Does It Run and Is That *Enough*? Revisiting Text-to-Chart Generation with a Multi-Agent Approach

James Ford and **Anthony Rios**
The University of Texas at San Antonio
{james.ford, anthony.rios}@utsa.edu

Abstract

Large language models can translate natural-language chart descriptions into runnable code, yet approximately 15% of the generated scripts still fail to execute, even after supervised fine-tuning and reinforcement learning. We investigate whether this persistent error rate stems from model limitations or from reliance on a single-prompt design. To explore this, we propose a lightweight multi-agent pipeline that separates drafting, execution, repair, and judgment, using only an off-the-shelf GPT-4o-mini model. On the TEXT2CHART31 benchmark, our system reduces execution errors to 4.5% within three repair iterations, outperforming the strongest fine-tuned baseline by nearly 5 percentage points while requiring significantly less compute. Similar performance is observed on the CHARTX benchmark, with an error rate of 4.6%, demonstrating strong generalization, and also with the VISTEXT benchmark. Under current benchmarks, execution success appears largely solved. However, manual review reveals that 6 out of 100 sampled charts contain hallucinations, and an LLM-based accessibility audit shows that only 33.3% (TEXT2CHART31) and 7.2% (CHARTX) of generated charts satisfy basic colorblindness guidelines. These findings suggest that future work should shift focus from execution reliability toward improving chart aesthetics, semantic fidelity, and accessibility.

1 Introduction

Natural language chart generation turns a request such as *plot quarterly revenue by region* into runnable code and a complete figure. A reliable solution would let non-programmers explore data, shorten the loops of professional analysts, and (potentially) create alternative renderings for low-vision users that are more accessible to screen readers. Two conditions decide whether the community can call the task solved: the generated script must execute without error, and the visual output must match the description.

One public benchmark has guided much of the recent progress. The dataset TEXT2CHART31 (Pesaran Zadeh et al., 2024) measures exact failure rates across thirty-one plot types along with various code similarity metrics. Zero-shot prompts for large language models (LLMs) initially failed on about forty percent of inputs. Supervised fine-tuning and reinforcement learning reduced that number below fifteen percent and brought pairwise plots close to six percent. Prompt engineering improved results a little more (Zhang et al., 2024b; Koh et al., 2025; Hu et al., 2024; Li et al., 2024b). Closed-source models such as GPT-4-turbo and Claude3Opus achieved similar levels, yet stubborn errors remained for complex surfaces and volumetric plots. Work on chart captioning and visual question answering (Ford et al., 2025; Ye et al., 2024; Han et al., 2023; Kim et al., 2025; Wang et al., 2024b) shows that even a small execution failure can break an entire downstream pipeline.

Instead of one large prompt, agent-based systems use several small prompts that plan, call tools, and verify interim results (Ji and Wang, 2025; Zhao et al., 2024). Code-repair agents update faulty scripts after runtime errors and often succeed within a few steps (Fan et al., 2023; de Fitero-Dominguez et al., 2024; Bouzenia et al., 2024; Wang et al., 2024a). This study asks two research questions. First, whether chart generation, defined in its present benchmark setting, can be improved using a simple agent-based framework. If the answer is yes, the field would need to revisit its evaluation criteria, shift resources away from training larger models for pure execution, and retire the current benchmarks in favour of more difficult scenarios such as noisy data or multi-step analytic sessions.

Second, if using traditional metrics and our approach effectively "solves", what objectives should

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 25160–25173
November 4-9, 2025 ©2025 Association for Computational Linguistics

guide new work when execution errors are already rare? For general image generation, this could extend chart evaluation work to move beyond a binary "runs or not" view. Structural similarity (SSIM) scores reflect low-level alignment. At the same time, multimodal LLM judges rate perceptual fidelity and semantic match (You et al., 2025; Wu et al., 2025; Yan et al., 2024; Goswami et al., 2025). Furthermore, a focus on aesthetics, readability, and accessibility could produce charts that follow color-contrast guidelines for color-blind users, place legends to avoid overlap, and include alt-text or tactile representations for screen readers and embossing devices. Such qualities matter for journalism, education, and government dashboards, where the chart must inform a wide audience rather than only run without crashing.

To answer these questions, we developed a lightweight multi-agent pipeline that uses an off-the-shelf GPT-4o-mini model without any additional training. The pipeline was also tested with three open-source models (Llama 3.1, Gemma 3, and Qwen 3). A DRAFT agent generates Python/Matplotlib code from natural-language chart descriptions, and a REPAIR agent iteratively debugs and rewrites the code—up to three times—when execution fails. On both TEXT2CHART31 and CHARTX, the system reduces execution errors by nearly 5 absolute percentage points, outperforming fine-tuned baselines while preserving image quality as measured by SSIM and multimodal LLM judgment, and similar patterns were evident with VISTEXT. A manual analysis of 100 sampled outputs found that 83% were accurate, with most remaining issues related to minor stylistic differences rather than semantic or data errors. These findings suggest that agentic pipelines are not only more robust than single-shot prompting but also capable of producing high-quality, visually faithful charts.

Overall, we make the following contributions.

- We introduce a multi-agent pipeline that achieves state-of-the-art execution success on two public benchmarks. Our system reduces the overall error rate of existing models by nearly 5% with no drop in performance in visual quality.

- We provide empirical evidence that execution is largely "solved" in current benchmarks and perform a comprehensive analysis to explore the missing gaps in current evaluation that should be targeted in the future.

2 Related Work

Chart Generation and Evaluation. Large language models (LLMs) are increasingly used to automate chart generation from natural language, but major challenges persist in quality control and evaluation. Datasets such as CHARTQA, PLOTQA, CHARTLLAMA, and CHARXIV (Masry et al., 2022; Methani et al., 2020; Han et al., 2023; Wang et al., 2024b) establish benchmarks for chart reasoning, captioning, and data extraction. These corpora focus primarily on chart understanding and question answering. Ford et al. (2025) and Ye et al. (2024) also examine chart captioning and visual QA.

To address text-to-chart generation directly, TEXT2CHART31, CHARTX, and VISTEXT pair instructions with both code and image outputs, allowing complete evaluation pipelines (Pesaran Zadeh et al., 2024; Xia et al., 2025; Tang et al., 2023). Still, these datasets remain small and often require human references. Prompt engineering, CoT prompting (Podo et al., 2024a; Li et al., 2024a), and image-based analysis methods (e.g., SSIM) have been proposed to assess generated charts (You et al., 2025; Wu et al., 2025). However, hallucinated or stylistically inconsistent outputs remain common (Podo et al., 2024b; Tian et al., 2025). Moreover, recent work questions the reliability of multimodal LLM judges for chart evaluation (Mukhopadhyay et al., 2024; Masry et al., 2024). This motivates the development of task-specific evaluation protocols combining low-level visual similarity (e.g., SSIM) as well as using LLM-as-a-judge (Yan et al., 2024; Goswami et al., 2025).

Code Generation and Repair. Chart generation with LLMs relies on producing syntactically and semantically valid code. Yet generated scripts often contain runtime errors, prompting the need for automated debugging frameworks. Several works propose feedback-driven or error-trace-guided repairs (Fan et al., 2023; de Fitero-Dominguez et al., 2024; Bouzenia et al., 2024; Wang et al., 2024a). These methods fall into agentless, agentic, and retrieval-augmented categories (Puvvadi et al., 2025), but still suffer from semantic hallucination and misalignment with user intent (Low et al., 2024).

Agentic Chart Code Generation. Agentic systems introduce structured coordination among LLM components for planning, generation, and validation. Tool-augmented and multi-agent pipelines

Plot Type	Data Points
Text2Chart31	
Pairwise Chart	472
Statistical Distribution Chart	452
Gridded Chart	192
Irregularly Gridded Chart	148
3D and Volumetric Chart	159
Total	1,423
ChartX	
General Chart	500
Fine-Grained Chart	500
Specific Chart	152
Total	1,152
VisText	
Bar	874
Area	466
Line	424
Total	1,764

Table 1: Dataset statistics.

have been proposed across domains (Cheng et al., 2024; Shang et al., 2024; Shen et al., 2024b; Zhang et al., 2024a; Zong et al., 2024; Shen et al., 2024a). Open-source ecosystems such as LANGCHAIN and CREWAI (Langchain; CrewAI) provide infrastructure to implement agentic workflows with memory and tool use.

In visualization tasks, SOCRATIC CHART orchestrates chart question answering through multiple agents with access to visual and textual tools (Ji and Wang, 2025), while LIGHTVA incorporates user feedback in an iterative refinement loop (Zhao et al., 2024). These systems exemplify how verification and repair agents can boost generation quality, even without direct reference data. Our work builds on this trend, highlighting the synergy between agentic reasoning, code correctness, and reference-free evaluation.

3 Data

We use three benchmark datasets. The statistics for the datasets used in our study are shown in Table 1.

Text2Chart31. The Text2Chart31 dataset (Pesaran Zadeh et al., 2024) consists of 11,128 data points with 31 types of plots, providing a robust environment to test the benefits of agentic visualizations. The dataset contains text plot descriptions, Python code to create a charts, image files of the created charts, and (for 8,166 data points) csv data files as well as reasoning steps. The dataset is segmented into 9,705 data points in the training set and 1,423 in the test set (for the purpose of this current study, only the test set was used). The dataset was created synthetically through the use of GPT-3.5-turbo and GPT-4. Topics span nearly 50 scientific, political, economic, and popular culture subjects. The 31 chart types are distributed in five categories, including pairwise, statistical, gridded, irregularly gridded, and 3D/volumetric data.

ChartX. The second dataset is ChartX (Xia et al., 2025). This benchmark data consists of 4,848 synthetically-generated chart examples in the validation set and 1,152 in the test set (similarly, for the purpose of this current study, only the test set was used), spanning 18 chart types. A total of 22 topics are represented in the charts, with similar cultural, economic, political, and scientific themes as the first dataset. The data file contains text descriptions of the charts, Python code, the raw data points, and chart image files.

VisText. The third dataset is VisText (Tang et al., 2023). This benchmark data consists of 12,441 bar, line and area charts constructed using real-world data from the Statista.com pertaining to topics such as technology, economics, and sports. The test set was used in this study, totaling 1,764 charts. The data file consists of text descriptions of the charts including the data points and chart image files. Note that the text descriptions of the charts were created automatically rather than human-generated.

4 Methodology

Figure 1 provides a high-level overview of our paper. Our framework has three main components. First, we use a DRAFTING agent that transforms the text into code to generate the chart. Second, the code is evaluated using a Python interpreter, and any resulting errors are passed to a RE-WRITER agent; otherwise, if there is an error, the final chart is returned. If there was an error, the RE-WRITER agent will try to fix the error. This process is repeated until the code runs or a maximum number of iterations is hit. We describe each part in detail below.

Step 1: DRAFTING Agent. First, we generated baseline Python code from the provided text descriptions in the dataset using the closed-source GPT4o-mini LLM. We conducted both zero-shot and few-shot runs for the baseline. In the zero-shot setting, the LLMs receive only the task instructions and the data without examples. The prompt fol-

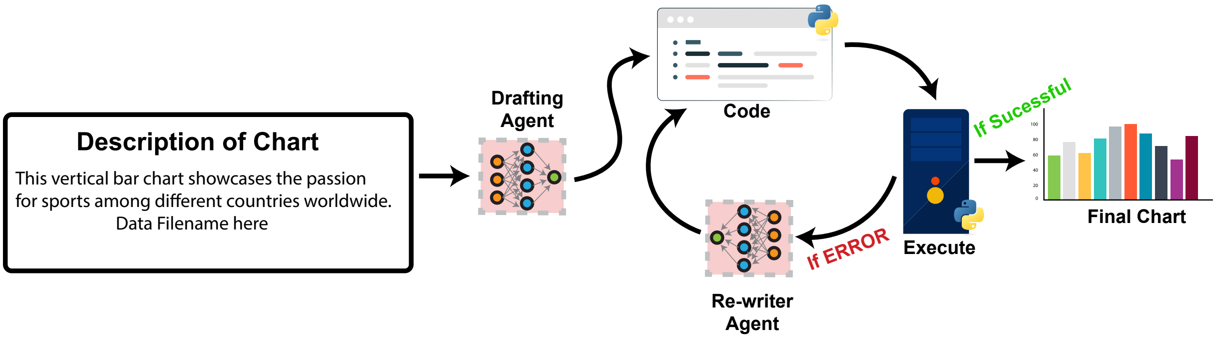


Figure 1: **Multi-agent text-to-chart pipeline**. A natural-language description is first passed to the *Drafting Agent*, which synthesises runnable Python/Matplotlib code. The script is executed; if a runtime exception occurs, the full traceback and source are forwarded to the *Re-writer Agent*, which edits the code and resubmits it. This repair loop repeats (up to three iterations in our experiments) until the code runs successfully, producing the final chart with no human intervention.

lows the format from the Text2Chart31 paper (Pesaran Zadeh et al., 2024). Specifically, the system prompt instructs the model to generate Python code for data visualization as follows:

System Prompt

You are good at generating complete python code from the given chart description.

Next, the user instruction prompt then provides the specific chart details

User Instructions

Your task is to generate a complete Python code for the given description. Make sure to include all necessary libraries.
`Description: Description_text`
Please generate the corresponding code that generates the plot that has the above description. `Code:`

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

where `Description_text` is the provided text description which is passed to the model. An example of the text description is as follows:

> This vertical bar chart showcases the passion for sports among different countries worldwide. The chart represents the percentage of individuals in each country who identify themselves as avid sports fans. The data provides insights into the level of sports fanaticism across countries, offering a comparative analysis.
>
> The X-axis denotes the countries included in the dataset, labeled as 'Country', while the Y-axis signifies the percentage of individuals who consider themselves passionate sports fans, denoted as 'Percentage of Sports Fanatics'.
>
> The data for this graph is stored in a CSV file named 'sports_fanatics.csv', which includes five columns: 'Country', 'Percentage of Sports Fanatics'. Here are the first six rows of the dataset:
>
> Country,Percentage of Sports Fanatics
> United States,45
> Germany,35
> ...

This prompt directs the LLM to generate the appropriate Python code to create the chart per the specifications. The system prompt is mentioned once in the few-shot setting, and two in-context examples are provided. These examples are supplied in pairs, containing (1) the user instruction with the chart description and data file and (2) the code associated with the data files to generate the chart.

Step 2: RE-WRITER Agent. Second, the code generated from Step 1 is passed to a Python interpreter. If the code runs, the final chart is generated. However, if the code results in an error, the error and original code are passed to the RE-WRITER Agent. Specifically, for the instances where execution failed, the supervising agent sends the code and the error message to the re-writer agent to correct and re-execute the code. The process is repeated for a maximum of three iterations. The system prompt is shown below (full prompt in the Appendix):

	Error Ratio					Code Similarity	
Model	Pairwise	Statistical distribution	(Irregularly) gridded	3D and Volumetric	Total	METEOR	CodeBLEU
CLI-7B	22.67	29.42	77.94	52.20	41.32	0.485	0.402
L3I-8B	20.76	28.98	66.76	34.59	35.91	0.519	0.437
SFT: L3I-8B	19.07	13.27	13.53	20.75	16.09	0.562	0.464
SFT+RLpref: L3I-8B	13.14	11.50	15.00	26.42	14.55	0.567	0.461
CLI-13B	18.86	29.42	71.76	57.23	39.14	0.489	0.413
StarCoder-15.5B	23.31	32.08	51.18	25.16	32.89	0.347	0.328
InstructCodeGen-16B	38.56	45.13	62.94	40.25	46.66	0.388	0.330
SFT: CLI-13B	6.36	6.19	12.06	22.64	9.49	**0.581**	**0.481**
SFT+RLpref: CLI-13B	6.36	5.53	12.35	21.38	9.21	0.566	0.467
GPT-3.5-turbo	11.02	13.50	28.82	19.59	18.62	0.524	0.453
GPT-4-0613	13.56	11.06	28.53	39.62	19.26	0.535	0.441
GPT-4-turbo	11.02	14.16	11.76	29.56	14.27	0.540	0.448
GPT-4o	13.98	6.86	13.53	26.42	13.00	0.552	0.450
Claude3Opus	7.84	7.74	30.59	23.27	14.90	0.515	0.435
ZS GPT 4o-mini Baseline	13.35	5.97	18.53	35.85	14.76	0.542	0.407
ZS GPT 4o-mini Agentic	2.54	5.09	9.41	18.24	6.75	0.510	0.406
FS GPT 4o-mini Baseline	14.62	8.85	12.45	31.45	14.13	0.557	0.433
FS GPT 4o-mini Agentic	1.48	**3.76**	**5.59**	13.21	**4.50**	0.532	0.447
FS Llama 3.1 Baseline	18.64	22.35	52.35	33.33	29.52	0.510	0.404
FS Llama 3.1 Agentic	4.03	7.08	30.29	**8.18**	11.74	0.449	0.401
FS Gemma 3 Baseline	16.95	28.54	59.71	64.78	36.19	0.535	0.440
FS Gemma 3 Agentic	**1.06**	5.54	26.76	23.90	11.18	0.467	0.443
FS Qwen 3 Baseline	22.98	5.31	12.06	22.64	14.69	0.556	0.445
FS Qwen 3 Agentic	10.59	5.53	66.76	11.95	8.22	0.530	0.421

Table 2: Performance of baseline, fine-tuned (SFT), and preference-optimized (SFT + RLpref) models on the TEXT2CHART-31 benchmark. Columns 2–6 report error ratios (↓) for Pairwise, Statistical-Distribution, Irregularly-Gridded, 3D/Volumetric, and Overall errors; lower values indicate more accurate chart generation. Columns 7–8 give code-similarity scores (↑) using METEOR and CODEBLEU, where higher is better. "ZS" denotes zero-shot models (all closed-source LLMs are ZS if not noted as FS), "FS" few-shot models, and "Agentic" indicates our agent-based prompting strategy. Best results for each metric are **bolded**.

System Prompt

You are an expert Python code rewriter. Your task is to rewrite Python code based strictly on the user's suggestions.
- DO NOT modify any part of the code that is not explicitly mentioned in the suggestion.
- Ensure that the rewritten code is functional, error-free, and adheres to Python syntax rules (e.g., indentation, brackets, braces).
- Return ONLY the complete revised code without explanations, comments, or Markdown formatting.
- Follow instructions EXACTLY as provided.

5 Results

Experimental Details. In addition to the closed-source GPT-4o-mini model, three open-source models executed the agentic pipeline. These included Llama-3.1-8B-Instruct, gemma-3-12b-it, and Qwen3-32B. All experiments are implemented using the LangChain framework (Langchain) with the exception of the Qwen 3 model which utilized direct Python functions to call the agents. Overall, our experiments cost ∼$150, including all experiments that did and did not work.

Evaluation Metrics. For our main metrics, we use code similarity and execution error rates. Execution error rates is calculated as the proportion of code snippets that do not run in the Python interpreter. Code similarity metrics are calculated with METEOR (Banerjee and Lavie, 2005) and CodeBLEU (Ren et al., 2020). Finally, image similarity analysis is conducted, comparing the generated charts with the provided ground truth chart images from the two datasets to gauge the accuracy of the data visualizations created by the agent. Image similarity was determined with both the Structural Similarity Index Measure (SSIM) as well as multimodal LLM as a judge (Yan et al., 2024; Goswami

Model	Error Rate
ZS GPT 4o-mini Baseline	11.11
ZS GPT 4o-mini Agentic	**3.13**
FS GPT 4o-mini Baseline	9.38
FS GPT 4o-mini Agentic	4.60
FS Llama 3.1 Baseline	24.38
FS Llama 3.1 Agentic	6.94
FS Gemma 3 Baseline	23.00
FS Gemma 3 Agentic	5.3
FS Qwen 3 Baseline	9.38
FS Qwen 3 Agentic	7.64

Table 3: Overall chart generation error rate (↓) on the CHARTX benchmark. "ZS" denotes zero-shot models, "FS" few-shot models, and "Agentic" indicates our agent-based prompting strategy. Lower values mean fewer generation errors; the best score is **bolded**.

Model	Error Rate
ZS GPT 4o-mini Baseline	4.15
ZS GPT 4o-mini Agentic	0.85
FS GPT 4o-mini Baseline	3.91
FS GPT 4o-mini Agentic	1.02
FS Llama 3.1 Baseline	14.73
FS Llama 3.1 Agentic	1.47
FS Gemma 3 Baseline	15.67
FS Gemma 3 Agentic	1.59
FS Qwen 3 Baseline	2.61
FS Qwen 3 Agentic	**0.79**

Table 4: Overall chart generation error rate (↓) on the VISTEXT benchmark. "ZS" denotes zero-shot models, "FS" few-shot models, and "Agentic" indicates our agent-based prompting strategy. Lower values mean fewer generation errors; the best score is **bolded**.

et al., 2025). SSIM measures structural differences between two images, ranging from 0 to 1, with 1 being perfect similarity. For the MM-LLM LLM-as-a-judge analysis, in-context (few-shot) prompting was used to compare the generated charts with the ground truth, with GPT4o-mini generating a perceptual quality score ranging from 0 to 100 for each chart when compared to the original (Wu et al., 2025). The appendix details the prompts used in the MM-LMM judge procedure. The SSIM and perceptual quality scores were averaged for the baseline and agentic process results for all generated images. Thus, the agentic calculations have more data points than baseline calculations.

Text2Chart31 Results. Table 2 presents the results of our agentic process compared to the models published in the Text2Chart31 paper (Pesaran Zadeh et al., 2024). For the comparison, the models labeled SFT and SFT+RL are the supervised fine-tuning and reinforcement learning versions from the benchmark study. Our baseline results are similar to other closed-source models such as the GPT offerings and Claude3Opus, as well as the SFT and SFT+RL 8B models. However, our agentic runs showed major decreases in error rates in executing the code, dropping by more than half to two-thirds over the baseline and surpassing the results of the previously published models by far. The agentic few-shot run had an error rate of 4.5 percent, less than half of the SFT+RL 13B model at 9.2 percent for total error. The agentic models also displayed strong performance across all chart types, with the weakest result being 13.2 percent error for 3D and Volumetric charts. Pairwise plots had an error rate of only 1.5 percent. While the open-source models generally did not match the performance of GPT 4o-mini, the agentic pipelines did display similar gains over the baseline models.

However, the metrics involving code similarity did not show similar improvements. While comparisons to the published results might suffer from inconsistent alignment with their methodologies, the fact that our agentic scores are no better than the baseline indicates that code similarity might be of lesser importance, given the decreased error rate, especially with the image quality findings discussed below. More specifically, code similarity is not as important if the code does not run. Yet, we still show better performance than stronger models (GPT-4o) for METEOR. We do note that prior research results indicate that code similarity may not be a good indicator of visualization quality (Chen et al., 2024). Furthermore, the agentic process lends itself to creative GenAI problem-solving, which, in turn, produces deviations from the original code.

ChartX Results. Table 3 lists the results using the ChartX dataset. Note that the ChartX dataset was not originally used for text-to-chart generation (Xia et al., 2025), hence we only compare to our baselines. As evidenced in the first dataset performance, the agentic methods greatly improve the successful execution of the generated code to produce the charts. Error rates decline by approximately half to two-thirds. Interestingly, while the few-shot approach improved the error rate for the baseline as expected, the result was reversed once the agentic process corrected the code. Also interesting is the

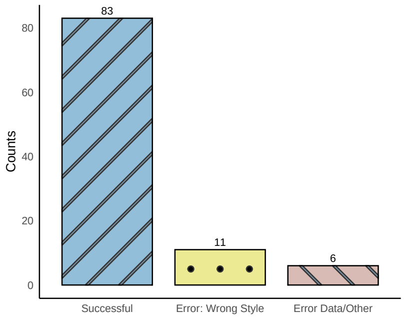


Figure 2: Iteration Human Study Review (n=100)

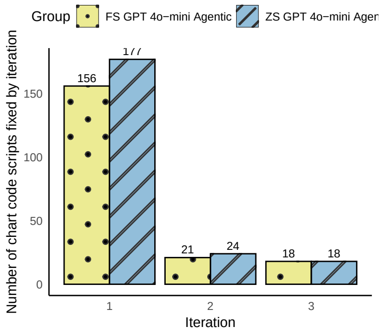


Figure 3: Iteration Analysis for Text2Chart31.

fact that the final zero-shot agentic results were almost identical between both the Text2Chart31 and this ChartX dataset at 4.5 and 4.6 percent for GPT 4o-mini, respectively.

VisText Results. Table 4 lists the results using the VisText dataset. Note that the VisText dataset was also not originally used for text-to-chart generation (Tang et al., 2023), hence we only compare to our baselines as well. Again, the agentic pipeline achieves much lower error rates than baseline.

Ablation and Discussion. Overall, we perform a comprehensive ablation and error analysis to understand the current state of text-to-chart generation using an agent-based process. Specifically, we evaluate the impact of the number of iterations, how well accessible the charts are (about colorblindness), the general image quality compared to the ground-truth results, and a manual error analysis.

Iteration Ablation. To evaluate the impact of iterative repair, we conducted ablation studies examining how many chart scripts were successfully fixed at each step of the agentic pipeline. Figures 3 and 4 show the number of scripts corrected during the first, second, and third iterations for the Text2Chart31 and ChartX datasets, respectively. As expected, the few-shot (FS) GPT-4o-mini agent started with fewer initial code failures than the zero-shot (ZS) variant. Nevertheless, both settings benefited from additional repair attempts, particularly in the first two iterations. The largest gains occurred during the first pass, with diminishing but meaningful returns in subsequent iterations. These results highlight the value of multi-step reasoning and self-correction within agentic systems.

Model	SSIM	MM-LLM as a Judge
	Text2Chart31	
GPT 4o-mini Agentic	0.670	67.4
GPT 4o-mini Baseline	0.672	66.7
	ChartX	
GPT 4o-mini Agentic	0.722	76.0
GPT 4o-mini Baseline	0.722	76.4
	VisText	
GPT 4o-mini Agentic	0.570	79.37
GPT 4o-mini Baseline	0.569	79.31

Table 5: Image Quality Analysis results on FS models.

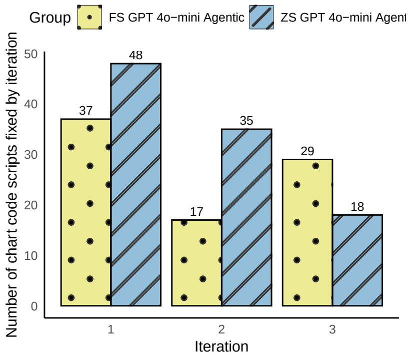
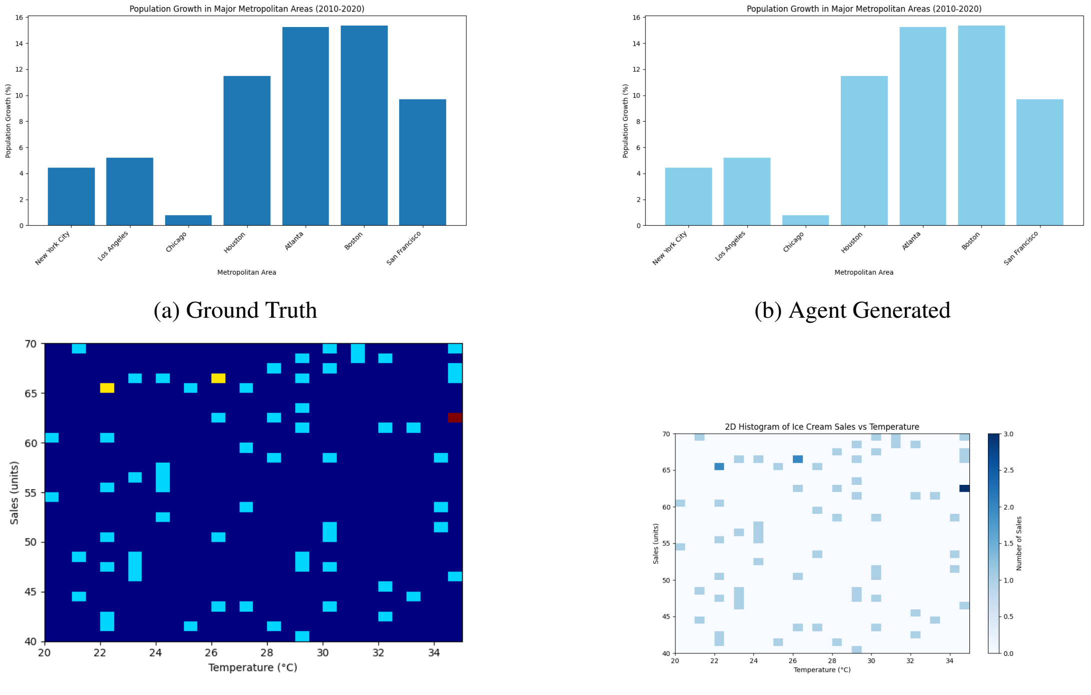


Figure 4: Iteration Analysis for the ChartX dataset

Color Blindness. Overall, we find that modern evaluation frameworks do not measure performance across many attributes needed to understand chart quality. Hence, we use an LLM-as-a-judge to evaluate how many generated charts pass standard colorblindness criteria (see the Appendix for the criteria and prompt). We manually evaluated the quality of the annotations on a random sample of 50 charts, and 46 were correctly labeled. For the Text2Chart31 dataset, we find that only 33.3% were appropriate for color blindness. For the ChartX dataset, only 7.2% of the generated charts were appropriate. This suggests that even though we can generate charts with a minimal error rate, substantial research is needed to make accessible charts.

Image Quality Analysis. Table 5 presents image quality scores for the agentic and baseline models across the three datasets. Structural Similarity In-



(a) Ground Truth

(b) Agent Generated

(c) Ground Truth

(d) Agent Generated

Figure 5: Examples of visual similarity and variation in generated charts compared to ground truth. (a) and (b) show a bar chart comparing metropolitan population growth, with the agent-generated chart (b) closely matching the structure and values of the ground truth (a). (c) and (d) depict a 2D histogram of ice cream sales vs. temperature, where the agent-generated chart (d) captures the overall distribution but varies in visual encoding and bin density relative to the ground truth (c).

Model/Error Message	**Count**
Text2Chart31	
stem() got an unexpected keyword argument "use_line_collection"	49
Argument Z must be 2-dimensional	19
name "np" is not defined	17
ChartX	
No module named "mplfinance"	39
No module named "squarify"	25
All arrays must be of the same length	18
VisText	
All arrays must be of the same length	56
unconverted data remains when parsing with format "%Y-%m"	7
time data doesn't match format "%B %d, %Y"	2

Table 6: Top 3 common Python interpreter errors for Iteration 1 on each dataset with GPT 4o-mini

dex Measure (SSIM) values are nearly identical between the agentic and baseline models, indicating that the improved execution rates seen in Tables 3 and 2 do not come at the cost of chart quality. In other words, the agent generates more executable charts while maintaining visual similarity to the ground truth.

Similarly, perceptual judgments by a multimodal LLM (MM-LLM) show comparable scores across methods (ranging from 67 to 76 on a 100-point scale), further confirming that the agentic process does not degrade the perceived quality of the charts while resulting in much better execution performance. We do note that the prompts provided to both models contained only high-level descriptions (e.g., chart type and data values), without detailed formatting instructions, so some stylistic deviation from ground truth is expected.

Figure 5 illustrates close matches and common variations in chart outputs. Panels (a) and (b) depict a bar chart with minimal deviation: the agent-generated chart closely mirrors the structure and data of the ground truth, differing only in bar shading. In contrast, panels (c) and (d) show a 2D histogram with more noticeable differences. While the agent chart preserves the overall distribution, it employs a different colormap and includes a legend and title not present in the ground truth. These examples demonstrate that even when stylistic variations occur, the generated visualizations are plausible, faithful to the underlying data, and often enhanced with additional chart elements

Error Analysis. To better understand the nature of remaining errors, we manually reviewed 100 randomly sampled charts generated by our few-shot agentic system. We provide a summary of the analysis in Figure 2. Each chart was labeled as either *Successful* or assigned one of two error categories: *Wrong Style* (e.g., incorrect chart type, missing stylistic elements) or *Error Data/Other* (e.g., incorrect values, axis misalignment, or malformed outputs not caught by execution checks). As shown in Figure 2, 83% of the charts were deemed fully successful. Among the 17 failures, 11 involved stylistic mismatches that still plausibly conveyed the intended message (e.g., a line chart instead of a bar chart). At the same time, only 6 were due to data-related or structural errors. This analysis suggests that while execution correctness has largely been addressed, future improvements should focus on semantic and stylistic fidelity.

To better understand the kinds of issues the agent addresses, we analyzed the most frequent runtime errors encountered during the first iteration of the repair process. As shown in Table 6, the top errors for the Text2Chart31 dataset included incorrect keyword arguments (e.g., unsupported parameters like use_line_collection), shape mismatches (e.g., expecting a 2-dimensional input), and missing or incorrectly referenced libraries (e.g., undefined variables like np). For ChartX, the most common errors involved missing third-party libraries (mplfinance, squarify) and array length mismatches. Similarly, the leading errors for VisText included array lengths and incorrect data formats. The agent typically responds to the first error surfaced by the Python runtime, treating the repair process as a sequential correction task. As a result, deeper or secondary issues are often uncovered in subsequent iterations, supporting the design choice of a multi-round repair loop.

Implications. Our findings suggest that text-to-chart generation, as currently defined by benchmarks like TEXT2CHART31, CHARTX, and VISTEXT, is approaching a performance ceiling concerning execution correctness. By introducing a lightweight multi-agent framework that separates drafting, execution, and iterative repair, we reduced error rates by over 5 percentage points without relying on supervised fine-tuning or reinforcement learning. This indicates that when paired with execution-aware self-correction, structured prompting alone can achieve or exceed the performance of more computationally expensive methods.

However, execution success alone does not guarantee semantic accuracy, visual clarity, or accessibility. While our system maintains visual quality (via SSIM and multimodal LLM judgment), manual analysis shows that some outputs still deviate stylistically or semantically from the intended chart. Additionally, only 7-33% of generated charts meet basic color accessibility standards, underscoring a critical gap in current evaluation protocols.

These results call for reorienting future work in this space, from reducing execution failure to enhancing chart readability, aesthetics, and inclusivity. Benchmarks must evolve to reflect real-world use cases where visual clarity and usability are as important as syntactic correctness. Agentic systems, by enabling structured reasoning and self-verification, offer a promising foundation for tackling these higher-level challenges.

6 Conclusion

We presented a lightweight agentic framework that significantly improves text-to-chart generation by reducing execution errors while preserving image quality, all without fine-tuning or reinforcement learning. Our analysis shows that single-prompt LLMs (whether zero- or few-shot) still fail frequently, whereas multi-agent repair loops offer more robust performance at low cost.

Beyond execution, we highlight the need to shift focus toward semantic fidelity, visual clarity, and accessibility. Paired with image quality analysis, agent-based approaches can help address these broader challenges. Future work should explore how these methods can enhance real-world applications, especially for users with visual or cognitive impairments (Gorniak et al., 2024; Seo et al., 2024).

Acknowledgments

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. 2145357.

Limitations

While our proposed framework demonstrates strong performance in generating LLM-based data visualizations using a multi-agent approach, several limitations remain. First, our evaluation focuses on three benchmark datasets, Text2Chart31, ChartX, and VisText. The first two datasets are synthetically generated and, while the third uses real-world data, its chart descriptions are created automatically. Thus, these datasets may not reflect the full complexity of real-world chart generation. Future work should consider additional datasets that include human-authored chart descriptions and real-world data to better evaluate generalizability. Second, the system uses a single proprietary model and three open-source models for drafting and repair. Exploring additional open-source LLMs as alternatives could improve transparency and reproducibility. Testing a broader set of LLM providers may also reveal model-specific strengths or weaknesses. Finally, our agentic framework uses a simple two-agent setup without memory, planning, or retrieval modules. More advanced agent designs supporting longer-term reasoning or richer tool use could improve performance, especially in more complex or multi-step charting scenarios. Finally, we note that we use AI to help improve our writing of this manuscript.

References

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. 2024. Repairagent: An autonomous, llm-based agent for program repair. *Preprint*, arXiv:2403.17134.

Nan Chen, Yuge Zhang, Jiahang Xu, Kan Ren, and Yuqing Yang. 2024. Viseval: A benchmark for data visualization in the era of large language models. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–11.

Xiaoxue Cheng, Junyi Li, Xin Zhao, Hongzhi Zhang, Fuzheng Zhang, Di Zhang, Kun Gai, and Ji-Rong Wen. 2024. Small agent can also rock! empowering small language models as hallucination detector. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14600–14615, Miami, Florida, USA. Association for Computational Linguistics.

CrewAI. `https://github.com/crewaiinc/crewai`.

David de Fitero-Dominguez, Eva Garcia-Lopez, Antonio Garcia-Cabot, and Jose-Javier Martinez-Herraiz. 2024. Enhanced automated code vulnerability repair using large language models. *Engineering Applications of Artificial Intelligence*, 138:109291.

Zhiyu Fan, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. 2023. Automated repair of programs from large language models. In *Proceedings of the 45th International Conference on Software Engineering*, ICSE '23, page 1469–1481. IEEE Press.

James Ford, Xingmeng Zhao, Dan Schumacher, and Anthony Rios. 2025. Charting the future: Using chart question-answering for scalable evaluation of LLM-driven data visualizations. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 7497–7510, Abu Dhabi, UAE. Association for Computational Linguistics.

Joshua Gorniak, Yoon Kim, Donglai Wei, and Nam Wook Kim. 2024. Vizability: Enhancing chart accessibility with llm-based conversational interaction. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, UIST '24, New York, NY, USA. Association for Computing Machinery.

Kanika Goswami, Puneet Mathur, Ryan Rossi, and Franck Dernoncourt. 2025. Plotedit: Natural language-driven accessible chart editing in pdfs via multimodal llm agents. In *Advances in Information Retrieval*, pages 130–134, Cham. Springer Nature Switzerland.

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. *Preprint*, arXiv:2311.16483.

Linmei Hu, Duokang Wang, Yiming Pan, Jifan Yu, Yingxia Shao, Chong Feng, and Liqiang Nie. 2024. Novachart: A large-scale dataset towards chart understanding and generation of multimodal large language models. In *Proceedings of the 32nd ACM International Conference on Multimedia*, MM '24, page 3917–3925, New York, NY, USA. Association for Computing Machinery.

Yuyang Ji and Haohan Wang. 2025. Socratic chart: Cooperating multiple agents for robust svg chart understanding. *Preprint*, arXiv:2504.09764.

Wonjoong Kim, Sangwu Park, Yeonjun In, Seokwon Han, and Chanyoung Park. 2025. Simplot: Enhancing chart question answering by distilling essentials. *Preprint*, arXiv:2405.00021.

Woosung Koh, Jang Han Yoon, MinHyung Lee, Youngjin Song, Jaegwan Cho, Jaehyun Kang, Taehyeon Kim, Se-Young Yun, Youngjae Yu, and Bongshin Lee. 2025. c^2: Scalable auto-feedback for llm-based chart generation. *Preprint*, arXiv:2410.18652.

Langchain. `https://python.langchain.com/docs/introduction/`.

Guozheng Li, Xinyu Wang, Gerile Aodeng, Shunyuan Zheng, Yu Zhang, Chuangxin Ou, Song Wang, and Chi Harold Liu. 2024a. Visualization generation with large language models: An evaluation. *Preprint*, arXiv:2401.11255.

Xinhang Li, Jingbo Zhou, Wei Chen, Derong Xu, Tong Xu, and Enhong Chen. 2024b. Visualization recommendation with prompt-based reprogramming of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13250–13262, Bangkok, Thailand. Association for Computational Linguistics.

En Low, Carmen Cheh, and Binbin Chen. 2024. Repairing infrastructure-as-code using large language models. In *2024 IEEE Secure Development Conference (SecDev)*, pages 20–27.

Ahmed Masry, Do Xuan Long, 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, Megh Thakkar, Aayush Bajaj, Aaryaman Kartha, Enamul Hoque, and Shafiq Joty. 2024. Chartgemma: Visual instruction-tuning for chart reasoning in the wild. *Preprint*, arXiv:2407.04172.

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

Srija Mukhopadhyay, Adnan Qidwai, Aparna Garimella, Pritika Ramu, Vivek Gupta, and Dan Roth. 2024. Unraveling the truth: Do VLMs really understand charts? a deep dive into consistency and robustness. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 16696–16717, Miami, Florida, USA. Association for Computational Linguistics.

Lisa Charlotte Muth. 2020. How your colorblind and colorweak readers see your colors. `https://blog.datawrapper.de/colors-for-colorblind-readers/`. Accessed: 2025-05-19.

Fatemeh Pesaran Zadeh, Juyeon Kim, Jin-Hwa Kim, and Gunhee Kim. 2024. Text2Chart31: Instruction tuning for chart generation with automatic feedback. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11459–11480, Miami, Florida, USA. Association for Computational Linguistics.

Luca Podo, Marco Angelini, and Paola Velardi. 2024a. V-recs, a low-cost llm4vis recommender with explanations, captioning and suggestions. *Preprint*, arXiv:2406.15259.

Luca Podo, Muhammad Ishmal, and Marco Angelini. 2024b. Vi(e)va llm! a conceptual stack for evaluating and interpreting generative ai-based visualizations. *Preprint*, arXiv:2402.02167.

Meghana Puvvadi, Sai Kumar Arava, Adarsh Santoria, Sesha Sai Prasanna Chennupati, and Harsha Vardhan Puvvadi. 2025. Coding agents: A comprehensive survey of automated bug fixing systems and benchmarks. In *2025 IEEE 14th International Conference on Communication Systems and Network Technologies (CSNT)*, pages 680–686.

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. *Preprint*, arXiv:2009.10297.

JooYoung Seo, Sanchita S. Kamath, Aziz Zeidieh, Saairam Venkatesh, and Sean McCurry. 2024. Maidr meets ai: Exploring multimodal llm-based data visualization interpretation by and with blind and low-vision users. In *Proceedings of the 26th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '24, New York, NY, USA. Association for Computing Machinery.

Chuyi Shang, Amos You, Sanjay Subramanian, Trevor Darrell, and Roei Herzig. 2024. TraveLER: A modular multi-LMM agent framework for video question-answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9740–9766, Miami, Florida, USA. Association for Computational Linguistics.

Leixian Shen, Haotian Li, Yun Wang, and Huamin Qu. 2024a. From data to story: Towards automatic animated data video creation with llm-based multi-agent systems. In *2024 IEEE VIS Workshop on Data Storytelling in an Era of Generative AI (GEN4DS)*, pages 20–27.

Weizhou Shen, Chenliang Li, Hongzhan Chen, Ming Yan, Xiaojun Quan, Hehong Chen, Ji Zhang, and Fei Huang. 2024b. Small LLMs are weak tool learners: A multi-LLM agent. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16658–16680, Miami, Florida, USA. Association for Computational Linguistics.

Benny Tang, Angie Boggust, and Arvind Satyanarayan. 2023. VisText: A benchmark for semantically rich chart captioning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7268–7298, Toronto, Canada. Association for Computational Linguistics.

Yuan Tian, Weiwei Cui, Dazhen Deng, Xinjing Yi, Yurun Yang, Haidong Zhang, and Yingcai Wu. 2025. Chartgpt: Leveraging llms to generate charts from abstract natural language. *IEEE Transactions on Visualization and Computer Graphics*, 31(3):1731–1745.

Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024a. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering*, 50(4):911–936.

Zirui Wang, Mengzhou Xia, Luxi He, Howard Chen, Yitao Liu, Richard Zhu, Kaiqu Liang, Xindi Wu, Haotian Liu, Sadhika Malladi, Alexis Chevalier, Sanjeev Arora, and Danqi Chen. 2024b. Charxiv: Charting gaps in realistic chart understanding in multimodal llms. *Preprint*, arXiv:2406.18521.

Tianhe Wu, Kede Ma, Jie Liang, Yujiu Yang, and Lei Zhang. 2025. A comprehensive study of multimodal large language models for image quality assessment. In *Computer Vision – ECCV 2024*, pages 143–160, Cham. Springer Nature Switzerland.

Renqiu Xia, Bo Zhang, Hancheng Ye, Xiangchao Yan, Qi Liu, Hongbin Zhou, Zijun Chen, Peng Ye, Min Dou, Botian Shi, Junchi Yan, and Yu Qiao. 2025. Chartx chartvlm: A versatile benchmark and foundation model for complicated chart reasoning. *Preprint*, arXiv:2402.12185.

Pengyu Yan, Mahesh Bhosale, Jay Lal, Bikhyat Adhikari, and David Doermann. 2024. Chartreformer: Natural language-driven chart image editing. In *Document Analysis and Recognition - ICDAR 2024*, pages 453–469, Cham. Springer Nature Switzerland.

Yilin Ye, Jianing Hao, Yihan Hou, Zhan Wang, Shishi Xiao, Yuyu Luo, and Wei Zeng. 2024. Generative ai for visualization: State of the art and future directions. *Visual Informatics*, 8(2):43–66.

Zhiyuan You, Zheyuan Li, Jinjin Gu, Zhenfei Yin, Tianfan Xue, and Chao Dong. 2025. Depicting beyond scores: Advancing image quality assessment through multi-modal language models. In *Computer Vision – ECCV 2024*, pages 259–276, Cham. Springer Nature Switzerland.

Lu Zhang, Tiancheng Zhao, Heting Ying, Yibo Ma, and Kyusong Lee. 2024a. OmAgent: A multi-modal agent framework for complex video understanding with task divide-and-conquer. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 10031–10045, Miami, Florida, USA. Association for Computational Linguistics.

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

Yuheng Zhao, Junjie Wang, Linbin Xiang, Xiaowen Zhang, Zifei Guo, Cagatay Turkay, Yu Zhang, and Siming Chen. 2024. Lightva: Lightweight visual analytics with llm agent-based task planning and execution. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–13.

Chang Zong, Yuchen Yan, Weiming Lu, Jian Shao, Yongfeng Huang, Heng Chang, and Yueting Zhuang. 2024. Triad: A framework leveraging a multi-role LLM-based agent to solve knowledge base question answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1698–1710, Miami, Florida, USA. Association for Computational Linguistics.

A Appendix

Prompts for Baseline The prompts used to create the baseline charts are derived from the code in the Text2Chart31 dataset (Pesaran Zadeh et al., 2024). These were modified for the few-shot run by including two example descriptions and generated code from the training set from that data source.

System Prompt

You are good at generating complete python code from the given chart description.

User Instructions

Your task is to generate a complete Python code for the given description. Make sure to include all necessary libraries.
`Description: Description_text`
Please generate the corresponding code that generates the plot that has the above description. `Code:`

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

Prompts for Agentic Process The prompts used in the agentic process are presented below. The LangChain framework uses multiple tools to complete the agentic tasks, thus the prompts are embedded within each tool.

Reflection Tool

The following Python code produced an error:
`code`
`Error: error`
Identify the root cause of the error. Provide a suggestion to fix ONLY the problematic lines, explicitly specifying which parts of the original code should REMAIN UNCHANGED. Return the complete code with the suggested modifications inserted.

Rewriter Tool

system_message = You are an expert Python code rewriter. Your task is to rewrite Python code based strictly on the user's suggestions. - DO NOT modify any part of the code that is not explicitly mentioned in the suggestion. - Ensure that the rewritten code is functional, error-free, and adheres to Python syntax rules (e.g., indentation, brackets, braces). - Return ONLY the complete revised code without explanations, comments, or Markdown formatting. - Follow instructions EXACTLY as provided."""
user_message = """Rewrite the following Python code based on this suggestion:
`Original Code: code`
`Suggestion: suggestion`

Prompts for MM-LLM as a judge For the MM-LLM as a judge analysis, GPT4o-mini was prompted to rate the quality of each of the baseline and agentic charts which were generated.

MM-LLM Prompt

system_message = For two shown images, the human perceptual quality score of the first image is 50. Based on this example, assign a perceptual quality score to the second image in terms of perceptual similarity. The score must range from 0 to 100, with a higher score denoting better image quality. Return the result as a JSON object in this format: s̈core:̈ <integer>.
user_message =
"type": "image_url",
"image_url":
"url": f"data:image/png;base64,base64_image1" ,
"type": "image_url",
"image_url":
"url": f"data:image/png;base64,base64_image2"

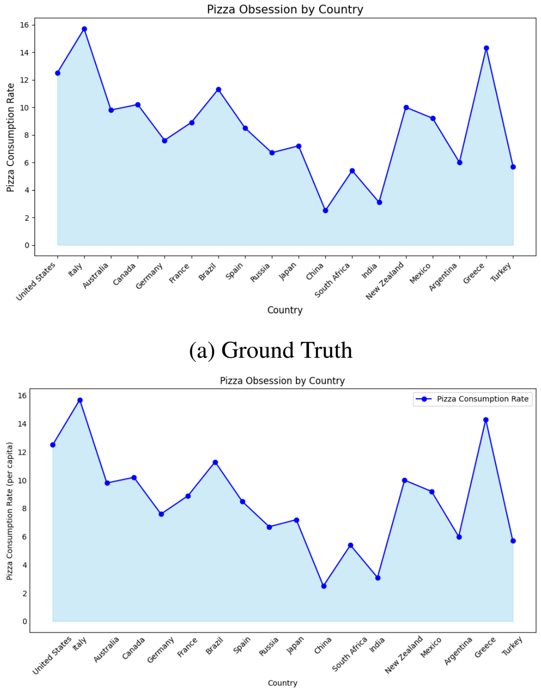
Color Blindness Analysis For the color blindness analysis, GPT4o-mini was prompted to evaluate whether the generated chart was appropriate or not based on color blindness criteria. To assess whether generated charts are appropriate for color-vision-deficient users, we created an LLM-based judgment prompt grounded in accessibility research. Specifically, we drew on practical recommendations from Muth (2020), who highlights common failure modes in data visualizations for colorblind readers, such as over-reliance on hue and the use of indistinguishable color pairs like red/green, purple/blue, or pink/grey when luminance is held constant. These combinations, while legible to users with typical vision, can render visual encodings ambiguous or unreadable for individuals with color deficiencies.

Our system prompt was designed to reflect these insights, emphasizing three critical criteria: (1) redundant encoding, such as pairing color with shape, text, or iconography; (2) contrast in lightness, which improves discernibility even when hue perception is impaired; and (3) legend readability, including whether text labels and gradients remain interpretable under color vision simulation.

Color Blindness Prompt

system_message = You will be given a data-visualization (image). Decide whether the visualization is appropriate for viewers with color-vision deficiencies. How to judge: 1. Do not rely on hue alone. Look for additional cues such as shapes, icons, text labels, patterns, or distinct light-dark contrasts. 2. Avoid problem pairs. Red + green, red + brown, green + brown, purple + blue, and pink + turquoise of similar lightness are hard to tell apart. 3. Prefer safe palettes. Blue vs. orange/red, or any two colors that differ clearly in lightness, usually work well. Check gradients. Gradients should vary in lightness, not just hue. Overall clarity. Annotations, legends, and labels must still be readable when colors are altered by common forms of color-blindness. Output format: Just return Äppropriateör Ñot appropriate,̈ do NOT return anything else. "Return the result as a JSON object in this format: J̈udgment:̈ <string>.
user_message =
"type": "image_url",
"image_url":
"url": f"data:image/png;base64,base64_image2"

Error Analysis Examples of various error either in the graph generation or the color blindness assessment.



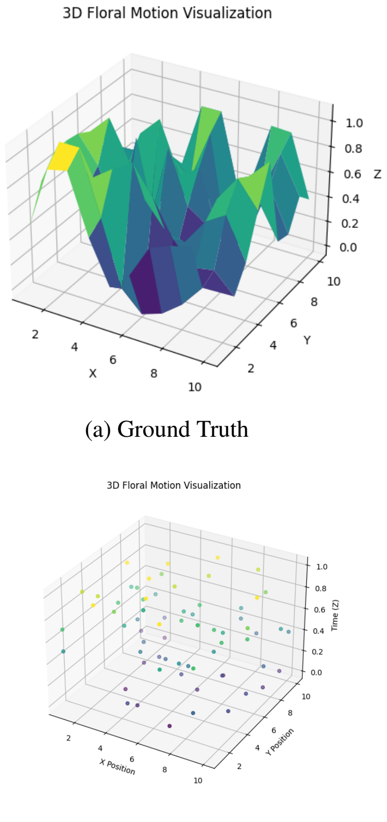
(a) Ground Truth

(b) Agentic

Figure 6: Example of the agent generating a successful chart

Figure 6 displays an example of the agentic process generating a successful chart. Figure 7 displays an example of the agentic process generating the wrong style of chart. Figure 8 displays an example of the agentic process generating the chart with incorrect data, as the labels are interchanged.

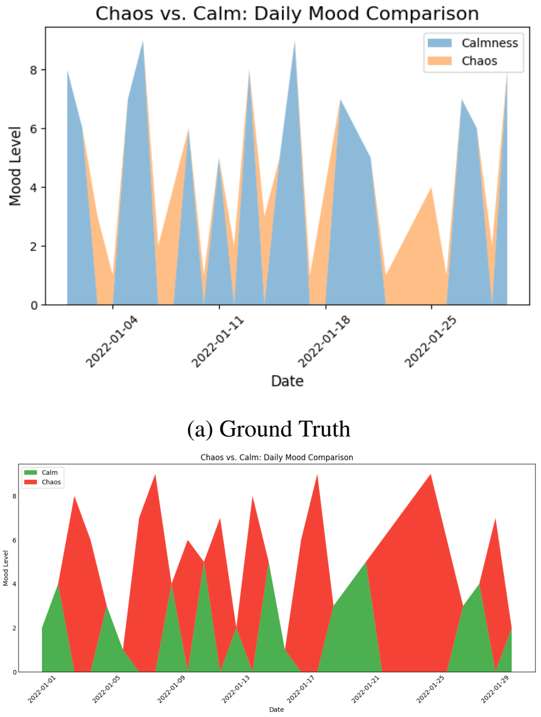
For the color blindness analysis, Figure 9 displays an example of GPT4o-mini correctly assessing a chart as being "Not Appropriate" for color blindness. The chart includes both blue and green, which may cause issues for viewers with visual impairments. In contrast, Figure 10 displays an example of GPT4o-mini incorrectly assessing a chart as being "Not Appropriate" for color blindness. The chart only uses one shade of blue along with the grey border and gridlines, so this is less likely to cause issues for the visually impaired.



(a) Ground Truth

(b) Agentic

Figure 7: Example of the agent generating the wrong style chart



(a) Ground Truth

(b) Agentic

Figure 8: Example of the agent generating the wrong data for a chart

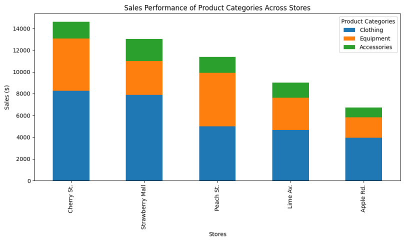


Figure 9: Color Blindness assessment correct

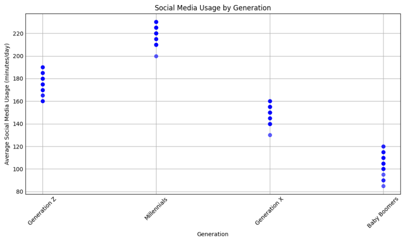


Figure 10: Color Blindness assessment error