Visual Sketchbook: Enabling Reflection and Refinement in MLLMs Chart-to-Code Generation

Anonymous ACL submission

Abstract

Chart-to-code is an emerging task with significant potential in data analysis, automated reporting, and education. It requires accurate visual interpretation of charts and the ability to translate this understanding into executable code. However, existing methods often struggle to generate precise code for more complex charts, resulting in non-executable code and inaccurate chart reconstructions. To address these challenges, we introduce **Visual Sketchbook**—a novel framework that employs a multistage optimization process through iterative multimodal feedback, inspired by recent test-time scaling techniques. Our method decomposes the generation process into reflection and refinement stages, allowing for progressive reasoning and verification. Experiments show that Visual Sketchbook achieves substantial improvements (on average a 12% gain with a maximum of 17%) in chart-to-code tasks compared to baseline methods. We further demonstrate that the effectiveness and generalizability of our proposed method through detailed analysis and ablation studies.

1 Introduction

Chart-to-code is a rapidly evolving research area focused on enabling multimodal large language models (MLLMs) to generate executable code that accurately reproduces visual charts (Zhao et al., 2025; He et al., 2024). It has significant potential for application in fields like data analysis, automated reporting, and education (Han et al., 2023; Yang et al., 2024; Bendeck and Stasko, 2024), where it can enhance efficiency and reduce manual effort in visualizing data.

Chart-to-code requires not only an accurate interpretation of a chart's visual features such as layout, scale, labels, and colors, but also their precise translation into well-structured functional code. However, exsiting approaches employ a single-pass generation strategy to produce the entire code at once based on a static chart input(Shi et al., 2024; Wu et al., 2024), resulting in deviations in chart details and reduced code executability.

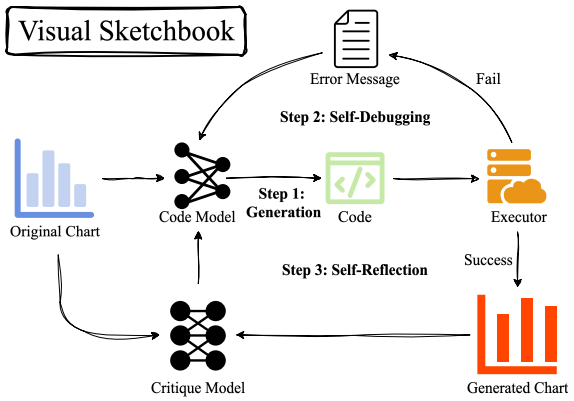

Figure 1: Overview of Visual Sketchbook framework.

To address the challenges, we draw inspiration from test-time scaling techniques, which allow models to reflect on and adjust their outputs during inference and improve their performance. Following self-correction and self-reflection studies in reasoning and programming tasks (Saunders et al., 2022; Ding et al., 2024; Xi et al., 2024), we propose **Visual Sketchbook**, an iterative reasoning framework that incorporates multimodal feedback to refine chart-to-code generation. Specifically, Visual Sketchbook provides the model with a space to generate initial code and chart drafts, which it can then review and adjust iteratively, much like refining a rough sketch into a final version. This process combines feedback from both code execution and visual analysis, allowing the model to detect and correct errors by comparing the generated output with the expected chart. Unlike single-pass methods, this iterative approach enables the model to improve with each round of interaction, resulting in more reliable chart reconstructions.

To evaluate the effectiveness of our proposed method, we conduct experiments on ChartMimic

benchmark (Shi et al., 2024). The experimental results show that Visual Sketchbook outperforms baselines in both chart fidelity and code executability, achieving an average gain of 12% with a maximum of 17% in task performance. In particular, our iterative feedback mechanism significantly reduces errors related to non-executable code and discrepant chart details. Additionally, we conduct ablation studies to assess the framework's robustness across different refinement strategies. Results highlight the adaptability of sequential refinement strategy, underscoring its potential to advance multimodal code generation tasks and offering valuable directions for future research. In summary, our contributions are as follows:

- We introduce Visual Sketchbook, a novel framework for chart-to-code tasks that leverages multimodal feedback to improve performance, and present a multi-aspect self-reflection approach that iteratively enhances the fidelity of chart replications.
- We validate the effectiveness of Visual Sketchbook through experiments on the ChartMimic dataset, demonstrating its performance in chart reconstruction.
- To further explore Visual Sketchbook, we conduct ablation experiments to analyze the impact of different improvement strategies on the final performance.

2 Visual Sketchbook

We propose the Visual Sketchbook framework to enhance the chart-to-code performance of models through an iterative self-reflection mechanism. This framework enables the model to iteratively refine its generated code by systematically analyzing and addressing discrepancies between the generated and original charts. It consists of two stages: (1) an initial code generation stage, and (2) an iterative refinement stage based on self-reflection. Figure 1 illustrates the workflow of Visual Sketchbook.

2.1 Code Generation

In the code generation stage, the model receives the original chart and task description (*i.e.,* replicating the original chart) as input and generates an initial version of the code. This generated code serves as the foundation for subsequent iterative refinement. To enhance the likelihood of successful code execution, we employ the Self-Debugging mechanism (Chen et al., 2023). Specifically, if the generated code encounters a run-time error, the model is instructed to debug based on the error messages provided by the Python interpreter. The self-debugging process continues until the code runs successfully or the maximum iteration limit is reached, which is set to three to prevent infinite loops. By resolving execution issues at this stage, the framework ensures that the subsequent iterative refinement stage operates on functional code, thereby focusing on improving visual accuracy rather than code correctness.

2.2 Iterative Refinement Based on Self-Reflection

During the iterative refinement stage, the model refines its generated code by iteratively focusing on different aspects of the chart, such as layout, text, color, and data representation. Each aspect contributes uniquely to the visual and structural fidelity of the generated chart:

- Layout: Adjusts the overall structure, including the relative positioning of axes, legends, and labels, to align as closely as possible with the original chart.
- Text: Improve the accuracy of textual elements in terms of both position and content, including labels, legends, and titles.
- Color: Modifies the color scheme, aligning the background color and legend hues with those of the original chart.

Since the code itself does not directly reflect the differences between the generated and original charts, the model first produces textual descriptions for each aspect of both charts. It then compares these descriptions to identify discrepancies, which serve as feedback for improving the code. This self-reflection process is designed to iteratively detect and address differences in each aspect, enhancing the visual consistency between the generated and original charts.

The iterative refinement stage can follow different strategies depending on the user-defined iteration budget. Some possible iteration strategies include:

- Sequential Refinement: The model sequentially optimizes each aspect in separate iterations, starting with layout, followed by text and color.

Method	Low-Level					High-Level
	Text	Layout	Type	Color	Avg.	GPT-4o
GPT-4o-mini	67.1	89.0	76.0	61.4	73.4	73.0
+ Self Debugging	73.2	92.1	77.6	66.1	77.2	77.4
+ layout	76.3	93.4	**78.2**	**66.5**	78.6	79.1
+ text	79.2	**93.6**	77.8	65.3	**79.0**	78.9
+ color **(Ours)**	**79.4**	**93.6**	77.7	65.3	**79.0**	**79.4**
GLM-4V-Flash	28.3	55.6	49.3	30.7	41.0	39.3
+ Self Debugging	36.3	62.1	50.4	37.2	46.5	43.9
+ layout	37.1	62.3	52.7	38.3	47.6	44.6
+ text	**37.6**	**63.0**	53.3	39.8	48.4	**46.2**
+ color **(Ours)**	36.9	62.7	**53.8**	**41.5**	**48.7**	**46.2**

Table 1: Performance comparison of Visual Sketchbook on GPT-4o-mini and GLM-4V-Flash, showing improvements in chart-to-code capability with Self-Debugging and iterative self-reflection.

- Grouped Refinement: The model improves multiple aspects within a single iteration, such as reflecting on layout, text, and color simultaneously.
- Single-Aspect Refinement: The model performs only one iteration, focusing on a single aspect.

The flexibility of iteration strategies allows adaptation to different task requirements, ensuring a balance between inference efficiency and refinement quality.

3 Experiments and Analysis

3.1 Experimental Setup

We evaluate the performance of Visual Sketchbook on chart-to-code tasks using two models: GPT-4o-mini (OpenAI, 2024a) and GLM-4V-Flash (Zhipu, 2024). For GPT-4o-mini, we set max_tokens to 4096 and temperature to zero. For GLM-4V-Flash, we use a lower max_tokens of 1024 and also set the temperature to zero to maintain consistency in results. We conduct all experiments on the ChartMimic benchmark (Shi et al., 2024).

3.2 Main Results

Table 1 presents the results of the Visual Sketchbook on various models. It shows an 8% performance improvement for GPT-4o-mini and a 17% improvement for GLM-4V-Flash. The self-debugging mechanism enhances code execution, and through sequential refinement of layout, text, and color, visual fidelity between the generated and original charts steadily improves. These results highlight Visual Sketchbook's potential in enhancing the chart-to-code capabilities of MLLMs.

Method	GPT-4o-mini		GLM-4V-Flash	
	Low-Level	High-Level	Low-Level	High-Level
w/o self-reflection	77.2	77.4	46.5	43.9
grouped refinement	77.7	78.6	47.3	45.6
sequential refinement **(Ours)**	**79.0**	**79.4**	**48.7**	**46.2**

Table 2: Performance comparison of sequential and grouped refinement strategies.

3.3 Ablation Study

We conduct various ablation studies to evaluate the impact of different factors on the Visual Sketchbook. These studies included comparisons between sequential and grouped refinement methods, as well as assessing the necessity of describing the charts prior to identifying discrepancies.

Table 2 compares the performance of sequential and grouped refinement across three key aspects: layout, text, and color. The results demonstrate that sequential refinement generally outperforms grouped refinement in both low-level and high-level evaluations. This superior performance is attributed to the independent and focused critiques provided for each aspect during sequential refinement. Conversely, although grouped refinement is more efficient due to fewer iterations, it tends to produce conflicting feedback between different aspects, leading to a slight reduction in overall performance.

To investigate the necessity of describing the charts in the self-reflection step, we conduct an ablation study comparing two approaches: one

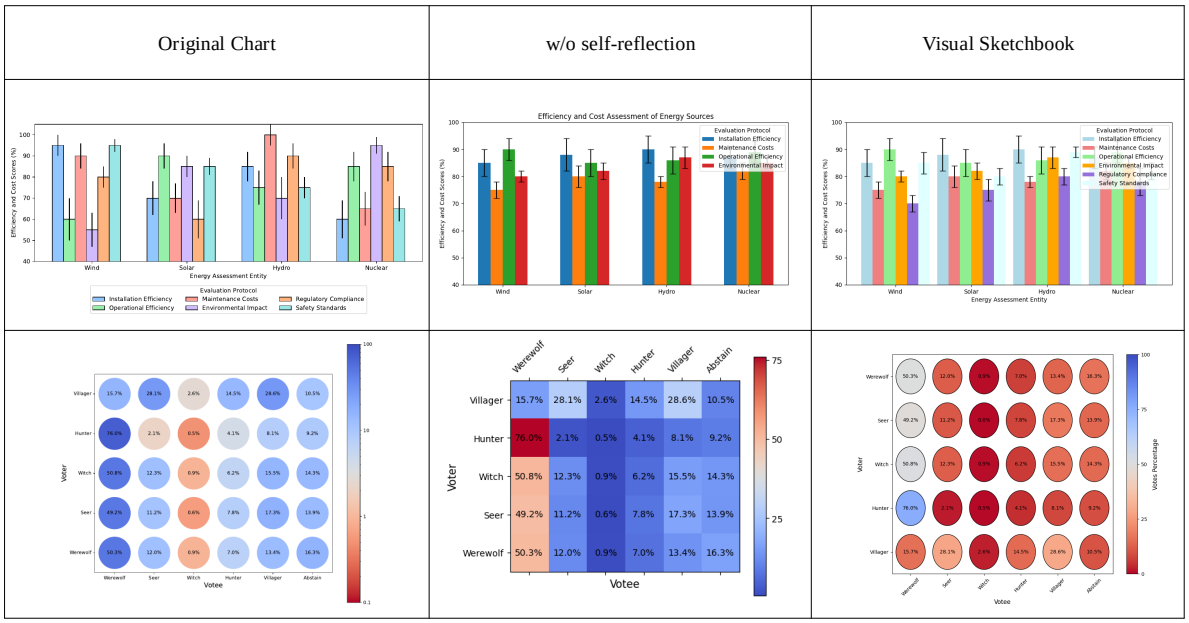

Figure 2: Case study of Visual Sketchbook.

Method	GPT-4o-mini	
	Low-Level	High-Level
w/o self-reflection	77.2	77.4
self-reflection w/o description	78.4	**82.4**
self-reflection w/ description **(Ours)**	**79.0**	79.4

Table 3: Performance comparison of sequential and grouped refinement strategies.

where the model first generates a description of the original chart and the code-generated chart, and the other where the model directly inputs both charts for discrepancy detection without a prior description. Tabel 3 shows that the approach without descriptions outperforms the description-based method in High-Level scores. This is likely because direct discrepancy detection aligns better with the High-Level evaluation format, where GPT-4o (Hurst et al., 2024) scores the similarity between the code-generated chart and the original chart. Although the no-description approach achieves a higher High-Level score, it requires more from the model, as the model must be able to simultaneously process both charts. Therefore, we choose the description-based self-reflection method, which balances performance and the requirements for model capability.

3.4 Case Study

We present two examples in Figure 2 to further demonstrate the advantages of Visual Sketchbook. In the first case, Visual Sketchbook identifies and corrects the omission of two entities from the original chart. It also adjusted the color scheme to more closely match the original chart, ensuring a more accurate visual representation of the data. In the second case, Visual Sketchbook corrects the chart type, but it does not fully address the issue with the color bar y-tick labels, leaving a minor inconsistency that could be improved in future iterations.

4 Conclusion

In this work, we introduce Visual Sketchbook, an iterative framework designed to improve chart-to-code generation by leveraging multimodal feedback. By combining visual analysis with code execution, Visual Sketchbook allows the model to refine its outputs through multiple cycles, addressing errors and enhancing chart fidelity. Our experiments demonstrate the superiority of this approach over traditional single-pass methods, achieving notable improvements in both code executability and chart accuracy. We believe that Visual Sketchbook offers a promising step forward in the field of multimodal code generation.

Limitations

There are some limitations to be addressed in future work. First, this study does not evaluate the Visual Sketchbook method on open-source models, but experiments on proprietary models demonstrate its effectiveness. Future research will expand testing to a wider range of open-source models, assessing its performance across models with varying architectures and performance levels. In addition, this study focuses primarily on chart replication, which limits task coverage. While the effectiveness of the Visual Sketchbook has been shown in this task, exploring other chart-related tasks will be essential. Future research will aim to broaden the scope of chart-to-code generation scenarios.

References

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

Alexander Bendeck and John Stasko. 2024. An empirical evaluation of the gpt-4 multimodal language model on visualization literacy tasks. *IEEE Transactions on Visualization and Computer Graphics*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*.

Yiwen Ding, Zhiheng Xi, Wei He, Zhuoyuan Li, Yitao Zhai, Xiaowei Shi, Xunliang Cai, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. Mitigating tail narrowing in llm self-improvement via socratic-guided sampling. *arXiv preprint arXiv:2411.00750*.

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*.

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

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Shankar Kantharaj, Rixie Tiffany Ko Leong, Xiang Lin, Ahmed Masry, Megh Thakkar, Enamul Hoque, and Shafiq Joty. 2022. Chart-to-text: A large-scale benchmark for chart summarization. *arXiv preprint arXiv:2203.06486*.

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

Haoyu Lu, Wen Liu, Bo Zhang, Bingxuan Wang, Kai Dong, Bo Liu, Jingxiang Sun, Tongzheng Ren, Zhuoshu Li, Hao Yang, et al. 2024. Deepseek-vl: towards real-world vision-language understanding. *arXiv preprint arXiv:2403.05525*.

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. *arXiv preprint arXiv:2203.10244*.

OpenAI. 2024a. Gpt-4o mini.

OpenAI. 2024b. Gpt-4v.

William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. 2022. Self-critiquing models for assisting human evaluators. *arXiv preprint arXiv:2206.05802*.

Chufan Shi, Cheng Yang, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran Xu, Xinyu Zhu, Siheng Li, Yuxiang Zhang, Gongye Liu, Xiaomei Nie, Deng Cai, and Yujiu Yang. 2024. Chartmimic: Evaluating lmm's cross-modal reasoning capability via chart-to-code generation. *arXiv preprint arXiv:2406.09961*.

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

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. *arXiv preprint arXiv:2405.07990*.

Zhiheng Xi, Dingwen Yang, Jixuan Huang, Jiafu Tang, Guanyu Li, Yiwen Ding, Wei He, Boyang Hong, Shihan Do, Wenyu Zhan, et al. 2024. Enhancing llm reasoning via critique models with test-time and training-time supervision. *arXiv preprint arXiv:2411.16579*.

Zhiyu Yang, Zihan Zhou, Shuo Wang, Xin Cong, Xu Han, Yukun Yan, Zhenghao Liu, Zhixing Tan, Pengyuan Liu, Dong Yu, et al. 2024. Matplotagent: Method and evaluation for llm-based agentic scientific data visualization. *arXiv preprint arXiv:2402.11453*.

Xuanle Zhao, Xianzhen Luo, Qi Shi, Chi Chen, Shuo Wang, Wanxiang Che, Zhiyuan Liu, and Maosong Sun. 2025. Chartcoder: Advancing multimodal large language model for chart-to-code generation. *arXiv preprint arXiv:2501.06598*.

Zhipu. 2024. Glm-4v-flash.

A Related Work

Multimodal Large Language Models. Recent developments in MLLMs have dramatically improved the processing of complex multimodal inputs. Notably, proprietary models such as GPT-4V (OpenAI, 2024b) and Gemini (Team et al., 2023) have showcased significant advancements. Concurrently, open-weight models like LLaVA (Liu et al., 2024), Qwen-VL (Bai et al., 2023), and DeepSeek-VL (Lu et al., 2024) have made substantial contributions to the domain, further advancing the state of the art.

Chart Understanding. Driven by advancements in computer vision and natural language processing, chart understanding has received considerable attention in recent years. This task involves extracting visual information from charts and integrating it with logical reasoning and semantic understanding to generate accurate descriptions or answer related questions. Key sub-tasks within chart understanding include chart question answering (Masry et al., 2022), chart-to-text summarization (Kantharaj et al., 2022), and chart-to-code generation, among others. Chart-to-code, a relatively underexplored subfield within chart understanding, aims to generate executable code (*e.g.,* Python scripts) from visual chart data. Early approaches typically combine computer vision techniques with language models to produce code that replicates chart patterns or constructs data processing pipelines (Jiang et al., 2024). However, existing methodologies have been constrained by limitations in code accuracy and flexibility (Shi et al., 2024; Wu et al., 2024). This paper seeks to address these challenges by proposing new advancements in the field.

B Prompts

Figures 3 to 6 show the prompts used in Visual Sketchbook.

The Prompt for Code Generation

You are a skilled AI agent specializing in generating code to replicate or create charts. Your responses must include well-commented, efficient, and accurate code in the specified programming language (Python with Matplotlib).

Here is the query:
{{query}}

When you complete a plot, remember to save it. The file name should be "**{{output_image_file}}**".

{{image}}

Figure 3: The prompt for code generation.

The Prompt for Chart Description

Layout: Please describe in detail the layout of the chart, including the positions and relative arrangements of the subplots, title, axes, legends, data points, grid lines, and other components. You just need to pay attention to the arrangement of each component.
Text: Please describe the content and positions of titles, labels, axis and annotations in the chart.
Color: Please describe the color information used in the chart, including background colors and the colors of data series.

{{image}}

Figure 4: The prompt for chart description.

The Prompt for Self-Reflection

You will receive two charts: the first one is the reference chart, and the second one is the AI-generated chart. Your task is to identify the most significant discrepancy or omission between the AI-generated chart and the reference chart.
{{seggestion}}

Your output should match this format:
{
"discrepancy": "<one key discrepancy where the AI-generated chart does not align with the reference chart>",

"correct": "<the correct information of the reference chart>"
}

Figure 5: The prompt for Self-Reflection.

The Prompt for Self-Refinement

You are an expert in chart visualizations using Matplotlib. Your task is to modify the provided Python code to ensure that the resulting chart closely matches the reference chart.

The following code is responsible for reproducing the reference chart:
{{code}}

The chart generated by this code, based on its description, does not match the reference chart in the following areas:
{{feedback}}

Now, please generate the improved code. Once you have completed the plot, remember to save it. The file should be named "**{{output_image_file}}**."

{{image}}

Figure 6: The prompt for Self-Refinement.

C Statements

C.1 Data Statement

All the data used in our paper are from open-source datasets, which are permitted for use in scientific research and publication.

C.2 AI Assistants Using Statement

We only used ChatGPT to assist with writing refinement, including correcting grammar errors and improving readability. However, we did not use the AI assistant for coding or research innovation.

C.3 Computational Budget

Our proposed method achieves accurate generation of code from charts at inference time. This approach avoids the need to train a new model, thereby saving significant computational resources. In our experiments, GLM-4V-Flash is free. On average, generating the code per 1,000 charts using GPT-4o-mini costs approximately $5, and evaluating these charts using ChartMimic (High-Level) costs about $10.