CoDePlot: Evaluating the Chart Code Generation Capabilities of Large Vision Language Models on Realistic Charts

Anonymous ACL submission

Abstract

Large vision language models (VLMs) are increasingly used to solve tasks involving non-natural images such as charts, figures and diagrams. While VLMs often exhibit impressive capabilities in processing these images, there remains a gap in evaluation. Indeed, despite the fact that non-natural images play a significant role in many real-world applications, the vast majority of current benchmarks still focuses on natural images. We take a step toward closing this gap by introducing the *CoDePlot* benchmark, a challenging, novel and realistic dataset of 3k (chart, code) pairs obtained via heavy VLM-based filtering of permissively licensed Python Notebooks from Github. Along with our benchmark, we introduce a fine-grained rating system for comparing two charts according to different aspects (e.g., style and faithfulness), which allows VLMs-as-a-judge to obtain a high correlation with human raters. Using this system, we find that chart code generation is hard even for the highest-performing VLMs, with Gemini 2.0 Flash scoring at 82.6% and the best Open Weight model lagging behind at 49.9% on the hard benchmark examples. Finally, we introduce a training method which views chart code generation as *Inverse Rendering* to improve VLMs on CoDePlot. We use Inverse Rendering Training to train a small PaliGemma-3B model to score 57.8% — better than its substantially larger counterparts.

1 Introduction

Recent years have seen a shift from text-only large language models (LLMs) toward models capable of processing additional modalities such as images and audio. In particular, vision-language models (VLMs) trained on images and text have become an active area of research, as attested by the release of commercial models such as Gemini (Gemini Team Google, 2023) and GPT4 (OpenAI, 2024) as well as Open Weights models such as PaliGemma (Beyer et al., 2024), Qwen2-VL (Yang et al., 2024) and Llama 3.2 (Grattafiori et al., 2024). These models obtain strong performance on visual understanding tasks, including image captioning, visual question answering and image segmentation. However, a gap in evaluation persists: evaluation typically strongly focuses on *natural* images, although a large portion of images is *non-natural*; this includes charts, figures and diagrams (e.g., Hsu et al., 2021). We endeavour toward closing this gap by introducing *CoDePlot*, a benchmark consisting of (code, chart) pairs scraped from permissively licensed Python Notebooks on Github.[1] We heavily filter the (code, chart) pairs using an automatic VLM-based assessment to create a challenging, realistic and novel dataset divided into 2,744 easy

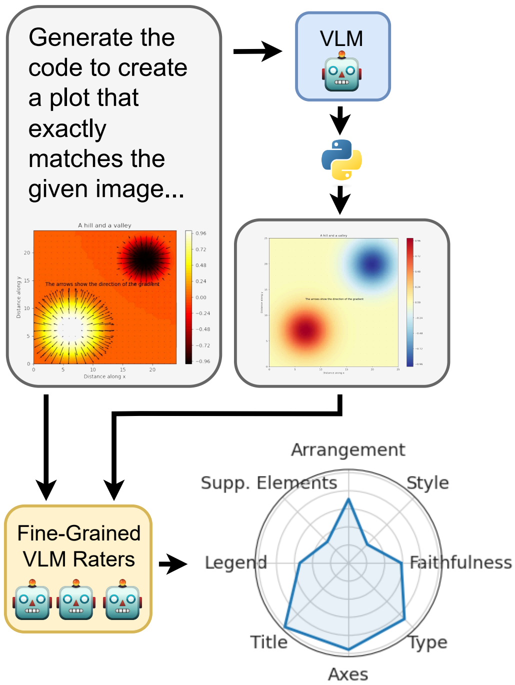


Figure 1: CoDePlot entails a dataset of (code, chart) pairs and a fine-grained chart comparison scheme.

[1]https://github.com

and 214 hard examples. The task associated with CoDePlot is *chart code generation*: given an image of a chart, generate the code used to create the chart. Figure 1 shows an example of the hard subset along with the corresponding predictions of state-of-the-art VLMs. Chart code generation requires multiple capabilities, including recovering the data underlying the chart (Liu et al., 2023a), compositional understanding of chart elements as well as code generation. Chart code generation is orthogonal to higher-level semantic tasks such as chart question answering (Masry et al., 2022).

Evaluation of chart code generation poses a substantial challenge: We need to compare the images of the ground truth chart and the chart generated from the predicted code, but traditional image similarity metric such as pixel-space mean squared error and SSIM (Wang et al., 2004) are too low-level for this task (Wu et al., 2024). Prior work thus resorts to using an VLM-as-a-judge tasked with rating the similarity of the two chart images from 1 to 10 (Wu et al., 2024). However, the VLM judge in this scenario only exhibits moderate correlation with human judgements. A single overall score also does not permit easily interpreting the rating results. To solve these problems, we introduce a *fine-grained* evaluation scheme for comparing two charts. We break down chart similarity into eight categories rated on a Likert scale (Likert, 1932) from 1 to 4. The ratings can then be averaged to obtain a single score. Besides providing more interpretable results, our rating scheme also permits VLMs-as-a-judge to obtain a correlation with human judgements on par with the correlation between human annotators (Section 7).

The CoDePlot benchmark is challenging, even for the strongest VLMs: the highest average score is 82.6%, achieved by Gemini 2.0 Flash, with Claude 3.5 Sonnet and GPT-4o behind. The hardest category (*faithfulness*, i.e., accuracy of the depicted data) is even more challenging at a maximum score of 66.9%. Open Weight VLMs substantially underperform commercial ones: Qwen2-VL-72B and Llama 3.2 90B Vision achieve average scores of 27.9% and 49.9%, respectively. To close this gap, we propose a way to generate diverse synthetic data for the chart code generation task by viewing it as *Inverse Rendering* (Section 5): this allows fine-tuning a PaliGemma (Beyer et al., 2024) model to perform competitively for its size at 84.2% and 57.8% average score on the easy and hard splits, respectively, with only 3B parameters. Finally, we conduct a thorough analysis of the VLMs' predictions on CoDePlot, finding low contamination, common failure modes and the ability of VLMs to iteratively improve their predictions, among other insights (c.f. Section 7).

Contributions **1)** We introduce a dataset of 3k realistic high-quality (code, chart) pairs from Github Notebooks containing Python code divided into an easy and a hard split along with the associated benchmark task of generating the code used to produce the charts. **2)** We propose a fine-grained set of evaluation criteria for comparing a reference and a ground-truth chart to enable nuanced comparison of predictions while also enabling VLMs-as-a-judge to achieve a correlation to human raters on par with humans. **3)** We show how viewing the chart code generation task as *inverse rendering* enables training a competitive fine-tuned PaliGemma model purely on synthetic data for this task and release the fine-tuned model.

2 Related Work

Vision Language Models. VLMs such as CLIP (Radford et al., 2021) and SigLIP (Zhai et al., 2023) are contrastively trained on image-text pairs. These models are capable of computing the similarity between an (image, text) pair, but are not capable of generating text on their own. Thus, it has become common practice to fuse them with a pretrained text decoder; e.g., PaliGemma has been trained by fusing a SigLIP vision encoder with the Gemma-2B language model (Beyer et al., 2024). Fuyu (Bavishi et al., 2023) and EVE (Diao et al., 2024) explore an alternative approach directly combining image and text in a single decoder-style model. Commerical models including Gemini (Gemini Team Google, 2023), GPT4 (OpenAI, 2024) and Claude 3.5 Sonnet (Anthropic, 2024) can also process images, however, their architectural details are not public. Some existing VLMs have been fine-tuned for processing charts, including ChartLlama (Han et al., 2023), ChartAssistant (Meng et al., 2024), ChartInstruct (Masry et al., 2024a) and ChartGemma (Masry et al., 2024b). These models are not well suited to the chart code generation task out-of-the-box since they have not been trained on code.

Related Benchmarks. Benchmarks on chart understanding include PlotQA (Methani et al., 2020) and ChartQA (Masry et al., 2022), where the task is visual question answering (e.g. answering "What is the peak value of the orange line?") based on chart images. HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) are commonly used to evaluate coding performance. HumanEval and MBPP consist of a set of natural language prompts specifying desired function behavior and corresponding unit tests. They assess code generation capabilities by testing whether the code snippet generated for the given prompt is functionally correct (i.e., passes all unit tests). The recently released HumanEval-V augments the HumanEval-style problem setup with additional input images required to solve the coding tasks (however, these are not necessarily charts; Zhang et al., 2024). The Plot2Code dataset (Wu et al., 2024) is the first to combine chart understanding and coding by introducing the chart code generation task (the same task as in CoDePlot) based on chart images scraped from the matplotlib[2] example gallery and their corresponding code snippets. While being a first step toward measuring chart code generation capabilities, we hypothesize (i) evaluating on Plot2Code is not robust due to its small size (132 examples), (ii) being sourced from example gallery charts, Plot2Code does not match the distribution of charts occurring in the wild and (iii) the metrics used in Plot2Code (mainly, a single overall VLM-as-a-judge rating) do not adequately capture all aspects of chart code generation. Moreover, we find evidence toward substantial contamination on Plot2Code (c.f. Section 7), presumably due to the charts from the example gallery being copied and re-uploaded to other platforms (e.g., Github) many times over and becoming a part of the training data prone to being memorized. Finally, while Plot2Code only covers matplotlib, we extend coverage to the popular *seaborn* library (Waskom, 2021).

3 Benchmark Construction

Initial Collection. We start from the raw data source used for creating the MatCha training data (Liu et al., 2023b) consisting of (chart, code) pairs obtained by crawling all GitHub IPython Notebooks with appropriate licenses and, for cells which have an image as output, storing the cell code and the image. The raw dataset consists of 20M (chart, code) pairs. However, the vast majority of code snippets are not executable on their own: they depend on data obtained from previous cells (such as the outputs of a training run) or external data (such as a dataset obtained from some URL). As a first step, we filter the dataset to keep only code snippets which are executable in a sandboxed Python environment; this removes >99% of the (chart, code) pairs, reducing the size of the dataset to 116k executable code snippets and the corresponding charts. After thorough deduplication via MinHash (Broder, 1997), ~37k pairs remain.

[2] https://matplotlib.org

	Easy	Hard
Num. Examples	2744	214
Bar	818	39
Contour	6	6
Line	1293	106
Pie	227	7
Scatter	311	34
Misc.	89	22
Code		
Lines	25 ± 21	42 ± 34
Image		
Width	661 ± 244	798 ± 340
Height	486 ± 129	581 ± 267

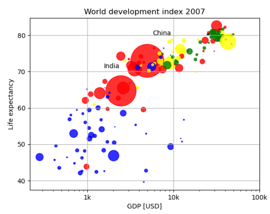
Table 1: Evaluation dataset statistics. The value following $\pm$ indicates a single standard deviation.

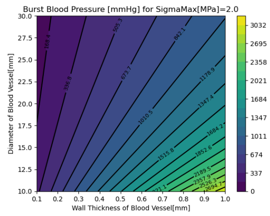
VLM Filtering. The 37k executable and deduplicated pairs form the basis of our dataset. To create a challenging and diverse set to evaluate on, we further filter the dataset using a VLM (in pratice, Gemini 1.5 Flash; Gemini Team Google, 2023). We formulate multiple axes among which to measure the quality of an example: (i) *Informativeness*, (ii) *Visual Appeal*, (iii) *Realisticness*, (iv) *Completeness* and (v) *Complexity* and prompt the VLM to rate examples on a scale from 1-10 in each category (prompt in Appendix A). We keep the examples with a harmonic mean of (i)-(iv) ≥ 7.5 and further divide the dataset into *easy* (Complexity ≤ 5) and *hard* (Complexity > 5) examples. Alongside prompting for the quality scores we also prompt the VLM to describe the chart type as free-form text. We then cluster the chart type responses by keywords to obtain 6 categories to divide the examples into. Table 1 shows summary statistics of our evaluation dataset and Figure 2 shows representative examples of each of the different categories.

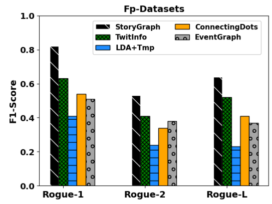
4 Metrics

Evaluating performance of a chart code generation model is challenging. We need a way to measure the similarity of the ground truth chart and the chart

Image







Code

```python
gdp_cap = [...]
life_exp = [...]
pop = [...]
col = [...]

plt.scatter(
    x=gdp_cap,
    y=life_exp,
    s=np.array(pop) * 2,
    c=col,
    alpha=0.8,
)

plt.xscale("log")
plt.xlabel("GDP [USD]")
plt.ylabel("Life expectancy")
plt.title("World development index 2007")
plt.xticks(
    [1000, 10000, 100000],
    ["1k", "10k", "100k"]
)

[...]
```

```python
n = 100
t = np.linspace(0.1, 1.0, n) / 1000.0
d = np.linspace(10.0, 30.0, n) / 1000.0

T, D = np.meshgrid(t, d)
T_plt, D_plt = np.meshgrid(
    t * 1000.0, d * 1000.0
)
sigma = 2.0 * 1.0e6  # [Pa]

P = sigma * (2 * T) / D  # [Pa]
P = P / 133.3223684  # [Pa]->[mmHg]

lvls = np.linspace(0, 3200, 20)

cf = plt.contourf(
    T_plt,
    D_plt, P,
    levels=lvls
)
cp = plt.contour(
    T_plt, D_plt, P, levels=lvls, colors="k"
)

[...]
```

```python
plt.rcParams["font.size"] = 13
plt.rcParams["font.weight"] = "bold"
# set width of bar
barWidth = 0.11
# set height of bar
bars1 = [0.82, 0.53, 0.64]
bars2 = [0.63, 0.41, 0.52]

[...]

# Make the plot
plt.bar(
    r1,
    bars1,
    color="black",
    width=barWidth,
    edgecolor="white",
    label="StoryGraph",
    hatch="\\",
)

[...]
```

Category scatter | contour | bar

Figure 2: Examples in the hard split of the CoDePlot benchmark.

which is the result of running the predicted code. Plot2Code (Wu et al., 2024) introduces a metric to measure whether text elements in the two charts align in terms of position and text value (referred to as *text match score*), and a metric relying on an VLM-as-a-judge to rate the similarity of two plots on a scale of one to ten. These scalar metrics are necessarily reductive (Keeney, 1993): there are many different ways in which two charts can be similar (or dissimilar). For example, it is not clear whether two charts which contain the *same data* but display it in *different ways* should be rated as more or less similar than two charts which present *different data* in the *same way*. In line with our approach to data filtering (Section 3), we thus propose evaluating across multiple axes: we create a fine-grained set of eight different categories according to which two charts can be similar, and employ an VLM-as-a-judge to evaluate similarity along these categories. The rating scheme encompasses the categories *Chart Type*, *Axes*, *Title*, *Legend*, *Supplementary Elements*, *Arrangement*, *Style* and *Faithfulness*. Each category is rated on a Likert scale as either 1 (incorrect), 2 (partially correct), 3 (mostly correct) or 4 (correct). Details on each category can be found in Table 2, and a set of comprehensive guidelines for comparing along these categories in Appendix A. Our fine-grained rating scheme disambiguates the comparison of two charts; analogously, prior research has introduced effective fine-grained rating schemes for different specialized areas (Burchardt, 2013; Lo et al., 2022).

Category	Description
Chart Type	Does the chart use the same chart type as the ground truth?
Axes	Does the chart use the same axes as the ground truth?
Title	Does the chart use the same title as the ground truth?
Legend	Does the chart use the same legend as the ground truth?
Supplementary Elements	Does the chart use the same supplementary elements as the ground truth? Supplementary elements are all text and visual elements which are not in the other categories (including subtitles, annotations, markers, . . .)
Arrangement	Is the placement of the visual elements in the chart consistent with the ground truth?
Style	Does the chart use the same color palette, font, fills and decorations as the ground truth?
Faithfulness	Is the information communicated by the chart consistent with the ground truth?

Table 2: Fine-grained chart comparison categories and their description; Appendix A contains our guidelines for rating along these categories.

Fine-grained evaluation has also been shown to be more suited toward LLMs-as-a-judge than a single overall score on natural language text tasks (Ye et al., 2024); we confirm the same is true for VLMs later in Section 7.

5 Inverse Rendering Training

Inspired by approaches to neural inverse rendering (Sengupta et al., 2019; Tewari et al., 2022,

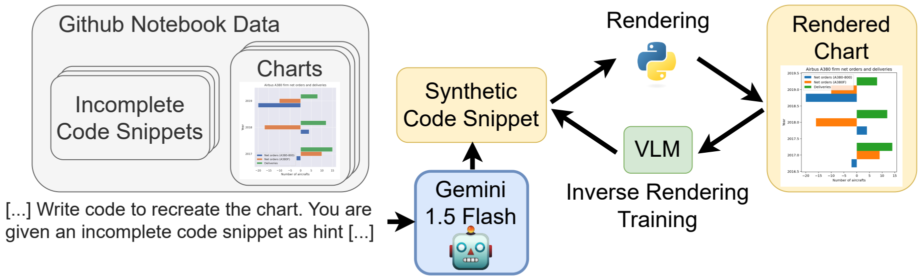


Figure 3: Synthetic data generation for Inverse Rendering Training: an existing VLM is prompted to generate a code snippet, conditioned on an existing chart and a corresponding incomplete code snippet, which is then rendered via the Python interpreter. The resulting (rendered chart, synthetic code) snippet is used to train another VLM.

	Model	Execution Rate	Type	Axes	Title	Leg.	Supp.	Arr.	Sty.	Fai.	*Avg.*
Easy	*Qwen2-VL-72B*	65.0	55.5	43.6	49.0	48.0	39.7	43.5	30.1	36.1	43.1
	Llama 3.2 90B Vision	91.6	78.5	69.7	72.7	74.3	58.2	64.7	49.5	55.0	64.7
	PaliGemma-3B-IR (ours)	92.1	91.0	82.6	90.0	88.8	85.5	84.2	84.2	70.1	84.2
	Gemini 1.5 Flash	86.1	84.7	77.3	83.2	82.4	79.6	77.5	77.4	66.2	78.3
	Gemini 2.0 Flash	**96.4**	**96.2**	**91.3**	**95.2**	**95.2**	**93.0**	**92.3**	**91.0**	**82.3**	**91.8**
	GPT-4o	93.7	92.6	81.6	90.4	88.5	81.7	81.9	73.0	67.7	81.5
	Claude 3.5 Sonnet	94.2	92.0	81.1	89.5	89.0	68.2	77.8	65.2	75.3	79.0
Hard	*Qwen2-VL-72B*	54.2	38.3	30.4	36.5	30.4	25.7	26.3	19.6	16.3	27.9
	Llama 3.2 90B Vision	84.1	65.1	56.5	62.5	59.9	41.4	46.1	36.3	33.6	49.9
	PaliGemma-3B-IR (ours)	71.5	67.0	60.7	66.4	63.6	55.2	55.5	57.0	38.0	57.8
	Gemini 1.5 Flash	71.5	66.4	61.7	67.0	64.2	59.0	55.8	57.3	45.7	59.6
	Gemini 2.0 Flash	91.1	89.8	**83.5**	**88.8**	**87.4**	**82.3**	**81.1**	**82.1**	**66.9**	**82.6**
	GPT-4o	89.3	86.1	73.4	81.7	78.1	71.3	70.3	61.6	54.6	71.9
	Claude 3.5 Sonnet	**93.0**	**91.6**	80.6	86.6	81.0	66.3	72.8	58.6	65.5	75.1

Table 3: Results of multiple Open Weight models (indicated via *italics*) and commercial models on the easy and hard splits of our evaluation dataset. Ratings are computed by using an ensembled Gemini Flash as the judge.

among others), we propose viewing chart code generation as an inverse rendering task. Traditionally, inverse rendering is concerned with recovering the properties (e.g., lighting, geometry) of a scene from one or more images. In chart code generation, we are tasked with recovering an abstract representation (i.e., the code) of a chart, knowing it has been rendered via the Python interpreter. In this case, we can view the generated chart image as the scene, and the underlying code as the property to recover. The chart code generation task then becomes the task of reversing the rendering conducted by the Python interpreter. Neural approaches to inverse rendering have been driven to a substantial extent by *differentiable* rendering (c.f. Kato et al., 2020). The Python interpreter, however, is not differentiable. Nonetheless, the Inverse Rendering perspective allows proposing a way to generate synthetic data for the chart code generation task.

Synthetic Inverse Rendering Data Generation. Our method hinges on the fact that there already exist VLMs which are proficient at generating Python code, even though they are not necessarily adept at chart code generation (as shown by the results on our benchmark; c.f. Section 6). Thus, we can prompt a VLM to generate the Python code for rendering a source chart, execute the synthetic code to obtain a synthetic chart, and use the resulting (rendered chart, synthetic code) pair to train a VLM to produce the synthetic code, conditioned on the rendered chart. This amounts to an inverse rendering process: Upon data creation, we run a forward pass through the Python interpreter to map code $\rightarrow$ chart, then, during training, the VLM learns the inverse chart $\rightarrow$ code mapping. However, in practice, unconditionally prompting the VLM leads to highly similar generations, making it necessary to condition on an external source of diversity (Chan et al., 2024). We condition generation on the same raw corpus used to create the CoDePlot dataset (c.f. Section 3), but use only *unexecutable* code snippets (code snippets relying on external variables, web URLs, etc.) to ensure there is no overlap with the evaluation data. The data generation and training process is illustrated in Figure 3.

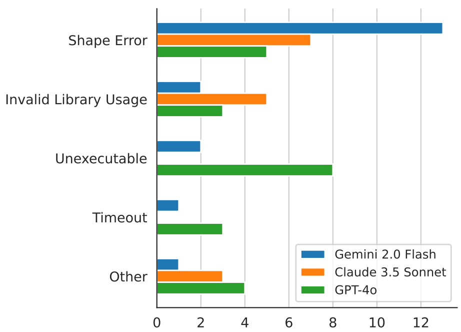


Figure 4: Error groups of a manual error analysis conducted on the errors made by the highest-scoring VLMs on the hard CoDePlot split. *Shape errors* (e.g., incompatible number of x/y data points) are predominant.

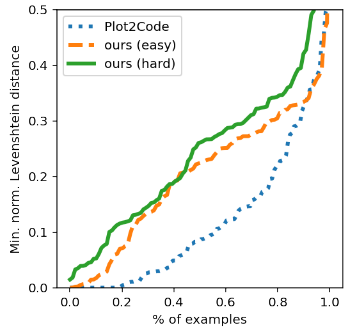


Figure 5: Minimum normalized Levenshtein distances between eight VLM completions and the ground truth when prompted with the chart and the first 50% of the corresponding code snippet.

Training. We fine-tune PaliGemma on a dataset of 2.6M synthetic (code, chart) pairs created via Gemini 1.5 Flash.[3] We tune hyperparameters starting from the settings recommended by Beyer et al. (2024); the final values are reported in Appendix B.

6 Benchmark Results

We evaluate multiple Open Weight and commercial models using our fine-grained evaluation, using the average scores of eight Gemini calls as the judge; we show later in Section 7 that this judge is highly correlated with human judgements. Results are shown in Table 3, and qualitative examples in Appendix D. Our PaliGemma Inverse Rendering fine-tune performs competitively, especially on the easy split, where it outperforms GPT4-o, Claude and Gemini 1.5 Flash (the model used to generate its training data). The fine-grained scores also permit drawing conclusions about model behavior: e.g. GPT-4o outperforms Claude in some categories, but has a lower average score due to being worse at axes and faithfulness. Furthermore, chart type, axes, title and legend are the least challenging, while supplementary elements and arrangement pose a greater challenge, and style and faithfulness are the most challenging. Low faithfulness is particularly problematic: it suggests the evaluated VLMs are unable to recover the data from the chart; this has implications for a wide range of applications.

7 Discussion

Unexecutable predicted code follows common patterns. We analyzed cases where models do not produce a valid chart due to the code not executing successfully and manually grouped errors into categories. The results are shown in Figure 4. A substantial portion of the errors made by all models are *shape errors*, that is, errors where the model e.g. produced arrays of different length for the x and the y axes. This is not surprising: VLMs have been shown to underperform at counting (Golovneva et al., 2024; Sterz et al., 2024), which is an implicit requirement for avoiding shape errors. The CoDePlot benchmark may be a useful measure of VLMs' *implicit* counting abilities, i.e., the ability to count when it is not the final goal but instead an intermediate step. Besides shape errors, all models make errors in their usage of the plotting libraries (e.g., passing invalid keyword arguments). GPT-4o and Gemini also sometimes output unexecutable code snippets (e.g., continuing to append elements to the data variables and running out of context length).

CoDePlot is not substantially contaminated. To measure how likely VLMs are to have memorized our data, we prompt Gemini to complete the code snippet, given the chart and the first 50% (rounded down) of lines of code. We sample eight completions, and use the minimum Levenshtein distance (Yujian and Bo, 2007) to the ground truth code among the eight samples as the measure of contamination.[4] Results are shown in Figure 5. 18% of Plot2Code examples have an exact match (Levenshtein distance = 0) among the eight completions, while this is the case for 2% and 0% of our easy and hard splits, respectively. This points toward the CoDePlot benchmark being unlikely to be memorized by state-of-the-art VLMs.

[3] Gemini 2.0 was not available at the time.

[4] Levenshtein distance can act as a proxy for memorization since the semantics of code are invariant to many surface-level characteristics (e.g., comments and formatting, but also swapping two lines as long as they are independent).

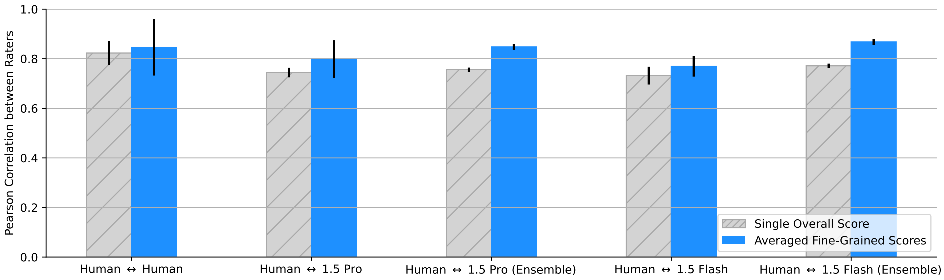


Figure 6: Pearson correlation coefficient between Human annotators and between Humans and VLMs-as-a-judge. Correlation of the VLMs-as-a-judge with human raters is positively affected by fine-grained rating and ensembling, where the VLM is called multiple (eight) times, then the results averaged.

Fine-grained evaluation exhibits high inter-annotator agreement. We let nine expert human annotators evaluate a total of 180 Gemini Flash predictions on our evaluation data according to our fine-grained evaluation guidelines (details in Appendix C). The task was to compare the true chart with the chart produced by the predicted code, without taking the true and predicted code into account.[5] Each annotator annotated 20 examples chosen to partially overlap such that every example is annotated by a maximum of five annotators. In addition, they were asked to provide a single overall chart similarity score (from 1-10). Inter-annotator agreement is generally high and consistent across multiple metrics (Table 5). *Arrangement* is the category with the lowest agreement: annotators might have trouble judging positioning *in aggregate* across the chart. Furthermore, averaging the scores of the fine-grained categories leads to a metric with slightly higher inter-annotator agreement than the single overall score.

Fine-grained evaluation is better suited to VLMs-as-a-judge than a single overall score. To measure how well VLMs-as-a-judge do at rating according to our guidelines, we prompt models of two different sizes to rate the examples which were also human-annotated. We then sequentially substitute each human annotator's ratings with the VLM rating and average the results to quantify Human $\leftrightarrow$ VLM agreement. In addition to performing one call of the VLM-as-a-judge for every example, we also experiment with *ensembling* the scores from multiple (in practice, eight) calls. Results are shown in Figure 6. Our findings are: (i) average fine-grained scores exhibit higher inter-annotator agreement than a single overall score (as also shown in Table 5), (ii) average fine-grained scores lead to higher Human $\leftrightarrow$ VLM agreement and (iii) average fine-grained scores benefit substantially more from ensembling multiple calls, enabling even the smaller Gemini Flash to attain human-level agreement, while this is not the case for the single overall score. We thus conclude that comparing two charts via ensembling multiple fine-grained scores

[5] We based this decision off of our view that the code is an intermediate representation which should not affect ratings.

Category	**LLM**	**RMSE**	**SSIM**	**cBLEU**	**cBLEU +SSIM**
Type	0.81	-0.05	0.06	0.15	0.15
Axes	0.63	-0.12	-0.25	0.16	-0.07
Title	0.68	0.14	0.26	0.16	0.24
Leg.	0.63	-0.31	0.22	-0.12	0.05
Supp.	0.55	-0.20	0.24	0.28	0.33
Arr	0.57	-0.30	0.15	0.04	0.13
Sty.	0.54	-0.35	0.15	-0.03	0.07
Fai.	0.74	0.13	0.15	0.31	0.33
Average	0.85	-0.20	0.11	0.23	0.25
Single Score	0.69	0.02	0.15	0.28	0.30

Table 4: Pearson correlation of automatic metrics to human ratings. *LLM* refers to using a Gemini 1.5 Flash ensemble as judge. *cBLEU+SSIM* is the sum of Code-BLEU (Ren et al., 2020) and SSIM (Wang et al., 2004).

Category	α	r	ρ
Chart Type	0.58	0.80	0.79
Axes	0.54	0.65	0.61
Title	1.00	1.00	1.00
Legend	0.86	0.86	0.78
Supp. Elements	0.76	0.85	0.86
Arrangement	0.38	0.37	0.41
Style	0.60	0.63	0.63
Faithfulness	0.80	0.82	0.82
Average	0.84	0.82	0.77
Single Overall Score	0.74	0.77	0.73

Table 5: Inter-annotator agreement measured via Krippendorff (2011)'s α, Pearson r and Spearman ρ.

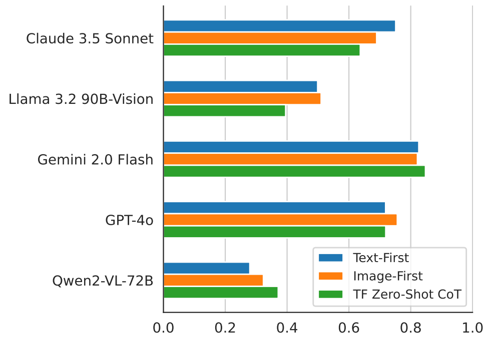


Figure 7: Sensitivity of multiple VLMs to different prompting strategies. In *Image-First*/*Text-First* the order of the image/text in the prompt is adapted. In *Text-First Zero-Shot CoT* the VLM is prompted to first plan out the necessary steps, then write the code.

	Model	R0	R1	R2
Avg. Score	Gemini 2.0 Flash	**82.6**	**84.6**	**87.0**
	Claude 3.5 Sonnet	75.1	79.5	81.1
	GPT-4o	71.9	77.5	79.5
Exec. Rate	Gemini 2.0 Flash	91.1	92.5	95.3
	Claude 3.5 Sonnet	**93.0**	**97.2**	**98.1**
	GPT-4o	89.3	93.4	94.9

Table 6: Average LLM scores when iteratively refining the previous iterations' predictions. *R0* are the initial predictions (no refinement).

from smaller VLMs is the best method, while also tending to be cheaper than a single call to a larger VLM.[6] In Table 4, we confirm that the ensembled fine-grained VLM-as-a-judge has higher correlation with human ratings than the lower-level RMSE, SSIM and CodeBLEU (Ren et al., 2020).

Variations in the prompt affect models differently. Following the Plot2Code benchmark (Wu et al., 2024), the prompt in our main experiments has the instructions first and the input chart image second. We also test the reverse: putting the image first, and the instructions second. Furthermore, we analyze prompting the model to plan its steps before writing the code, akin to Zero-Shot Chain-of-Thought prompting (Kojima et al., 2022). The effect of these two dimensions of variation is shown in Figure 7. The optimal prompting strategy varies heavily across models: the Llama 3.2, GPT-4o, and Qwen2 models benefit from putting the image first, while Claude and Gemini perform better in the text-first setting.[7] Zero-Shot CoT improves performance for Qwen2 and Gemini, but decreases performance for the other models. The prompts are shown in Appendix A. The results from this analysis highlight the difficulty of fairly comparing different VLMs. However, our main observation that the CoDePlot benchmark is challenging for all existing VLMs is not affected by these findings.

VLMs are capable of iteratively improving their answers. For humans, a natural way to solve the chart code generation task would be to write some initial code, look at the chart it produces, then iteratively refine the code until the produced chart matches the ground truth chart. Conversely, in many cases, humans would presumably not be able to solve the task without the continuous feedback from the Python interpreter. We thus also test a setup where the VLM is able to iteratively improve its prediction: we run multiple rounds of *refinement*, where the VLM is given its previously produced code alongside the input image, as well as either (a) the chart produced by rendering the predicted code or (b) the error produced by running the predicted code, if the code did not successfully produce a chart (prompts in Appendix A). As shown in Table 6, iterative refinement improves the predictions: the first round is especially effective, but more rounds lead to continued improvements.

8 Conclusion

We have introduced the CoDePlot benchmark: a challenging benchmark of highly realistic charts with the associated task of chart code generation. By employing VLMs-as-a-judge which use a fine-grained rating scheme specifically designed to compare two charts, we found that this is a challenging task for state-of-the-art VLMs, with the best one achieving an average score of 82.6%. We have verified the credibility of our rating scheme by confirming that VLMs-as-a-judge obtain high correlation with human judgements, especially if the judge is ensembled. Furthermore, we introduced *Inverse Rendering Training* for chart code generation, a way to obtain synthetic data for this task, which allowed fine-tuning a small PaliGemma-3B model to perform competitively with an average score of 57.8% on CoDePlot. Finally, we thoroughly analyzed the models' predictions on CoDePlot, finding low contamination, common error patterns and the ability of the VLMs to iteratively improve their predictions, among other insights.

[6] At the time of writing, this is the case for the Gemini Family: eight calls to Gemini Flash are cheaper than one call to Gemini Pro (https://ai.google.dev/pricing).

[7] The sensitivity to text/image order has already been observed by Wardle and Susnjak (2024), who found that the task plays a role, but did not observe differences across models.

Limitations

One limitation of our dataset construction process is that the data comes from public sources, so it is possible that VLMs have already seen the data during their training process. However, by analyzing contamination (Section 7), we found that our dataset has likely not been seen in this form during training. Another limitation is the choice of prompting methods: for simplicity and scalability we have opted to use the same prompt to evaluate all VLMs in our main results, however, we also found substantial variation in which prompting method works best for any given VLM (c.f. Section 7). Future work could investigate systematic ways to mitigate the prompt variance of LLM and VLM evaluation. A limitation of our synthetic training dataset is becoming dated quickly as new VLMs get released (e.g., during this work, Gemini 2.0 was released, making Gemini 1.5 outdated), however, we provide full details to re-generate the dataset with more up-to-date VLMs to mitigate this limitation. Finally, although our VLMs-as-a-judge exhibit high correlation with human raters, they can likely not completely replace humans due to potential subtle biases (Li et al., 2025).

References

Anthropic. 2024. Claude 3.5 sonnet.

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

Rohan Bavishi, Erich Elsen, Curtis Hawthorne, Maxwell Nye, Augustus Odena, Arushi Somani, and Sağnak Taşırlar. 2023. Introducing our multimodal models.

Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, Thomas Unterthiner, Daniel Keysers, Skanda Koppula, Fangyu Liu, Adam Grycner, Alexey Gritsenko, Neil Houlsby, Manoj Kumar, Keran Rong, Julian Eisenschlos, Rishabh Kabra, Matthias Bauer, Matko Bošnjak, Xi Chen, Matthias Minderer, Paul Voigtlaender, Ioana Bica, Ivana Balazevic, Joan Puigcerver, Pinelopi Papalampidi, Olivier Henaff, Xi Xiong, Radu Soricut, Jeremiah Harmsen, and Xiaohua Zhai. 2024. Paligemma: A versatile 3b vlm for transfer.

Andrei Z Broder. 1997. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE.

Aljoscha Burchardt. 2013. Multidimensional quality metrics: a flexible system for assessing translation quality. In *Proceedings of Translating and the Computer 35*, London, UK. Aslib.

Xin Chan, Xiaoyang Wang, Dian Yu, Haitao Mi, and Dong Yu. 2024. Scaling synthetic data creation with 1,000,000,000 personas. *arXiv preprint arXiv:2406.20094*.

Mark Chen, Jerry Tworek, Heewoo Jun, et al. 2021. Evaluating large language models trained on code.

Haiwen Diao, Yufeng Cui, Xiaotong Li, Yueze Wang, Huchuan Lu, and Xinlong Wang. 2024. Unveiling encoder-free vision-language models.

Gemini Team Google. 2023. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Olga Golovneva, Tianlu Wang, Jason Weston, and Sainbayar Sukhbaatar. 2024. Contextual position encoding: Learning to count what's important.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. 2024. The llama 3 herd of models.

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

Ting-Yao Hsu, C Lee Giles, and Ting-Hao Huang. 2021. SciCap: Generating captions for scientific figures. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3258–3264, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. 2020. Differentiable rendering: A survey. *arXiv preprint arXiv:2006.12057*.

Ralph L Keeney. 1993. *Decisions with multiple objectives: Preferences and value tradeoffs*. Cambridge university press.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*.

Klaus Krippendorff. 2011. Computing krippendorff's alpha-reliability.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng, and Huan Liu. 2025.

From generation to judgment: Opportunities and challenges of llm-as-a-judge.

Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of Psychology*.

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

Fangyu Liu, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Yasemin Altun, Nigel Collier, and Julian Eisenschlos. 2023b. MatCha: Enhancing visual language pretraining with math reasoning and chart derendering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12756–12770, Toronto, Canada. Association for Computational Linguistics.

Leo Yu-Ho Lo, Ayush Gupta, Kento Shigyo, Aoyu Wu, Enrico Bertini, and Huamin Qu. 2022. Misinformed by visualization: What do we learn from misinformative visualizations? In *Computer Graphics Forum*, volume 41, pages 515–525. Wiley Online Library.

Ahmed Masry, Xuan Long Do, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. 2022. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2263–2279, Dublin, Ireland. Association for Computational Linguistics.

Ahmed Masry, Mehrad Shahmohammadi, Md Rizwan Parvez, Enamul Hoque, and Shafiq Joty. 2024a. Chartinstruct: Instruction tuning for chart comprehension and reasoning.

Ahmed Masry, Megh Thakkar, Aayush Bajaj, Aaryaman Kartha, Enamul Hoque, and Shafiq Joty. 2024b. Chartgemma: Visual instruction-tuning for chart reasoning in the wild.

Fanqing Meng, Wenqi Shao, Quanfeng Lu, Peng Gao, Kaipeng Zhang, Yu Qiao, and Ping Luo. 2024. ChartAssistant: A universal chart multimodal language model via chart-to-table pre-training and multitask instruction tuning. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 7775–7803, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Nitesh Methani, Pritha Ganguly, Mitesh M. Khapra, and Pratyush Kumar. 2020. Plotqa: Reasoning over scientific plots. In *The IEEE Winter Conference on Applications of Computer Vision (WACV)*.

OpenAI. 2024. Gpt-4 technical report.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*.

Soumyadip Sengupta, Jinwei Gu, Kihwan Kim, Guilin Liu, David W Jacobs, and Jan Kautz. 2019. Neural inverse rendering of an indoor scene from a single image. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8598–8607.

Hannah Sterz, Jonas Pfeiffer, and Ivan Vulić. 2024. Dare: Diverse visual question answering with robustness evaluation.

Ayush Tewari, Justus Thies, Ben Mildenhall, Pratul Srinivasan, Edgar Tretschk, Wang Yifan, Christoph Lassner, Vincent Sitzmann, Ricardo Martin-Brualla, Stephen Lombardi, et al. 2022. Advances in neural rendering. In *Computer Graphics Forum*, volume 41, pages 703–735. Wiley Online Library.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612.

Grant Wardle and Teo Susnjak. 2024. Image first or text first? optimising the sequencing of modalities in large language model prompting and reasoning tasks.

Michael L. Waskom. 2021. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021.

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

An Yang, Baosong Yang, Binyuan Hui, et al. 2024. Qwen2 technical report.

Seonghyeon Ye, Doyoung Kim, Sungdong Kim, Hyeonbin Hwang, Seungone Kim, Yongrae Jo, James Thorne, Juho Kim, and Minjoon Seo. 2024. FLASK: Fine-grained language model evaluation based on alignment skill sets. In *The Twelfth International Conference on Learning Representations*.

Li Yujian and Liu Bo. 2007. A normalized levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1091–1095.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. 2023. Sigmoid loss for language image pre-training.

Fengji Zhang, Linquan Wu, Huiyu Bai, Guancheng Lin, Xiao Li, Xiao Yu, Yue Wang, Bei Chen, and Jacky Keung. 2024. Humaneval-v: Evaluating visual understanding and reasoning abilities of large multimodal models through coding tasks. *arXiv preprint arXiv:2410.12381*.

A Prompt Templates

Default Prompt

You are a helpful assistant that can generate Python code using matplotlib and seaborn. Generate the code to create a plot that exactly matches the given image. The generated code should be surrounded by ```python and ```

Prompt for the VLM-as-a-judge

Your job is to compare two plots and rate their similarity according to some guidelines. The guidelines are as follows:

Guidelines for annotating charts for their similarity to a reference (‘ground truth’) chart.

Annotations are on a scale of 1-4, and n/a if the criterion is not applicable.
1: incorrect
2: partially correct
3: mostly correct
4: correct

In general, choose incorrect if the criterion is not fulfilled at all, partially correct if the criterion is <= 50% fulfilled, mostly correct if it is >50% fulfilled and correct if it is completely fulfilled.

See below for the criteria and examples.

Choose n/a for the questions P1 if the given element is not present in the ground truth chart. The questions in P2, P3, and F1 should only be rated as n/a in exceptional circumstances, for example if there is no data to rate in the ground truth plot.

P1: Consistency Does the chart use the same visual elements as the ground truth? Rate questions in this category irrespective of the placement of the element (placement is rated in “P2: Arrangement”). Focus on the aspects pertaining to the content of the element, for example text of the title, number of ticks and tick values of the axis. Do not rate the style (style is rated in “P3: Style”), unless it plays a major role in the content of the element, for example if all-black lines instead of colored lines make the elements of the legend indistinguishable.

1. Chart Type: Does the chart use the same chart type as the ground truth?
- Choose “mostly correct” for example if the data is present in a scatter plot but an overlaid line chart is missing.
- Choose "partially correct" for example if 1/2 subplots use the correct chart type.
2. Axes: Does the chart use the same axes as the ground truth?
- Some charts (like pie charts) typically don’t have axes, choose "n/a" in this case.
- Choose “mostly correct” if e.g. x and y axis are present but both the tick labels are incorrect.
- Choose "partially correct" for example if the y axis is missing but present in the ground truth.
3. Title: Does the chart use the same title as the ground truth?
- Choose “mostly correct” for example if the title is “Count of customers per month” when the true title is “Monthly count of customers”.
- Choose "partially correct" for example if the title is “Customers” when the true title is “Monthly count of customers”
4. Legend: Does the chart use the same legend as the ground truth?
- Choose “mostly correct” for example if the legend has 1/4 items missing.
- Choose "partially correct" for example if the legend has 1/2 items missing.
5. Supp. Elements: Does the chart use the same supplementary elements as the ground truth? Supplementary elements are all text and visual elements in the chart which are not in the other categories (including subtitles, annotations, markers, . . .)
- Choose “mostly correct” for example if a subtitle with the same semantics (but different wording) is present.
- Choose “mostly correct” for example if 1/3 scatter plot categories use a different marker.
- Choose "partially correct" for example if a subtitle is present but an annotation containing text and an arrow is missing.
- Choose "partially correct" for example if the connection between points in a scatter/line plot is missing.

P2: Arrangement The question in this category rates whether the placement of the visual elements is consistent with the ground truth. It is intended to measure in aggregate how many of the visual elements from P1 are placed correctly.

1. Does the chart place the visual elements in the same arrangement as the ground truth?
- Make sure to rate the data separately. For example, if a line is below another although it shouldn’t be because of incorrect data, this belongs in the Faithfulness category, not arrangement.
- Choose “mostly correct” for example if the bars in a stacked bar chart are next to each other instead of on top of each other (everything else being correct), or the legend is placed in the wrong corner.
- Choose "partially correct" for reversals such as two subplots being in the wrong order, or bars being sorted ascending instead of descending (or vice versa).
- Choose “no” for example if bars are arranged randomly instead of ascending/descending.

P3: Style The question in this category rates the stylization of the plot. This includes styling of particular elements (for example a border around the legend), but does not involve the positioning or contents. Importantly, this category is not about the data content. Judge style independently of the data! For example, if a line is wrongly positioned, but the color is correct, then the style is correct. The same holds true e.g. for heatmaps, where the *color palette* pertains to the style, but the *data* itself is not part of the style.

1. Does the chart use the same color palette, font, fills and decorations as the ground truth?
- Weigh text by font size * number of characters, for example if a large title is in the wrong font (but the rest is correct), choose "partially correct" instead of “mostly correct”.
- Choose “mostly correct” for example if 1/3 lines have the wrong color, the shade of red is wrong (but the color is correct), or borders are missing.
- Choose "partially correct" for example if the backdrop, axes and title use the same colors but the data is shown in different colors.

F1: Faithfulness The question in this category measures the faithfulness of the underlying data, irrespective of its placement and visuals.

1. Is the information contained in the chart accurate?
- Choose “correct” if the data contained in the chart looks correct, up to some small numerical errors, and there is no additional information not present in the ground truth.
- Choose “mostly correct” if the data shows the same trend as in the ground truth (for example line A below line B except at point C), with similar, but not exactly the same absolute values.
- Choose "partially correct" if the data exhibits clear similarities, but is fundamentally different (for example cosine instead of sine).
- Choose “incorrect” if the data bears little resemblance to the ground truth.

{{true_image}}

Above was the ground truth plot. Below is the plot to compare against:

{{predicted_image}}

Above is the chart to compare against. First, describe the contents of each chart in detail, with particular attention to the elements listed in the guidelines. Then, break down the differences and similarities between the two charts. Only afterwards is it time to rate the chart. State your final justification for each category (before the score), then the score, and output your verdict as JSON contained in a ```json markdown code block with the keys "chart_type", "axes", "title", "legend", "supp_elements", "arrangement", "style" and "faithfulness" and the rating "incorrect", "partially correct", "mostly correct", "correct" or "n/a".

Zero-Shot CoT Prompt

You are a helpful assistant that can generate Python code using matplotlib and seaborn. Generate the code to create a plot that exactly matches the given image. The generated code should be surrounded by ```python and ```. Before writing the code, ensure you create a detailed plan containing the steps to produce the given chart. Then, execute these steps by writing the code.

Iterative Refinement Prompt

You are a helpful assistant that can generate Python code using matplotlib and seaborn. Generate the code to create a plot that exactly matches the given image. The generated code should be surrounded by ```python and ```.

{{true_image}}

As helpful additional information, here is the code of a previous try: {{previous_predicted_code}}

If previous try produced a chart:

The code of the previous try generated the following image. You can adapt the previous code to fix its mismatches.

{{previous_predicted_image}}

If previous try produced an error:

The code of the previous try did not execute successfully. It executed with the following error: {{previous_prediction_error}}
You can adapt the previous code to fix its error.

Synthetic Code Generation Prompt

{{reference_chart}}

Your task it to write matplotlib code to recreate the above chart. You are given a hint, which is an incomplete fragment of the code rendering the chart:

{{reference_code}}

You will do this task in multiple steps:

First, describe what the chart is about, what are the subplots and major elements of the chart.

Second, transcribe the chart title, legend, axis labels, etc.

Third, you will define the data in one of three ways:
(a) If the data is not prohibitively large (say less than 20 points for each series) or well described by a mathematical function then define the data ahead of time exactly (for example as numpy arrays).

(b) Otherwise, if the data looks like a random distribution define randomized data instead (for example using np.random) and use that to generate the plot. (c) Otherwise, define smaller dummy data instead and then use it to generate the plot. Your data should preserve the general look and conclusions of the plot. Use this only as a last resort if (a) and (b) are not possible.

Write which way are you choosing and why. When writing down the data add a comment after each array describing its length.

Make sure to report the mode you have chosen via either (a), (b) or (c). Finally output the matplotlib code rendering the chart. Make sure to include the labels, and only use matplotlib or seaborn. Do not use any other plotting library like plotly.

B PaliGemma Training Hyperparameters

Parameter	Value
Learning Rate	3e-5
Resolution	448 x 448
Batch Size	256
Max. Global Gradient Norm	1.0
Weight Decay	0.0
Num. Epochs	2

Table 7: Hyperparameters used for training the PaliGemma-3B-IR model.

Hyperparameters used for training the PaliGemma-3B-IR model are reported in Table 7.

C Human Rating Details

The instructions given to the human annotators for fine-grained evaluation were equivalent to the VLM-as-a-judge instructions (i.e., “Prompt for the VLM-as-a-judge” in Appendix A). The nine expert annotators were recruited from the wider research group and worked pro bono under the clear understanding that their score annotations would be used for research purposes. Given the nature of the task, an ethics review was deemed unnecessary.

D Example Predictions and Ratings

Fine-grained ratings and predictions on the easy and hard CoDePlot splits are shown in Figure 8 and Figure 9, respectively.

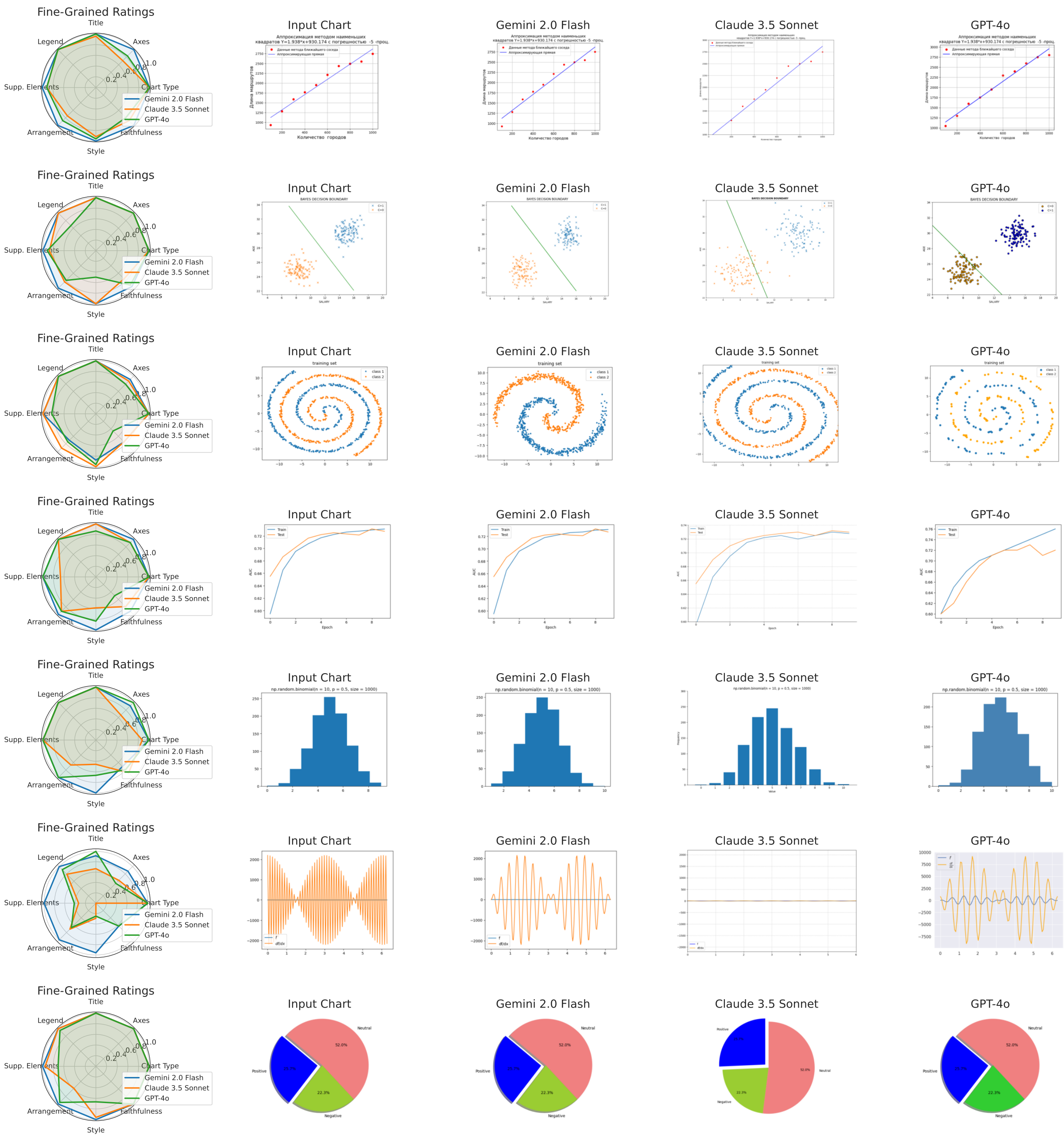


Figure 8: Examples of fine-grained ratings and predictions on the easy CoDePlot split.

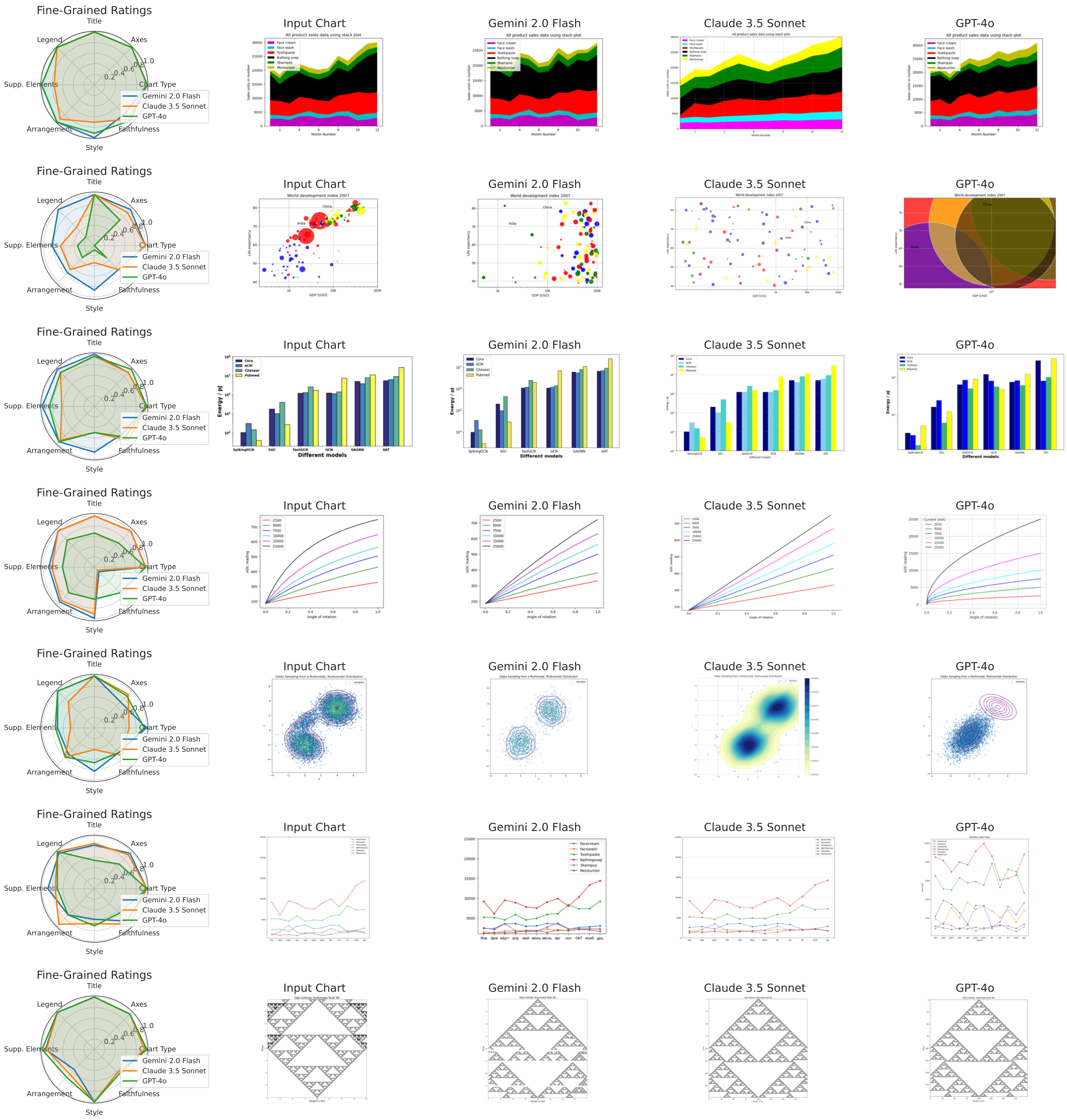


Figure 9: Examples of fine-grained ratings and predictions on the hard CoDePlot split.