CHARTMIMIC: EVALUATING LMM'S CROSS-MODAL REASONING CAPABILITY VIA CHART-TO-CODE GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce a new benchmark, ChartMimic, aimed at assessing the visually-grounded code generation capabilities of large multimodal models (LMMs). ChartMimic utilizes information-intensive visual charts and textual instructions as inputs, requiring LMMs to generate the corresponding code for chart rendering. ChartMimic includes $4,800$ human-curated (figure, instruction, code) triplets, which represent the authentic chart use cases found in scientific papers across various domains (e.g., Physics, Computer Science, Economics, etc). These charts span 18 regular types and 4 advanced types, diversifying into 201 subcategories. Furthermore, we propose multi-level evaluation metrics to provide an automatic and thorough assessment of the output code and the rendered charts. Unlike existing code generation benchmarks, ChartMimic places emphasis on evaluating LMMs' capacity to harmonize a blend of cognitive capabilities, encompassing visual understanding, code generation, and cross-modal reasoning. The evaluation of 3 proprietary models and 14 open-weight models highlights the substantial challenges posed by ChartMimic. Even the advanced GPT-4o, InternVL2-Llama3-76B only achieve an average score across Direct Mimic and Customized Mimic tasks of 82.2 and 61.6, respectively, indicating significant room for improvement. We anticipate that ChartMimic will inspire the development of LMMs, advancing the pursuit of artificial general intelligence.

1 INTRODUCTION

Code generation (Sun et al., 2024) is a rather demanding task that requires advanced abstract thinking and logical reasoning, reflecting the unique intelligence of human beings. Recently, advances in artificial general intelligence (AGI) have demonstrated the potential of large foundation models (Google, 2023; OpenAI, 2024; Anthropic, 2024; AI@Meta, 2024) to solve the tasks that are once the exclusive domain of human abilities (Achiam et al., 2023; Zhu et al., 2024). However, existing code generation benchmarks (Chen et al., 2021; Austin et al., 2021; Hendrycks et al., 2021; Lai et al., 2023) solely use text as input, while humans receive information from multiple modalities when coding (Liang et al., 2023; Fan et al., 2024). Such real-life scenarios have yet to be fully explored.

Taking a common scene in Fig. 1, researchers often need to write code for data visualization and may already have preferred chart templates at hand. However, they usually lack either the source code or the expertise to reproduce these chart templates. As a result, they turn to large multimodal models (LMMs) as assistants to aid in code generation. In this scenario, coding for scientific charts entails *code generation grounded on visual understanding* (i.e., chart-to-code generation), which necessitates LMMs to integrate a variety of advanced cognitive capabilities, including visual understanding, code generation, and cross-modal reasoning. Therefore, evaluating the performance of LMMs on this real-world task also enables researchers to pinpoint potential areas for improving models' capabilities.

To this end, we present **ChartMimic** (Fig. 2), a multimodal code generation benchmark. ChartMimic is characterized by its *(1) information-intensive visual inputs*, *(2) diverse chart types*, and *(3) multi-level evaluation metrics*. Specifically, compared to natural images, scientific charts convey nuanced semantic meanings through intricate visual logic, thereby exhibiting higher information density. Based on this, we define two tasks, Direct Mimic and Customized Mimic (Sec. 2.1), which utilize charts and textual instructions as inputs. These tasks challenge LMMs to generate the corresponding

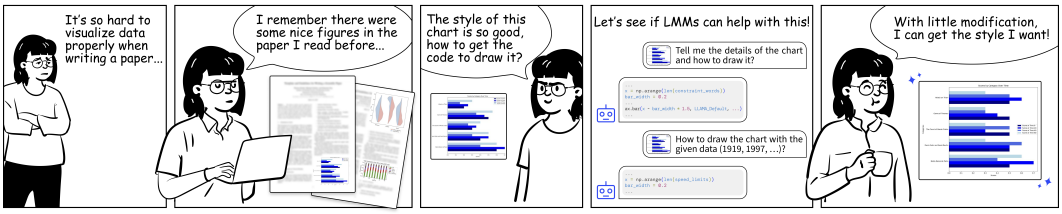


Figure 1: The real-world example. LMMs assist scientists and researchers in understanding, interpreting and creating charts during the reading and writing of academic papers. These models serve as assistants that enhance the comprehension and presentation of data in scholarly communications.

code for a given chart or to incorporate new data specified in the instructions, respectively. Through the collection of academic documents and scientific papers, we identify 22 commonly used chart types and 201 subcategories. Subsequently, we manually annotate a total of $4,800$ (figure, instruction, code) triplets for these types (Sec. 2.2). Furthermore, we establish automatic evaluation metrics from both high-level and low-level perspectives to thoroughly assess the performance of LMMs (Sec. 2.4).

We conduct examination of 17 LMMs on ChartMimic (Sec. 3.2), including 3 proprietary models and 14 open-weight models across parameter sizes from 2.2B to 76.0B. We observe that while several open-weight models can match the performance of proprietary models such as GPT-4o on public leaderboards (OpenCompass, 2023), a significant performance gap still persists on ChartMimic. Specifically, the best open-weight model, InternVL2-Llama3-76B, lags behind GPT-4o, with an average score gap of 20.6 on two tasks, indicating substantial room for improvement in open-source community. Our analysis of prompting methods (Sec. 4.2) reveals that GPT-4o can improve itself through self-reflection, which is a key manifestation of System 2 reasoning (Sloman, 1996; Kumar et al., 2024). This discovery highlights the vital role that System 2 reasoning process plays in LMMs when tackling the complex challenges presented by ChartMimic. Meanwhile, Correlation analysis (Sec. 4.3) demonstrates a high correlation between our automatic metrics and human evaluation, validating the effectiveness of these metrics. Further error analysis (Sec. 4.4) reveals that hallucinations notably hinder the performance of LLMs on ChartMimic, as they lead to the insertion of non-existent text into ground-truth figures and confusion between similar types of charts.

We envision ChartMimic as a comprehensive suite of benchmarks designed to guide researchers in understanding their LMMs' capabilities. By providing a comprehensive evaluation framework, ChartMimic aims to facilitate the growth of foundation models for the community, offering insights into various aspects such as visual understanding, code generation, and cross-modal reasoning.

2 THE CHARTMIMIC BENCHMARK

In this section, we first introduce the definition of two tasks involved in ChartMimic (Sec. 2.1), and then delineate the data curation process (Sec. 2.2). Subsequently, we conduct a quantitative analysis to assess the quality and diversity of ChartMimic (Sec. 2.3), establish evaluation metrics (Sec. 2.4), and compare it with existing related benchmarks (Sec. 2.5).

2.1 TASK DEFINITION

LMMs' ability to generate chart-rendering code demonstrates their visual understanding, coding, and cross-modal reasoning skills. Specifically, given the chart X and the instruction I, the LMM f is expected to output the code C that satisfies the requirements outlined in the instruction:

$$C = f(X, I). \tag{1}$$

As shown in Fig. 2, based on the information provided in the instructions, we propose two tasks:

Direct Mimic. The LMMs are tasked to directly generate code that can reproduce the provided chart, thereby assessing their visual comprehension and reasoning capabilities.

Customized Mimic. The LMMs are requested to generate code for a new chart that incorporates customized data provided in the instruction while preserving the original chart's aesthetic and design, assessing their ability to integrate visual and textual information.

After obtaining the code generated by LMMs, we execute it to render the corresponding chart and subsequently compare its similarity with the ground-truth chart. Illustrative examples of the two tasks

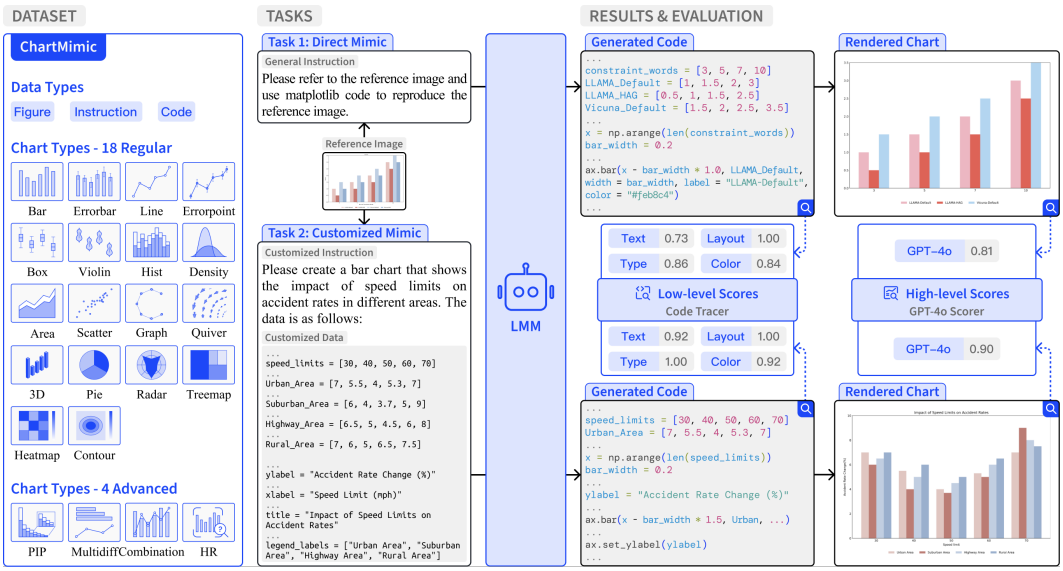


Figure 2: The pipeline of ChartMimic. We provide 4,800 human-curated (figure, instruction, code) triplets. We use ChartMimic to evaluate LMMs' proficiency in the multimodal chart-to-code generation, resulting in both high-level and low-level evaluation results.

are shown in Fig. 2. To accommodate the defined tasks above, we propose ChartMimic, a benchmark designed to evaluate the comprehension of charts and their conversion into executable code.

2.2 Data Curation Process

ChartMimic distinguishes itself through 4 fundamental considerations: *(1) diversity of chart types, (2) balance of chart complexity, (3) reduction of data leakage, (4) integration of authentic user requirements.* We keep on these four principles to complete the data curation for ChartMimic through a five-step pipeline. Here, we provide an overview here and more details in Appendix A.

General Filtering. We scrape figures from source files of publications on arXiv[1] that hold a CC BY 4.0 license with a publication date after February 2024 with PDF format. This yields approximately 174,100 figures across various domains (e.g., Physics, Computer Science, Economics, etc). We then filter the figures based on the criteria of how designers select inspiring visualization examples, including clarity and visual appeal, color schemes (Bako et al., 2022; Quispel et al., 2018), and uniqueness of the chart within its category, resulting in a refined collection of about 15,800 figures.

Diversity and Information Density Filtering. The filtering process involves two stages. In the first stage, we establish a data pool and categorize chart types. Charts with significant differences in complexity or information density are added to ensure diversity and effective communication (Bako et al., 2022). In the second stage, five experts from various fields independently evaluate the data, creating separate selection pools. We preserve the intersection of their selections and finalize the set through a voting process. This meticulous approach refines our collection to 279 charts.

Manual Selection and Filling. In addition to sourcing from arXiv, we also collect charts from various platforms, including the Matplotlib gallery, Stack Overflow, and plotting-related forums on Twitter and Reddit. These charts are selected for their unique styles not represented in our arXiv curated 279 charts. To mitigate the risk of data leakage, we rigorously process the data and color styling of these charts, replacing existing color schemes with those unseen in our data pool while maintaining their aesthetic appeal. Consequently, we obtain 600 prototype charts for ChartMimic.

Code and Instruction Writing. We propose to manually write codes and instructions for ChartMimic based on the collected 600 prototype charts. Initially, annotators are tasked with meticulously reproducing the 600 prototype charts using Python code, resulting in a set of 600 (figure, code, instruction) triplets for the Direct Mimic task. Although other coding languages such as JavaScript and R can be used to create charts, current LMMs perform poorly when doing chart-to-code using these other languages (Sun et al., 2024). Therefore, we focus on Python code generation in the current

[1] https://arxiv.org/

Table 1: Statistics of ChartMimic. We measure code length in terms of tokens, utilizing the Llama3 tokenizer. In the level count, "A/B/C" denotes the number of "easy/medium/hard" level, respectively.

Type	Bar	Line	ErrorBar	Heatmap	Box	Scatters	Hist	Radar	3D	Pie	ErrorPoint	Violin
Count	320	280	120	120	100	100	80	80	80	80	80	80
Subcategories	16	8	12	4	6	4	3	6	5	8	5	3
Code Length (AVG.)	689.6	794.0	681.2	685.8	689.0	655.0	529.6	779.8	655.4	418.4	624.3	975.6
Code Length (STD.)	237.8	244.4	144.7	258.7	228.2	253.0	147.1	144.3	241.4	99.5	197.7	252.3
Level Count	176 / 120 / 24	256 / 24 / 0	68 / 52 / 0	0 / 76 / 44	60 / 40 / 0	80 / 20 / 0	52 / 28 / 0	52 / 28 / 0	8 / 48 / 24	52 / 28 / 0	44 / 28 / 8	32 / 44 / 4

Type	Area	Contour	Density	Graph	Quiver	Treemap	Combination	HR	Muiltidiff	PIP	Total
Count	80	80	80	80	80	80	120	100	100	80	2400
Subcategories	2	3	4	4	4	4	30	25	25	20	201(101 + 100)
Code Length (AVG.)	774.4	489.4	540.0	564.5	893.4	342.2	697.4	718.9	798.2	1083.9	696.0
Code Length (STD.)	161.8	87.8	104.7	117.5	631.0	36.3	163.6	265.5	271.2	290.1	278.4
Level Count	52 / 28 / 0	0 / 28 / 52	44 / 32 / 4	56 / 24 / 0	0 / 52 / 28	52 / 28 / 0	12 / 76 / 32	4 / 16 / 80	0 / 48 / 52	0 / 0 / 80	1100 / 868 / 432

version of ChartMimic. Subsequently, to simulate the scenario of Customized Mimic, annotators are instructed to modify the chart data in the Direct Mimic task by integrating new data from various domains. They are then required to modify the corresponding code and instructions, leading to the 600 (figure, code, instruction) triplets for the Customized Mimic task. Consequently, we establish the ChartMimic benchmark, comprising $1,200$ high-quality seed data.

Data Augmentation. Following the development of seed triplets, we initiate a process of manual data augmentation. Annotators are tasked with altering various elements of each seed triplet, including data, color schemes, mark styles, etc., to produce augmented triplets. For each seed triplet, we create 3 additional augmented triplets. This process enhances our dataset, yielding a total of $4,800$ triplets that reflect a wide range of realistic and practical chart use cases.

2.3 Data Statistics and Division

The (figure,code,instruction) triplets for both Direct and Customized Mimic tasks share the same figure. Therefore, we detail the data statistics for the $2,400$ triplets in the Direct Mimic task. As depicted on the left side of Fig. 2, ChartMimic encompasses a total of 22 categories, with 18 types of regular charts and 4 types of advanced charts. For the 18 regular chart types, we identify 101 subcategories, with the definitions and examples of each subcategory provided in Appendix C. The advanced chart types such as Plot-in-Plot (PIP), Multidiff, and Combination are distinct forms of amalgamating multiple chart sets into a singular chart. Given the variety of their internal combination elements, each of these can be treated as a unique subcategory. Meanwhile, the Hard-to-Recognize (HR) category encapsulates unclassifiable charts, with each chart in the seed data being considered a category unto itself. When we factor in the additional 100 subcategories represented by these advanced chart types, ChartMimic encompasses a total of 201 subcategories. This extensive diversity underscores the comprehensive nature of our benchmark. We employ Llama3 (AI@Meta, 2024) tokenizer to measure the code length. As shown in Tab. 1, ChartMimic has average code token length of 696.0 with a standard deviation of 278.4. Addtionally, we manually categorize charts into three complexity levels: easy (1,100), medium (868), and hard (432). The detailed categorization criteria and assessment methodology are thoroughly documented in Appendix A.

We further divide the $4,800$ examples of ChartMimic into two subsets: test and testmini set. The test set comprises $3,600$ examples, while the testmini set is composed of $1,200$ examples. The testmini set is designed for rapid model development validation. Our partitioning strategy ensures each chart type is proportionally represented, preserving a distribution in the testmini set that closely aligns with the test set. Detailed comparative experimental results, discussed in Appendix B, demonstrate the consistency across two subsets. Unless otherwise stated, we report results on the testmini set.

2.4 Evaluation Metrics

For tasks within ChartMimic, an appropriate evaluation necessitates comparing the visual similarity between the generated and ground-truth figures. To achieve this, we propose multi-level metrics (i.e., high-level and low-level) to assess the similarity at different granularities. Specifically, the high-level metric encompasses GPT-4o Score and the low-level metric encompasses Text, Layout, Type and Color Score. We compute the average score between the **high-level** and **low-level scores**, ranging from 0 to 100, as the **overall score**. The illustration of the multi-level metrics is depicted in Fig. 2.

GPT-4o Score. Following the successful use of large foundation models for evaluation in both natural language processing (Zheng et al., 2024; Li et al., 2023; Dubois et al., 2024) and computer vision (Zhang et al., 2023; Yang et al., 2024; Wu et al., 2024b), we adopt GPT-4o score as our high-

Table 2: A comparison of our proposed ChartMimic to other benchmarks. "I" and "NL" indicate "Image" and "Natural Language" respectively.

Benchmarks	Source	# of Chart Types	# of Test Instances	Input Format	Output Format	Evaluation Metric
Chart Understanding Benchmarks						
ChartQA (Masry et al., 2022)	Human Curated	3	10K	I+NL	NL	Accuracy
Chart-to-Text (Kantharaj et al., 2022)	Crawl	6	44K	I+NL	NL	Match-based
ChartSumm (Rahman et al., 2023)	Human Curated	3	84K	I+NL	NL	Match-based
CharArXiv (Wang et al., 2024b)	Human Curated	18	93K	I+NL	NL	GPT-4 Score
Code Generation Benchmarks						
HumanEval (Chen et al., 2021)	Human Curated	-	164	Code	Code	Pass Rate
MBPP (Austin et al., 2021)	Human Curated	-	500	NL+Code	Code	Pass Rate
MMCode (Li et al., 2024b)	Crawl	-	263	I+NL	Code	Pass Rate
MatPlotBench (Yang et al., 2024)	Human Curated	13	100	NL	Code	GPT-4 Score
Plot2Code (Wu et al., 2024a)	Crawl	15	132	I+NL	Code	Multi-Level
Design2Code (Si et al., 2024)	Crawl	HTML	484	I+NL	Code	Multi-Level
ChartMimic (Ours)	Human Curated	22	4,800	I+NL	Code	Multi-Level

level metric. Specifically, we input both the ground-truth figure and the generated figure into GPT-4o, and instruct it to output a high-level similarity score ranging 0 to 100. Although CLIP Score (Radford et al., 2021) is widely used for assessing image similarity, in our preliminary experiments, it has struggled to distinguish variations in types and other critical elements in charts, resulting in a low correlation with human evaluation results. Therefore, we use only GPT-4o Score as our high-level evaluation metric. The detailed description of GPT-4o Score can be found in Appendix D.

In addition to high-level similarity, evaluating the similarity among low-level elements between generated and ground-truth figures can provide a more fine-grained analysis of LMMs. Therefore, we propose to evaluate four key low-level elements in charts (Savva et al., 2011; Poco & Heer, 2017): text, layout, type and color. Extracting them from figures is a challenging task, as existing extraction models often fall short in terms of accuracy (Meng et al., 2024). Considering that figures are rendered based on the code, we design a code tracer to monitor the execution processes of the ground-truth code and generated code. The code tracer records the text, layout, type and color information. We calculate the F1 score of these elements as their corresponding score. And we average text, layout, type and color scores to get the low-level score. The methodology for obtaining these elements is briefly introduced as follows, with detailed descriptions provided in Appendix D.

Text Score. During the code execution process, for the function responsible for adding text elements to the rendered figures, the code tracer monitors it and records each text element parameter.

Layout Score. Layout refers to the arrangement of subplots in the figure. At the end of the code execution, the code tracer traverses all subplots in the figure and obtains their layout information.

Type Score. For each plot function, which serves the purpose of adding a specific chart type instance to the figure, the code tracer monitors its invocation status and records every invoked plot function.

Color Score. For each plot function, it will return the chart type instance at the end of the function invocation. The code tracer accesses the color attributes of these chart type instances.

It is important to note that code execution success rate is a standard metric for code generation tasks (Sun et al., 2024). We have implicitly incorporated this aspect into our high-level and low-level scores. Specifically, if the code fails to execute successfully, both the low-level and high-level scores are assigned a value of 0. Therefore, we do not separately weight it into the overall score.

2.5 Comparisons with Existing Benchmarks

To further distinguish the difference between ChartMimic and other existing ones, we elaborate the benchmark details in Tab. 2. From the chart understanding perspective, the prior benchmarks (Masry et al., 2022; Kantharaj et al., 2022; Rahman et al., 2023; Wang et al., 2024b) are only focused on questions about the data in the charts without necessitating the advanced cognitive capabilities of LMMs, which include visual understanding, code generation, and cross-modal reasoning. In the code generation aspect, HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021) and MatPlotBench (Yang et al., 2024) only consider tasks with text inputs, which may not meet requirements in the era of LMMs. Recently, MMCode (Li et al., 2024b) attempted to create a benchmark for multimodal code generation, but the vision inputs for their task are still overly simple and only have a single pass rate evaluation metric. Design2Code (Si et al., 2024) and Plot2Code (Wu et al., 2024a) are the most similar one to ours. Although they use multi-level evaluation metrics like ours, their

Table 3: The ChartMimic leaderboard with **Direct Mimic** task. The best scores are in **bold**. We also include the code execution success rate (Exec. Rate) and model size (Params).

Model	Params	Exec. Rate	Low-Level					High-Level	Overall
			Text	Layout	Type	Color	Avg.	GPT-4o	
Proprietary									
GeminiProVision	-	68.2	52.6	64.2	51.3	47.1	53.8	53.3	53.6
Claude-3-opus	-	83.3	66.8	83.1	49.9	42.1	60.5	60.1	60.3
GPT-4o	-	**93.2**	**81.5**	**89.8**	**77.3**	**67.2**	**79.0**	**83.5**	**81.2**
Open-Weight									
IDEFICS2-8B	7.6B	49.0	6.2	33.1	9.2	9.0	14.4	17.6	16.0
DeepSeek-VL-7B	7.3B	41.3	15.3	26.6	19.7	14.5	19.0	20.4	19.7
LLaVA-Next-Yi-34B	34.8B	50.2	15.9	29.6	17.6	15.2	19.6	20.6	20.1
LLaVA-Next-Mistral-7B	7.6B	59.7	14.0	31.1	19.8	17.8	20.7	21.3	21.0
Qwen2-VL-2B	2.6B	47.0	20.1	29.5	21.3	17.9	22.2	23.4	22.8
Cogvlm2-llama3-chat-19B	19.2B	50.5	21.3	31.8	18.4	17.0	22.1	24.5	23.3
InternVL2-2B	2.2B	52.5	23.6	35.8	16.0	15.4	22.7	24.2	23.5
Qwen2-VL-7B	8.2B	67.0	26.4	51.0	31.0	23.3	32.9	35.0	34.0
InternVL2-4B	4.2B	66.2	34.7	51.7	25.2	23.6	33.8	38.4	36.1
InternVL2-8B	8.1B	61.8	31.5	51.1	28.6	26.2	34.4	38.9	36.6
MiniCPM-Llama3-V-2.5	8.4B	80.3	30.7	49.6	38.6	27.6	36.6	42.1	39.4
Phi-3-Vision-128K	4.2B	66.7	37.5	49.6	37.4	29.8	38.6	41.0	39.8
InternVL2-26B	26.0B	69.3	39.2	58.7	35.9	31.8	41.4	47.4	44.4
InternVL2-Llama3-76B	76.0B	**83.2**	**54.1**	**74.5**	**49.2**	**41.5**	**54.8**	**62.2**	**58.5**

test are crawled directly from the internet or existing dataset, which may pose a risk of data leakage. Our ChartMimic benchmark gives complex scientific charts as inputs, demanding capabilities on grounding visual understanding into code generation, and provides multi-level evaluation metrics.

3 EXPERIMENT

3.1 BASELINE SETUP

We benchmark 17 widely utilized proprietary and open-source models currently available in the field. For proprietary models, we consider 3 representative models: GPT-4o (OpenAI, 2024), Claude-3-opus (Anthropic, 2024) and GeminiProVision (Google, 2023). For the open-weight models, we choose 14 competitive models with total parameter size from 2.2B to 76.0B: InternVL2(2B, 4B, 8B, 26B, 76B) (Chen et al., 2023), Qwen2-VL(2B, 7B) (Wang et al., 2024a), Phi-3-Vision (phi, 2024), DeepSeek-VL-7B (Lu et al., 2024), LLaVA-Next(7B, 34B) (Li et al., 2024a), IDEFICS2-8B (Laurençon et al., 2024), MiniCPM-Llama3-V2.5 (Xu et al., 2024) and Cogvlm2-llama3-chat-19B (Wang et al., 2023a). We start with the direct prompting, which provides the reference chart with direct instructions. The specific instructions and model configurations can be found in Appendix E.

3.2 MAIN RESULTS

We present the main results of 17 LMMs on ChartMimic. Tab. 3 and Tab. 4 show the results on the Direct Mimic and Customized Mimic task, respectively. The key findings are as follows:

GPT-4o performs best among proprietary models, while InternVL2-Llama3-76B excels among open-weight models. In proprietary models, GPT-4o achieves an overall score of 81.2 in Direct Mimic and 83.2 in Customized Mimic. Within open-weight models, InternVL2-Llama3-76B, reaches the overall score of 58.5 in Direct Mimic and 64.7 in Customized Mimic, which have compareble performance with proprietary model like Claude-3-opus (60.3 in Direct Mimic and 65.4 in Customized Mimic). Meanwhile, Phi-3-Vision-128K despite its 4.2B parameters, reaches the overall score of 39.8 in Direct Mimic and 42.1 in Customized Mimic, which outperforms LLaVA-Next-Yi-34B (20.1 in Direct Mimic and 35.3 in Customized Mimic). This indicates that even models with fewer parameters can achieve decent performance through a refined training process.

A performance gap persists between open-weight LMMs and proprietary ones. Although several open-weight LMMs can exhibit performance comparable to GPT-4o across various benchmarks (OpenCompass, 2023), even the best-performing InternVL2-Llama3-76B falls short of achieving the performance of GPT-4o both Direct Mimic and Customized Mimic. This apparent performance

Table 4: The ChartMimic leaderboard with **Customized Mimic** task. The best scores are in **bold**. We also include the code execution success rate (Exec. Rate) and model size (Params).

Model	Params	Exec. Rate	Low-Level					High-Level	Overall
			Text	Layout	Type	Color	Avg.	GPT-4o	
Proprietary									
GeminiProVision	-	76.2	52.2	70.9	56.0	49.4	57.1	59.6	58.4
Claude-3-opus	-	88.2	75.2	86.8	54.1	44.3	65.1	65.7	65.4
GPT-4o	-	**96.5**	**88.5**	**92.9**	**79.2**	**67.6**	**82.1**	**84.3**	**83.2**
Open-Weight									
Qwen2-VL-2B	2.6B	35.8	17.4	23.9	19.7	16.5	19.4	21.4	20.4
Cogvlm2-llama3-chat-19B	19.2B	38.7	19.0	27.9	16.5	15.7	19.8	21.6	20.7
LLaVA-Next-Mistral-7B	7.6B	49.0	20.0	32.0	22.6	19.9	23.6	24.7	24.2
IDEFICS2-8B	7.6B	49.2	21.6	32.2	18.1	12.2	21.0	27.3	24.2
InternVL2-2B	2.2B	49.3	22.2	35.4	20.0	18.1	23.9	27.8	25.9
LLaVA-Next-Yi-34B	34.8B	64.2	28.7	44.8	32.9	27.7	33.5	37.1	35.3
DeepSeek-VL-7B	7.3B	59.3	27.5	47.5	36.8	31.5	35.8	39.3	37.6
Phi-3-Vision-128K	4.2B	67.8	29.7	52.5	42.3	36.5	40.3	44.0	42.1
InternVL2-4B	4.2B	74.0	41.3	55.6	39.6	33.1	42.4	47.8	45.1
Qwen2-VL-7B	8.2B	73.3	41.0	56.3	43.5	34.2	43.8	47.8	45.8
InternVL2-8B	8.1B	73.0	43.1	54.4	39.9	35.4	43.2	48.9	46.1
MiniCPM-Llama3-V-2.5	8.4B	78.7	40.8	58.0	44.8	33.2	44.2	51.5	47.9
InternVL2-26B	26.0B	73.7	43.9	62.3	43.5	34.3	46.0	51.1	48.6
InternVL2-Llama3-76B	76.0B	**89.8**	**57.8**	**79.0**	**63.5**	**50.5**	**62.7**	**66.7**	**64.7**

disparity demonstrates the challenging nature of our ChartMimic benchmark for current open-weight LMMs. Moreover, open-weight LMMs even exhibit notable deficiencies in generating executable code, with a majority of them showing an execution rate below 75%. These findings highlight that there is still a considerable scope for the open-source community to enhance LMMs' capabilities in terms of complex visual understanding, code generation and cross-modal reasoning.

When given additional user-customized data, LMMs exhibit performance improvements. In the Customized Mimic task, most LMMs get performance improvement compared to the Direct Mimic task, particularly in terms of Text Score. This enhancement can be attributed to the provision of user-customized data, which alleviates the burden on LMMs to recognize textual and data information within the charts. Notably, Cogvlm2-llama3-chat-19B and Qwen2-VL-2B experience a decrease in there overall score and execution rate. This decline may arise from the reason that user-customized data increases the burden on the models to process different modalities of information. Overall, except for GPT-4o, the performance of the remaining LMMs is still below 65.0 in the Customized Mimic task. This indicates that, in addition to the capability to recognize data in the chart, understanding the layout of the chart and code generation are also key factors for LMMs to better accomplish the task.

4 Discussion

4.1 Different Complexity Levels

We report the performance of the top-6 models across different complexity levels for ChartMimic in Fig. 3. There is a consistent decline in performance across all tasks as the difficulty increased. For example, in Direct Mimic task, the performance of GPT-4o at easy, medium, and hard levels are 86.5, 77.7, and 74.8, respectively. These results demonstrate that ChartMimic benchmark are inherently challenging and confirms the efficacy of the established difficulty levels. Additionally, for weak LMM such as MiniCPM-Llama3-V-2.5, providing customized data at the Easy Level can help it perform tasks better (from 46.8 to 59.7). However, at the Hard Level, even when given customized data, their performance do not show improvement (from 31.9 to 31.5). This indicates that for Easy Level, complex reasoning abilities are not

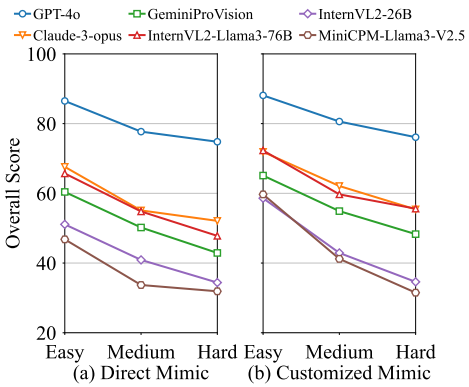


Figure 3: Overall scores of top-6 performed models at different complexity levels.

Table 5: Results for different prompting methods for GPT-4o and different size of InternVL2 LMMs on **Direct Mimic** task. The best scores for each model are in **bold**.

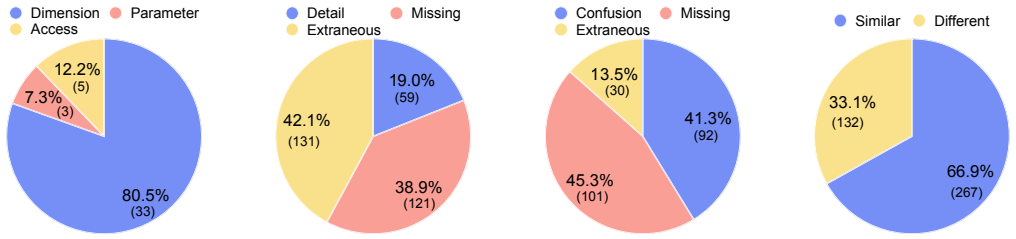
Model	Params	Method	Exec. Rate	Low-Level					High-Level	Overall
				Text	Layout	Type	Color	Avg.	GPT-4o	
GPT-4o	-	Direct	93.2	81.5	89.8	77.3	67.2	79.0	83.5	81.2
		HintEnhanced	92.0	80.9	89.7	78.2	67.9	79.2	83.0	81.1
		SelfReflection	95.2	86.6	93.7	81.1	68.2	82.4	87.1	**84.8**
		Scaffold	91.0	81.3	90.6	75.1	63.9	77.7	80.3	79.0
InternVL2-Llama3-76B	76.0B	Direct	83.2	54.1	74.5	49.2	41.5	54.8	62.2	58.5
		HintEnhanced	78.0	54.3	71.7	50.2	41.6	54.5	60.3	57.4
		SelfReflection	82.5	57.0	76.2	56.4	45.0	58.7	63.6	**61.1**
		Scaffold	76.7	52.1	71.4	49.1	40.5	53.3	60.1	56.7
InternVL2-26B	26.0B	Direct	69.3	39.2	58.7	35.9	31.8	41.4	47.4	**44.4**
		HintEnhanced	66.8	40.4	54.8	36.6	33.0	41.2	47.0	44.1
		Self	69.0	39.5	57.5	35.6	30.5	40.8	47.1	43.9
		Scaffold	64.3	36.0	53.5	35.4	30.3	38.8	44.0	41.4
InternVL2-8B	8.1B	Direct	61.8	31.5	51.1	28.6	26.2	34.4	38.9	**36.6**
		HintEnhanced	58.7	30.1	38.3	30.1	27.5	31.5	35.9	33.7
		SelfReflection	56.8	27.4	39.8	25.1	22.9	28.8	33.4	31.1
		Scaffold	61.8	22.8	43.5	24.8	20.2	27.8	32.0	29.9
InternVL2-2B	2.1B	Direct	52.5	23.6	35.8	16.0	15.4	22.7	24.2	**23.5**
		HintEnhanced	41.5	17.3	19.4	14.3	12.9	16.0	18.5	17.2
		SelfReflection	35.5	16.3	22.2	12.8	14.8	16.5	17.3	16.9
		Scaffold	38.0	6.5	17.8	3.2	2.2	7.4	15.6	11.5

required, and providing data is sufficient to complete the task well. However, for Hard Level charts, LMMs need complex code generation capabilities grounded on visual understanding. In this case, even when given data, if the LMMs cannot perform the corresponding code reasoning, they still cannot complete the task. Moreover, when given addtional data, if the data is too long and complex, the LLM may still become confused, leading to a certain decrease in performance.

4.2 Different Prompting Methods

We further examine the impact of different prompting methods on performance of ChartMimic benchmark. Specifically, we choose the GPT-4o and InternVL2 series LMMs (2B, 8B, 26B, 76B) to study their performance on the Direct Mimic task. We select three representative prompting methods: HintEnhanced, SelfReflection, and Scaffold Prompting. HintEnhanced uses prompt with chain-of-thought (Wei et al., 2022), explicitly prompting the LMMs to pay attention to important details (e.g., layout, type, text, etc). SelfReflection (Shinn et al., 2024) involves inputting the LMMs' own output and the corresponding rendered chart as additional information, instructing the LMM' to self-reflect their output. Scaffold Prompting (Lei et al., 2024) overlays a dot matrix within the figure as visual information anchors and leverages multi-dimensional coordinates as textual positional references. We detail the experimental setups in Appendix E.

As shown in Tab. 5, SelfReflection enables GPT-4o and InternVL2-Llama3-76B to reflect on and correct their outputs, demonstrating notable improvements over Direct Prompting. This underscore self-reflection as a key manifestation of System 2 reasoning (Sloman, 1996; Kumar et al., 2024). For GPT-4o, SelfReflection enhances Text, Layout, and Type Scores, though Color Score remains unchanged due to persistent challenges in fine color discrimination. However, LMMs with less developed reasoning capabilities (InternVL2-26B, 8B, and 2B) show no improvement or even decline with SelfReflection. This indicates that only LMMs with substantial reasoning capabilities can effectively engage in self-reflection and result optimization, underscoring the critical role of System 2 reasoning in addressing ChartMimic's challenges. Regarding HintEnhanced and Scaffold methods, they do not enhance the performance of models and reduce performance to varying degrees. Upon examining the cases, we find that the HintEnhanced method, which involves generating captions first, can introduce hallucinations that lead to errors in the subsequently generated code. As for the Scaffold method, introduction of an additional dot matrix can interfere with existing coordinate axis information in charts with high information density, thereby negatively impacting performance. These negative effects intensify as LMMs' reasoning capabilities decrease, further highlighting the importance of advanced reasoning capabilities in handling complex prompting strategies. We provide case studies of three prompting methods in Appendix E.



(a) Code (b) Text (c) Type (d) Color

Figure 4: Error analysis of GPT-4o across four error types on the Direct Mimic task. The number in brackets indicates the count of error case. Error examples can be found in Appendix Appendix H.

4.3 Correlation with Human Evaluation

To evaluate the reliability of the proposed multi-level metrics, we calculate their correlation with human evaluations. Specifically, we collect $1,200$ charts collected from GPT-4o using four different prompting methods (Sec. 4.2) on Direct Mimic. Each charts is evaluated by three individual evaluators, who assign scores ranging from 0 to 100 based on the similarity with the ground-truth chart. Details about the human evaluation process are provided in Appendix F. As shown in Tab. 6, the Pearson correlation coefficient (r) indicates that both the high-level ($r = 0.7041$) and low-level metric ($r = 0.7681$) have a high correlation with human judgment, demonstrating the reliability of our multi-level metrics. Moreover, the low-level metric demonstrates a higher correlation with human judgments compared to the high-level metric. This is attributed to its dependence on code execution logic, enabling it to capture detailed elements.

Table 6: Pearson correlation coefficient between multi-level evaluation metric and human evaluation.

Metric	Coefficient	p-value
High-Level	0.7041	< 0.0001
Low-Level	0.7681	< 0.0001

4.4 Error Analysis

We conduct a detailed error analysis of the state-of-the-art GPT-4o model to elucidate the current limitations of LMMs on ChartMimic, thereby identifying potential areas for further improvement. Our analysis identifies and category four main types of errors, detailed below.

Code-related Errors. (1) Dimension: Errors associated with data dimension, e.g., data dimension does not satisfy required conditions for operations; (2) Access: Errors occurring when accessing iterable elements out of bounds or accessing undeclared variables; (3) Parameter: Errors related to passing incorrect parameters when invoking functions. As observed in Fig. 4 (a), the majority of errors stem from Dimension, suggesting analysis and operation on data poses a challenge for GPT-4o.

Text-related Errors. (1) Detail: The text in the generated chart is largely consistent with the ground-truth chart text but contains minor discrepancies, such as mixing up "-" and "_"; (2) Missing: The generated chart omits text present in the ground-truth chart; (3) Extraneous: The generated chart contains text not found in the ground-truth chart, e.g., adding text for titles, reflecting the model's own interpretation of the chart; As depicted in Fig. 4 (b), the majority of errors are Missing and Extraneous, which indicates that for GPT-4o, due to intensive information in charts, it is challenging for it to comprehend the whole scope of the charts, even the basic text recognition task.

Type-related Errors. (1) Confusion: GPT-4o misinterprets the chart type as one that appears or functions similarly, such as mistaking violin plots for box plots, and so on; (2) Missing: The generated chart omits chart types present in the ground-truth chart; (3) Extraneous: The generated chart includes chart types not present in the ground-truth chart. As demonstrated in Fig. 4 (c), although GPT-4o exhibits remarkable capability in object recognition for natural images, it still struggles with scientific charts, which contain more nuanced semantic meanings through visual logic.

Color-related Errors. (1) Similar: The colors are not the same as the ground-truth colors but appear analogous. (2) Different: The colors are entirely dissimilar to the ground-truth colors. As shown in Fig. 4 (d), though GPT-4o can not exactly recognize the accurate colors, it can identify similar ones.

To sum up, GPT-4o still faces challenges in code generation and exhibits notable visual understanding deficiencies on ChartMimic. It has difficulty accurately recognizing visual elements in figures or may hallucinate incorrect elements and struggle with complex data analysis. Combining these insights, there is a need for further improvement in both visual understanding and code generation.

5 Related Work

Large Multimodal Models. The proprietary LMMs such as GPT-4o (OpenAI, 2024), Gemini (Google, 2023), and Claude-3 (Anthropic, 2024) have enabled complex multimodal interactions. Similarly, emerging open-weight LMMs such as LLaVA (Xu et al., 2024; Li et al., 2024a), InternVL (Chen et al., 2023), Qwen-VL (Bai et al., 2023), DeepSeek-VL (Lu et al., 2024) have contributed to the community. Despite these advancements, the effective evaluation of LMMs remains a major challenge. However, effectively evaluating LMMs remains a challenge, with open-source models performing well in benchmarks (Xu et al., 2023; Liu et al., 2023; Masry et al., 2022; Chen et al., 2024; Yue et al., 2023; Lu et al., 2023) yet falling short in practical applications (Xie et al., 2024; Koh et al., 2024; Si et al., 2024). This gap emphasizes the need for real-world-based evaluations that reflect authentic use cases. ChartMimic addresses this by requiring LMMs to translate complex visual information into code, testing their visual and coding capabilities.

Code Generation. Tasks such as HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), APPS (Hendrycks et al., 2021), and DS-1000 (Lai et al., 2023) are important benchmarks in natural language processing, but the inputs for these tasks are single-modal, consisting only of text, limiting their scope. With the emergence of LMMs, the evaluation of multimodal code generation has become increasingly critical for assessing real-world capabilities. On top of this, MMCode (Li et al., 2024b) attempts to address this issue, but their visual inputs may be simplistic and face the problem of a single evaluation method. Design2Code (Si et al., 2024) evaluate LMMs' code generation abilities through HTML web page generation. However, their test data comes from the C4 (Raffel et al., 2020) dataset, which may pose a risk of data leakage. Meanwhile, they just imitate the HTML and do not take the customized instructions; we take this into consideration. Recently, Plot2Code (Wu et al., 2024a) undertakes work similar to ours, aiming to measure models' code generation abilities through chart-to-code generation. Similarly, their approach of directly scraping data from the matplotlib gallery poses a risk of data leakage. Our ChartMimic provides a new set of manually curated $4,800$ data pairs, ensuring chart diversity and offering more fine-grained evaluation methods.

Chart Understanding. ChartMimic evaluates the capabilities of LMMs in grounding chart understanding into code generation, bridging visual and programmatic domains. Previous works focus on chart question answering (Masry et al., 2022; Methani et al., 2020; Xu et al., 2023; Wang et al., 2024b; Li et al., 2024c) and chart captioning (Rahman et al., 2023; Kantharaj et al., 2022). They assess the LMMs' ability to understand specific data characteristics or summarize key information into text. ChartMimic advances the field by introducing a chart-to-code task, transforming the LMMs' understanding of charts into code, which is neglected before but a realistic scenario for practical, real-world usage. This approach enables a comprehensive evaluation of the LMMs' overall comprehension of charts and their ability to express this understanding in code form. Leveraging the linguistic properties of code, our benchmark introduces fine-grained metrics to assess LMMs' chart understanding capabilities across multiple dimensions, including text, chart type, layout and color.

6 Conclusions

In this study, we develop the ChartMimic benchmark to evaluate LMMs' proficiency capability via chart-to-code generation. ChartMimic focuses on real-world applications for data visualization, aiming to assess LMMs' ability to harmonize a blend of cognitive capabilities, including visual understanding, code generation, and cross-modal reasoning. We propose two distinct levels of evaluation metrics (low and high level) to provide a comprehensive assessment. ChartMimic directly contributes to the understanding of progress towards artificial general intelligence, reflecting the expertise and reasoning abilities expected of skilled adults in various professional fields. Despite its comprehensive nature, ChartMimic, like any benchmark, has limitations. The manual curation process, although thorough, may introduce biases. Additionally, using scientific charts as information-intensive visual inputs to measure LMMs' multimodal code generation capabilities, while effective, still encounters domain-specific challenges. Our evaluation metric, despite considering most elements' similarity, does not uniformly score details of sub-icons, such as markers. We anticipate that ChartMimic will inspire the development of LMMs, advancing the pursuit of artificial general intelligence. Future research could explore various aspects, such as multimodal reasoning prompt strategies, to further reduce the gap between open-weight LMMs and proprietary ones.

REFERENCES

New models added to the phi-3 family, available on microsoft azure, May 2024. URL https://azure.microsoft.com/en-us/blog/new-models-added-to-the-phi-3-family-available-on-microsoft-azure/.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

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

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*, 2023.

Hannah K Bako, Xinyi Liu, Leilani Battle, and Zhicheng Liu. Understanding how designers find and use data visualization examples. *IEEE Transactions on Visualization and Computer Graphics*, 29 (1):1048–1058, 2022.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, et al. Are we on the right way for evaluating large vision-language models? *arXiv preprint arXiv:2403.20330*, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, Bin Li, Ping Luo, Tong Lu, Yu Qiao, and Jifeng Dai. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. *arXiv preprint arXiv:2312.14238*, 2023.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.

Stephanie DH Evergreen. *Effective data visualization: The right chart for the right data*. SAGE publications, 2019.

Li Fan, Lee Ching-Hung, Han Su, Feng Shanshan, Jiang Zhuoxuan, and Sun Zhu. A new era in human factors engineering: A survey of the applications and prospects of large multimodal models. *arXiv preprint arXiv:2405.13426*, 2024.

Google. Gemini: A family of highly capable multimodal models. *ArXiv*, abs/2312.11805, 2023. URL https://api.semanticscholar.org/CorpusID:266361876.

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.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

Shankar Kantharaj, Rixie Tiffany Ko Leong, Xiang Lin, et al. Chart-to-text: A large-scale benchmark for chart summarization. *arXiv preprint arXiv:2203.06486*, 2022.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pp. 18319–18345. PMLR, 2023.

Hugo Laurençon, Léo Tronchon, Matthieu Cord, and Victor Sanh. What matters when building vision-language models?, 2024.

Xuanyu Lei, Zonghan Yang, Xinrui Chen, Peng Li, and Yang Liu. Scaffolding coordinates to promote vision-language coordination in large multi-modal models. *arXiv preprint arXiv:2402.12058*, 2024.

Bo Li, Kaichen Zhang, Hao Zhang, Dong Guo, Renrui Zhang, Feng Li, Yuanhan Zhang, Ziwei Liu, and Chunyuan Li. Llava-next: Stronger llms supercharge multimodal capabilities in the wild, May 2024a. URL `https://llava-vl.github.io/blog/2024-05-10-llava-next-stronger-llms/`.

Kaixin Li, Yuchen Tian, Qisheng Hu, Ziyang Luo, and Jing Ma. Mmcode: Evaluating multimodal code large language models with visually rich programming problems. *arXiv preprint arXiv:2404.09486*, 2024b.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacaeval: An automatic evaluator of instruction-following models. `https://github.com/tatsu-lab/alpaca_eval`, 2023.

Zekun Li, Xianjun Yang, Kyuri Choi, Wanrong Zhu, Ryan Hsieh, HyeonJung Kim, Jin Hyuk Lim, Sungyoung Ji, Byungju Lee, Xifeng Yan, et al. Mmsci: A multimodal multi-discipline dataset for phd-level scientific comprehension. *arXiv preprint arXiv:2407.04903*, 2024c.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500. IEEE, 2023.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. Mmbench: Is your multi-modal model an all-around player? *arXiv preprint arXiv:2307.06281*, 2023.

Haoyu Lu, Wen Liu, Bo Zhang, Bingxuan Wang, Kai Dong, Bo Liu, Jingxiang Sun, Tongzheng Ren, Zhuoshu Li, Yaofeng Sun, Chengqi Deng, Hanwei Xu, Zhenda Xie, and Chong Ruan. Deepseek-vl: Towards real-world vision-language understanding, 2024.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*, 2023.

M Ronnier Luo, Guihua Cui, and Bryan Rigg. The development of the cie 2000 colour-difference formula: Ciede2000. *Color Research & Application: Endorsed by Inter-Society Color Council, The Colour Group (Great Britain), Canadian Society for Color, Color Science Association of Japan, Dutch Society for the Study of Color, The Swedish Colour Centre Foundation, Colour Society of Australia, Centre Français de la Couleur*, 26(5):340–350, 2001.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. *arXiv preprint arXiv:2203.10244*, 2022.

Fanqing Meng, Wenqi Shao, Quanfeng Lu, Peng Gao, Kaipeng Zhang, Yu Qiao, and Ping Luo. Chartassisstant: A universal chart multimodal language model via chart-to-table pre-training and multitask instruction tuning. *arXiv preprint arXiv:2401.02384*, 2024.

Nitesh Methani, Pritha Ganguly, Mitesh M Khapra, and Pratyush Kumar. Plotqa: Reasoning over scientific plots. In *CVPR*, pp. 1527–1536, 2020.

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

OpenCompass. Opencompass: A universal evaluation platform for foundation models. `https://github.com/open-compass/opencompass`, 2023.

Paul Parsons. Understanding data visualization design practice. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):665–675, 2021.

Jorge Poco and Jeffrey Heer. Reverse-engineering visualizations: Recovering visual encodings from chart images. In *Computer graphics forum*, volume 36, pp. 353–363. Wiley Online Library, 2017.

Xuedi Qin, Yuyu Luo, Nan Tang, and Guoliang Li. Making data visualization more efficient and effective: a survey. *The VLDB Journal*, 29(1):93–117, 2020.

Annemarie Quispel, Alfons Maes, and Joost Schilperoord. Aesthetics and clarity in information visualization: The designer's perspective. In *Arts*, volume 7, pp. 72. MDPI, 2018.

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

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Raian Rahman, Rizvi Hasan, Abdullah Al Farhad, et al. Chartsumm: A comprehensive benchmark for automatic chart summarization of long and short summaries. *arXiv preprint arXiv:2304.13620*, 2023.

Manolis Savva, Nicholas Kong, Arti Chhajta, Li Fei-Fei, Maneesh Agrawala, and Jeffrey Heer. Revision: Automated classification, analysis and redesign of chart images. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pp. 393–402, 2011.

Chufan Shi, Haoran Yang, Deng Cai, Zhisong Zhang, Yifan Wang, Yujiu Yang, and Wai Lam. A thorough examination of decoding methods in the era of llms. *arXiv preprint arXiv:2402.06925*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

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.

Steven A Sloman. The empirical case for two systems of reasoning. *Psychological bulletin*, 119(1):3, 1996.

Qiushi Sun, Zhirui Chen, Fangzhi Xu, Kanzhi Cheng, Chang Ma, Zhangyue Yin, Jianing Wang, Chengcheng Han, Renyu Zhu, Shuai Yuan, et al. A survey of neural code intelligence: Paradigms, advances and beyond. *arXiv preprint arXiv:2403.14734*, 2024.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024a.

Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, Jiazheng Xu, Bin Xu, Juanzi Li, Yuxiao Dong, Ming Ding, and Jie Tang. Cogvlm: Visual expert for pretrained language models, 2023a.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Chandu, David Wadden, Kelsey MacMillan, Noah A Smith, Iz Beltagy, et al. How far can camels go? exploring the state of instruction tuning on open resources. *Advances in Neural Information Processing Systems*, 36:74764–74786, 2023b.

Zirui Wang, Mengzhou Xia, Luxi He, Howard Chen, Yitao Liu, Richard Zhu, Kaiqu Liang, Xindi Wu, Haotian Liu, Sadhika Malladi, et al. Charxiv: Charting gaps in realistic chart understanding in multimodal llms. *arXiv preprint arXiv:2406.18521*, 2024b.

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, Yixiao Ge, Qiushan Guo, Jiahao Wang, Zhixuan Liang, Zeyu Lu, Ying Shan, and Ping Luo. Plot2code: A comprehensive benchmark for evaluating multi-modal large language models in code generation from scientific plots. *arXiv preprint arXiv:2405.07990*, 2024a.

Tianhe Wu, Kede Ma, Jie Liang, Yujiu Yang, and Lei Zhang. A comprehensive study of multimodal large language models for image quality assessment. *arXiv preprint arXiv:2403.10854*, 2024b.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

Ruyi Xu, Yuan Yao, Zonghao Guo, Junbo Cui, Zanlin Ni, Chunjiang Ge, Tat-Seng Chua, Zhiyuan Liu, and Gao Huang. LLaVA-UHD: an lmm perceiving any aspect ratio and high-resolution images. *arXiv preprint arXiv:2403.11703*, 2024.

Zhengzhuo Xu, Sinan Du, Yiyan Qi, Chengjin Xu, Chun Yuan, and Jian Guo. Chartbench: A benchmark for complex visual reasoning in charts. *arXiv preprint arXiv:2312.15915*, 2023.

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. *arXiv preprint arXiv:2402.11453*, 2024.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. *arXiv preprint arXiv:2311.16502*, 2023.

Xinlu Zhang, Yujie Lu, Weizhi Wang, An Yan, Jun Yan, Lianke Qin, Heng Wang, Xifeng Yan, William Yang Wang, and Linda Ruth Petzold. Gpt-4v (ision) as a generalist evaluator for vision-language tasks. *arXiv preprint arXiv:2311.01361*, 2023.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36, 2024.

Zheng Zhu, Xiaofeng Wang, Wangbo Zhao, Chen Min, Nianchen Deng, Min Dou, Yuqi Wang, Botian Shi, Kai Wang, Chi Zhang, et al. Is sora a world simulator? a comprehensive survey on general world models and beyond. *arXiv preprint arXiv:2405.03520*, 2024.

APPENDIX

A Data Annotation **16**

A.1 Data Annotation Principles 16

A.2 Data Annotation Pipeline 16

A.3 Complexity Levels 18

A.4 Instruction Examples 19

B Correlation Between Test Set and Testmini set **23**

C Chart Taxonomy **24**

D Details of Evaluation Metrics **33**

D.1 GPT-4o Score 33

D.2 Text Score 38

D.3 Layout Score 38

D.4 Type Score 39

D.5 Color Score 39

E Model Configurations and Prompting Methods **41**

E.1 Generation Configurations 41

E.2 Prompts 41

E.3 Cases of Different Prompting Methods 44

E.4 Details of Models 48

F Correlation with Human Evaluation **48**

G Correlation with Chart Understanding and Code Generation **50**

H Cases of Error Analysis **51**

I Error Analysis of Open-Weight Models **54**

J Ethics, Societal Impact and Scalability of ChartMimic **55**

A DATA ANNOTATION

A.1 DATA ANNOTATION PRINCIPLES

Diversity of Chart Types. Data visualization has become an essential tool for conveying information in various fields, and the design practices and requirements for different types of charts vary significantly(Parsons, 2021). Most of previous work (Han et al., 2023; Xu et al., 2023; Masry et al., 2022; Kantharaj et al., 2022; Rahman et al., 2023) focus only on line, bar, pie charts, etc, which are commonly used within the field of computer science. However, with the increasing integration of LMMs into everyday tasks, people from all fields are starting to use generative techniques as daily assistants to enhance their design and creative processes when creating visualizations. In this light, enriching the spectrum of chart diversity is crucial for evaluating LMMs' proficiency in multimodal chart-to-code generation.

Balance of Charts Complexity. Charts serve as visual aids for data presentation, enabling users to promptly grasp the underlying patterns and significance within the data. It is essential to adopt the appropriate chart type and complexity level to effectively convey the information (Evergreen, 2019). Previous works have focused mainly on charts with only a single data format and low information density, which are rarely encountered in practical settings such as academic writing. Noting that LMMs like GPT-4o have already demonstrated outstanding data visualization capabilities, our focus is on using charts that are actually employed in practice, such as those from research papers, and on selecting charts with varying levels of complexity when constructing our benchmark.

Reduction of Data Leakage. Recognizing that augmenting training data is a primary method for enhancing the performance of LMMs has led researchers to more comprehensively exploit all accessible data during pre-training. However, this approach introduces the possibility of data leakage, especially when pre-training data might already include resources such as the matplotlib gallery[2] or other pre-existing datasets, potentially resulting in inaccurate evaluations. To mitigate this issue, we deliberately avoid the use of code that can be readily found online or code that could be auto-generated by large language models for chart creation in constructing our dataset, thereby reducing the probability of data leakage. To further illustrate the potential data leakage in matplotlib gallery, we select the code for 20 charts from the matplotlib gallery. Specifically, we provide the first half of the code as a prefix for Llama3-8B (AI@Meta, 2024) and let the model complete the remaining code. Then we calculate the edit distance between the generated complete code block and the ground truth. Similarly, we apply the same process to the code in Direct Mimic, calculating the edit distance between the generated results and the ground truth. We find that the edit distance for the code in the matplotlib gallery is 22.1, while the edit distance for Direct Mimic's code is 39.8. This indicates that the code in Direct Mimic have a larger edit distance, which further reduces the risk of data leakage compared to the matplotlib gallery ones.

Integration of Authentic User Requirements. Users from diverse domains such as finance, healthcare, education, and engineering demonstrate unique needs and preferences for data visualization. These requirements extend beyond mere diversity; they demand the incorporation of charts capable of articulating complex and multi-dimensional data, and conforming to domain-specific aesthetic preferences(Qin et al., 2020; Evergreen, 2019). By aligning our dataset construction process with these real-world demands, we enable a more relevant and precise evaluation of LMMs. Adopting this approach not only reflects actual user patterns but also steers the research community in the iterative improvement of LMMs. This focused development will also lead to an enhanced user experience and increased user satisfaction, as the models become more proficient in meeting the sophisticated and varied needs of users.

A.2 DATA ANNOTATION PIPELINE

We present the detailed description about the five-step data annotation pipeline in this section. Fig. 5 demonstrates an illustration of the data annotation pipeline.

General Filtering. To obtain a high-quality dataset aligned with real-world use cases and to avoid data leakage, we initially scrape figures from source files of publications on arXiv[3] that hold a CC BY

[2] https://matplotlib.org/
[3] https://arxiv.org/

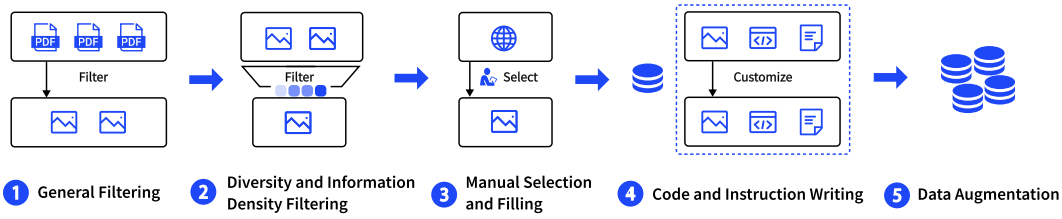


Figure 5: An illustration of the data annotation pipeline, which encompasses five key steps.

4.0 license and have a publication date after February 2024. We then extract figures in PDF format, yielding approximately 174,100 figures across various domains (such as Physics, Mathematics, Computer Science, Quantitative Biology, Quantitative Finance, Statistics, Electrical Engineering and Systems Science, Economics). We filter these figures based on file format and generation method, retaining only Matplotlib-generated PDFs, indicating that these figures can be reproduced using Python. This process results in a refined collection of 15,800 figures.

Diversity and Information Density Filtering. This stage involves a two-phase process conducted by five domain experts from information visualization, digital media, industrial design, visual communication design, and computer science. The information visualization expert has three years of research experience in scientific visualization and visual analytics. The digital media specialist focuses on human-computer interaction and multimedia design. The industrial design expert brings perspectives from user experience and product visualization. The visual communication design expert specializes in graphic design principles and information aesthetics. The computer science expert has extensive experience in data visualization programming and scientific computing.

In the first phase, these experts conduct a 7-day manual review of the 15,800 figures, focusing on visual diversity and information communication effectiveness. They reference the Matplotlib gallery, gradually identifying and finalizing chart type while reviewing the figures, and build corresponding type pools. For each new figure, they assess its visual elements—such as layout, axes, line styles, marker styles, and colors—against existing figures in their corresponding type pool. If the figure as long as exhibits a distinctive difference in at least one of these aspects, it is retained; otherwise, it is excluded. This process results in 1,295 figures being selected for the second phase.

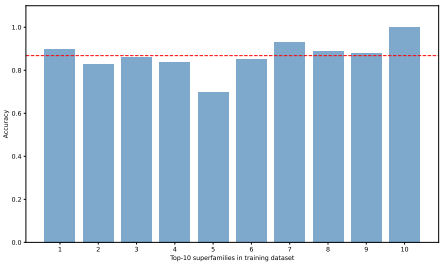
In the second phase, these experts independently review the 1,295 figures and further select those figures they deem to exhibit significant distinctions and diversity. Figures selected unanimously by all experts are directly included, while the remaining figures are subjected to a majority voting system requiring at least 3/5 votes for inclusion. This rigorous process, which takes less than 3 days to complete, results in a final set of 279 figures.

Manual Selection and Filling. In addition to sourcing from arXiv, we curate chart figures from diverse platforms such as the matplotlib gallery, Stack Overflow, and plotting-related forums on Twitter and Reddit. These charts are deliberately chosen for their distinctive styles, which are not present in our arXiv dataset. Consequently, we obtain 600 prototype charts for ChartMimic. This stage took us less than a week to complete the data selection.

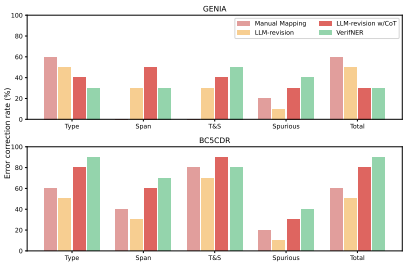
Code and Instruction Writing. We propose to manually write codes and instructions for ChartMimic based on the collected 600 prototype charts. To ensure annotation quality, a team of skilled Python users—Python annotators—master's students in computer science with 6+ years of Python and matplotlib experience—reproduce 600 prototype charts using Python 3.9.0 and matplotlib v3.8.4. Since the unannotated data in the figures cannot be fully restored, they can only be approximated when writing the code. This process generates 600 (figure, code, instruction) triplets for the Direct Mimic task and another 600 triplets for the Customized Mimic task by integrating data from various domains into the corresponding code and instructions, comprising 1,200 high-quality seed data.

Data Augmentation. After developing the seed triplets, we proceed with a manual data augmentation process. Python Annotators modify various elements of each seed triplet, including data, color schemes, and mark styles, to create augmented triplets. For each seed triplet, we generate three additional augmented triplets. This process enhances our dataset, resulting in a total of $4,800$ triplets. The "Code and Instruction Writing" and "Data Augmentation" stages together take the data annotators approximately 1.5 months to complete.

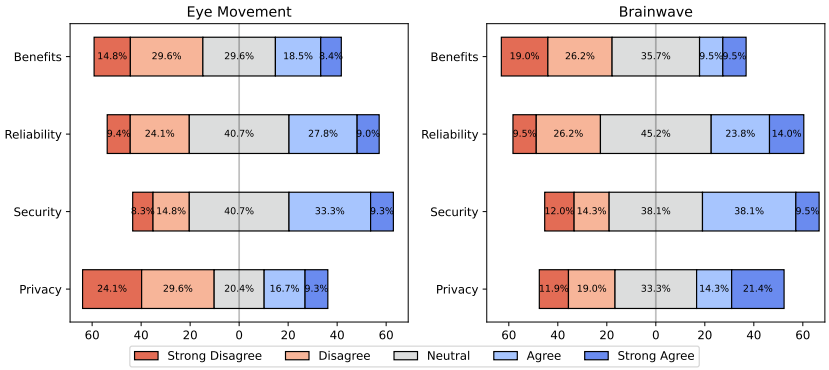
A.3 COMPLEXITY LEVELS



(a) Easy: bar_8



(b) Medium: bar_9



(c) Hard: bar_21

Figure 6: Representative examples of chart complexity in ChartMimic (Easy, Medium, Hard).

Our complexity assessment framework is established upon two fundamental criteria, designed to comprehensively evaluate both the visual and technical aspects of chart generation:

Visual Elements Complexity We systematically evaluate the sophistication of each chart through a comprehensive scoring mechanism applied to its constituent visual components. This encompasses the chart typology, data grouping structures, marker configurations, textual elements, chromatic schemes, compositional layouts, and coordinate systems. Each visual component is quantitatively assessed based on both its frequency and sophistication level (designated as 1-3 points for low/medium/high complexity, respectively). This multi-dimensional scoring approach ensures a thorough evaluation of the visual complexity inherent in each chart.

Implementation Complexity We incorporate code complexity as a quantitative metric, measured primarily through code length and structural intricacy. This parameter effectively captures the technical sophistication required for accurate chart reproduction, including the complexity of data preprocessing, visualization logic, and stylistic customizations. The implementation complexity provides insights into the programming challenges associated with each chart type.

Consequently, charts in ChartMimic are systematically categorized into three distinct complexity levels, each representing a specific combination of visual and implementation challenges:

Easy: Fundamental chart configurations featuring minimal visual complexity and straightforward implementation requirements (e.g., in Fig. 6(a), monochromatic bar charts with sparse textual elements and simplified data representation).

Medium: Charts exhibiting intermediate complexity in visual element composition or implementation requirements (e.g., in Fig. 6(b), dual-subplot bar charts incorporating grouped data structures, diverse chromatic schemes, and moderate textual annotations).

Hard: Charts demonstrating sophisticated visual elements or advanced implementation (e.g., in Fig. 6(c), complex dual-subplot bar charts featuring divergent data patterns, extensive color schemes, comprehensive textual annotations, and substantial code complexity).

A.4 INSTRUCTION EXAMPLES

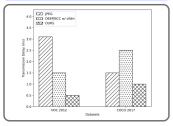
To illustrate our instruction format and task requirements, we present examples (bar_28 and CB_29) with their corresponding (figure, instruction, code) triplets for both Direct Mimic and Customized Mimic tasks. Fig. 7 and Fig. 8 show the Direct Mimic and Customized Mimic tasks for a bar chart (bar_28), respectively. The instruction provides guidance on creating the visualization. Similarly, Fig. 9 and Fig. 10 demonstrate the tasks for a more complex combination chart (CB_29).

Task 1: Direct Mimic - bar_28

Instruction

You are an expert Python developer who specializes in writing matplotlib code based on a given picture. I found a very nice picture in a STEM paper, but there is no corresponding source code available. I need your help to generate the Python code that can reproduce the picture based on the picture I provide.
Note that it is necessary to use figsize=(6, 5) to set the image size to match the original size.
Now, please give me the matplotlib code that reproduces the picture below.

Figure



Code

1/3

```
# ===================
# Part 1: Importing Libraries
# ===================
import matplotlib.pyplot as plt
import numpy as np

# ===================
# Part 2: Data Preparation
# ===================
# Data
datasets = ["VOC 2012", "COCO
2017"]
jpeg = [3.1, 1.5]
deepjscc = [1.5, 2.5]
ours = [0.5, 1.0]

# X-axis positions
x = np.arange(len(datasets))

# Bar width
width = 0.2
labels = ["JPEG", "DEEPJSCC w/
ofdm", "OURS"]
ylim = [0, 4.3]
ylabel = "Transmission Delay (ms)"
xlabel = "Datasets"
```

2/3

```
# ===================
# Part 3: Plot Configuration and
Rendering
# ===================
# Plotting
fig, ax = plt.subplots(figsize=(6,
5))

# Adjusting figure size to match
the original image's dimensions
ax.bar(x - width, jpeg, width,
label=labels[0], hatch="//",
edgecolor="black", color="white")

ax.bar(
    x,
    deepjscc,
    width,
    label=labels[1],
    hatch="..",
    edgecolor="black",
    color="white",
)

ax.bar(x + width, ours, width,
label=labels[2], hatch="xx",
edgecolor="black", color="white")
```

3/3

```
# Labels and Title
ax.set_ylim(ylim)
ax.set_ylabel(ylabel)
ax.set_xlabel(xlabel)
ax.set_xticks(x)
ax.set_xticklabels(datasets)
ax.legend(loc="upper left")

# ===================
# Part 4: Saving Output
# ===================
plt.tight_layout()
plt.savefig("bar_28.pdf",bbox_inch
es="tight")
```

Ground-truth Figure (Rendered from Code)

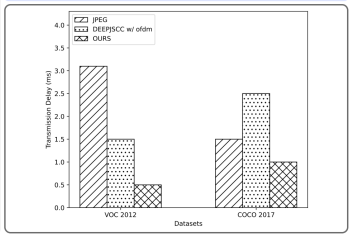


Figure 7: An Example of the Direct Mimic task (bar_28), showing the (figure, instruction, code) triplet. Additionally, we also display the ground truth figure rendered from the code for illustration.

Task 2: Customized Mimic - bar_28

Instruction

There is a reference image for you and I also have a set of data about fuel efficiency of different types of transport in urban and rural areas. Please refer to the reference image and generate Python matplotlib code that can create a chart that exactly mirrors the style of the reference image and accurately represents my data.

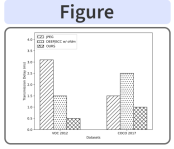
datasets = ["Urban Transport", "Rural Transport"]
electric = [2.7, 3.4]
hybrid = [1.9, 2.1]
diesel = [3.5, 2.8]

# X-axis positions
x = np.arange(len(datasets))

# Bar width
width = 0.2
labels = ["Electric", "Hybrid", "Diesel"]
ylim = [0, 4.5]
ylabel = "Fuel Efficiency (km/l)"
xlabel = "Transport Types"

Now, please output the Python matplotlib code that can create a chart that exactly mirrors the style of the reference image and accurately represents my data.
Note: You must copy the data I provide completely into your code, and ensure that the code you generate contains the data and is directly executable.
Note that it is necessary to use figsize=(6, 5) to set the image size.

Figure



Code

1/3
```
# ===================
# Part 1: Importing Libraries
# ===================
import matplotlib.pyplot as plt
import numpy as np;

# ===================
# Part 2: Data Preparation
# ===================
datasets = ["Urban Transport",
"Rural Transport"]
electric = [2.7, 3.4]
hybrid = [1.9, 2.1]
diesel = [3.5, 2.8]

# X-axis positions
x = np.arange(len(datasets))

# Bar width
width = 0.2
labels = ["Electric", "Hybrid",
"Diesel"]
ylim = [0, 4.5]
ylabel = "Fuel Efficiency (km/l)"
xlabel = "Transport Types"
```

2/3
```
# ===================
# Part 3: Plot Configuration and
Rendering
# ===================
# Plotting
fig, ax = plt.subplots(figsize=(6,
5))

# Adjusting figure size to match
the original image's dimensions
ax.bar(x - width, electric, width,
label=labels[0], hatch="//",
edgecolor="black", color="white")
ax.bar(
    x,
    hybrid,
    width,
    label=labels[1],
    hatch="..",
    edgecolor="black",
    color="white",
)
ax.bar(
    x + width, diesel, width,
label=labels[2], hatch="xx",
edgecolor="black", color="white"
)
```

3/3
```
# Labels and Title
ax.set_ylim(ylim)
ax.set_ylabel(ylabel)
ax.set_xlabel(xlabel)
ax.set_xticks(x)
ax.set_xticklabels(datasets)
ax.legend(loc="upper left")

# ===================
# Part 4: Saving Output
# ===================
plt.tight_layout()
plt.savefig('bar_28.pdf',
bbox_inches='tight')
```

Ground-truth Figure (Rendered from Code)

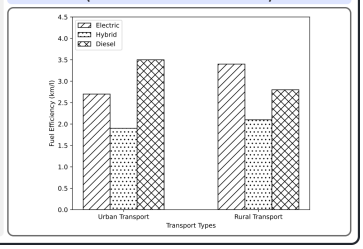


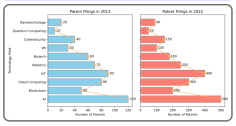
Figure 8: An Example of the Customized Mimic task (bar_28), showing the (figure, instruction, code) triplet. Additionally, we also display the ground truth figure rendered from the code for illustration.

Task 1: Direct Mimic - CB_29

Instruction

You are an expert Python developer who specializes in writing matplotlib code based on a given picture. I found a very nice picture in a STEM paper, but there is no corresponding source code available. I need your help to generate the Python code that can reproduce the picture based on the picture I provide.
Note that it is necessary to use figsize=(10, 5) to set the image size to match the original size.
Now, please give me the matplotlib code that reproduces the picture below.

Figure



Code

1/3

```
# ===================
# Part 1: Importing Libraries
# ===================
import matplotlib.pyplot as plt
import numpy as np

# ===================
# Part 2: Data Preparation
# ===================
# Technology fields
tech_fields = [
    "AI",
    "Blockchain",
    "Cloud Computing",
    "IoT",
    "Robotics",
    "Biotech",
    "VR",
    "Cybersecurity",
    "Quantum Computing",
    "Nanotechnology",
]

# Patent filings in 2013
patents_2013 = [120, 50, 80, 90,
70, 60, 30, 40, 10, 20]

# Patent filings in 2022
patents_2022 = [500, 200, 300,
400, 250, 180, 100, 150, 50, 90]
x_label = "Number of Patents"
y_label = "Technology Field"
ax1_title = "Patent Filings in
2013"
ax2_title = "Patent Filings in
2022"
```

2/3

```
# ===================
# Part 3: Plot Configuration and
Rendering
# ===================
# Create a figure with two
subplots (1x2) and shared y-axis
fig, axes = plt.subplots(1, 2,
figsize=(10, 5), sharey=True)
colors = ["skyblue", "salmon"]

# Plotting the bar graphs
for i, (patents, color) in
enumerate(zip([patents_2013,
patents_2022], colors)):
    axes[i].barh(tech_fields,
patents, color=color,
edgecolor="gray")
    axes[i].set_xlabel(x_label)
    # Adding line graph on the
same axes
    axes[i].plot(patents,
tech_fields, "o-",
color="#f7b267")

# Adding data labels
for ax, patents in zip(axes,
[patents_2013, patents_2022]):
    for index, value in
enumerate(patents):
        ax.text(value + 1, index,
f" {value}", va="center")

# Set labels and titles
axes[0].set_title(ax1_title)
axes[1].set_title(ax2_title)
axes[0].set_ylabel(y_label)
```

3/3

```
# ===================
# Part 4: Saving Output
# ===================
# Adjust layout to prevent
clipping and overlap
plt.tight_layout()
plt.savefig("CB_29.pdf",
bbox_inches="tight")
```

Ground-truth Figure (Rendered from Code)

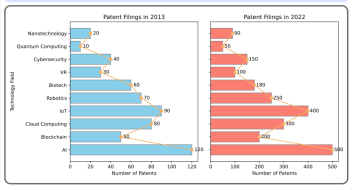


Figure 9: An Example of the Direct Mimic task (CB_29), showing the (figure, instruction, code) triplet. Additionally, we also display the ground truth figure rendered from the code for illustration.

Task 2: Customized Mimic - CB_29

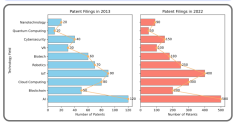
Instruction

There is a reference image for you and I also have a set of data about the number of registered vehicles for different transportation modes in 2013 and 2022. Please refer to the reference image and generate Python matplotlib code that can create a chart that exactly mirrors the style of the reference image and accurately represents my data.

```
# Transportation modes
transport_modes = [
  "Automobiles",
  "Bicycles",
  "Buses",
  "Trains",
  "Planes",
  "Ships",
  "Motorcycles",
  "Scooters",
  "Trams",
  "Helicopters",
]

# Number of registered vehicles in 2013 (in thousands)
vehicles_2013 = [50000, 10000, 15000, 20000, 8000, 3000, 7000, 5000, 2000, 1000]
# Number of registered vehicles in 2022 (in thousands)
vehicles_2022 = [55000, 12000, 16000, 22000, 9000, 3500, 7500, 6000, 2500, 1200]
x_label = "Number of Registered Vehicles (in thousands)"
y_label = "Transportation Mode"
ax1_title = "Registered Vehicles in 2013"
ax2_title = "Registered Vehicles in 2022"
```

Now, please output the Python matplotlib code that can create a chart that exactly mirrors the style of the reference image and accurately represents my data.
Note: You must copy the data I provide completely into your code, and ensure that the code you generate contains the data and is directly executable.
Note that it is necessary to use figsize=(10, 5) to set the image size.

Figure



Code

1/3

```
# ===================
# Part 1: Importing Libraries
# ===================
import matplotlib.pyplot as plt
import numpy as np;

# ===================
# Part 2: Data Preparation
# ===================
# Transportation modes
transport_modes = [
    "Automobiles",
    "Bicycles",
    "Buses",
    "Trains",
    "Planes",
    "Ships",
    "Motorcycles",
    "Scooters",
    "Trams",
    "Helicopters",
]

# Number of registered vehicles in 2013 (in thousands)
vehicles_2013 = [50000, 10000, 15000, 20000, 8000, 3000, 7000, 5000, 2000, 1000]
# Number of registered vehicles in 2022 (in thousands)
vehicles_2022 = [55000, 12000, 16000, 22000, 9000, 3500, 7500, 6000, 2500, 1200]
```

2/3

```
x_label = "Number of Registered Vehicles (in thousands)"
y_label = "Transportation Mode"
ax1_title = "Registered Vehicles in 2013"
ax2_title = "Registered Vehicles in 2022"

# ===================
# Part 3: Plot Configuration and Rendering
# ===================
# Create a figure with two subplots (1x2) and shared y-axis
fig, axes = plt.subplots(1, 2, figsize=(10, 5), sharey=True)
colors = ["skyblue", "salmon"]

# Plotting the bar graphs
for i, (vehicles, color) in enumerate(zip([vehicles_2013, vehicles_2022], colors)):
    axes[i].barh(transport_modes, vehicles, color=color, edgecolor="gray")
    axes[i].set_xlabel(x_label)
    # Adding line graph on the same axes
    axes[i].plot(vehicles, transport_modes, "o-", color="#f7b267")
```

3/3

```
# Adding data labels
for ax, vehicles in zip(axes, [vehicles_2013, vehicles_2022]):
    for index, value in enumerate(vehicles):
        ax.text(value + 50, index, f" {value}", va="center")

# Set labels and titles
axes[0].set_title(ax1_title)
axes[1].set_title(ax2_title)
axes[0].set_ylabel(y_label)

# ===================
# Part 4: Saving Output
# ===================
# Adjust layout to prevent clipping and overlap
plt.tight_layout()
plt.savefig('CB_29.pdf', bbox_inches='tight')
```

Ground-truth Figure (Rendered from Code)

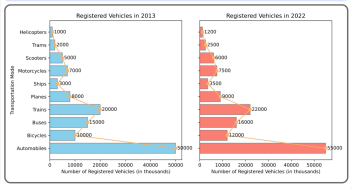


Figure 10: An Example of the Customized Mimic task (CB_29), showing the (figure, instruction, code) triplet. Additionally, we also display the ground truth figure rendered from the code for illustration.

Table 7: The ChartMimic benchmark with **Direct Mimic** task. We report the results of both Testmini and Test set.

Model	Params	Test set	Exec. Rate	Low-Level					High-Level	Overall
				Text	Layout	Type	Color	Avg.	GPT-4o	
Proprietary										
GPT-4o	-	Testmini	93.2	81.5	89.8	77.3	67.2	79.0	83.5	81.2
		Test	93.0	83.4	90.2	76.7	66.0	79.1	84.1	81.6
Open-Weight										
InternVL2-2B	2.2B	Testmini	52.5	23.6	35.8	16.0	15.4	22.7	24.2	23.5
		Test	51.8	23.3	34.8	15.9	15.9	22.5	25.6	24.0
InternVL2-8B	8.1B	Testmini	61.8	31.5	51.1	28.6	26.2	34.4	38.9	36.6
		Test	61.6	33.4	47.4	25.7	24.2	32.7	39.9	36.3
InternVL2-26B	26.0B	Testmini	69.3	39.2	58.7	35.9	31.8	41.4	47.4	44.4
		Test	69.9	41.7	58.0	35.6	31.0	41.6	48.1	44.9
InternVL2-Llama3-76B	76.0B	Testmini	83.2	54.1	74.5	49.2	41.5	54.8	62.2	58.5
		Test	83.3	55.6	73.5	50.4	40.8	55.1	62.7	58.9

Table 8: The ChartMimic benchmark with **Customized Mimic** task. We report the results of both Testmini and Test set.

Model	Params	Test set	Exec. Rate	Low-Level					High-Level	Overall
				Text	Layout	Type	Color	Avg.	GPT-4o	
Proprietary										
GPT-4o	-	Testmini	96.5	88.5	92.9	79.2	67.6	82.1	84.3	83.2
		Test	96.2	87.2	91.7	80.1	66.4	81.4	84.8	83.1
Open-Weight										
InternVL2-2B	2.2B	Testmini	49.3	22.2	35.4	20.0	18.1	23.9	27.8	25.9
		Test	49.6	22.4	33.9	19.2	19.6	23.8	28.4	26.1
InternVL2-8B	8.1B	Testmini	73.0	43.1	54.4	39.9	35.4	43.2	48.9	46.1
		Test	73.5	43.7	54.1	41.1	34.1	43.3	49.8	46.5
InternVL2-26B	26.0B	Testmini	73.7	43.9	62.3	43.5	34.3	46.0	51.1	48.6
		Test	74.7	44.7	66.3	46.8	35.1	48.2	50.8	49.5
InternVL2-Llama3-76B	76.0B	Testmini	89.8	57.8	79.0	63.5	50.5	62.7	66.7	64.7
		Test	88.1	57.9	79.6	65.6	51.7	63.7	68.2	66.0

B CORRELATION BETWEEN TEST SET AND TESTMINI SET

Tab. 7 and 8 reports the performance of GPT-4o and InternVL2 series LMMs (2B, 8B, 26B, Llama3-76B) on Direct Mimic and Customized Mimic task. The minor differences between scores on the test subset and the testmini subset suggest that testmini effectively mirrors the test subset, serving as a valuable evaluation subset for model development, especially for those who have limited computing resources.

C CHART TAXONOMY

This section presents the chart taxonomy in ChartMimic. It encompasses a structure of 22 categories according to chart type characteristics and data composition. The categories comprise of:

- 18 regular types, ordered as follows: Bar, Heatmap, Scatter, Box, Errorbar, Errorpoint, Line, Violin, Radar, Pie, Density, Graph, Quiver, Contour, Histogram, Tree, Area, and 3D charts.
- 4 advanced types: PIP (Plot-in-Plot), Multidiff (Multiple Differences), Combination, and HR (Hard-to-Recognize).

The regular types are further divided into subcategories according to chart feature or data characteristics, whereas each chart of the advanced types represents a unique subcategory. The taxonomy showcases examples to showcase diversity of each category.

Bar: Bar chart uses rectangular bars to represent data and can be distinguished by its orientation, horizontal or vertical, with its nuanced data attributes. There are 8 subcategories of the data attributes for each orientation, consisting of 16 subcategories in total, as shown in Figure 11:

1. Base (single positive data set, unordered)
2. Sorted (data in ascending or descending sequence)
3. Grouped (multiple positive data sets, adjacent)
4. Stacked (multiple positive data sets laying atop one another)
5. Normalized (proportioned stacks of positive data summing to one, a Stacked variant)
6. Diverging (multiple stacked data sets expanding from a central axis)
7. With-Negative (data sets including negative values)
8. Reverse (exclusively negative data sets)

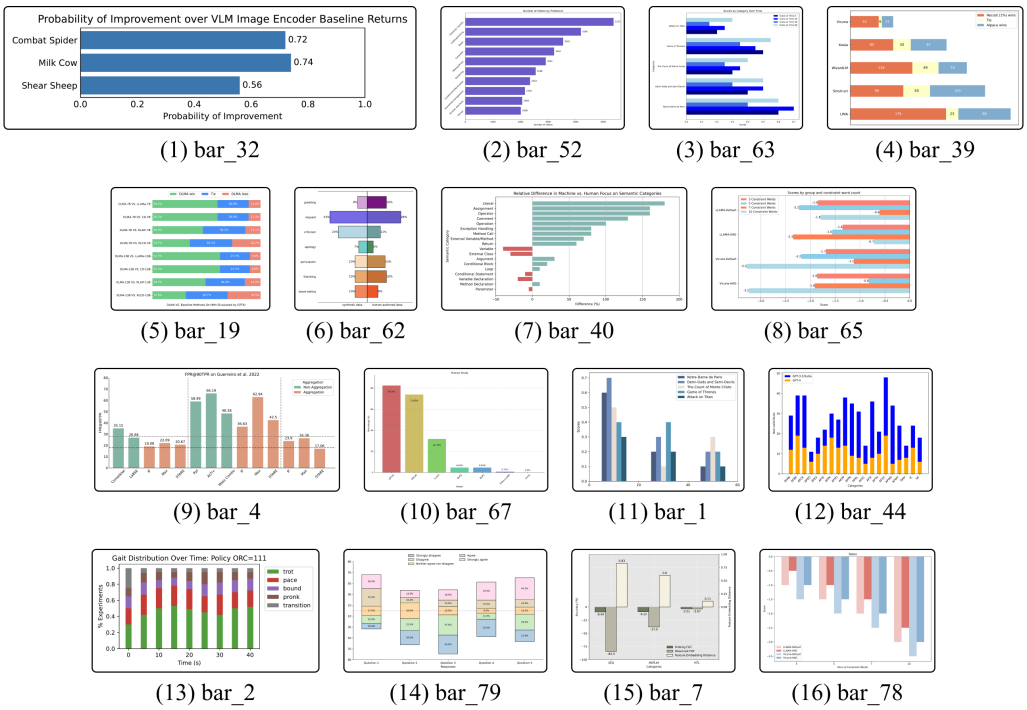
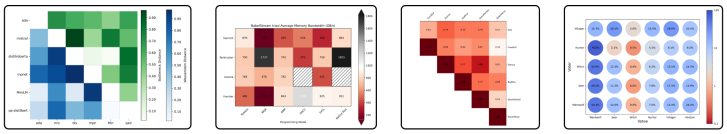


Figure 11: Examples of Bar chart subcategories.

Heatmap: Based on the layout and visual representations, Heatmap is categorized into 4 subcategories, each reflecting unique aspects of data presentation, as shown in Figure 12.

1. Base (the general layout of a typical heatmap)
2. Missing-Data (visualization indicating the absence of data points)
3. Triangle-Layout (heatmap configured in a triangular layout)
4. Other-Shaped (heatmap comprising elements in non-rectangular forms, such as circles)

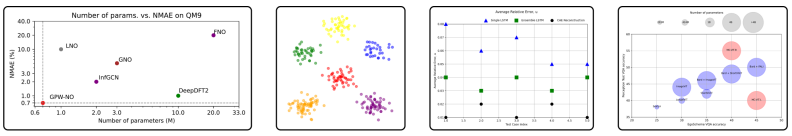


(1) heatmap_3 (2) heatmap_6 (3) heatmap_1 (4) heatmap_10

Figure 12: Examples of Heatmap subcategories.

Scatter: Scatter plot is classified based on the dot characteristics and data distribution. The taxonomy is divided into 4 subcategories as shown in Figure 13.

1. Base (basic scatter plot, uniform dot size, color may vary)
2. Diff-Shape (different dot shapes)
3. Diff-Size (different dot sizes, such as a bubble chart)
4. Clustered (scatter plot with clear clustering)



(1) scatters_7 (2) scatters_4 (3) scatters_13 (4) scatters_12

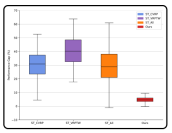
Figure 13: Examples of Scatter plot subcategories.

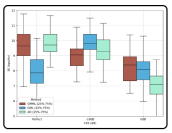
Box: Box plot is characterized by the orientation of boxes and data characteristics. The orientation can be horizontal or vertical, while data characteristics include the grouping of data and the presence of missing lines. The taxonomy is divided into 3 subcategories for each orientation, consisting of 6 subcategories in total, as shown in Figure 14.

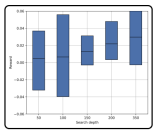
1. Base (single group of data, complete box shape)
2. Grouped (multiple groups of data, complete box shape)
3. Missed-Line (missing parts below the first quartile line and above the third quartile line)

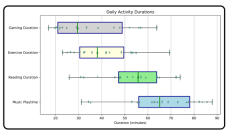
Error Bar: An Error bar chart is an enhanced variant of the basic bar chart, augmented with error margins to represent the variability or uncertainty within the data. Unlike Bar chart, Error Bar chart typically does not include the categories "Normalized" and "Sorted" in the dimension of data attributes, as these are less common. Therefore, Error Bar chart is classified into 6 data attributes subcategories for each orientation, resulting in 12 subcategories in total, as shown in Figure 15.

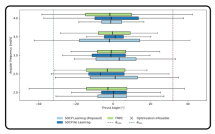
1. Base (single positive data set, unordered)
2. Grouped (multiple positive data sets, adjacent)
3. Stacked (multiple positive data sets laying atop one another)
4. Diverging (multiple stacked data sets expanding from a central axis)
5. With-Negative (data sets including negative values)
6. Reverse (exclusively negative data sets)


(1) box_3


(2) box_6


(3) box_22


(4) box_10


(5) box_25

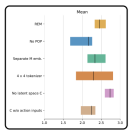
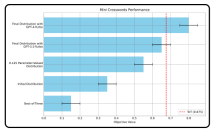
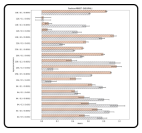
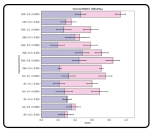
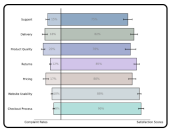

(6) box_16

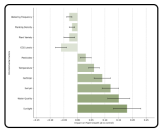
Figure 14: Examples of Box chart subcategories.

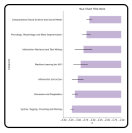

(1) errorbar_7

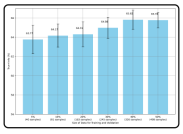

(2) errorbar_3

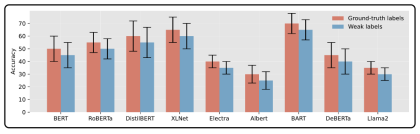

(3) errorbar_16


(4) errorbar_26

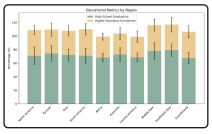

(5) errorbar_28

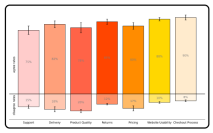

(6) errorbar_9

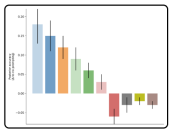

(7) errorbar_5



(8) errorbar_8


(9) errorbar_24


(10) errorbar_27


(11) errorbar_14

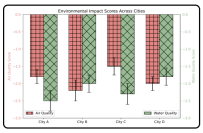
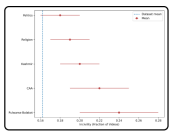
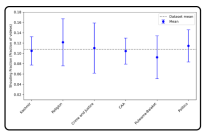

(12) errorbar_30

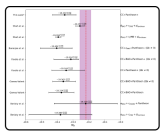
Figure 15: Examples of Error Bar chart subcategories.

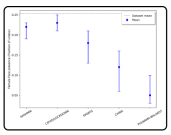
Error Point: Error Point chart enhances the classic Scatter chart by introducing error bars to each data point, conveying the inherent variability or uncertainty of the data. When classifying Error Point chart, a pivotal consideration lies in the orientation and symmetry of the error bars, leading us to define 5 key characteristics that govern their taxonomy. These characteristics—symmetry and asymmetry in both horizontal and vertical orientations, coupled with a composite category encompassing both directions—culminate into 5 comprehensive subcategories, as illustrated in Figure 16. Here we contend the following delineations:

1. Vertical-Horizontal Orientation: Distinguishing the direction of error bars, which can profoundly affect the interpretation of the data.
2. Symmetry-Asymmetry: Acknowledging whether error bars exhibit a mirrored consistency or an uneven distribution across data points.


(1) errorpoint_2


(2) errorpoint_1


(3) errorpoint_3


(4) errorpoint_5

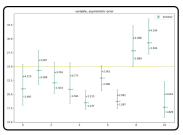
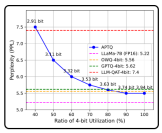

(5) errorpoint_8

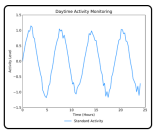
Figure 16: Examples of Error Point chart subcategories.

Line: Line chart is classified based on three primary attributes: data grouping, error visualization methods, and dot-line characteristics for grouped data, as shown in Figure 17.

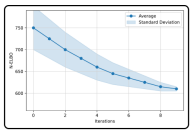
1. Data grouping: Single (individual dataset) or Grouped (multiple datasets).
2. Error visualization methods: Base (no error), Striped-Error (striped fill patterns), and Marker-Error (markers above and below data points).
3. Dot-Line Characteristics: For single dataset, No-Marker (data points without markers), Marker (data points with markers). For grouped datasets, Diff-Color (different colors for each group), Diff-Marker (different marker shapes for each group), and Diff-Line (different line styles for each group).



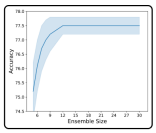
(1) line_22



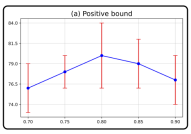
(2) line_46



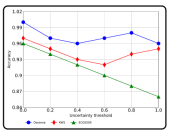
(3) line_24



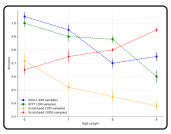
(4) line_37



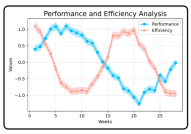
(5) line_52



(6) line_16



(7) line_31

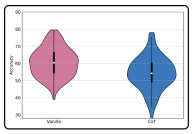


(8) line_61

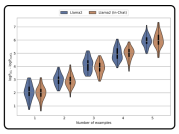
Figure 17: Examples of Line chart subcategories.

Violin: Violin chart is a combination of the Box chart and kernel Density chart. It provides a deep insight into the distribution of the data, indicating where individual data points fall within the overall data range. Based on the number of data groups and the shape and distribution of the violin form, the taxonomy is divided into 3 subcategories, as shown in Figure 18.

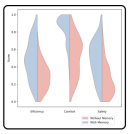
1. Base (standard shape, single data group)
2. Grouped-Symmetrical (standard shape, multiple data groups)
3. Grouped-Departed (half shapes joined, multiple data groups)



(1) violin_6



(2) violin_1

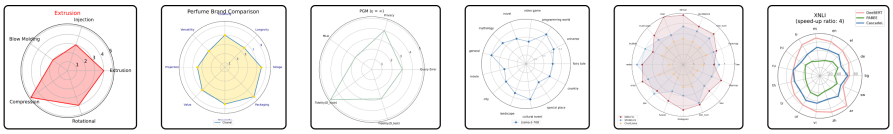


(3) violin_2

Figure 18: Examples of Violin chart subcategories.

Radar: Radar chart, also known as spider chart or star plot, is a graphical method of displaying multivariate data on a two-dimensional plane. It is particularly useful for showing performance metrics or skill assessments across multiple areas. Radar chart is classified based on three primary attributes: the number of data grouping, area fill, and dot-line characteristics for grouped data, as shown in Figure 19.

1. Data Grouping: Base (a single dataset) or Grouped (multiple datasets).
2. Area Fill: FillArea (areas within the radar chart filled) or NoFillArea (areas without fill to emphasize the outline).
3. Dot-Line Characteristics: For single dataset, NoMarker (dots without markers), Marker (dots with markers). For grouped datasets, Diff-Color (different colors for each group), Diff-Line (different line styles for each group), and Diff-Marker (different marker shapes for each group).

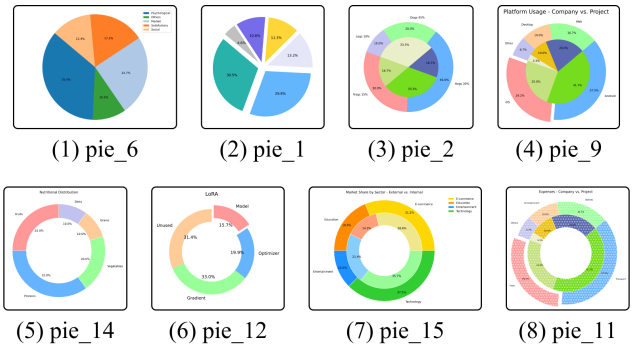


(1) radar_17 (2) radar_20 (3) radar_4 (4) radar_13 (5) radar_7 (6) radar_10

Figure 19: Examples of Radar chart subcategories.

Pie: Pie chart is a circular statistical graphic that divides a circle into slices to illustrate numerical proportion, whereas ring charts, also known as donut charts, utilize a hollow circle to serve a similar purpose. The presence of a ring, the number of layers, and the highlighted segment are the primary attributes in the classification of pie and ring charts, as shown in Figure 20.

1. Hollowness: Pie (no ring) or Ring (with a ring).
2. Layering: SingleLayer (single data series) or MultiLayer (multiple series or categories).
3. Highlighting: Base (without highlighted segments) or Explode (with one or more segments emphasized to capture viewer attention).



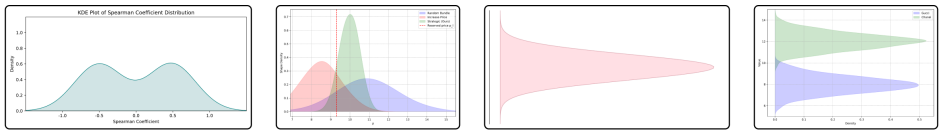
(1) pie_6 (2) pie_1 (3) pie_2 (4) pie_9

(5) pie_14 (6) pie_12 (7) pie_15 (8) pie_11

Figure 20: Examples of Pie and Ring chart subcategories.

Density: Density chart conveys the concentration and distribution of data within a space, often used to depict the magnitude or frequency across different areas or intervals. The orientation and data grouping are the primary attributes of the taxonomy of density plots, as shown in Figure 21.

1. Orientation: Vertical (y-axis as density) or Horizontal (x-axis as density).
2. Data Grouping: Base (single dataset) or Grouped (multiple datasets).

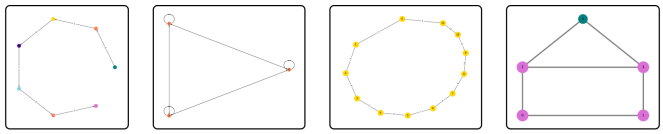


(1) density_3 (2) density_4 (3) density_2 (4) density_5

Figure 21: Examples of Density chart subcategories.

Graph: Graph chart commonly represents relationships and interconnected data through nodes (also known as vertices) and edges. It is widely used to depict networks, pathways, and complex inter-dependencies. The taxonomy of graph charts is based on the directionality and weight of the edges, resulting in 4 subcategories, as shown in Figure 22.

1. Directionality: Directed (edges have direction) or Undirected (edges have no direction).
2. Weight: Weighted (edges have weight) or Unweighted (edges have no weight).

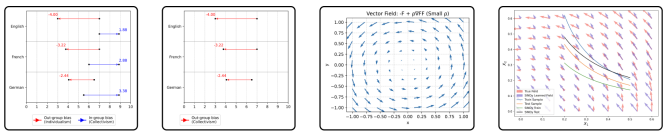


(1) graph_1 (2) graph_5 (3) graph_3 (4) graph_4

Figure 22: Examples of Graph chart subcategories.

Quiver: Quiver chart, also known as vector field plot, is used to display the magnitude and direction of vectors across a two-dimensional plane. This type of chart is particularly useful in physics and engineering to represent velocity fields, gradients, or other vector-based data spatially. The taxonomy of quiver charts is based on the vector quantity and data grouping, resulting in 4 subcategories, as shown in Figure 23.

1. Vector Quantity: Simple (limited vectors) or Field (vector field).
2. Data Groups: Single (single group) or Grouped (multiple groups).

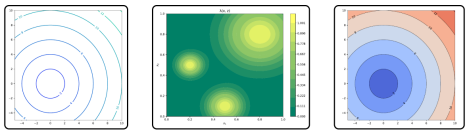


(1) quiver_2 (2) quiver_1 (3) quiver_4 (4) quiver_3

Figure 23: Examples of Quiver chart subcategories.

Contour: Contour chart, also called contour map or isoline graph, is used to represent three-dimensional data on a two-dimensional plane by plotting contour lines that connect points of equal value. This method is especially useful in fields like meteorology and geography, where it visually communicates variations in terrain elevation or changes in meteorological elements like temperature and pressure. The taxonomy of Contour chart is based on the representation of the contour lines, resulting in 3 subcategories, as shown in Figure 24.

1. Line (line representation)
2. Fill-Area (color-filled representation)
3. Combination (both line and color-filled)

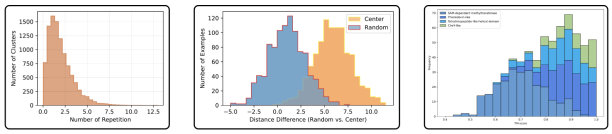


(1) contour_4 (2) contour_2 (3) contour_5

Figure 24: Examples of Contour chart subcategories.

Histogram: Histogram chart, often referred as hist chart, is a representation of data distribution where the data is grouped into ranges or “bins” and illustrated as bars to show the frequency of data points within each bin. It is particularly useful for identifying patterns or anomalies in the data, such as skewness, peaks, or gaps in the distribution. The taxonomy of histogram charts is based on the data grouping and positioning, resulting in 3 subcategories, as shown in Figure 25.

1. Base (single dataset)
2. Overlaid (overlapping multiple datasets)
3. Stacked (stacked multiple datasets)

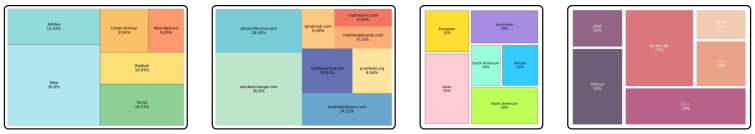


(1) hist_5 (2) hist_4 (3) hist_2

Figure 25: Examples of Histogram chart subcategories.

Treemap: Treemap displays hierarchical data through nested rectangles, where each branch of the tree is represented by a rectangle that contains smaller rectangles corresponding to sub-branches. This method allows for efficient use of space, enabling the viewer to quickly compare sizes and proportions within the hierarchy, and is especially useful for analyzing large datasets to reveal relationships and patterns. The taxonomy of treemap charts is based on the compactness and edge presence, resulting in 4 subcategories, as shown in Figure 26.

1. Tight-Edge (compact with border)
2. Tight-NoEdge (compact without border)
3. Loose-Edge (loose with border)
4. Loose-NoEdge (loose without border)

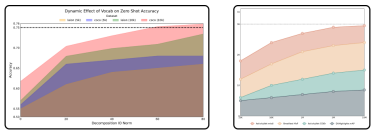


(1) treemap_3 (2) treemap_1 (3) treemap_5 (4) treemap_2

Figure 26: Examples of Treemap chart subcategories.

Area: Area chart is a graphical representation where data points are connected by line segments and the area between the line and the axis is filled with color or patterns, providing a sense of volume. It is particularly useful for visualizing the cumulative magnitude of values over time, allowing for a clear perception of trends and changes in the data series. The taxonomy of area charts is based on the presence of markers, resulting in 2 subcategories as shown in Figure 27.

1. Base (without markers)
2. Marker (with markers)



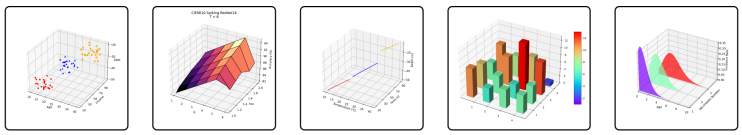
(1) area_3 (2) area_1

Figure 27: Examples of Area chart subcategories.

3D charts: 3D chart extends two-dimensional charting into three dimensions with spatial representations, offering an added layer of depth to represent additional data variables or to enhance visual appeal. Based on the above chart types and the additional surface representation in 3D space, we classify 3D chart into 5 subcategories, as shown in Figure 28.

1. Scatter (3D scatter chart)
2. Surface (3D surface chart)
3. Line (3D line chart)

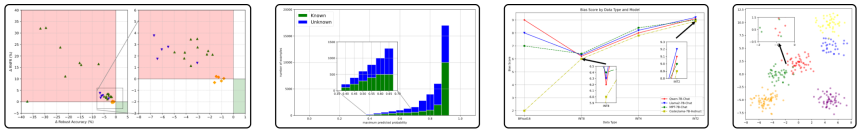
4. Bar (3D bar chart)
5. Density (3D density plot)



(1) 3D_8 (2) 3D_3 (3) 3D_9 (4) 3D_7 (5) 3D_14

Figure 28: Examples of 3D chart subcategories.

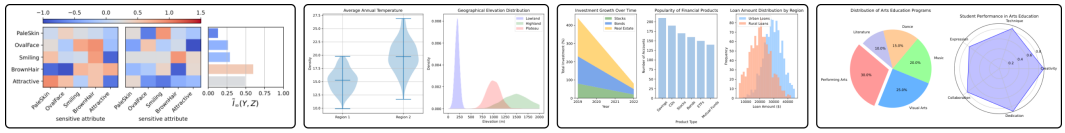
PIP: PIP chart insets a magnified or reduced portion of the main plot to highlight key data features, as in Figs. 29 (1) and (2). They enhance readability for complex datasets and facilitate comparative analyses, as demonstrated in Figs. 29 (3) and (4).



(1) PIP_1 (2) PIP_2 (3) PIP_3 (4) PIP_9

Figure 29: Examples of PIP charts.

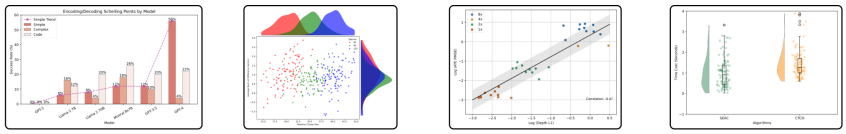
Multidiff: A Multidiff chart combines at least two different chart types across multiple subplots, with each subplot presenting one type. Derived from the categories above and using diverse layouts, Multidiff charts offer numerous configurations, as shown in Fig. 30.



(1) multidiff_10 (2) multidiff_20 (3) multidiff_21 (4) multidiff_24

Figure 30: Examples of Multidiff charts.

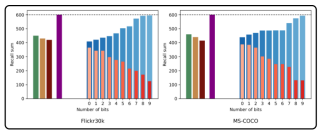
Combination: Combination chart merges features from different chart types into one plot, offering multilayered presentation. Unlike Multidiff chart with multiple subplots for different categories, Combination chart displays multiple categories in a single plot. As in Fig. 31 (2), a scatter plot illustrates the data distribution while adjacent density plots detail the axis-specific spread. Additional examples are shown in Fig. 31.



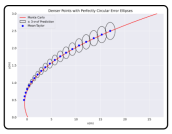
(1) Combination_2 (2) Combination_6 (3) Combination_11 (4) Combination_26

Figure 31: Examples of Combination charts.

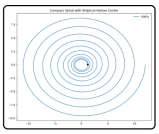
HR: An HR chart is one that defies the above 21 categories or is challenging to identify. HR chart is typically modified from common charts with distinctive features like custom visual arrangement or atypical markers, as shown in Fig.32.



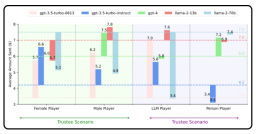
(1) HR_3



(2) HR_4



(3) HR_10



(4) HR_20

Figure 32: Examples of HR charts.

D DETAILS OF EVALUATION METRICS

In this section, we present the details of evaluation metrics, including GPT-4o score for high-level metrics and Text Score, Layout Score, Type Score and Color Score for low-level metrics. For more information, please refer to our source code.

D.1 GPT-4O SCORE

We employ GPT-4o (OpenAI, 2024) to assess the extend to which the generated figure corresponds to the ground-truth figure. The specific content of the prompt is presented in Fig. 33. Specifically, we input both the generated and the ground-truth figures into the GPT-4o simultaneously. Then, GPT-4o is instructed to evaluate the similarity between the two figures, taking into account six dimensions: text, layout, type, data, style, and clarity. Subsequently, GPT-4o outputs a score ranging from 0 to 100 to represent the degree of similarity between the figures.

Prompt for GPT-4o Score

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 33: Prompt for GPT-4o Score.

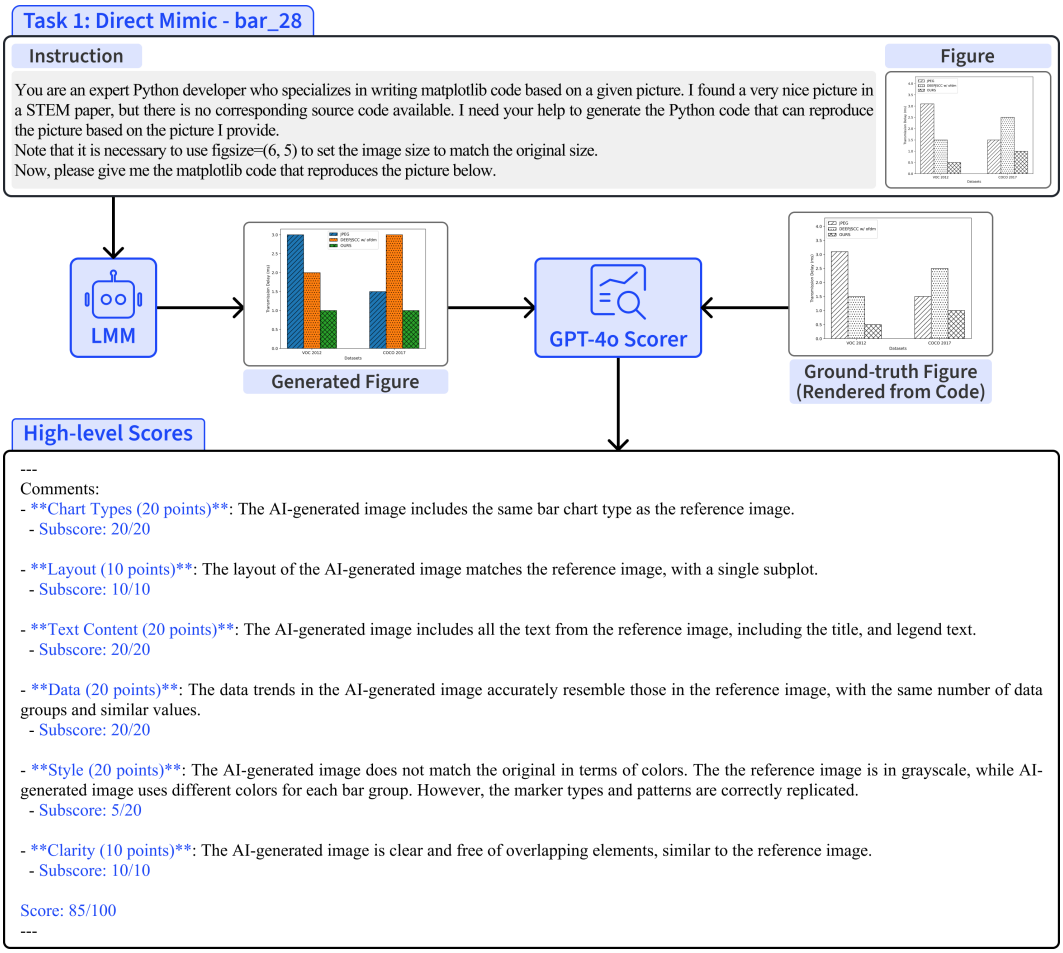


Figure 34: Example of GPT-4o's scoring results for a Direct Mimic example (bar_28).

Stability of Evaluation with GPT-4o Score. We conduct the high-level evaluation for GPT-4o on the Direct Mimic task for 5 times to assess the stability of GPT-4o Score. The result indicate a mean GPT-4o Score of 83.4 with a standard deviation of 0.08, demonstrating the stability of GPT-4o Score.

Cost of Evaluation with GPT-4o Score. A single-round evaluation with GPT-4o Score on the Direct Mimic task approximately costs $5.25. Utilizing OpenAI's batch services further reduces this cost to $2.63 per round.

Longevity of Evaluation with GPT-4o Score. The GPT-4o Score metric is designed with long-term viability in mind. Our approach provides a meta-evaluation framework that can adapt to evolving language models. While currently leveraging GPT-4o, the method is model-agnostic and can be implemented with other advanced LLMs such as Claude or Gemini. This adaptability ensures that as more capable models emerge, the evaluation process can seamlessly transition to utilize these superior systems. Importantly, this approach has demonstrated increasing correlation with human assessments as model capabilities improve, as evidenced by studies like AlpacaEval (Li et al., 2023) and MT-Bench (Zheng et al., 2023).

GPT-4o Score Examples. To demonstrate our evaluation framework, we present GPT-4o's scoring examples for both Direct Mimic and Customized Mimic tasks. Figs. 34 to 37 show the evaluation results for two representative charts of different complexity levels (bar_28 and CB_29). These examples demonstrate how GPT-4o systematically evaluates various aspects of chart reproduction across different chart complexities and task types.

Correlation Coefficient Comparison with CLIP Score. CLIP Score (Radford et al., 2021) is widely used for assessing figure similarity. However, our preliminary experiments indicate that it struggles

Task 2: Customized Mimic - bar_28

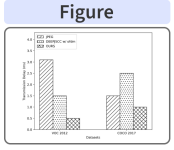
Instruction

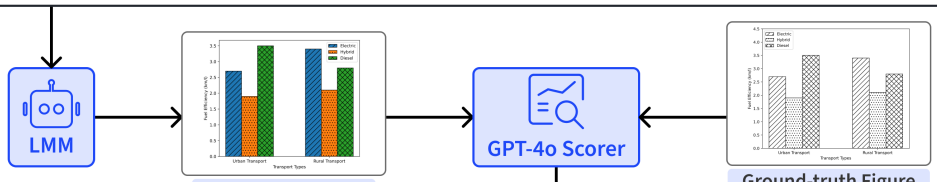
There is a reference image for you and I also have a set of data about fuel efficiency of different types of transport in urban and rural areas. Please refer to the reference image and generate Python matplotlib code that can create a chart that exactly mirrors the style of the reference image and accurately represents my data.

```
datasets = ["Urban Transport", "Rural Transport"]
electric = [2.7, 3.4]
hybrid = [1.9, 2.1]
diesel = [3.5, 2.8]

# X-axis positions
x = np.arange(len(datasets))

# Bar width
width = 0.2
labels = ["Electric", "Hybrid", "Diesel"]
ylim = [0, 4.5]
ylabel = "Fuel Efficiency (km/l)"
xlabel = "Transport Types"
```

Now, please output the Python matplotlib code that can create a chart that exactly mirrors the style of the reference image and accurately represents my data.
Note: You must copy the data I provide completely into your code, and ensure that the code you generate contains the data and is directly executable.
Note that it is necessary to use figsize=(6, 5) to set the image size.

Figure





High-level Scores

Comments:
- **Chart Types (20 points)**: The AI-generated image includes the same bar chart type as the reference image.
  - Subscore: 20/20

- **Layout (10 points)**: The arrangement of the subplots in the AI-generated image matches the reference image, with a single subplot in both.
  - Subscore: 10/10

- **Text Content (20 points)**: The AI-generated image includes all the text from the reference image, such as the title, axis labels, and legend.
  - Subscore: 20/20

- **Data (20 points)**: The data trends in the AI-generated image accurately resemble those in the reference image, and the number of data groups is the same.
  - Subscore: 20/20

- **Style (20 points)**: The AI-generated image does not match the original in terms of colors. The reference image is in black and white, while the AI-generated image uses colors (blue, orange, green) . The marker types (hatch patterns) are correctly replicated, but the absence of color is a significant deviation.
  - Subscore: 5/20

- **Clarity (10 points)**: The AI-generated image is clear and free of overlapping elements. The bars and text are well-defined.
  - Subscore: 10/10

Score: 85/100

Figure 35: Example of GPT-4o's scoring results for a Customized Mimic example (bar_28).

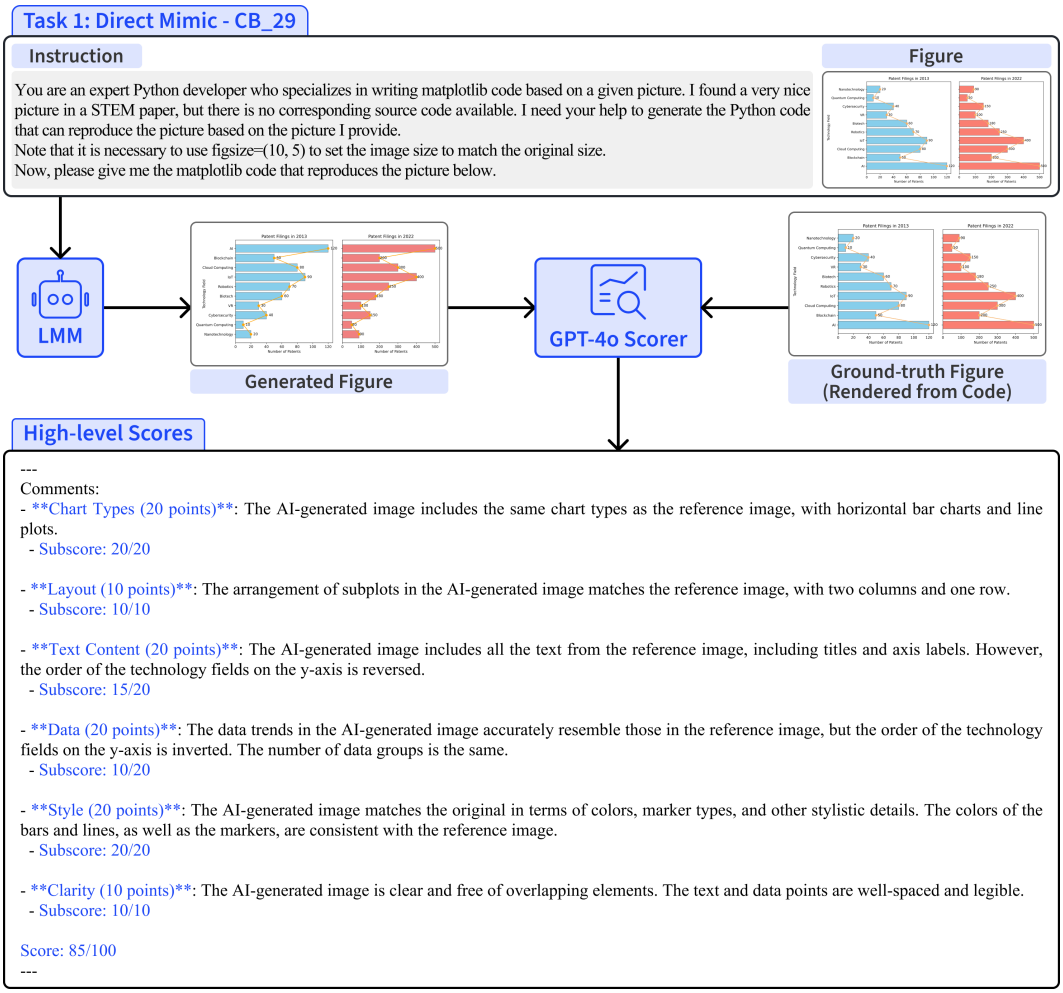


Figure 36: Example of GPT-4o's scoring results for a Direct Mimic example (CB_29).

Task 2: Customized Mimic - CB_29

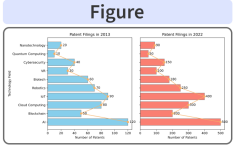
Instruction

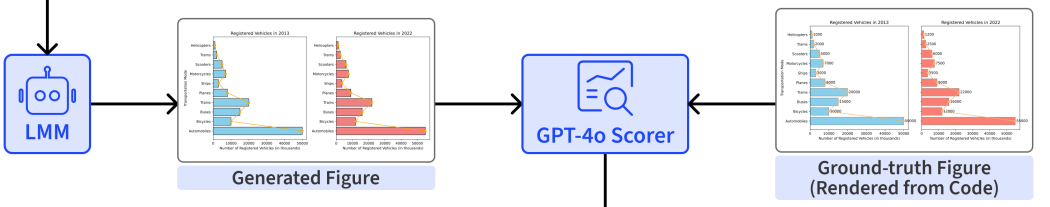
There is a reference image for you and I also have a set of data about the number of registered vehicles for different transportation modes in 2013 and 2022. Please refer to the reference image and generate Python matplotlib code that can create a chart that exactly mirrors the style of the reference image and accurately represents my data.

```
# Transportation modes
transport_modes = [
  "Automobiles",
  "Bicycles",
  "Buses",
  "Trains",
  "Planes",
  "Ships",
  "Motorcycles",
  "Scooters",
  "Trams",
  "Helicopters",
]

# Number of registered vehicles in 2013 (in thousands)
vehicles_2013 = [50000, 10000, 15000, 20000, 8000, 3000, 7000, 5000, 2000, 1000]
# Number of registered vehicles in 2022 (in thousands)
vehicles_2022 = [55000, 12000, 16000, 22000, 9000, 3500, 7500, 6000, 2500, 1200]
x_label = "Number of Registered Vehicles (in thousands)"
y_label = "Transportation Mode"
ax1_title = "Registered Vehicles in 2013"
ax2_title = "Registered Vehicles in 2022"
```

Now, please output the Python matplotlib code that can create a chart that exactly mirrors the style of the reference image and accurately represents my data.
Note: You must copy the data I provide completely into your code, and ensure that the code you generate contains the data and is directly executable.
Note that it is necessary to use figsize=(10, 5) to set the image size.

Figure





High-level Scores

Comments:
- **Chart Types (20 points)**: The AI-generated image includes all the chart types present in the reference image, which are horizontal bar charts with overlaid line plots.
  - Subscore: 20/20

- **Layout (10 points)**: The arrangement of subplots in the AI-generated image matches the reference image, with two subplots arranged side by side.
  - Subscore: 10/10

- **Text Content (20 points)**: The AI-generated image includes all the text from the reference image, such as titles and axis labels. However, the the reference image includes additional text annotations on the bars, which are not present in AI-generated image.
  - Subscore: 10/20

- **Data (20 points)**: The data trends in the AI-generated image resemble those in the original image, and the number of data groups is the same.
  - Subscore: 20/20

- **Style (20 points)**: The AI-generated image matches the original in terms of colors, marker types, and other stylistic details. However, the reference image includes additional data labels on the bars, which are not present in the AI-generated image.
  - Subscore: 18/20

- **Clarity (10 points)**: The AI-generated image is clear and free of overlapping elements. The additional data labels do not obscure the visualization.
  - Subscore: 10/10

Score: 88/100

Figure 37: Example of GPT-4o's scoring results for a Customized Mimic example (CB_29).

to differentiate variations in types and other critical elements in charts, resulting in a low correlation coefficient of 0.53 with human evaluation. In contrast, as stated in Sec. 4.3, GPT-4o Score achieves a correlation coefficient of 0.70 with human evaluation. Therefore, we adopt GPT-4o Score as the high-level evaluation metric.

D.2 TEXT SCORE

Listing 1: An exemplary Python code for logging text information.

```
from matplotlib.backends.backend_pdf import RendererPdf

drawed_texts = []

def log_function(func):
   def wrapper(*args, **kwargs):
      global drawed_texts

      text_string = args[4]
      drawed_texts.append( text_string )

      return func(*args, **kwargs)

   return wrapper

RendererPdf.draw_text = log_function(RendererPdf.draw_text)
```

In order to accurately capture the textual content presented in the rendered figures, our code tracer monitors the function used to add text to the rendered PDF, logging each textual element. An exemplary Python code is provided in Listing 1, where we record the text elements by adding a log wrapper to the "`draw_text()`" function in the matplotlib package. Subsequently, we employ this approach to extract the text information from both the generated code and the ground-truth code.

Based on these two groups of texts, precision is defined as the ratio of the number of correctly captured ground-truth text to the total number of text in the generated figure. Recall is defined as the ratio of the number of correctly captured ground-truth text to the total number of text in the ground-truth figure. The F1-score, calculated using precision and recall, serves as Text Score.

D.3 LAYOUT SCORE

Listing 2: An exemplary Python code for logging layout information.

```
def get_gridspec_layout_info(fig):

   """
   Get the layout information of a given figure.
   Args:
      fig (matplotlib.figure.Figure): The figure to extract layout
         information from.
   Returns:
      layout_info (list): A list of dictionaries, each containing layout
         information for each subplot in the figure.
   """

   layout_info = {}
   for ax in fig.axes:
      spec = ax.get_subplotspec()
      if spec is None:
         continue
      gs = spec.get_gridspec()
      nrows, ncols = gs.get_geometry()
      row_start, row_end = spec.rowspan.start, spec.rowspan.stop - 1
      col_start, col_end = spec.colspan.start, spec.colspan.stop - 1
```

```
        layout_info[ax] = dict(nrows=nrows, ncols=ncols,
            row_start=row_start, row_end=row_end, col_start=col_start,
            col_end=col_end)
    layout_info = list(layout_info.values())
    return layout_info

layout_info = get_gridspec_layout_info(fig=plt.gcf())
```

The layout refers to the organization of subplots within a chart figure. In each figure implemented using matplotlib, multiple axes objects are present, each containing layout information that specifies its position within the figure. To analyze the layout, we iterate through each axis object in the figure and obtain their respective layout information. An exemplary Python code to accomplish this process is provided in Listing 2, where the position information of each axis is recorded. Subsequently, we gather the layout information from both the ground-truth code and the generated code.

Based on these two groups of layout information, precision is defined as the ratio of the number of correctly identified ground-truth layouts to the total number of layouts in the generated figure. Recall is defined as the ratio of the number of correctly identified ground-truth layouts to the total number of layouts in the ground-truth figure. The F1-score, calculated using precision and recall, serves as Layout Score.

D.4 Type Score

Listing 3: An exemplary Python code for logging type information.

```
from matplotlib.axes import Axes
import inspect

called_functions = {}

def log_function(func):
    def wrapper(*args, **kwargs):

        file_name = inspect.getfile(func)
        name = file_name + "/" + func.__name__
        called_functions[name] = called_functions.get(name, 0) + 1
        result = func(*args, **kwargs)
        return func(*args, **kwargs)

    return wrapper

Axes.bar = log_function(Axes.bar)
```

The matplotlib package provides a variety of functions for easily generating diverse chart types, such as using "`axes.bar()`" to create bar charts. By monitoring the invocations of these functions, we can identify the types of charts being utilized. A successful invocation of a plot functions indicates the incorporation of a specific chart type in the final visualization. Listing 3 presents an exemplary Python code that demonstrates this approach, where we introduce a logger to the "`bar()`" function. Subsequently, we gather the chart types from both the generated and ground-truth code.

Based on these two groups of chart types, precision is defined as the ratio of the number of correctly identified ground-truth chart types to the total number of chart types in the generated figure. Recall is defined as the ratio of the number of correctly identified ground-truth chart types to the total number of chart types in the ground-truth figure. The F1-score, calculated using precision and recall, serves as Type Score.

D.5 Color Score

Listing 4: An exemplary Python code for logging color information.

```
from matplotlib.axes import Axes
```

```
import inspect

drawed_colors = []

def log_function(func):
   def wrapper(*args, **kwargs):

      func_name = inspect.getfile(func) + "/" + func.__name__
      result = func(*args, **kwargs)

      for item in result:
         color = item.get_facecolor()
         drawed_colors.append( func_name + "--" + color )

Axes.bar = log_function(Axes.bar)
```

In the matplotlib package, each plot function returns a chart type instance at the end of the function invocation. These instances contain various attributes, including those related to color properties, such as facecolor, edgecolor and colormap. To assess the color attributes, we employ the code tracer that captures the color information of each chart type instance. An example Python code demonstrating this approach is provided in Listing 4. Subsequently, we gather the color information from both the ground-truth code and the generated code and calculate the similarity between them.

It is noteworthy that through preliminary experiments, we find that using exact color matching resulted in very low color similarity, as even slight variations would result in a similarity score of zero. To address this issue, we employ the CIEDE2000 color difference formula (Luo et al., 2001), which converts the matching value between two colors from a discrete $[0, 1]$ scale to a continuous range between 0 and 1. Finally, we calculate the maximum color similarity between the sets of colors in the ground-truth code and the generated code. Precision is defined as the ratio of maximum color similarity to the total number of color in the generated figure. Recall is defined as the ratio of maximum color similarity to the total number of color in the ground-truth figure. The F1-score, calculated using precision and recall, serves as Color Score.

E MODEL CONFIGURATIONS AND PROMPTING METHODS

E.1 GENERATION CONFIGURATIONS

Following previous setups (Wang et al., 2023b; Shi et al., 2024), for open-weight models, we set the temperature $\tau = 0.1$ to achieve optimal results, while for proprietary models, we set the temperature $\tau = 0$ for greedy decoding. For all models, we set the maximum generation length to 4096. Additionally, we use BF16 for model inference for open-weight models. All models are inferred on A100 80G GPU.

E.2 PROMPTS

We provide prompts for Direct, HintEnhanced, SelfReflection and Scaffold Prompting in Figs. 38 to 42. for Direct Prompting, we meticulously design a separate prompt for open-source models to achieve optimal results, as shown in Fig. 38. while the prompt for proprietary Models is shown in Fig. 39.

Prompt for Direct Prompting (Open-Weight Models)

You are an expert Python developer who specializes in writing matplotlib code based on a given picture. I found a very nice picture in a STEM paper, but there is no corresponding source code available. I need your help to generate the Python code that can reproduce the picture based on the picture I provide.

Please note that it is necessary to use figsize=({width}, {height}) to set the image size to match the original size. Additionally, I will not provide you with the actual data in the image, so you have to extract the actual data by yourself and based on the extracted data to reproduce the image. Ensure that the code you provide can be executed directly without requiring me to add additional variables.
Now, please give me the matplotlib code that reproduces the picture below.

Figure 38: Prompt for Direct Prompting (Open-Weight Models). {text} in blue font represents placeholders, which varies according to different test examples.

Prompt for Direct Prompting (Proprietary Models)

You are an expert Python developer who specializes in writing matplotlib code based on a given picture. I found a very nice picture in a STEM paper, but there is no corresponding source code available. I need your help to generate the Python code that can reproduce the picture based on the picture I provide.

Note that it is necessary to use figsize=({width}, {height}) to set the image size to match the original size.
Now, please give me the matplotlib code that reproduces the picture below.

Figure 39: Prompt for Direct Prompting (Proprietary Models). {text} in blue font represents placeholders, which varies according to different test examples.

Prompt for HintEnhanced Prompting

You are an expert Python developer who specializes in writing matplotlib code based on a given picture. I found a very nice picture in a STEM paper, but there is no corresponding source code available. I need your help to generate the Python code that can reproduce the picture based on the picture I provide.

To ensure accuracy and detail in your recreation, begin with a comprehensive analysis of the figure to develop an elaborate caption. This caption should cover, but not be limited to, the following aspects:
1. Layout Analysis: e.g., identify the picture's composition, noting the presence and arrangement of any subplots.
2. Chart Type Identification: e.g., determine how many charts within a subplot. Are they independent, or do they share a common axis?
3. Data Analysis: e.g., summarize the data trend or pattern.
4. Additional Features: e.g.,identify any supplementary elements such as legends, colormaps, tick labels, or text annotations that contribute to the figure's clarity or aesthetic appeal.

Now, given the picture below, please first output your comprehensive caption and then use the caption to assist yourself to generate matplotlib code that reproduces the picture. Note that it is necessary to use figsize=({width}, {height}) to set the image size to match the original size.

Figure 40: Prompt for HintEnhanced Prompting. {text} in blue font represents placeholders, which varies according to different test examples.

Prompt for Scaffold Prompting

You are an expert Python developer who specializes in writing matplotlib code based on a given picture. I found a very nice picture in a STEM paper, but there is no corresponding source code available. I need your help to generate the Python code that can reproduce the picture based on the picture I provide.

I will provide you with two images. The first image is the original picture. The second image is the picture overlaid with a dot matrix of a shape of {dot_matrix_height} * {dot_matrix_width} to help you with your task, and each dot is labeled with two-dimensional coordinates (x,y). Within each column, the x-coordinate increases from top to bottom, and within each row, the y-coordinate increases from left to right.

Please first use this dot matrix as reference anchors to generate the description of the picture (e.g., between dot A and dot B is something) and generate matplotlib code that reproduces the picture.

Note that it is necessary to use figsize=({width}, {height}) to set the image size to match the original size.

Figure 41: Prompt for Scaffold Prompting. {text} in blue font represents placeholders, which varies according to different test examples.

Prompt for SelfRevision Prompting

You are an expert Python developer who specializes in writing matplotlib code based on a given picture. I have a code for implementing the reference picture as follows:

Now, I have the Python matplotlib code for implementing the reference picture as follows:
```python
{python_code}
```
The rendered picture of the code is:

Now, please compare whether the renderer picture is the same as the reference picture. The difference may cover, but not be limited to, the following aspects:
1. Chart Types: Does the AI-generated image include all chart types present in the reference image (e.g., line charts, bar charts, etc.)?
2. Layout: Does the arrangement of subplots in the AI-generated image match the reference image (e.g., number of rows and columns)?
3. Text Content: Does the AI-generated image include all text from the reference image (e.g., titles, annotations, axis labels), excluding axis tick labels?
4. Data: 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: 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: Is the AI-generated image clear and free of overlapping elements?

- If the generated picture matches the reference, please output the original implementation code.
- If there are discrepancies, first list the specific differences between the two pictures. Then, modify the existing code to address these differences, ensuring the revised code is capable of reproducing the reference picture. Finally, output the revised code.

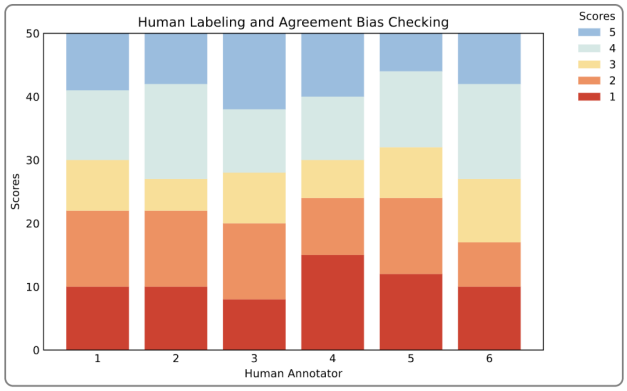
Note that it is necessary to use figsize=({width}, {height}) to set the image size to match the original size.

Figure 42: Prompt for SelfRevision Prompting. {text} in blue font represents placeholders, which varies according to different test examples.

E.3 CASES OF DIFFERENT PROMPTING METHODS

We provide cases of HintEnhanced, SelfReflection and Scaffold Prompting in Figs. 43 to 46. For an analysis of error cases related to Direct Prompting, please refer to Appendix H.

<HintEnhanced Prompting>



Response

Comprehensive Caption: The figure presented is a bar chart featuring six grouped sets of vertical bars, each representing different human annotators labeled from 1 to 6 along the x-axis. **Within each group, there are five bars corresponding to various score categories, making it a grouped bar chart.** The bars are color-coded based on scores ranging from 1 to 5, allowing for a clear visualization of the distribution of scores for each annotator.

The height of each bar reflects the count of annotations assigned to a specific score. Notably, the distribution of scores varies among the annotators. For instance, some annotators tend to have a higher frequency of lower scores (1 and 2), while others display more balanced trends or different patterns altogether.

.....

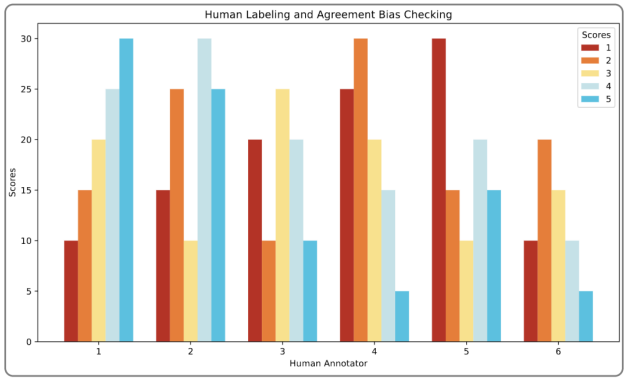
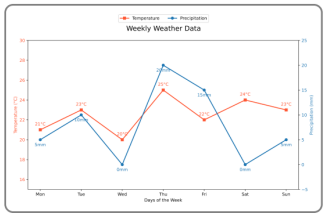
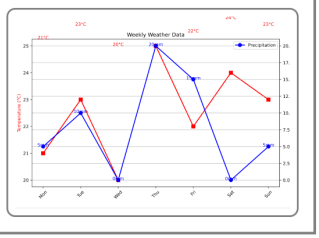


Figure 43: An error case of HintEnhanced Prompting. The **text** highlighted in red is LMM's incorrect understanding regarding provided figures. The remaining part of response is omitted with ".....".

<SelfReflection Prompting>





Response

Comparison of Images:
1. Chart Types:
• Both images include line charts for temperature and precipitation.
2. Layout:
• Both images have a single chart layout without subplots.
3. Text Content:
• **The title "Weekly Weather Data" appears in generated images should be changed to "Weekly Weather Report" to match the reference image.**
• Axis labels for temperature and precipitation are included in both images.
4. Data:
• The data points for temperature and precipitation trends appear consistent across both images.
• The number of data groups is the same.
5. Style:
• **Color differences: The generated image appears to have some variations in colors. The generated image should use green for temperature and orange for precipitation.**
.....

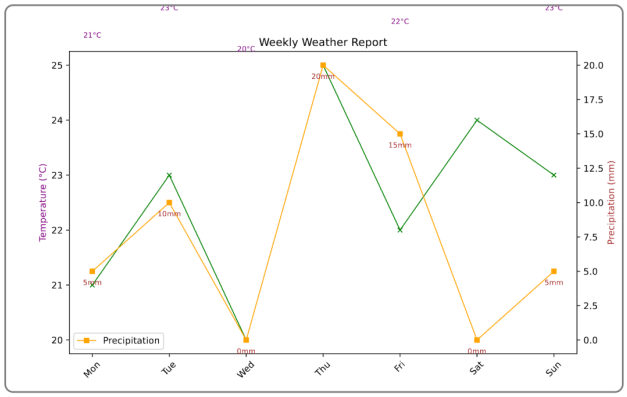
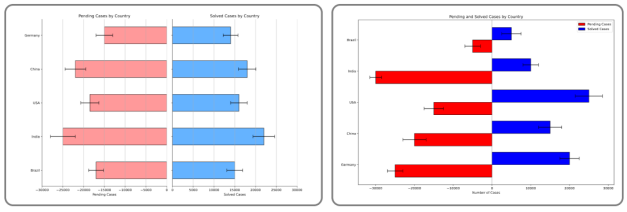


Figure 44: An error case of SelfReflection Prompting. The **text** highlighted in red is LMM's incorrect understanding regarding provided figures. The remaining part of response is omitted with ".....".

<SelfReflection Prompting>



Response

Here are the specific differences between the two images:

1. Chart Types:
• **The reference image uses two separate bar charts (pending and solved cases), while the generated image combines both into one chart.**
2. Layout:
• **The layout of the reference image displays two distinct sections for pending and solved cases, while the generated image merges them into a single horizontal bar chart.**
3. Text Content:
• **The titles differ; the reference image has "Pending Cases by Country" and "Solved Cases by Country," whereas the generated image has "Pending and Solved Cases by Country."**
• **The x-axis label in the reference image specifies "Pending Cases" and "Solved Cases" separately, while the generated image only uses "Number of Cases."**
.....

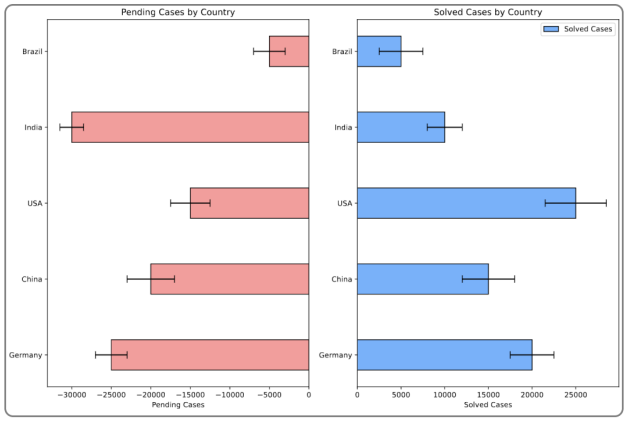


Figure 45: An good case of SelfReflection Prompting. The **text** highlighted in blue is LMM's correct reflection regarding provided figures. The remaining part of response is omitted with "……".

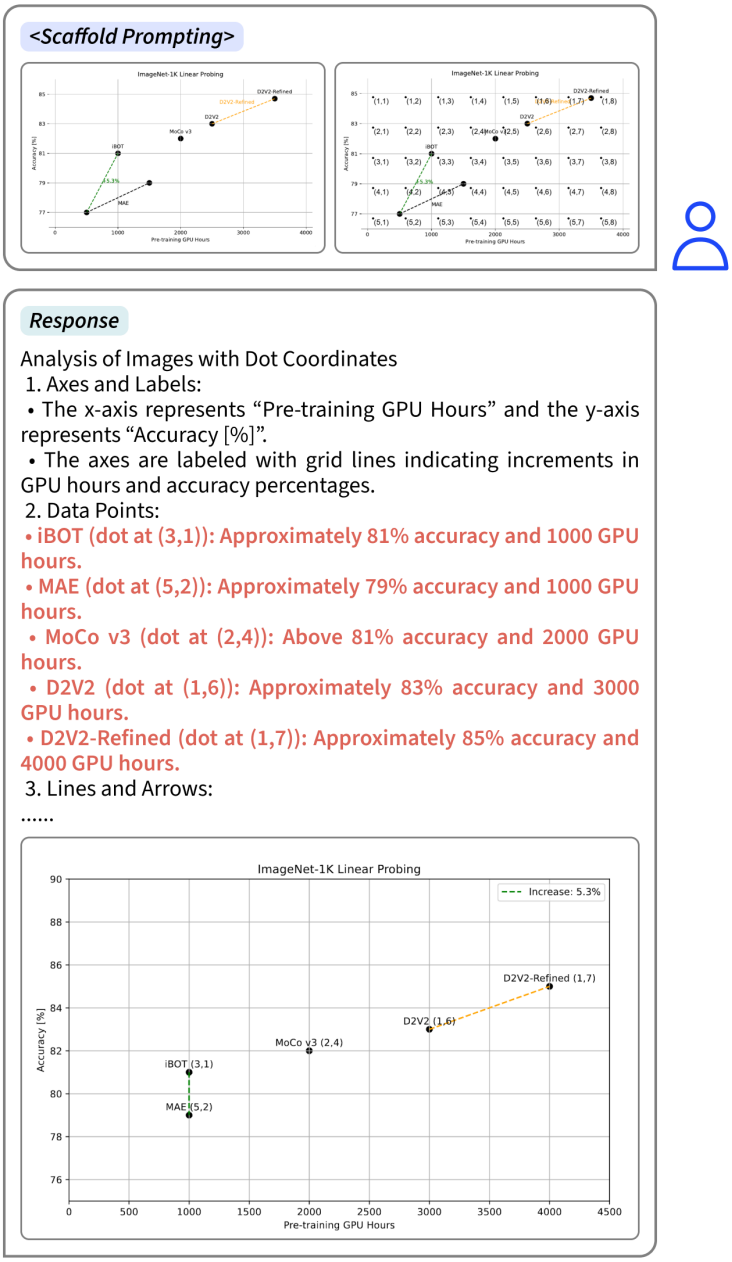


Figure 46: An error case of Scaffold Prompting. The **text** highlighted in red is LMM's incorrect understanding regarding provided figures. The remaining part of response is omitted with ".....".

E.4 DETAILS OF MODELS

We list the evaluated models in Tab. 9.

Model	Params	Language Model	Vision Model	Model Code/API
GPT-4o	-	-	-	`gpt-4o-2024-05-13`
Claude-3-opus	-	-	-	`claude-3-opus-20240229`
GeminiProVision	-	-	-	`gemini-pro-vision`
InternVL2-Llama3-76B	76.0B	Llama-3-70B-Instruct	InternViT-6B	`OpenGVLab/InternVL2-Llama3-76B`
LLaVA-Next-Yi-34B	34.8B	Yi-34B	CLIP ViT-L/14	`llava-hf/llava-v1.6-34b-hf`
InternVL2-26B	26.0B	InternLM2-20B	InternViT-6B	`OpenGVLab/InternVL2-26B`
Cogvlm2-llama3-chat-19B	19.2B	Llama-3-8B-Instruct	EVA2-CLIP-E	`THUDM/cogvlm2-llama3-chat-19B`
Phi-3-Vision-128K-Instruct	4.2B	Phi-3	CLIP ViT-L/14	`microsoft/Phi-3-vision-128k-instruct`
IDEFICS2-8B	7.6B	Mistral-7B	SigLip-400M	`HuggingFaceM4/idefics2-8b`
LLaVA-Next-Mistral-7B	7.6B	Mistral-7B	CLIP ViT-L/14	`llava-hf/llava-v1.6-mistral-7b-hf`
DeepSeek-VL-7B	7.3B	DeekSeek-7B	SAM-B & SigLIP-L	`deepseek-ai/deepseek-vl-7b-chat`
MiniCPM-Llama3-V2.5	8.4B	Llama3-8B-Instruct	SigLip-400M	`openbmb/MiniCPM-Llama3-V-2_5`
Qwen2-VL-7B	8.2B	Qwen2-7B	ViT-600M	`Qwen/Qwen2-VL-7B-Instruct`
InternVL2-8B	8.1B	InternLM2.5-7B	InternViT-300M	`OpenGVLab/InternVL2-8B`
InternVL2-4B	4.2B	Phi-3	InternViT-300M	`OpenGVLab/InternVL2-4B`
Qwen2-VL-2B	2.6B	Qwen2-1.5B	ViT-600M	`Qwen/Qwen2-VL-2B-Instruct`
InternVL2-2B	2.2B	InternLM2-1.8B	InternViT-300M	`OpenGVLab/InternVL2-2B`

Table 9: Model code/API of the evaluated models.

F CORRELATION WITH HUMAN EVALUATION

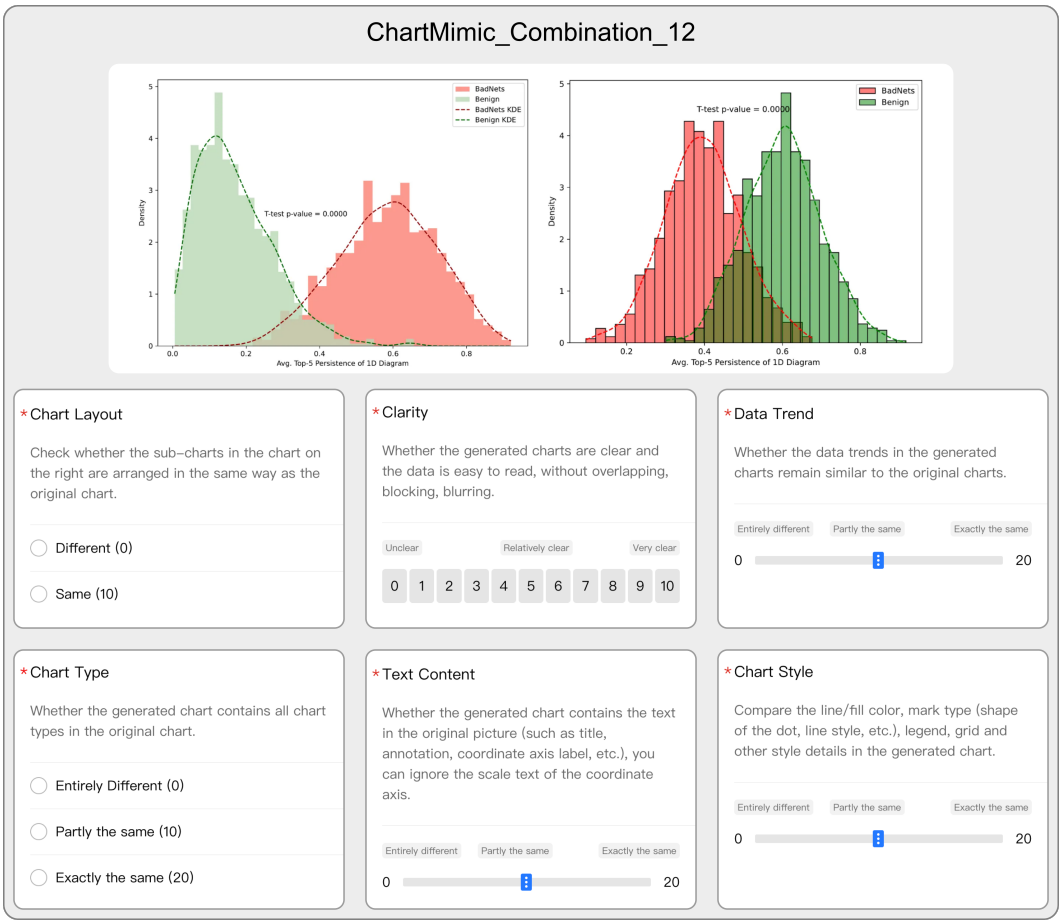


Figure 47: A screenshot of the human evaluation questionnaire.

In order to evaluate the reliability of the proposed automatic metrics, we calculate their correlation with human evaluations. Specifically, a selection of 300 test examples from the Direct Mimic task is utilized. Subsequently, we gather the results generated by GPT-4o using four different prompting methods (Sec. 4.2) on these test examples. This process yields $1,200$ figures to be assessed.

Each figure is independently assessed by three evaluators through a questionnaire, who assign scores on a scale of 0 to 100 based on the similarity between the generated figures and the ground-truth figures. The final score for each figure is calculated as the mean of scores given by the three evaluators. Our evaluators comprise volunteer graduate students holding bachelor's degrees in computer science. To facilitate their assessments, we provide a scoring rubric in the questionnaire, which closely aligns with the low-level and high-level metric evaluations. The scoring criteria encompass the following dimensions: chart type, layout, textual content, data, style, and clarity. A screenshot of the final questionnaire is available in Fig. 47.

Upon collecting the human evaluation results, we calculate the Pearson correlation coefficient to ascertain the correlation between our automatic metrics and human evaluation.

G CORRELATION WITH CHART UNDERSTANDING AND CODE GENERATION

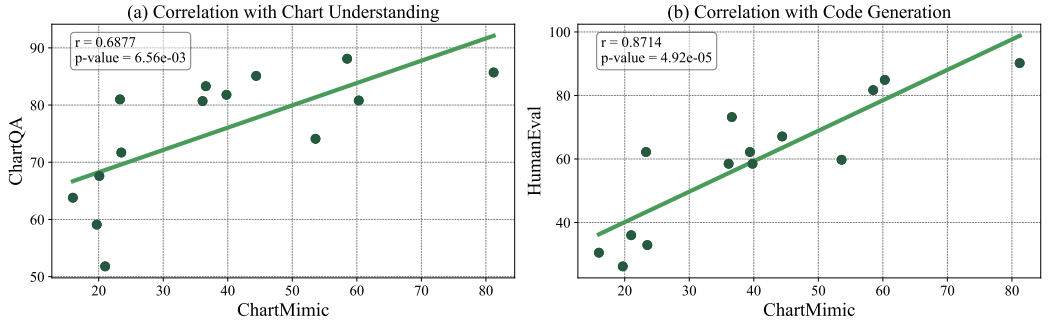


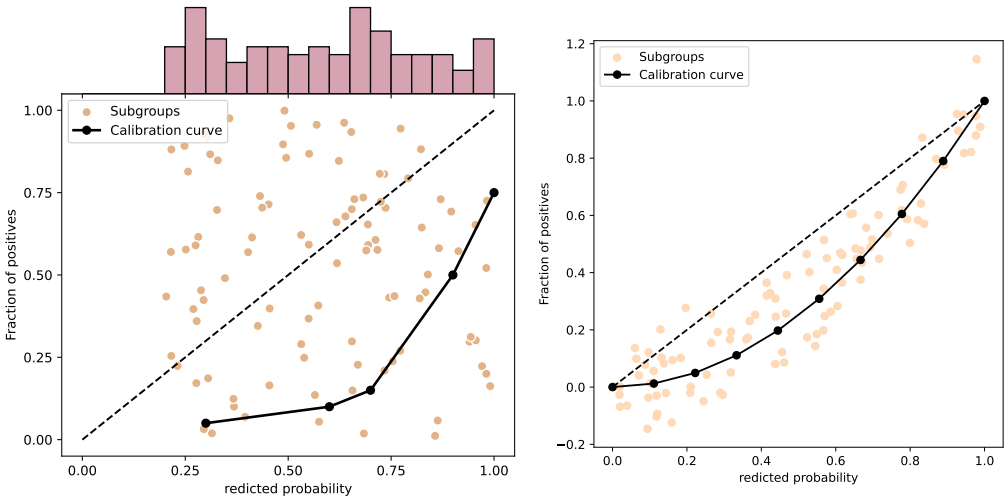
Figure 48: Performance correlation of ChartMimic with benchmarks assessing chart understanding and code generation capabilities.

In an effort to shed light on the factors enhancing performance on ChartMimic, we explore the performance correlation between ChartMimic and existing benchmarks that evaluate chart understanding and code generation capabilities. Specifically, we select ChartQA (Masry et al., 2022) as the benchmark for chart understanding and HumanEval (Chen et al., 2021) as the benchmark for code generation. We then calculate two Pearson correlation coefficients to quantify these relationships: The correlation coefficient between the performance of LMMs in the Direct Mimic task and their performance in the ChartQA task, denoted as r_{chart}, and the correlation coefficient between the performance of LMMs in the Direct Mimic task and the performance of their corresponding LLMs in the HumanEval task, denoted as r_{code} .

As depicted in Fig. 48, the calculated values are r_{chart} = 0.6877 and r_{code} = 0.8714. These results indicate that both chart understanding and code generation abilities influence the performance in the Direct Mimic task, with code generation having a more significant impact. This implies that for future model development aimed at enhancing multimodal code generation capabilities, it is crucial to focus on the foundational code generation abilities of LMMs.

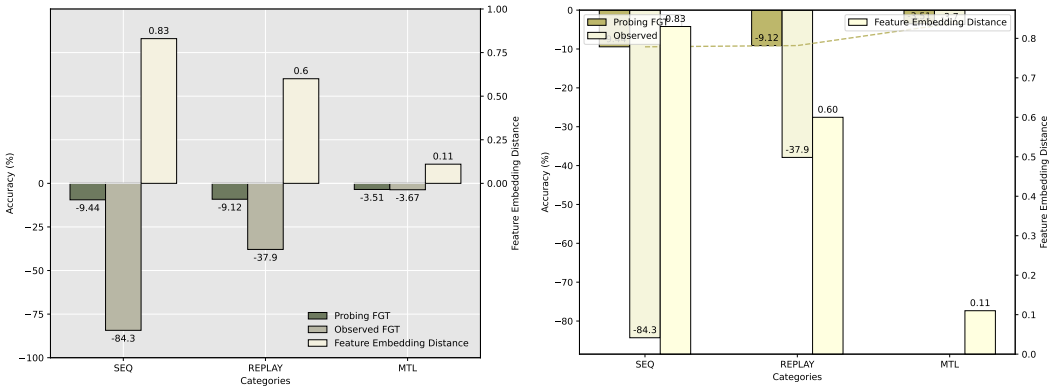
H CASES OF ERROR ANALYSIS

We provide cases of text-related, type-related and color-related errors in Figs. 49 to 54. These cases encompass various error types mentioned in Sec. 4.4.



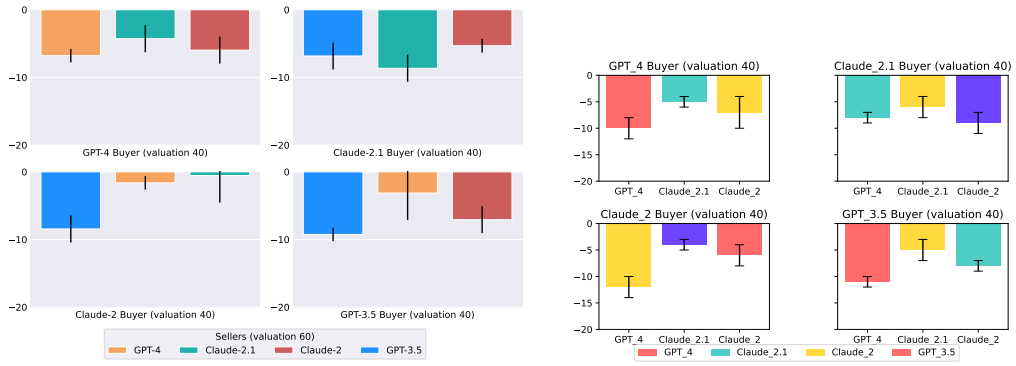
(a) Ground-truth figure (Combination_20) (b) Generated figure

Figure 49: Error Case 1. In this case, the errors include text-related errors of the *Missing* type and type-related errors of the *Missing* type.



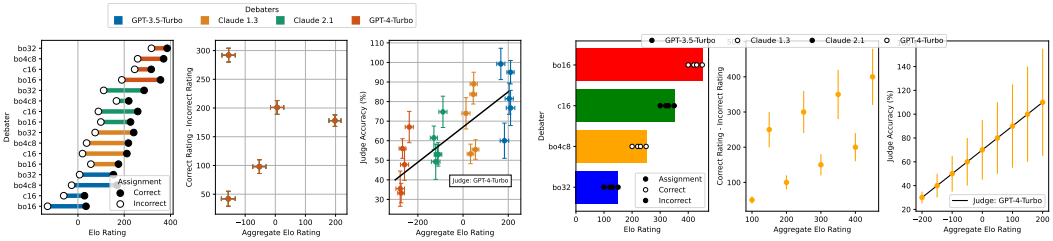
(a) Ground-truth figure (bar_7) (b) Generated figure

Figure 50: Error Case 2. In this case, the errors include type-related errors of the *Extraneous* type, and color-related errors of the *Different* type.



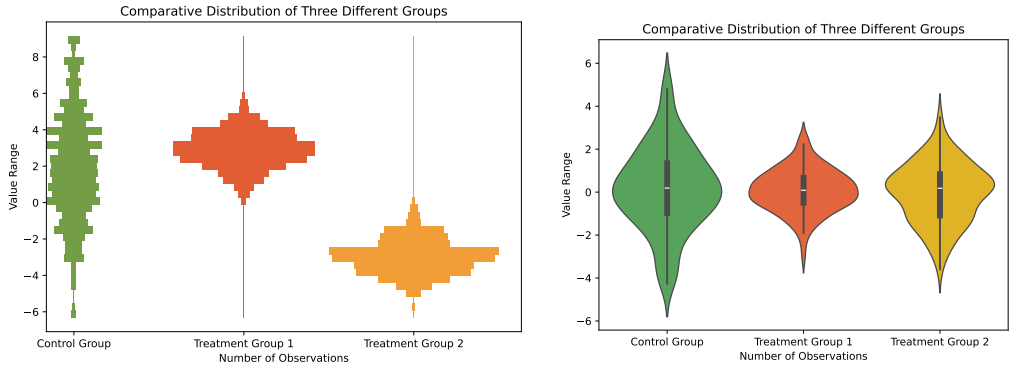
(a) Ground-truth figure (errorbar_12) (b) Generated figure

Figure 51: Error Case 3. In this case, the errors include text-related errors of the *Detail* and *Missing* type, and color-related errors of the *Different* type.



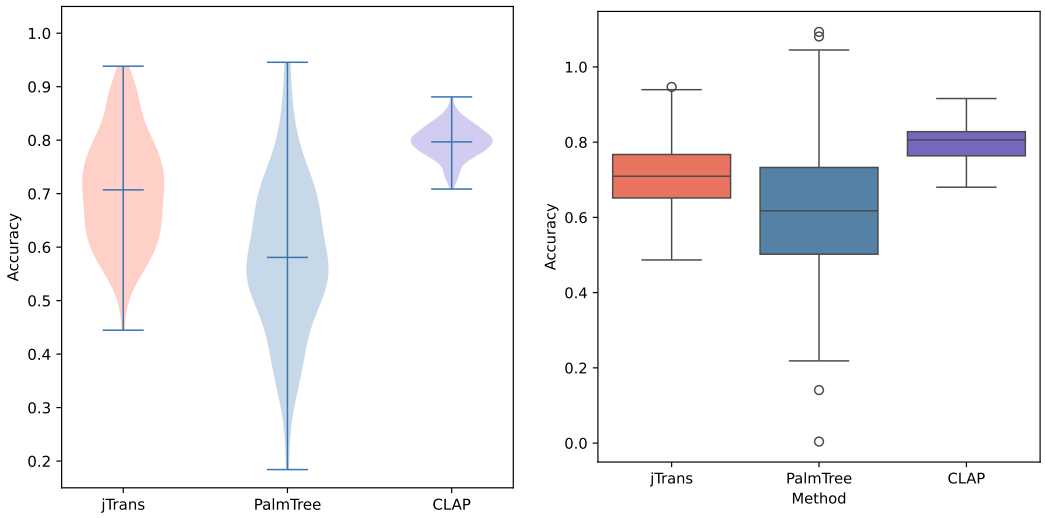
(a) Ground-truth figure (multidiff_9) (b) Generated figure

Figure 52: Error Case 4. In this case, the errors include text-related errors of the *Missing* type, type-related errors of the *Missing* type, and color-related errors of the *Different* type.



(a) Ground-truth figure (HR_22) (b) Generated figure

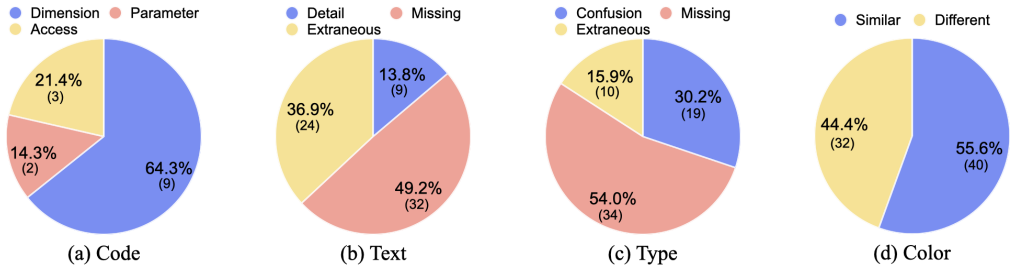
Figure 53: Error Case 5. In this case, the errors include type-related errors of the *Confusion* type, and color-related errors of the *Similar* type.



(a) Ground-truth figure (violin_4)

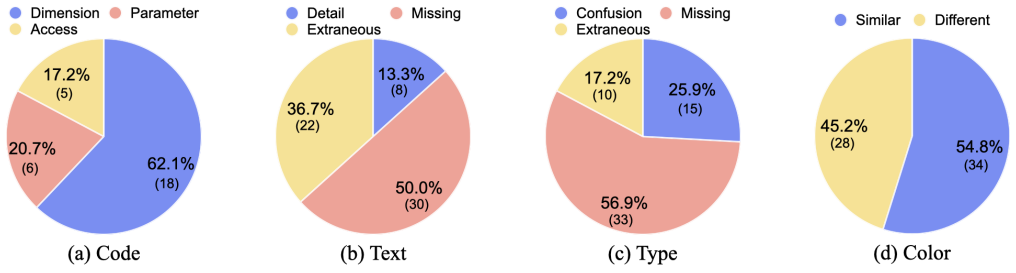
(b) Generated figure

Figure 54: Error Case 6. In this case, the errors include text-related errors of the *Extraneous* type, type-related errors of the *Confusion* type and color-related errors of the *Similar* type.



(a) Code (b) Text (c) Type (d) Color

Figure 55: Error analysis of InternVL2-Llama3-76B across four error types on the Direct Mimic task. The number in brackets indicates the count of error case.

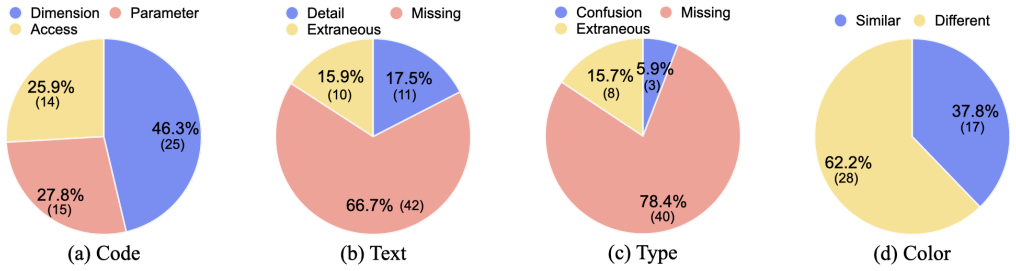


(a) Code (b) Text (c) Type (d) Color

Figure 56: Error analysis of InternVL2-26B across four error types on the Direct Mimic task. The number in brackets indicates the count of error case.

I Error Analysis of Open-Weight Models

We conduct detailed error analysis on representative open-weight models of different scales, sampling 100 cases for each model. For large-scale models such as InternVL2-Llama3-76B and InternVL2-26B (Fig. 55, Fig. 56), the error patterns align closely with those observed in GPT-4o. Specifically, Code-related Errors predominantly arise from dimensional inconsistencies, while Text-related Errors primarily manifest as missing elements. Type-related Errors mainly comprise missing components cases, and Color recognition demonstrates the capability to identify similar, if not identical, chromatic properties. In contrast, smaller-scale models like DeepSeek-VL-7B (Fig. 57) exhibit markedly different error distributions. Code-related issues show a significant increase in Access and Parameter Errors (approximately 53.7%), while Text-related errors demonstrate a higher prevalence of missing elements (approximately 66.7%). Type-related cases reveal a substantial increase in missing errors (approximately 78.4%). Moreover, the degradation in color recognition is more severe, with the model failing to identify even similar colors. These findings indicate that while larger models encounter challenges similar to GPT-4o, smaller models exhibit more fundamental difficulties in both code generation and visual comprehension.



(a) Code (b) Text (c) Type (d) Color

Figure 57: Error analysis of DeepSeek-VL-7B across four error types on the Direct Mimic task. The number in brackets indicates the count of error case.

J Ethics, Societal Impact and Scalability of ChartMimic

ChartMimic is designed as a comprehensive evaluation framework for advancing LMMs, with a strong emphasis on ethical considerations, societal impact and scalability. Its primary objective is to provide researchers with a rigorous, ethically sound tool for assessing LMMs' capabilities across critical domains, including visual understanding, code generation and cross-modal reasoning.

Ethical Considerations. The ethical integrity of ChartMimic is foundational to its design and implementation. Our dataset is derived from scientific domains, specifically using figures from arXiv papers distributed under the CC BY 4.0 license. This focus on scientific content significantly mitigates potential ethical concerns related to bias, representation, or sensitive personal information that often arise in datasets derived from social media or general web content. The scientific nature of our data ensures: Minimal risk of perpetuating societal biases or stereotypes; Absence of personally identifiable information; Content that is generally neutral and objective, focusing on factual scientific representations. Our manual annotation process for code generation incorporates stringent ethical controls. Annotators are trained to ensure that the information conveyed in the code does not contain any biased content and strictly adheres to the factual representation of the scientific charts. This process further reinforces the ethical robustness of our dataset.

Societal Impact. By providing a benchmark rooted in scientific data, ChartMimic contributes to the development of LMMs that can better understand and interact with complex, data-driven visualizations. This capability has far-reaching positive implications for scientific communication, data analysis, and knowledge dissemination across various fields.

Scalability. The architecture of ChartMimic is inherently extensible, featuring a modular codebase that facilitates the seamless integration of additional chart types and evaluation metrics. Our data collection methodology leverages the continuous update cycle of arXiv, enabling sustainable expansion of the dataset while maintaining its focus on ethically sound, scientific content. The evaluation framework also demonstrates notable scalability: The concept of low-level metric can be readily adapted to other programming environments such as JavaScript and R. The high-level metric incorporates a meta-evaluation approach using LMMs, which allows for sustainable alignment with human preferences as stronger models emerge.