Plot2Code: A Comprehensive Benchmark for Evaluating Multi-modal Large Language Models in Code Generation from Scientific Plots

Chengyue Wu
The University of Hong Kong

Yixiao Ge
ARC Lab, Tencent PCG
`yixiaoge@tencent.com`

Qiushan Guo
The University of Hong Kong

Jiahao Wang
The University of Hong Kong

Zhixuan Liang
The University of Hong Kong

Zeyu Lu
Shanghai Jiao Tong University

Ying Shan
ARC Lab, Tencent PCG

Ping Luo
The University of Hong Kong

Abstract

The remarkable progress of Multi-modal Large Language Models (MLLMs) has attracted significant attention due to their superior performance in visual contexts. However, their capabilities in turning visual figure to executable code, have not been evaluated thoroughly. To address this, we introduce **Plot2Code**, a comprehensive visual coding benchmark designed for a fair and in-depth assessment of MLLMs. We carefully collect 132 manually selected high-quality matplotlib plots across six plot types from publicly available matplotlib galleries. For each plot, we carefully offer its source code, and an descriptive instruction summarized by GPT-4. This approach enables Plot2Code to extensively evaluate MLLMs' code capabilities across various input modalities. Furthermore, we propose three automatic evaluation metrics, including code pass rate, text-match ratio, and GPT-4V overall rating, for a fine-grained assessment of the output code and rendered images. Instead of simply judging pass or fail, we employ GPT-4V to make an overall judgement between the generated and reference images, which has been shown to be consistent with human evaluation. The evaluation results, which include analyses of 14 MLLMs such as the proprietary GPT-4V, Gemini-Pro, and the open-sourced Mini-Gemini, highlight the substantial challenges presented by Plot2Code. With Plot2Code, we reveal that most existing MLLMs struggle with visual coding for text-dense plots, heavily relying on textual instruction. We hope that the evaluation results from Plot2Code on visual coding will guide the future development of MLLMs. All data involved with Plot2Code are available at `https://huggingface.co/datasets/TencentARC/Plot2Code`.

1 Introduction

In the wake of significant advancements in big data and computational power, Large Language Models (LLMs) [33, 4, 12, 15], such as ChatGPT [27] and GPT-4 [28], have become focal points of interest in both commercial and academic spheres. To extend their versatility across various contexts, Multi-modal Large Language Models (MLLMs) [8, 23, 29] have rapidly evolved, as exemplified by the latest models such as GPT-4V [29], Gemini [9], Claude-3 [1], and the open-source models LLaVA [21, 22], Mini-GPT [44, 5] and so on [8, 7]. Concurrently, a diverse array of
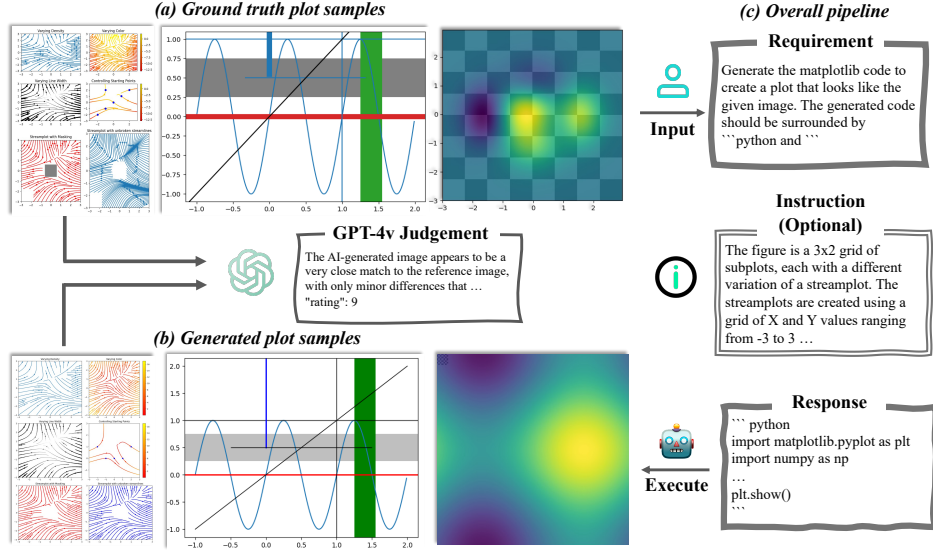
Figure 1: **Overview of Plot2Code.** *Left:* (a) Representative samples from the ground truth plots in our Plot2Code dataset. (b) Plot samples generated by multi-modal LLMs using the reference image. *Right:* (c) The comprehensive pipeline employed to assess the code generation ability of multi-modal LLMs. We consider two distinct settings: **Direct Asking** and **Conditional Asking**.

evaluation benchmarks [17, 16, 41, 39] are curated to assess their visual comprehension performance across different domains. However, there remains a notable gap in attention towards diagrams within text-dense images, which are crucial for assessing the multi-modal reasoning proficiency of MLLMs [24, 25].

In line with Richard Feynman's philosophy, "What I cannot create, I do not understand," evaluating the capability of MLLMs to generate code that renders a provided image effectively further showcases their multi-modal understanding and reasoning prowess. This particular challenge demands MLLMs to accurately interpret the visual elements present in input diagrams, correlate them with textual context provided, and finally derive executable code to generate the plots. Although the development of code generation from uni-modal natural language has experienced rapid progress in recent years [30, 10, 37], the exploration of code generation using multi-modal inputs remains an active area of research. Previous efforts, *e.g.* HumanEval [6] and MBPP [2], have concentrated on uni-modal code problems, while the more recent Design2Code [31] has expanded the scope to include user-interface (UI) design, particularly HTML files, for evaluating MLLMs. However, these studies focus on unimodal scenarios (*e.g.*, text-only [6, 2] or image-only [31] inputs) and have limited capabilities when evaluating models with multimodal inputs.

To this end, our work underscores the critical importance and motivation by addressing three key challenges in evaluating MLLMs' coding capabilities:

1. **Do the evaluation settings accommodate all modalities, including text and images, for both input and output?** This fundamental question pertains to the scope of visual coding. By employing extensive evaluation settings, we can conduct thorough ablation analyses of MLLMs' performance across various input modalities or their combinations, while also assessing the output across different modalities.

2. **Are the evaluation metrics accurate, straightforward, and comprehensive?** Most existing code benchmarks rely on unit tests to obtain binary evaluation results. While this approach may suffice for uni-modal code tasks, it falls short for visual coding tasks that require not only the code pass rates but also assessments of image fidelity.

3. **Are the evaluations for visual coding tasks relevant to real-world applications?** It is imperative that benchmarks align with real-world uses and applications, particularly in coding tasks. Employing the commonly used multiple-choice format for evaluating code tasks would be inadequate and incongruous.

Hence, in response to the aforementioned challenges, we present Plot2Code, a comprehensive and specialized multi-modal code benchmark crafted to evaluate the multi-modal understanding, reasoning, and coding capabilities of MLLMs. This benchmark comprises a carefully curated dataset comprising 132 matplotlib plots across 6 plot types, incorporating a total of 293 subplots sourced from matplotlib galleries. Each plot is paired with its corresponding code and a detailed description generated by GPT-4. To cater to diverse input and output formats, Plot2Code includes two evaluation settings, **Direct Asking** and **Conditional Asking**, supporting automatic metric-based evaluations for both text and image outputs. MLLMs can be evaluated using text, images, or a blend of both as inputs, while the text and image outputs can be assessed based on the code pass rate and GPT-4V overall rating, which consistently aligns with human evaluations.

We evaluate 14 publicly accessible MLLMs across various evaluation settings to determine optimal performance. Our findings underscore the significant challenges posed by Plot2Code, with GPT-4V only achieving an overall score of 7.68/10, indicating considerable room for enhancement in visual coding tasks.

The contributions of this study can be summarized as follows:

- We construct a novel evaluation benchmark, Plot2Code, tailored for multi-modal code tasks, enabling the assessment of advancements in multi-modal understanding and reasoning.

- Development of a diverse array of evaluation settings for Plot2Code, accommodating varied modalities for input and output through image-code pairs and automatic evaluation metrics.

- Evaluations of various publicly available MLLMs on Plot2Code, revealing that current MLLMs like GPT-4V, Gemini-Pro, and Claude-3, demonstrate modest performance in visual coding tasks.

We anticipate that Plot2Code will stimulate the research community to further explore and advance the realm of MLLMs, propelling us towards the realization of truly intelligent multi-modal systems.

2 Related Work

2.1 Multi-modal Large Language Models

With the remarkable progress achieved by Large Language Models (LLMs) [27, 33, 28], incorporating multi-modal input signals into the backbone LLMs has garnered significant interest from both academia and industry [21, 9, 23, 29, 9]. The primary focus of multi-modal LLM research involves developing additional encoders that enable compatibility and processing of multi-modal inputs by the backbone LLMs. To enhance practical applications, several studies [11, 23, 9] concentrate on processing text-dense images, such as documents and charts, using high-resolution vision encoders. Our goal is to conduct a thorough and comprehensive investigation of MLLMs and their potential by assessing their ability to generate code from reference plots, showcasing their visual coding proficiency.

2.2 Multi-modal Code Benchmark

Building upon the growing proficiency of LLMs, a subset of specialized models, known as Code LLMs [30, 19, 10], has emerged. These models focus specifically on programming code, offering numerous appealing applications such as code completion and infilling. Code tasks effectively reflect the in-depth reasoning abilities of (M)LLMs. Uni-modal code benchmarks, like HumanEval and MBPP [6, 2], test the generated code using single-round unit tests with the Pass@k metric. More recently, LLM agents have been evaluated in more complex multi-turn interactive code settings [35, 38]. Extending beyond the uni-modal context, MMCode [18] incorporates image input into code tasks, while Design2Code [31] evaluates MLLMs' generated HTML files through CLIP scores and HTML blocks. Our work proposes a comprehensive benchmark, Plot2Code, which supports a wide range of evaluation scenarios and accommodates both uni-modal and multi-modal inputs. The metrics encompass text-based measures such as code pass rate and generated plot similarity, serving as an all-encompassing evaluation suite for assessing MLLMs' in-depth understanding and reasoning capabilities. See Table 2 for detailed comparisons with related benchmarks.

3 Dataset Collection

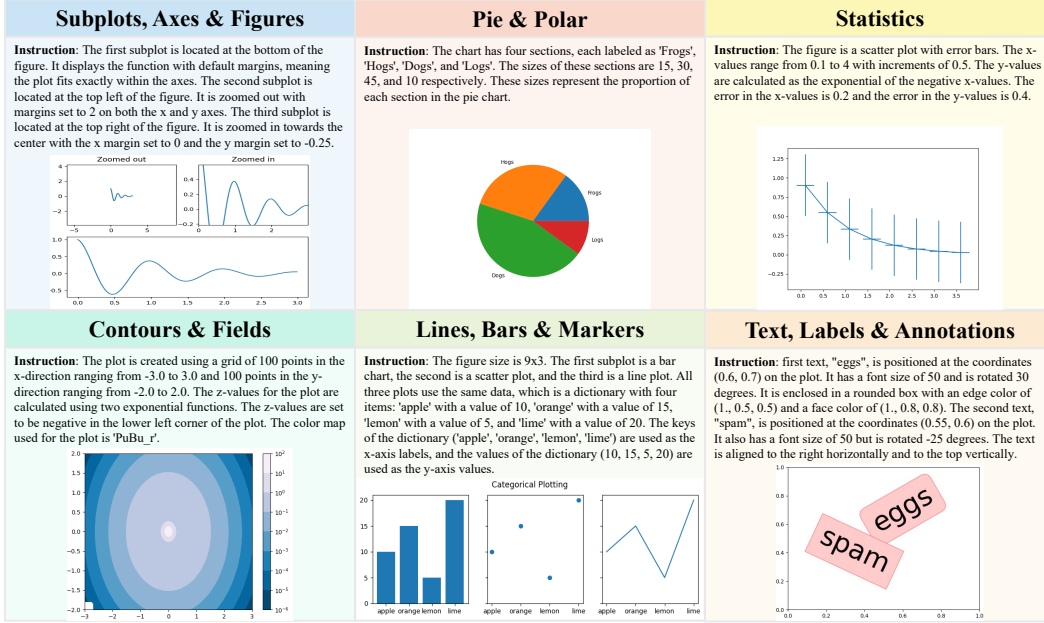| Subplots, Axes & Figures	Pie & Polar	Statistics
Instruction: The first subplot is located at the bottom of the figure. It displays the function with default margins, meaning the plot fits exactly within the axes. The second subplot is located at the top left of the figure. It is zoomed out with margins set to 2 on both the x and y axes. The third subplot is located at the top right of the figure. It is zoomed in towards the center with the x margin set to 0 and the y margin set to -0.25.	**Instruction**: The chart has four sections, each labeled as 'Frogs', 'Hogs', 'Dogs', and 'Logs'. The sizes of these sections are 15, 30, 45, and 10 respectively. These sizes represent the proportion of each section in the pie chart.	**Instruction**: The figure is a scatter plot with error bars. The x-values range from 0.1 to 4 with increments of 0.5. The y-values are calculated as the exponential of the negative x-values. The error in the x-values is 0.2 and the error in the y-values is 0.4.
Contours & Fields	**Lines, Bars & Markers**	**Text, Labels & Annotations**
Instruction: The plot is created using a grid of 100 points in the x-direction ranging from -3.0 to 3.0 and 100 points in the y-direction ranging from -2.0 to 2.0. The z-values for the plot are calculated using two exponential functions. The z-values are set to be negative in the lower left corner of the plot. The color map used for the plot is 'PuBu_r'.	**Instruction**: The figure size is 9x3. The first subplot is a bar chart, the second is a scatter plot, and the third is a line plot. All three plots use the same data, which is a dictionary with four items: 'apple' with a value of 10, 'orange' with a value of 15, 'lemon' with a value of 5, and 'lime' with a value of 20. The keys of the dictionary ('apple', 'orange', 'lemon', 'lime') are used as the x-axis labels, and the values of the dictionary (10, 15, 5, 20) are used as the y-axis values.	**Instruction**: first text, "eggs", is positioned at the coordinates (0.6, 0.7) on the plot. It has a font size of 50 and is rotated 30 degrees. It is enclosed in a rounded box with an edge color of (1., 0.5, 0.5) and a face color of (1., 0.8, 0.8). The second text, "spam", is positioned at the coordinates (0.55, 0.6) on the plot. It also has a font size of 50 but is rotated -25 degrees. The text is aligned to the right horizontally and to the top vertically.

Figure 2: **Examples of Plot2Code benchmark.** We show different-type plots with their corresponding instructions.

In this section, we delineate the process of curating and processing our benchmark data. Initially, we crawl every website link enumerated in the matplotlib gallery[1]. Subsequently, we extract the code block from each corresponding HTML file. This procedure yields a total of 841 distinct code blocks, which are subjected to further filtering and processing as explicated in the subsequent sections.

3.1 Test Set Curation

The primary objective is to acquire well-structured plot-code pairs that effectively evaluate the code generation capabilities of MLLM. It is important to note that the initially crawled Python code may not always be suitable for generating high-quality plots for evaluation purposes. To address this, we employ a combination of automatic processing and manual filtering, as detailed below.

Generation Filtering During our analysis, we observe that a single HTML file may encompass multiple code segments, with some segments being unable to produce plots due to their focus on import lines and initialization functions. To overcome this limitation, we exclusively extract code from HTML files containing a single code block. This ensures that the extracted code encompasses all essential components and can generate a plot without necessitating additional dependencies. We then filter out all the code that can not generate the plots and obtain 529 plot-code pairs.

Type Filtering Our analysis operates under the assumption that the plots are simple, static figures devoid of animation and interaction. Consequently, each plot can be regarded as an image file rendered by the matplotlib engine. To maintain this simplicity, we filter out any plots associated with specific tags, such as animation, widget, and event handling, found in their corresponding URLs. A detailed breakdown of the plot-code pair types in our dataset is provided in Figure 3.

Manual Curation Subsequent to the aforementioned processing, we conduct a final round of manual curation to filter examples based on the criteria outlined below:

- The plot is devoid of any external file dependencies and can be directly rendered using the corresponding code.

---

[1] https://matplotlib.org/stable/gallery/index.html

Table 1: **Key Statistics of Plot2Code.** Tokens are counted by LLaMA-2 tokenizer.

Statistic	Number
Total Samples	132
- Contours & Fields	30 (22.7%)
- Lines, Bars & Markers	37 (28.0%)
- Texts, Labels & Annotations	14 (10.6%)
- Statistics	17 (12.9%)
- Subplots, Axes & Figures	25 (18.9%)
- Pie & Polar	9 (6.8%)
Total Subplot Count	293
Code Length (tokens)	401 ± 281
- Minimum Length	60
- Maximum Length	1823
Instruction Length (tokens)	279 ± 115
- Minimum Length	72
- Maximum Length	628
Text Count	23 ± 13

Figure 3: **Type Distribution of Plot2Code.**
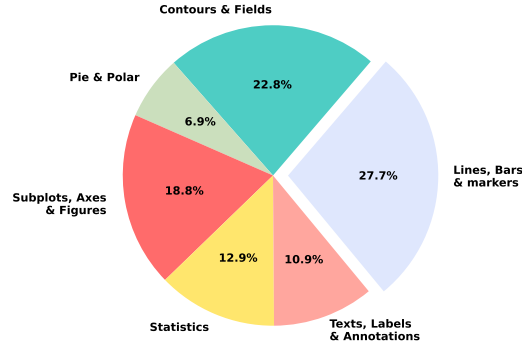
- The plots exhibit a wide array of diversity in terms of size, text, colors, and types, thereby serving as a comprehensive benchmark for evaluation that encompasses a wide array of commonly used charts and plots.

- The plots are uniformly distributed across various difficulty levels, ranging from beginner to specialized levels.

During the manual filtering process, we adopt a more stringent approach to retain only high-quality plots. Ultimately, we procure 132 test examples that serve as our benchmark.

3.2 Evaluation Setting

We assess the test set, curated in the previous step, under two distinct evaluation scenarios: direct asking and conditional asking. To facilitate convenient extraction of code from the MLLM-generated responses, we request the code to be enclosed between specific markers, enabling the use of regular expressions for extraction.

Direct Asking This setting means giving the MLLM an image as input and requiring it to generate executable code that produces a graph closely resembling the input image. The specific prompt can be found in Appendix A.1. Figure 7 illustrates an example in this case.

Conditional Asking For MLLMs, this setup means receiving an image and conditions (text instructions) as input and generating executable code that produces results in line with the specified conditions. For LLMs, the input includes conditions only, with other requirements being consistent with those for MLLMs. We employ GPT-4 to extract these instructions from the ground truth code, instructing it to retain all essential information for reproduction while avoiding exposure of code implementation details. The prompt used to construct these instructions can be found in Appendix A.2. Figure 8 illustrates an example in this case.

3.3 Data Statistics

Key Statistics To gauge the difficulty levels of the test examples, we present several statistics in Table 1. We enumerate the total number of subplots present in our test samples, as a single plot may comprise multiple subplots. In total, there are 293 subplots (min=1, max=18). We tokenize the scraped code files using the LLaMA-2 tokenizer [33]. The ground truth code exhibits an average token count of 409, with a standard deviation of 291. Additionally, we tokenize the instructions accompanying our test samples, yielding an average of 242 tokens and a standard deviation of

58. Utilizing PaddleOCR[2], we calculate the text count for each plot, with an average of 23 and a standard deviation of 13. Collectively, these metrics indicate that our benchmark examples are indeed challenging and encompass a broad spectrum of complexities found in scientific plots.

Type Distribution   To gain insight into the variety of plot types encompassed by our test set, we depict the type distribution via a pie chart in Figure 3. For each sample, we determine its type based on the tag present in its URL, sourced from the matplotlib gallery. These categories are defined by the matplotlib gallery itself. The most prevalent types include lines, bars, and markers, while other categories comprise contours, fields, pie charts, polar plots, subplots axes, statistical representations, and text labels and annotations.

3.4   Evaluation Metrics

Unlike unimodal code generation tasks, which can be assessed simply by conducting unit tests and recording the code pass rate, a piece of code might execute flawlessly without any errors but fail to render an image similar to the provided reference image. This discrepancy necessitates the development of precise and automatic evaluation metrics for multimodal code generation tasks. Consequently, we propose a suite of evaluation metrics encompassing various aspects to conduct this evaluation. Our evaluation metrics include the code pass rate, low-level metrics, text match rate, and GPT-4v judgement score. We will elaborate on these metrics in detail in the subsequent paragraphs.

Code Pass Rate   The MLLM is expected to generate code that can be rendered into an image using matplotlib. Consequently, we compute the code pass rate to determine whether the MLLM can generate executable code given the input reference image and instruction. The subsequent metrics will only be applied to images that have been correctly generated.

GPT-4V Judgement   We devise an evaluation pipeline leveraging the GPT-4v model to assess the high-level similarity between generated plots and ground truth plots. This pipeline assigns a rating to test samples on a scale of 1-10, taking into account various aspects such as overall appearance, colors, shapes, positions, and other visual elements of the images. The comprehensive prompt utilized for evaluation can be found in the Appendix A.3.

Text-Match Ratio   While the high-level similarity evaluation provided by the GPT-4v judgement is valuable, it does not account for detailed plot components such as text, which is crucial for plot interpretation. To compensate for this, we introduce the text-match ratio, aiming to assess the fine-grained similarity between the generated and ground truth plots. This metric evaluates the precision of the text present in the ground truth sample, ensuring that all text elements are accurately reproduced in the plot under assessment and there are no extra texts in the generated image.

3.5   Comparison with Other Datasets

As depicted in Table 2, our dataset encompasses the most extensive range of evaluation settings and metrics compared to all other uni-modal and multi-modal code benchmarks.

Table 2: **Comparison with other uni-modal and multi-modal code benchmarks.**"I" represents images,"T" represents text, and "I+T" stands for the multi-modal information with images and text.

Dataset	Task Type	Input Format		Output Eval Format				Evaluation Metric	
		T	I+T	I	T	I+T	Pass Rate	Component Match	Rating
HumanEval [6]	Programming	✔	✗	✗	✔	✗	✔	✗	✗
SVGEditBench [26]	SVG	✔	✗	✔	✗	✗	✗	✗	✗
MMcode [18]	Algorithm	✔	✔	✗	✔	✗	✔	✗	✗
Design2Code [31]	Websites	✗	✔	✔	✗	✗	✗	✔	✔
Plot2Code	Plots	✔	✔	✔	✔	✔	✔	✔	✔

---

[2]https://github.com/PaddlePaddle/PaddleOCR

Table 3: **Quantitative results for 14 MLLMs across two settings, Direct Asking and Conditional Asking.** The maximum value of GPT-4V overall rating is **bolded**.

Model	Backbone LLM	Direct Asking			Conditional Asking		
		Pass Rate	Text-Match	Rating	Pass Rate	Text-Match	Rating
LLMs							
ChatGPT [27]	ChatGPT [27]	-	-	-	80.3	56.7	6.59
GPT-4 [28]	GPT-4 [28]	-	-	-	80.3	68.0	7.36
GPT-4 (CoT) [28]	GPT-4 [28]	-	-	-	78.8	66.0	7.09
GPT-4 (PS+) [28]	GPT-4 [28]	-	-	-	77.3	66.8	7.26
Closed-source MLLMs							
Claude-3-Opus [1]	Claude-3 [1]	84.1	57.5	4.37	78.0	69.7	7.68
Claude-3-Sonnet [1]	Claude-3 [1]	75.8	46.7	5.38	65.9	57.0	7.20
Gemini-Pro [9]	Gemini [9]	68.2	53.6	5.06	55.3	66.9	7.10
GPT-4V [29]	GPT-4 [28]	84.1	57.7	**6.48**	81.8	70.7	7.68
GPT-4V (CoT) [29]	GPT-4 [28]	89.4	56.3	6.30	81.8	69.7	7.75
GPT-4V (PS+) [29]	GPT-4 [28]	86.4	55.3	6.25	85.6	71.4	**7.83**
Open-source MLLMs (Low resolution setting)							
Mini-Gemini-2B [20]	Gemma-2B [32]	39.4	21.4	1.96	22.7	31.8	2.80
Mini-Gemini-8x7B [20]	Mixtral-8x7B [14]	75.8	33.9	3.76	62.1	52.3	5.74
Mini-Gemini-34B [20]	Yi-34B [40]	67.4	30.5	2.78	50.0	51.2	4.79
Open-source MLLMs (High resolution setting)							
DeepSeek-VL-7B [23]	DeepSeek-7B [3]	72.0	38.7	3.69	56.8	50.1	5.19
LLaVA-1.6-Mistral-7B [21]	Mistral-7B [13]	64.4	32.6	3.06	42.4	45.1	4.48
LLaVA-1.6-34B [21]	Yi-34B [40]	72.0	34.6	3.18	53.0	50.7	5.60
Mini-Gemini-8x7B-HD [20]	Mixtral-8x7B [14]	73.5	40.7	3.87	58.4	53.7	6.08
Mini-Gemini-34B-HD [20]	Yi-34B [40]	55.8	34.0	3.06	43.4	46.1	5.35

4 Experiments

In this section, we evaluate a variety of multi-modal large language models and methods on our Plot2Code benchmark to compare their performance. This includes both closed-source commercial API models and state-of-the-art open-source models.

4.1 Evaluation Details

Evaluated (M)LLMs To ensure a comprehensive evaluation, we assess 14 representative closed-source and open-source (M)LLMs that vary in parameters, resolution settings, and backbone LLMs, such as GPT [27], DeepSeek [3], Mistral [13], Mixtral [14], and Yi [40]. The quantitative evaluation is provided in Sec. We also explore different prompt strategies, including Chain-of-Thought [36] and Plan-and-Solve [34]. 4.2. We investigate the influence of different designs of MLLMs on the performance of our benchmark in Sec. 4.3.

Evaluation Methods As mentioned in Sec. 3.2, we employ two distinct evaluation settings: Direct Asking and Conditional Asking. For LLMs lacking vision capabilities, we evaluate them solely in the Conditional Asking setting with instruction input. Furthermore, we extend the GPT-4V judgement setting to conduct pairwise evaluations between two (M)LLMs and perform a correlative analysis between GPT-4V judgement and human evaluation. More details are provided in Sec. 4.4.

4.2 Overall Evaluation

We showcase the quantitative results of (M)LLMs on our Plot2Code benchmark here. The code pass rate, text-match ratio, and GPT-4V overall rating for both direct asking and conditional asking scenarios are reported in Table 3.

The comprehensive challenge of Plot2Code. The benchmark poses considerable challenges, as even advanced models like Claude-3-Opus, Gemini-Pro, and GPT-4V achieve only 7.68, 7.10, and 7.68, respectively, in the overall assessment for the conditional asking scenario, indicating substantial room for improvement. In addition to the overall rating, the pass rate also presents challenges for MLLMs, particularly when instructions are added. For example, Gemini-Pro's pass rate decreases

Figure 4: **Pair evaluation results in the conditional asking setting.** We use GPT-4V without prompt strategies as the baseline (this method is not shown in the table as it serves as the basis for pairwise comparison).
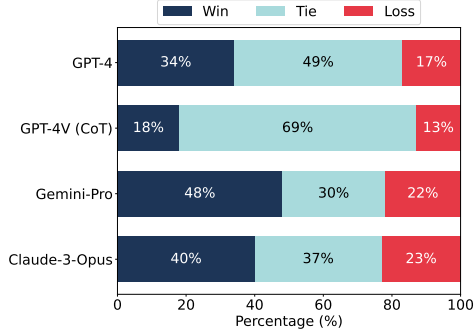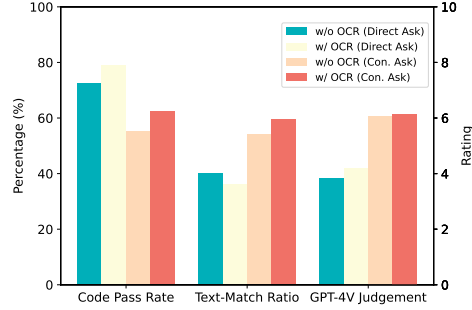

Figure 5: **Ablation experiments involving the addition of OCR tokens or not.** The base model is Mini-Gemini-8x7B-HD. OCR tokens are extracted using PaddleOCR, which is supported by the official implementation of Mini-Gemini.

from 68.2% to 55.3% after incorporating the instruction, as the added requirements will make it harder to generate the corresponding code. In contrast to widely used benchmarks like MT-bench and HumanEval, where recent advanced models attain ratings above 9.00 and code pass rates exceeding 80%, Plot2Code necessitates both visual understanding and reasoning abilities to analyze the plot, generate executable code, and create a plot resembling the reference plot. This heightened challenge for (M)LLMs serves as a more rigorous examination of their visual reasoning comprehension and reasoning capabilities.

The gap between closed-source and open-source models. The performance of open-source models lags considerably behind that of closed-source models. We evaluate recently advanced open-source MLLMs, including DeepSeek-VL [23], Mini-Gemini [9], and LLaVA-Next [21]. The best performance among the open-source MLLMs is achieved by Mini-Gemini-8x7B-HD, which scores 6.08 in the GPT-4V judgement with a 58.4% code pass rate. However, this performance is still not on par with that of commercial closed-source MLLMs. There is a need for the open-source community to develop more powerful models that can compete with, or even surpass, the capabilities of advanced proprietary models.

4.3 Influence of difference settings.

We analyze the results from various perspectives, encompassing prompt strategies, backbone LLMs, and the resolution settings. The key findings are summarized as follows.

The influence of LLMs. As depicted in Table 3, there is a strong correlation between model performance and the backbone LLM used, evident in both Mini-Gemini and LLava. This suggests that the Plot2Code task may require powerful backbone LLMs to facilitate the reasoning process and generate executable code.

Different evaluation settings. As mentioned in Sec. 3.2, there are two distinct evaluation settings. In Table 3, it can be observed that in the conditional asking setting, MLLMs generally achieve a lower pass rate and higher similarity compared to the direct asking setting. We attribute this to the fact that the added instruction imposes stricter restrictions or requirements on the generated code, making it more challenging for models to generate executable code. However, the additional instruction can enhance the similarity of the generated image to the reference image. Beyond these two evaluation settings, we also investigate the influence of different prompt strategies, including Chain-of-Thought and Plan-and-Solve. We find that prompts encouraging MLLMs to engage in deeper thinking do not show a clear advantage over our default prompt, indicating that the exploration of multi-modal reasoning prompts is still in progress.

Different image resolution settings. In addition to the backbone LLMs, we examine the vision encoder settings, specifically focusing on image resolution settings. Many MLLMs utilize vision

Table 4: **Comparison of different indicators in distinguishing image groups with and without significant differences.**

Indicators	t-statistic	p-value	Reject H_0
MSE	-1.24	0.22	✗
SSIM	1.28	0.21	✗
CLIP-Score	4.23	1.24×10^{-4}	✔
Text-Match Ratio	5.69	9.62×10^{-7}	✔
GPT-4V Judgement	**9.07**	1.22×10^{-11}	✔

Figure 6: **Illustration of inaccuracy of traditional low-level metrics.**

🤔 *Traditional low-level metrics can not precisely reflect the quality of the test sample.*
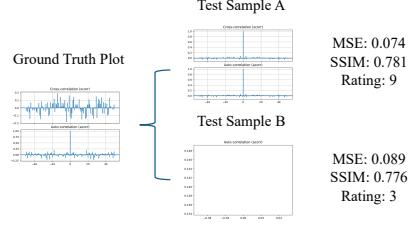
encoders with higher resolution capabilities to provide more detailed information from the input images. We observe that MLLMs with higher resolution consistently improve performance, a trend similar to that seen in ChartQA [24] and DocQA [25]. We also conduct an experiment wherein we add OCR tokens extracted by the official PaddleOCR implementation used in Mini-Gemini, as shown in Figure 5. This approach yields performance improvements akin to those achieved with high-resolution settings, suggesting current MLLMs may still require more powerful vision encoders to capture detailed information from images.

4.4 Pairwise Model Comparison

In accordance with the conventional practice of pairwise model evaluation [42, 43], we expand the GPT-4V judgement setting to perform pairwise evaluations between two (M)LLMs. The detailed prompt employed for pairwise model comparison can be found in the Appendix A.3. For each reference sample, we request GPT-4V to determine which generated image is more similar when comparing a pair of MLLMs. To mitigate the influence of differing positions, we swap the two generated images for an additional evaluation. A model is considered victorious only if it wins both rounds; otherwise, the result is deemed a tie. We utilize GPT-4V as the baseline for comparison. The results are illustrated in Figure 4, from which we can infer that: (i) Compared to GPT-4, the inclusion of image input for GPT-4V is beneficial in generating higher quality plots. (ii) The commonly used prompt strategy, Chain-of-Thought, does not yield additional advantages in our benchmark.

5 Statistical Analysis

In this section, we perform several statistical analyses to justify the design of our benchmark. Since our benchmark involves comparing the generated image with the reference image, we first investigate whether the proposed metrics can effectively indicate image similarity. Additionally, we conduct a correlative analysis to determine the relationship between GPT-4v judgement and human evaluation, further substantiating the basic intuition behind our proposed metric and benchmark.

Hypothesis tests for image similarity metrics. Since plot images typically exhibit similarity due to their common white background and overall structure, hypothesis tests are conducted to evaluate the effectiveness of different indicators in distinguishing between image groups with and without visual differences. We first select two image groups: one group consists of images generated by GPT-4V with high visual similarity, while the other group contains images from Mini-Gemini-2B with considerably lower similarity. We then perform a hypothesis test using several metrics, including traditionally used Mean Squared Loss (MSE), Structural Similarity (SSIM), as well as the metrics we proposed, namely text-match ratio and GPT-4V rating, to determine whether the metric can effectively differentiate between the two image groups with varying levels of visual similarity.

Specifically, we conduct two-sample t-test (independent t-test).

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{1}$$

9

Table 5: **Correlation coefficient comparison between GPT evaluations and human evaluations.**

Kendall's Tau		Pearson		Spearman	
Coefficient	p-value	Coefficient	p-value	Coefficient	p-value
0.437	8.68×10^{-49}	0.479	6.89×10^{-54}	0.469	1.57×10^{-51}

where $\bar{X}_1$ and $\bar{X}_2$ are the sample means of the two groups, s_1^2 and s_2^2 are the sample variances, and n_1 and n_2 are the sample sizes. The null hypothesis (H_0) assumed that there was no significant difference in the scores of the indicators between the two image groups. The alternative hypothesis (H_1) assumed that there was a significant difference. The p-value, a measure of the strength of evidence in support of a null hypothesis, was calculated for each indicator. A smaller p-value indicates stronger evidence against the null hypothesis. In this case, a significance level of 0.05 was chosen, meaning that if the p-value was less than 0.05, the null hypothesis would be rejected in favor of the alternative hypothesis, with a 95% probability of correctly rejecting the null hypothesis. The results of our hypothesis test can be found in Table 6.

For the MSE and SSIM indicators, the p-values were 0.22 and 0.21, respectively, both of which were greater than the significance level. Therefore, we could not reject the null hypothesis for these two indicators, suggesting that they might not effectively distinguish between the two image groups.

On the other hand, for the GPT-4V-Judgement and Text-Match Ratio indicators, the p-values were 1.22×10^{-11} and 9.62×10^{-7}, respectively, both of which were less than the significance level and CLIP-score's p-value (1.24×10^{-4}). Therefore, we rejected the null hypothesis for these two indicators, suggesting that they could effectively distinguish between the two image groups.

It should be noted that hypothesis testing only provides statistical evidence based on the collected data, and does not prove whether the null hypothesis or the alternative hypothesis is absolutely true.

Correlative analysis between GPT-4V judgement and human evaluation. To investigate the similarity between the GPT-4V judgement and human evaluation, a correlative analysis was performed using different correlation coefficients, including Kendall's Tau, Pearson correlation coefficient, and Spearman's rank correlation coefficient. The sample size for this analysis was 920. Details can be found in the Appendix C.

As shown in Table 5, All three correlation coefficients indicated a moderate positive relationship between the GPT-4V judgement and human evaluation. Moreover, the p-values were all smaller than the significance level of 0.05, suggesting that the correlations were statistically significant.

These findings imply that the GPT-4V judgement is in general agreement with human evaluation, demonstrating its effectiveness in assessing the similarity between generated images and real images.

6 Conclusion

In this study, we have presented a comprehensive benchmark, Plot2Code, for the evaluation of multi-modal language model's code generation ability. This benchmark encompasses a wide range of complexities and types of scientific plots, making it a robust tool for assessing the performance of different models. We have proposed a suite of evaluation metrics, including code pass rate, text-match ratio, and GPT-4v judgement score, which together provide a holistic evaluation of a model's performance.

Our evaluation of various models on the Plot2Code benchmark has revealed significant differences in performance, highlighting the challenges posed by this task and the room for improvement in current models. We have found that while some models can generate executable code and produce plots similar to the reference image, accurately reproducing all text elements and fine-grained details remains a challenge.

In future work, we believe our Plot2Code benchmark can stimulate further exploration of multi-modal reasoning, text-dense image understanding, and complex code generation capabilities of MLLMs. There are numerous aspects that are worth exploring, including multi-modal reasoning prompt strategies, the design of vision encoders that are compatible with text-dense images. In future work, we believe our Plot2Code benchmark can stimulate further exploration of multi-modal reasoning,

text-dense image understanding, and complex code generation capabilities of MLLMs. There are numerous aspects that are worth exploring, including multi-modal reasoning prompt strategies, the design of vision encoders that are compatible with text-dense images. We hope more research related to this aspects can further reduce the gap between the open-source community MLLMs and closed-source commercial APIs.

References

[1] Anthropic. Claude 3 haiku: our fastest model yet. 2024. Available at: `https://www.anthropic.com/news/claude-3-haiku`.

[2] 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.

[3] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[5] Jun Chen, Deyao Zhu, Xiaoqian Shen, Xiang Li, Zechun Liu, Pengchuan Zhang, Raghuraman Krishnamoorthi, Vikas Chandra, Yunyang Xiong, and Mohamed Elhoseiny. Minigpt-v2: large language model as a unified interface for vision-language multi-task learning. *arXiv preprint arXiv:2310.09478*, 2023.

[6] 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.

[7] Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Zhong Muyan, Qinglong Zhang, Xizhou Zhu, Lewei Lu, et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. *arXiv preprint arXiv:2312.14238*, 2023.

[8] Yuying Ge, Sijie Zhao, Jinguo Zhu, Yixiao Ge, Kun Yi, Lin Song, Chen Li, Xiaohan Ding, and Ying Shan. Seed-x: Multimodal models with unified multi-granularity comprehension and generation. *arXiv preprint arXiv:2404.14396*, 2024.

[9] Google Gemini Team. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[10] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

[11] Wei Haoran, Kong Lingyu, Chen Jinyue, Zhao Liang, Ge Zheng, Yang Jinrong, Sun Jianjian, Han Chunrui, and Zhang Xiangyu. Vary: Scaling up the vision vocabulary for large vision-language models. *arXiv preprint arXiv:2312.06109*, 2023.

[12] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[13] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[14] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[15] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[16] Bohao Li, Yuying Ge, Yixiao Ge, Guangzhi Wang, Rui Wang, Ruimao Zhang, and Ying Shan. Seed-bench-2: Benchmarking multimodal large language models. *arXiv preprint arXiv:2311.17092*, 2023.

[17] Bohao Li, Rui Wang, Guangzhi Wang, Yuying Ge, Yixiao Ge, and Ying Shan. Seed-bench: Benchmarking multimodal llms with generative comprehension. *arXiv preprint arXiv:2307.16125*, 2023.

[18] Kaixin Li, Yuchen Tian, Qisheng Hu, Ziyang Luo, and Jing Ma. Mmcode: Evaluating multi-modal code large language models with visually rich programming problems. *arXiv preprint arXiv:2404.09486*, 2024.

[19] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

[20] Yanwei Li, Yuechen Zhang, Chengyao Wang, Zhisheng Zhong, Yixin Chen, Ruihang Chu, Shaoteng Liu, and Jiaya Jia. Mini-gemini: Mining the potential of multi-modality vision language models. *arXiv:2403.18814*, 2023.

[21] Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. Llava-next: Improved reasoning, ocr, and world knowledge, January 2024.

[22] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024.

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

[24] 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.

[25] Minesh Mathew, Dimosthenis Karatzas, and CV Jawahar. Docvqa: A dataset for vqa on document images. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 2200–2209, 2021.

[26] Kunato Nishina and Yusuke Matsui. Svgeditbench: A benchmark dataset for quantitative assessment of llm's svg editing capabilities. *arXiv preprint arXiv:2404.13710*, 2024.

[27] OpenAI. Chatgpt. `https://chat.openai.com`, 2023.

[28] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

[29] OpenAI. GPT-4V(ision) system card, 2023.

[30] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[31] 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.

[32] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[33] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[34] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*, 2023.

[35] Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691*, 2023.

[36] 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.

[37] Chengyue Wu, Yukang Gan, Yixiao Ge, Zeyu Lu, Jiahao Wang, Ye Feng, Ping Luo, and Ying Shan. Llama pro: Progressive llama with block expansion. *arXiv preprint arXiv:2401.02415*, 2024.

[38] John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

[39] Kaining Ying, Fanqing Meng, Jin Wang, Zhiqian Li, Han Lin, Yue Yang, Hao Zhang, Wenbo Zhang, Yuqi Lin, Shuo Liu, et al. Mmt-bench: A comprehensive multimodal benchmark for evaluating large vision-language models towards multitask agi. *arXiv preprint arXiv:2404.16006*, 2024.

[40] Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, et al. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*, 2024.

[41] 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.

[42] 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.

[43] Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36, 2024.

[44] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*, 2023.

A Prompt Template

In this section, we introduce the prompt template used for the experiment.

A.1 Prompt for Code Generation

We use the following for the direct asking setting.

> You are a helpful assistant that can generate Python code using matplotlib. Generate the matplotlib code to create a plot that looks like the given image, as similar as possible. The generated code should be surrounded by ```python and ```
>
> *<image_token><image_token><image_token>...*

For the conditional asking setting, we add the instruction of the reference plot at the front of the direct asking prompt.

> *<instruction>*
>
> You are a helpful assistant that can generate Python code using matplotlib. Generate the matplotlib code to create a plot that looks like the given image, as similar as possible. The generated code should be surrounded by ```python and ```
>
> *<image_token><image_token><image_token>...*

A.2 Prompt for Instruction Generation

Here is the prompt for generating each plot's corresponding instruction. We requite the GPT-4 to examine the code for each plot and summarize the key information in it without any implementation details.

> Please review the Python code provided below, which uses matplotlib.pyplot to generate figures. Your job is to identify key details, like type, texts, etc., required to recreate a figure from the given code: *<code>*
>
> Remember, your response should not include any code and avoid implementation details. Do not describe any detailed variables or functions in the code. Instead, use everyday language to describe the necessary information. If the code uses random seed, you should extract it for reproduction. Reveal the data used in the figure for recreation. Strictly follow the rule that do not expose any variables or functions used in the code. Summarize the crucial information as follows:

A.3 Prompt for Evaluation

We use the following prompt for GPT-4V overall rating. We will provide both the ground truth image and the test image generated by the MLLM assistant for GPT-4V to rate the similarity.

> You are a helpful assistant. Please evaluate the similarity between a reference image created using matplotlib and an image generated by code provided by an AI assistant. Consider factors such as the overall appearance, colors, shapes, positions, and other visual elements of the images. Begin your evaluation by providing a short explanation. Be as objective as possible. After providing your explanation After providing your explanation, you must rate the response on a scale of 1 to 10 by strictly following this format: "[[rating]]", for example: "Rating: [[5]]",
>
> *<gt_image_token><gt_image_token><gt_image_token>...*
>
> *<test_image_token><test_image_token><test_image_token>...*

In the pair evaluation, we utilize the following prompt to determine which generated image, either from Assistant A or Assistant B, is more similar to the ground truth image.

> You are a helpful assistant. Please act as an impartial judge and evaluate the quality of the generated images provided by two AI assistants given the ground truth image displayed below. You should choose the assistant that generate the more similar image. Your evaluation should consider factors such as the overall appearance, colors, shapes, positions, and other visual elements of the images.
>
> Here is the ground truth image.
>
> *<gt_image_token><gt_image_token><gt_image_token>...*
>
> Here is the image generated by the assistant A.
>
> *<test_image_A_token><test_image_A_token><test_image_A_token>...*
>
> Here is the image generated by the assistant B.
>
> *<test_image_B_token><test_image_B_token><test_image_B_token>...*
>
> Begin your evaluation by comparing the two responses and provide a short explanation. Avoid any biases and ensure that the order in which the responses were presented does not influence your decision.

A.4 Prompt Strategy

Additionally, we experiment with alternative prompt strategies that promote more reasoning by MLLM assistants, such as Chain-of-Thought (CoT) [36] and Plan-and-Solve (PS) [34].

The Chain-of-Thought (CoT) prompt is demonstrated below, wherein a specific sentence is added at the commencement of the assistant's response.

> *<USER>*
>
> You are a helpful assistant that can generate Python code using matplotlib ...
>
> -
>
> *<Assistant>*
>
> Let us think step by step. ...

We modify Plan-and-Solve (PS) strategy to make it compatible with our visual coding task and call it as PS+ in Table 3. It first encourage MLLM assistants to make a detailed plan.

> *<USER>*
>
> You are a helpful assistant that can generate Python code using matplotlib ...
>
> -
>
> *<Assistant>*
>
> Let us first describe the plot and make a detailed plan step by step ...

If the assistant outputs the code during the first step, the strategy will terminate. Otherwise, the second step will be employed, prompting the assistant to produce the final answer based on the plan described in the first stage.

> *Previous messages...*
>
> -
>
> *<Assistant>*
>
> Based on the above description, now we are prepared to generate the code. The generated code is surrounded by ```python and ``` to make it easier to be extracted by regular expressions. Therefore, the code is:

B Case Study

In this section, we present several examples using GPT-4V as the model under evaluation. We showcase cases from the direct asking setting (Figure 7), the conditional asking setting (Figure 8), and the pair-evaluation setting compared to Gemini-Pro (Figure 9), respectively. All the samples are drawn with the default prompt strategy.

C Correlation Analysis

In this section, we discuss the details of the correlation analysis. We select 20 pair evaluation samples with GPT-4v as the baseline in the conditional asking setting (10 compared to Gemini-Pro, 10 compared to Claude-3-Opus). Subsequently, we use these 20 samples to create an online questionnaire and invite colleagues from the lab, who hold at least a bachelor's degree, to participate. Each question in the questionnaire presents the ground truth image, the generated image from Assistant A, and the generated image from Assistant B. Participants are asked to choose one of the following three options:

- Assistant A's generated image is more similar to the ground truth image
- Assistant B's generated image is more similar to the ground truth image
- the level of similarity is close

In the end, we receive 46 completed questionnaires, resulting in 46 x 20 = 920 samples for conducting the correlation analysis.

Direct Asking

<USER>
Direct Asking Prompt (illustrated in Appendix A.1)
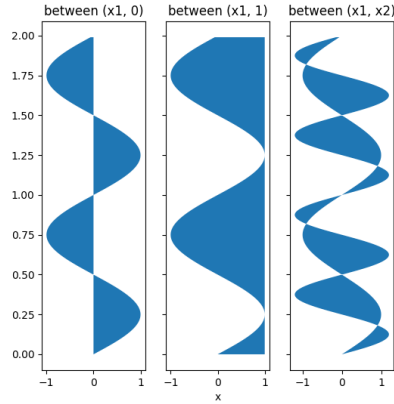
<GPT-4V>
```python
import matplotlib.pyplot as plt
import numpy as np
# Data for plotting
x = np.linspace(0, 2 * np.pi, 500)
y1 = np.sin(x)
y2 = np.sin(3 * x)
fig, axs = plt.subplots(1, 3, figsize=(9, 6), sharey=True)
# Plot with fill between y1 and 0
axs[0].fill_between(x, y1, 0, color='blue')
axs[0].set_title('between (x1, 0)')
# Plot with fill between y1 and 1
axs[1].fill_between(x, y1, 1, color='blue')
axs[1].set_title('between (x1, 1)')
# Plot with fill between y1 and y2
axs[2].fill_between(x, y1, y2, color='blue')
axs[2].set_title('between (x1, x2)')
# Set the x label only on the last subplot
for ax in axs:
    ax.label_outer()
    ax.set_xlim([0, 2*np.pi])
    ax.set_xticks([])  # Remove x-axis labels
plt.tight_layout()
plt.show()
```
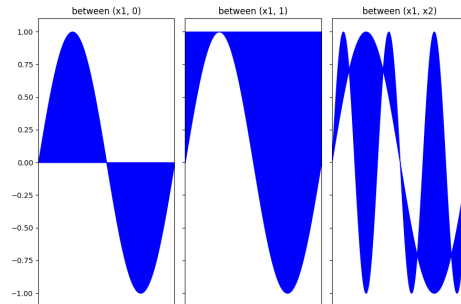
<Evaluation>
To evaluate the similarity between the reference image and the generated image, we will consider several factors:

1. Overall Appearance: The layout of both images includes three separate plots aligned horizontally. However, the reference image has a square aspect ratio for each plot, while the generated image has a rectangular aspect ratio, making the plots wider.
2. Colors: Both images use a similar shade of blue for the filled areas within the plots.
3. Shapes: The shapes within the plots are significantly different. The reference image contains smooth, wave-like shapes that are filled between certain bounds. The generated image also has wave-like shapes, but they are more sinusoidal and regular, with sharper peaks and troughs.
4. Positions: The positioning of the plots relative to each other is consistent in both images, with each plot directly adjacent to the next.
5. Visual Elements: The axes, labels, and titles are present in both images, but the font size and the exact positioning may vary slightly due to the different aspect ratios of the plots.

Considering these factors, the images share some similarities in layout and color but differ significantly in the shapes and aspect ratios of the plots. The overall concept seems to be similar, but the execution and the details of the plots are quite different.
Rating: [[3]]

Figure 7: A case of Direct Asking, showcasing the generated code, plot, and evaluation result.

Conditional Asking

<USER>
The figure created by this code consists of three subplots arranged horizontally, all sharing the same y-axis. The figure size is 6 by 6. The first subplot is a filled area plot between the y-axis and a curve defined by the sine of 2 times pi times y. The title of this subplot is 'between (x1, 0)'. The second subplot is a filled area plot between a vertical line at x=1 and the same sine curve as in the first subplot. The title of this subplot is 'between (x1, 1)', and it also has an x-axis label 'x'. The third subplot is a filled area plot between the sine curve and another curve defined by 1.2 times the sine of 4 times pi times y. The title of this subplot is 'between (x1, x2)'. The y values range from 0 to 2 with a step of 0.01. The x values for the first curve are the sine of 2 times pi times y, and for the second curve are 1.2 times the sine of 4 times pi times y.

Direct Asking Prompt (illustrated in Appendix A.1)
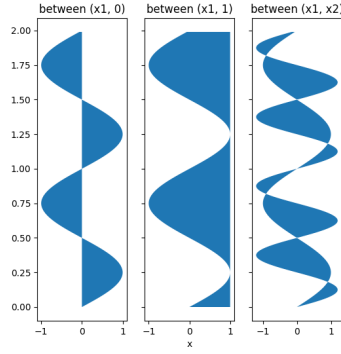
<GPT-4V>
```python
import matplotlib.pyplot as plt
import numpy as np
# Define the y values
y = np.arange(0, 2, 0.01)
# Define the x values for the curves
x1 = np.sin(2 * np.pi * y)
x2 = 1.2 * np.sin(4 * np.pi * y)
# Create the figure and subplots
fig, axs = plt.subplots(1, 3, figsize=(6, 6), sharey=True)
# First subplot
axs[0].fill_betweenx(y, x1, 0, color='blue')
axs[0].set_title('between (x1, 0)')
# Second subplot
axs[1].fill_betweenx(y, x1, 1, color='blue')
axs[1].set_title('between (x1, 1)')
axs[1].set_xlabel('x')
# Third subplot
axs[2].fill_betweenx(y, x1, x2, color='blue')
axs[2].set_title('between (x1, x2)')
# Display the plot
plt.tight_layout()
plt.show()
```
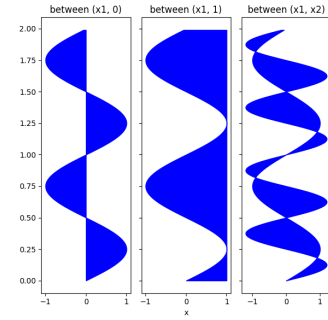

<Evaluation>
To evaluate the similarity between the reference image and the AI-generated image, we will compare the following elements:
1. Overall Appearance: We will look at the layout and composition of the images.
2. Colors: We will compare the color schemes used in both images.
3. Shapes: We will assess the shapes present in the images and their accuracy.
4. Positions: We will examine the positioning of the elements within the images.
5. Other Visual Elements: We will consider any additional visual elements that contribute to the likeness or differences between the images.

Upon comparing the two images:
- The overall appearance of both images is quite similar, with three panels displaying what appears to be filled curves.
- The colors used in both images are shades of blue, although the reference image has a lighter shade compared to the AI-generated image's deeper blue.
- The shapes of the filled curves are very similar, with slight variations in the curvature that could be attributed to the randomness in the data generation or rendering process.
- The positions of the curves within the panels are consistent, with each panel showing a progression of the shape's complexity or the interaction between two variables.
- Other visual elements such as the axes, labels, and titles appear to be consistent across both images.
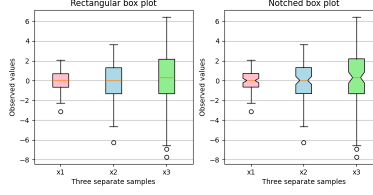
Considering these factors, the images are quite similar, with only minor differences in the shade of blue and slight variations in the shapes' curvature.
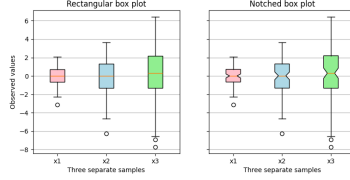Rating: [[9]]

Figure 8: A case of Conditional Asking, showcasing the generated code, plot, and evaluation result.
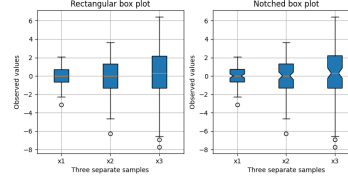
Pair Evaluation

Ground Truth Image

Assistant A　　　　　　　　　　　　**Assistant B**

GPT-4V Judgment (when A is placed in the first position):

Upon comparing the two responses to the ground truth image, it is clear that the image generated by Assistant A is more similar to the ground truth image than the image generated by Assistant B.

Assistant A's image closely matches the ground truth in terms of overall appearance, colors, shapes, and positions. The colors of the box plots (red, blue, green) are identical to the ground truth, and the layout of the plots, including the titles and axis labels, are accurately reproduced.

Assistant B's image, while maintaining the correct structure of a box plot, does not replicate the colors of the box plots accurately. All the boxes are blue, which differs from the ground truth image that has distinct colors for each box plot. Additionally, the titles and axis labels are missing, which are present in the ground truth.

Therefore, based on the evaluation criteria of overall appearance, colors, shapes, positions, and other visual elements, Assistant A has generated an image that is more similar to the ground truth image.

GPT-4V Judgment (when B is placed in the first position):

Upon comparing the images generated by Assistant A and Assistant B with the ground truth image, it is clear that Assistant B has produced an image that is more similar to the ground truth.

Assistant A's image differs significantly in color, using a uniform blue for all box plots, whereas the ground truth image uses distinct colors for each box plot (red, blue, green). Additionally, Assistant A's image does not replicate the notched feature in the right-hand side box plots, which is a key characteristic of the ground truth image.

Assistant B's image, on the other hand, closely matches the ground truth in terms of color, with each box plot accurately reflecting the colors in the ground truth (red, blue, green). Furthermore, Assistant B has successfully replicated the notched feature in the box plots on the right-hand side, just as in the ground truth image.

Therefore, based on the evaluation criteria of overall appearance, colors, shapes, and positions, Assistant B's image is more similar to the ground truth image.

Figure 9: A case of pair evaluation. We interchange the order of responses from the two assistants and conduct the evaluation twice. Both results are presented here.