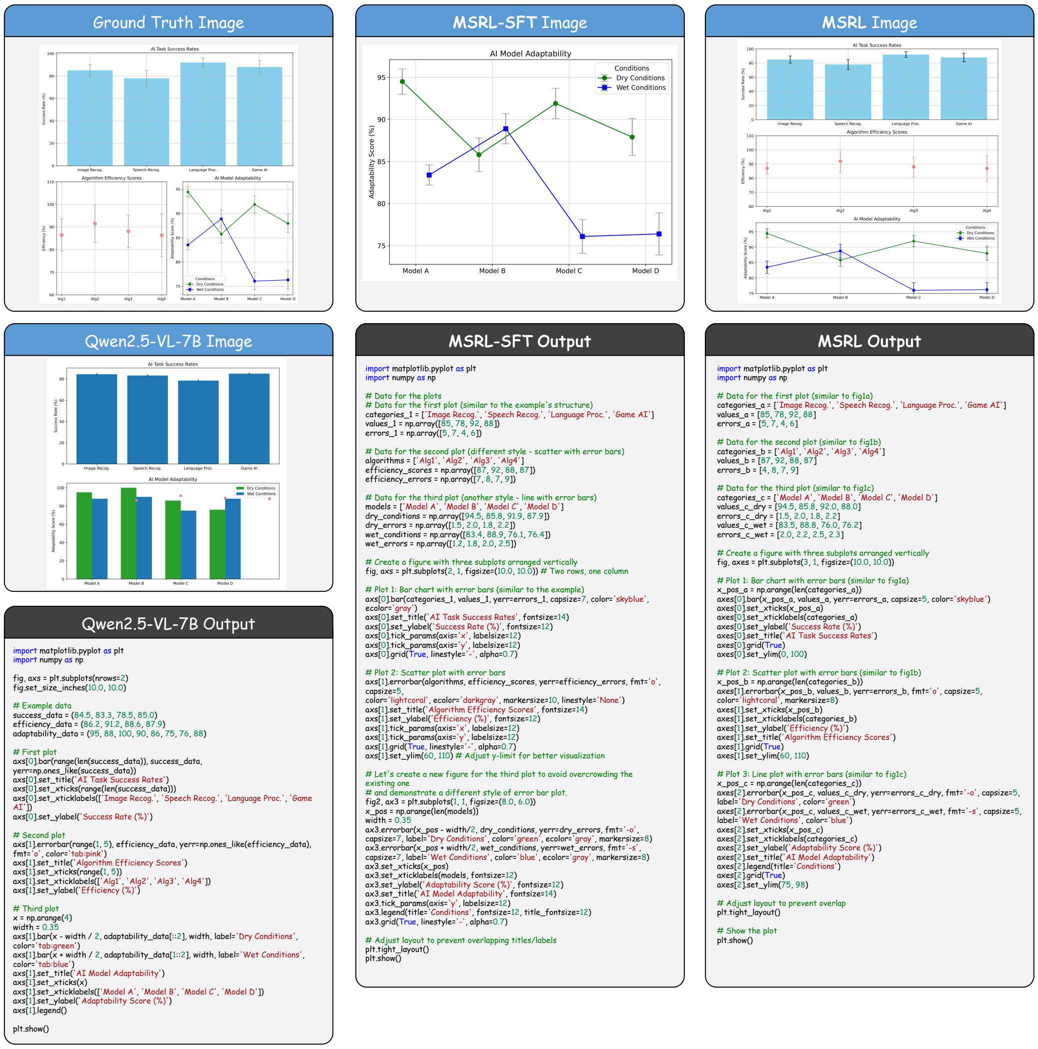
MSRL-SFT Output

```
import matplotlib.pyplot as plt
import numpy as np

# Data from the table
methods = ['Linear probing', 'BitFit', 'Prompt Tuning', 'Full finetuning', 'LoRA',
'Ours (LoSA)']
accuracy = [72.1, 76.1, 78.0, 81.9, 80.0, 81.1]
train_sec_per_img = [0.21, 0.40, 0.59, 1.00, 0.50, 0.25]
inference_gflops = [1.00, 1.00, 1.00, 1.00, 1.20, 0.90]
param_log10 = [0.30, 0.70, 1.40, 1.00, 0.75, 0.75] # Using log10 for better
visualization of large range
train_memory = [0.59, 0.50, 1.80, 1.00, 0.50, 0.50]

# Create a figure and axes for subplots
fig, ax = plt.subplots(figsize=(10, 5))

# Set the width of the bars
bar_width = 0.2

# Set the positions of the bars on the x-axis
r1 = np.arange(len(methods))
r2 = [x + bar_width for x in r1]
r3 = [x + bar_width for x in r2]
r4 = [x + bar_width for x in r3]

# Plotting the bars for each metric
ax.bar(r1, train_sec_per_img, color='#1f77b4', width=bar_width,
edgecolor='grey', label='Train sec / img')
ax.bar(r2, inference_gflops, color='#ff7f0e', width=bar_width, edgecolor='grey',
label='Inference GFLOPs')
ax.bar(r3, param_log10, color='#2ca02c', width=bar_width, edgecolor='grey',
label='Param, log10')
ax.bar(r4, train_memory, color='#d62728', width=bar_width, edgecolor='grey',
label='Train Memory')

# Add accuracy as points on a secondary y-axis
ax2 = ax.twinx()
ax2.scatter(r1 + 1.5 * bar_width, accuracy, color='black', marker='o', s=200,
zorder=5, label='Accuracy')

# Add labels and title
ax.set_xlabel('Methods', fontsize=14)
ax.set_ylabel('Normalized Requirements (lower is better)', fontsize=14)
ax2.set_ylabel('Accuracy, iNaturalist 2021', fontsize=14)

# Set the position of the x ticks
ax.set_xticks([r + 1.5 * bar_width for r in range(len(methods))])
ax.set_xticklabels(methods, rotation=0, ha='center', fontsize=12)

# Add legend
lines, labels = ax.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax.legend(lines + lines2, labels + labels2, loc='upper left', bbox_to_anchor=(0,
1.05), ncol=2, fontsize=12)

# Show the plot
plt.tight_layout()
plt.show()
```

MSRL Output

```
import matplotlib.pyplot as plt
import numpy as np

# Data from the table
methods = ['Linear probing', 'BitFit', 'Prompt Tuning', 'Full finetuning', 'LoRA',
'Ours (LoSA)']
train_sec_img = [0.2, 0.4, 0.6, 1.0, 0.5, 0.25]
inference_gflops = [1.0, 1.0, 1.0, 1.0, 1.2, 0.9]
param_log10 = [0.3, 0.7, 1.4, 1.0, 0.75, 0.75]
train_memory = [0.6, 0.5, 1.8, 1.0, 0.5, 0.5]
accuracy = [72.0, 76.0, 78.0, 82.0, 80.0, 81.2]
x = np.arange(len(methods)) # the label locations
width = 0.18 # the width of the bars

fig, ax1 = plt.subplots(figsize=(10, 5))

# Plotting the bar charts for requirements
rects1 = ax1.bar(x - 1.5 * width, train_sec_img, width, label='Train sec / img')
rects2 = ax1.bar(x - 0.5 * width, inference_gflops, width, label='Inference
GFLOPs')
rects3 = ax1.bar(x + 0.5 * width, param_log10, width, label='Param, log10')
rects4 = ax1.bar(x + 1.5 * width, train_memory, width, label='Train Memory')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax1.set_xlabel('Methods', fontsize=12)
ax1.set_ylabel('Normalized Requirements (lower is better)', fontsize=12)
ax1.set_title('Comparison of Methods on iNaturalist 2021', fontsize=14)
ax1.set_xticks(x)
ax1.set_xticklabels(methods, fontsize=12)
ax1.legend(loc='upper left')
ax1.tick_params(axis='y', labelsize=10)

# Create a second y-axis for accuracy
ax2 = ax1.twinx()
line1 = ax2.plot(x, accuracy, color='black', marker='o', linestyle='-', linewidth=2,
markersize=10, label='Accuracy')
ax2.set_ylabel('Accuracy, iNaturalist 2021', color='black', fontsize=12)
ax2.tick_params(axis='y', labelcolor='black', labelsize=10)
ax2.legend(loc='upper right')

# Highlight the best performing method (Ours (LoSA))
best_method_index = methods.index('Ours (LoSA)')
ax1.scatter(x[best_method_index] - 1.5 * width,
train_sec_img[best_method_index], color='black', s=150, zorder=5)
ax1.scatter(x[best_method_index] - 0.5 * width,
inference_gflops[best_method_index], color='black', s=150, zorder=5)
ax1.scatter(x[best_method_index] + 0.5 * width,
param_log10[best_method_index], color='black', s=150, zorder=5)
ax1.scatter(x[best_method_index] + 1.5 * width,
train_memory[best_method_index], color='black', s=150, zorder=5)
ax2.scatter(x[best_method_index], accuracy[best_method_index],
color='black', s=150, zorder=5)

fig.tight_layout()
plt.show()
```

Figure E: An example of comparing the code generated by different models with rendered images.



MSRL-SFT Output

```
import matplotlib.pyplot as plt
import numpy as np

# Data for the plots
# Data for the first plot (similar to the example's structure)
categories_1 = ['Image Recog.', 'Speech Recog.', 'Language Proc.', 'Game AI']
values_1 = np.array([85, 78, 92, 88])
errors_1 = np.array([5, 7, 4, 6])

# Data for the second plot (different style - scatter with error bars)
algorithms = ['Alg1', 'Alg2', 'Alg3', 'Alg4']
efficiency_scores = np.array([87, 92, 88, 87])
efficiency_errors = np.array([7, 8, 7, 9])

# Data for the third plot (another style - line with error bars)
models = ['Model A', 'Model B', 'Model C', 'Model D']
dry_conditions = np.array([94.5, 85.8, 91.9, 87.9])
dry_errors = np.array([1.5, 2.0, 1.8, 2.2])
wet_conditions = np.array([83.4, 88.9, 76.1, 76.4])
wet_errors = np.array([1.2, 1.8, 2.0, 2.5])

# Create a figure with three subplots arranged vertically
fig, axs = plt.subplots(2, 1, figsize=(10.0, 10.0)) # Two rows, one column

# Plot 1: Bar chart with error bars (similar to the example)
axs[0].bar(categories_1, values_1, yerr=errors_1, capsize=7, color='skyblue',
ecolor='gray')
axs[0].set_title('AI Task Success Rates', fontsize=14)
axs[0].set_ylabel('Success Rate (%)', fontsize=12)
axs[0].tick_params(axis='x', labelsize=12)
axs[0].tick_params(axis='y', labelsize=12)
axs[0].grid(True, linestyle='-', alpha=0.7)

# Plot 2: Scatter plot with error bars
axs[1].errorbar(algorithms, efficiency_scores, yerr=efficiency_errors, fmt='o',
capsize=5,
color='lightcoral', ecolor='darkgray', markersize=10, linestyle='None')
axs[1].set_title('Algorithm Efficiency Scores', fontsize=14)
axs[1].set_ylabel('Efficiency (%)', fontsize=12)
axs[1].tick_params(axis='x', labelsize=12)
axs[1].tick_params(axis='y', labelsize=12)
axs[1].grid(True, linestyle='-', alpha=0.7)
axs[1].set_ylim(60, 110) # Adjust y-limit for better visualization

# Let's create a new figure for the third plot to avoid overcrowding the
existing one
# and demonstrate a different style of error bar plot.
fig2, ax3 = plt.subplots(1, 1, figsize=(8.0, 6.0))
x_pos = np.arange(len(models))
width = 0.35
ax3.errorbar(x_pos - width/2, dry_conditions, yerr=dry_errors, fmt='-o',
capsize=7, label='Dry Conditions', color='green', ecolor='gray', markersize=8)
ax3.errorbar(x_pos + width/2, wet_conditions, yerr=wet_errors, fmt='-s',
capsize=7, label='Wet Conditions', color='blue', ecolor='gray', markersize=8)
ax3.set_xticks(x_pos)
ax3.set_xticklabels(models, fontsize=12)
ax3.set_ylabel('Adaptability Score (%)', fontsize=12)
ax3.set_title('AI Model Adaptability', fontsize=14)
ax3.tick_params(axis='y', labelsize=12)
ax3.legend(title='Conditions', fontsize=12, title_fontsize=12)
ax3.grid(True, linestyle='-', alpha=0.7)

# Adjust layout to prevent overlapping titles/labels
plt.tight_layout()
plt.show()
```

MSRL Output

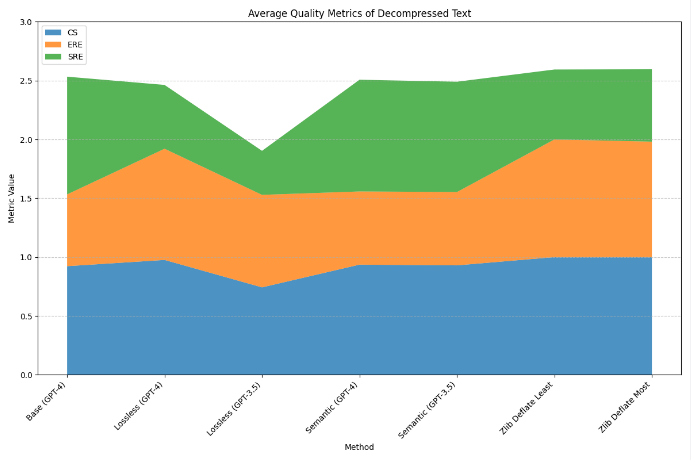
```
import matplotlib.pyplot as plt
import numpy as np

# Data for the first plot (similar to fig1a)
categories_a = ['Image Recog.', 'Speech Recog.', 'Language Proc.', 'Game AI']
values_a = [85, 78, 92, 88]
errors_a = [5, 7, 4, 6]

# Data for the second plot (similar to fig1b)
categories_b = ['Alg1', 'Alg2', 'Alg3', 'Alg4']
values_b = [87, 92, 88, 87]
errors_b = [4, 8, 7, 9]

# Data for the third plot (similar to fig1c)
categories_c = ['Model A', 'Model B', 'Model C', 'Model D']
values_c_dry = [94.5, 85.8, 92.0, 88.0]
errors_c_dry = [1.5, 2.0, 1.8, 2.2]
values_c_wet = [83.5, 88.8, 76.0, 76.2]
errors_c_wet = [2.0, 2.2, 2.5, 2.3]

# Create a figure with three subplots arranged vertically
fig, axes = plt.subplots(3, 1, figsize=(10.0, 10.0))

# Plot 1: Bar chart with error bars (similar to fig1a)
x_pos_a = np.arange(len(categories_a))
axes[0].bar(x_pos_a, values_a, yerr=errors_a, capsize=5, color='skyblue')
axes[0].set_xticks(x_pos_a)
axes[0].set_xticklabels(categories_a)
axes[0].set_ylabel('Success Rate (%)')
axes[0].set_title('AI Task Success Rates')
axes[0].grid(True)
axes[0].set_ylim(0, 100)

# Plot 2: Scatter plot with error bars (similar to fig1b)
x_pos_b = np.arange(len(categories_b))
axes[1].errorbar(x_pos_b, values_b, yerr=errors_b, fmt='o', capsize=5,
color='lightcoral', markersize=8)
axes[1].set_xticks(x_pos_b)
axes[1].set_xticklabels(categories_b)
axes[1].set_ylabel('Efficiency (%)')
axes[1].set_title('Algorithm Efficiency Scores')
axes[1].grid(True)
axes[1].set_ylim(60, 110)

# Plot 3: Line plot with error bars (similar to fig1c)
x_pos_c = np.arange(len(categories_c))
axes[2].errorbar(x_pos_c, values_c_dry, yerr=errors_c_dry, fmt='-o', capsize=5,
label='Dry Conditions', color='green')
axes[2].errorbar(x_pos_c, values_c_wet, yerr=errors_c_wet, fmt='-s', capsize=5,
label='Wet Conditions', color='blue')
axes[2].set_xticks(x_pos_c)
axes[2].set_xticklabels(categories_c)
axes[2].set_ylabel('Adaptability Score (%)')
axes[2].set_title('AI Model Adaptability')
axes[2].legend(title='Conditions')
axes[2].grid(True)
axes[2].set_ylim(75, 98)

# Adjust layout to prevent overlap
plt.tight_layout()

# Show the plot
plt.show()
```

Qwen2.5-VL-7B Output

```
import matplotlib.pyplot as plt
import numpy as np

fig, axs = plt.subplots(nrows=2)
fig.set_size_inches(10.0, 10.0)

# Example data
success_data = (84.5, 83.3, 78.5, 85.0)
efficiency_data = (86.2, 91.2, 88.6, 87.9)
adaptability_data = (95, 88, 100, 90, 86, 75, 76, 88)

# First plot
axs[0].bar(range(len(success_data)), success_data,
yerr=np.ones_like(success_data))
axs[0].set_title('AI Task Success Rates')
axs[0].set_xticks(range(len(success_data)))
axs[0].set_xticklabels(['Image Recog.', 'Speech Recog.', 'Language Proc.', 'Game
AI'])
axs[0].set_ylabel('Success Rate (%)')

# Second plot
axs[1].errorbar(range(1, 5), efficiency_data, yerr=np.ones_like(efficiency_data),
fmt='o', color='tab:pink')
axs[1].set_title('Algorithm Efficiency Scores')
axs[1].set_xticks(range(1, 5))
axs[1].set_xticklabels(['Alg1', 'Alg2', 'Alg3', 'Alg4'])
axs[1].set_ylabel('Efficiency (%)')

# Third plot
x = np.arange(4)
width = 0.35
axs[1].bar(x - width / 2, adaptability_data[::2], width, label='Dry Conditions',
color='tab:green')
axs[1].bar(x + width / 2, adaptability_data[1::2], width, label='Wet Conditions',
color='tab:blue')
axs[1].set_title('AI Model Adaptability')
axs[1].set_xticks(x)
axs[1].set_xticklabels(['Model A', 'Model B', 'Model C', 'Model D'])
axs[1].set_ylabel('Adaptability Score (%)')
axs[1].legend()

plt.show()
```

Figure F: An example of comparing the code generated by different models with rendered images.

Ground Truth Image

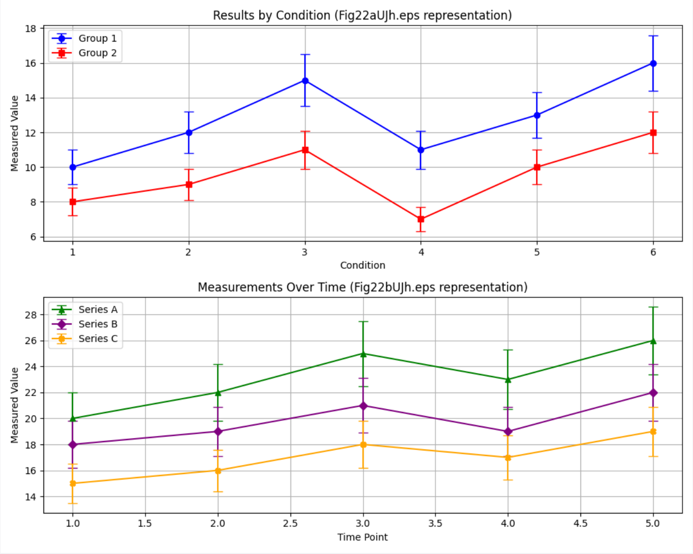


Ground Truth Code

```
import matplotlib.pyplot as plt
import numpy as np

# Data from the table
methods = [
'Base (GPT-4)',
'Lossless (GPT-4)',
'Lossless (GPT-3.5)',
'Semantic (GPT-4)',
'Semantic (GPT-3.5)',
'Zlib Deflate Least',
'Zlib Deflate Most'
]
cs_values = [0.923, 0.976, 0.743, 0.936, 0.93, 1.0, 1.0]
ere_values = [0.61, 0.945, 0.786, 0.622, 0.623, 1.0, 0.981]
sre_values = [1.0, 0.542, 0.374, 0.949, 0.937, 0.594, 0.615]

# Create a figure and axes
fig, ax = plt.subplots(figsize=(12, 8))
# Convert methods to numerical x-axis values
x = np.arange(len(methods))
# Plot the area chart
ax. stackplot(x, cs_values, ere_values, sre_values, labels=['CS', 'ERE', 'SRE'], alpha=0.8)
# Add labels and title
ax.set_xlabel('Method')
ax.set_ylabel('Metric Value')
ax.set_title('Average Quality Metrics of Decompressed Text')
ax.set_xticks(x)
ax.set_xticklabels(methods, rotation=45, ha='right')
# Add a legend
ax.legend(loc='upper left')
# Set y-axis limits
ax.set_ylim(0, 3.0) # Adjust based on the sum of values
# Add a grid for better readability
ax.grid(axis='y', linestyle='--', alpha=0.7)

# Ensure layout is tight to prevent labels from overlapping
plt.tight_layout()
# Show the plot
plt.show()
```

Figure G: A case of chart-code pairs for area charts in our Chart2Code dataset.

Ground Truth Image

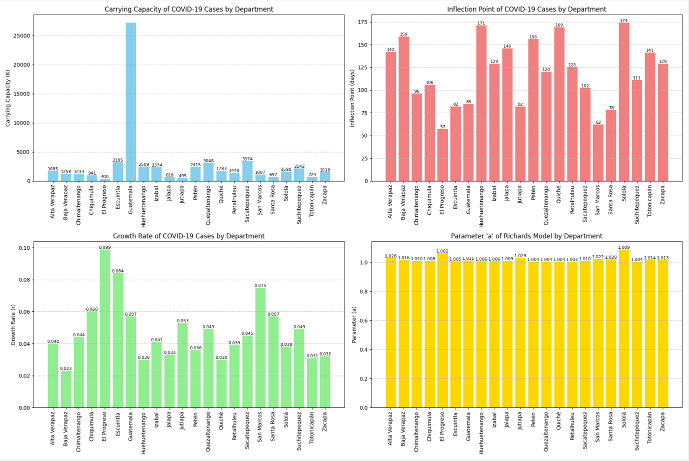


Ground Truth Code

```python
import matplotlib.pyplot as plt
import numpy as np


# Data for the first plot (corresponding to Fig22aUJh.eps)
x_values1 = np.array([1, 2, 3, 4, 5, 6])
data_points1_group1 = np.array([10, 12, 15, 11, 13, 16])
errors1_group1 = np.array([1, 1.2, 1.5, 1.1, 1.3, 1.6])
data_points1_group2 = np.array([8, 9, 11, 7, 10, 12])
errors1_group2 = np.array([0.8, 0.9, 1.1, 0.7, 1.0, 1.2])

# Data for the second plot (corresponding to Fig22bUJh.eps)
x_values2 = np.array([1, 2, 3, 4, 5])
data_points2_seriesA = np.array([20, 22, 25, 23, 26])
errors2_seriesA = np.array([2, 2.2, 2.5, 2.3, 2.6])
data_points2_seriesB = np.array([18, 19, 21, 19, 22])
errors2_seriesB = np.array([1.8, 1.9, 2.1, 1.9, 2.2])
data_points2_seriesC = np.array([15, 16, 18, 17, 19])
errors2_seriesC = np.array([1.5, 1.6, 1.8, 1.7, 1.9])

# Create a figure with two subplots (arranged vertically)
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))
# Plot the first errorbar chart on the first subplot (ax1)
ax1.errorbar(x_values1, data_points1_group1, yerr=errors1_group1, label='Group 1', fmt='-o', capsize=5,
color='blue')
ax1.errorbar(x_values1, data_points1_group2, yerr=errors1_group2, label='Group 2', fmt='-s', capsize=5,
color='red')
ax1.set_title('Results by Condition (Fig22aUJh.eps representation)')
ax1.set_xlabel('Condition')
ax1.set_ylabel('Measured Value')
ax1.legend()
ax1.grid(True)

# Plot the second errorbar chart on the second subplot (ax2)
ax2.errorbar(x_values2, data_points2_seriesA, yerr=errors2_seriesA, label='Series A', fmt='-^',
capsize=5, color='green')
ax2.errorbar(x_values2, data_points2_seriesB, yerr=errors2_seriesB, label='Series B', fmt='-D',
capsize=5, color='purple')
ax2.errorbar(x_values2, data_points2_seriesC, yerr=errors2_seriesC, label='Series C', fmt='-X',
capsize=5, color='orange')
ax2.set_title('Measurements Over Time (Fig22bUJh.eps representation)')
ax2.set_xlabel('Time Point')
ax2.set_ylabel('Measured Value')
ax2.legend()
ax2.grid(True)

# Adjust layout to prevent overlap
plt.tight_layout()
# Show the plot
plt.show()
```

Figure H: A case of chart-code pairs for errorbar charts in our Chart2Code dataset.

Ground Truth Image



Ground Truth Code

```python
import matplotlib.pyplot as plt
# Data from the table
departments = [
'Alta Verapaz', 'Baja Verapaz', 'Chimaltenango', 'Chiquimula', 'El Progreso',
'Escuintla', 'Guatemala', 'Huehuetenango', 'Izabal', 'Jalapa', 'Jutiapa',
'Petén', 'Quezaltenango', 'Quiché', 'Retalhuleu', 'Sacatepequez', 'San Marcos',
'Santa Rosa', 'Sololá', 'Suchitepequez', 'Totonicapán', 'Zacapa'
]
a = [1.028, 1.016, 1.010, 1.008, 1.062, 1.005, 1.011, 1.006, 1.006, 1.009, 1.029, 1.004, 1.004, 1.004, 1.003, 1.010,
1.022, 1.020, 1.089, 1.004, 1.014, 1.013]
growth_rate = [0.040, 0.023, 0.044, 0.060, 0.099, 0.084, 0.057, 0.030, 0.041, 0.033, 0.053, 0.036, 0.049,
0.030, 0.039, 0.045, 0.075, 0.057, 0.038, 0.049, 0.031, 0.032]
carrying_capacity = [1685, 1258, 1233, 941, 400, 3195, 27230, 2509, 2374, 618, 495, 2415, 3048, 1783,
1448, 3374, 1087, 697, 1598, 2142, 723, 1518]
inflection_point = [142, 159, 96, 106, 57, 82, 85, 171, 129, 146, 82, 156, 120, 169, 125, 102, 62, 78, 174, 111,
141, 129]

# Create subplots
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(18, 12))
axes = axes.flatten()
# Plotting Carrying Capacity
ax1 = axes[0]
bars_cc = ax1.bar(departments, carrying_capacity, color='skyblue')
ax1.set_ylabel('Carrying Capacity (K)')
ax1.set_title('Carrying Capacity of COVID-19 Cases by Department')
ax1.tick_params(axis='x', rotation=90)
ax1.grid(axis='y', linestyle='--', alpha=0.7)
for bar in bars_cc:
        height = bar.get_height()
        if height < 10000:  # Only show values for smaller bars to avoid clutter
                ax1.text(bar.get_x() + bar.get_width()/2., height,
                '%d' % int(height), ha='center', va='bottom', fontsize=8)
# Plotting Inflection Point
ax2 = axes[1]
bars_ip = ax2.bar(departments, inflection_point, color='lightcoral')
ax2.set_ylabel('Inflection Point (days)')
ax2.set_title('Inflection Point of COVID-19 Cases by Department')
ax2.tick_params(axis='x', rotation=90)
ax2.grid(axis='y', linestyle='--', alpha=0.7)
for bar in bars_ip:
        height = bar.get_height()
        ax2.text(bar.get_x() + bar.get_width()/2., height,
        '%d' % int(height), ha='center', va='bottom', fontsize=8)
# Plotting Growth Rate
ax3 = axes[2]
bars_gr = ax3.bar(departments, growth_rate, color='lightgreen')
ax3.set_ylabel('Growth Rate (r)')
ax3.set_title('Growth Rate of COVID-19 Cases by Department')
ax3.tick_params(axis='x', rotation=90)
ax3.grid(axis='y', linestyle='--', alpha=0.7)
for bar in bars_gr:
        height = bar.get_height()
        ax3.text(bar.get_x() + bar.get_width()/2., height,
        '%.3f' % height, ha='center', va='bottom', fontsize=8)
# Plotting 'a' parameter
ax4 = axes[3]
bars_a = ax4.bar(departments, a, color='gold')
ax4.set_ylabel('Parameter (a)')
ax4.set_title('Parameter \'a\' of Richards Model by Department')
ax4.tick_params(axis='x', rotation=90)
ax4.grid(axis='y', linestyle='--', alpha=0.7)
for bar in bars_a:
        height = bar.get_height()
        ax4.text(bar.get_x() + bar.get_width()/2., height,
        '%.3f' % height, ha='center', va='bottom', fontsize=8)
plt.tight_layout()
plt.show()
```

Figure I: A case of chart-code pairs for bar_num charts in our Chart2Code dataset.