RealChart2Code: Bridging the Gap in Real-World Chart-to-Code Generation via Multi-Task Evaluation

Jiajun Zhang[1,5]* **Yuying Li**[2]* **Zhixun Li**[3]* **Xingyu Guo**[4,5] **Jingzhuo Wu**[6]
Leqi Zheng[2] **Yiran Yang**[7] **Jianke Zhang**[2] **Qingbin Li**[4] **Shannan Yan**[2]
Changguo Jia[8] **Junfei Wu**[4,5] **Zilei Wang**[1] **Qiang Liu**[4,5] **Liang Wang**[4,5]†
[1]USTC [2]THU [3]CUHK [4]UCAS [5]CASIA [6]BNU [7]BUPT [8]PKU
zhangjiajun519@gmail.com

Dataset: https://huggingface.co/datasets/zjj1233/RealChart2Code

Abstract

Vision-Language Models (VLMs) have demonstrated impressive capabilities in code generation across various domains. However, their ability to replicate complex, multi-panel visualizations from real-world data remains largely unassessed. To address this gap, we introduce **RealChart2Code**, a new large-scale benchmark with over 2,800 instances grounded in authentic datasets and featuring tasks with clear analytical intent. Crucially, it is the first benchmark to systematically evaluate chart generation from large-scale raw data and assess iterative code refinement in a multi-turn conversational setting. Our comprehensive evaluation of 14 leading VLMs on RealChart2Code reveals significant performance degradation compared to simpler benchmarks, highlighting their struggles with complex plot structures and authentic data. Our analysis uncovers a substantial performance gap between proprietary and open-weight models and confirms that even state-of-the-art VLMs often fail to accurately replicate intricate, multi-panel charts. These findings provide valuable insights into the current limitations of VLMs and guide future research directions. We release the benchmark and code at https://github.com/Speakn0w/RealChart2Code.

1 Introduction

Recent advancements in AI research have demonstrated the powerful code generation capabilities of LLMs (OpenAI, 2023, 2025; Anthropic, 2023; Team, 2024; Rozière et al., 2023; Hui et al., 2024; MistralAI, 2024; Team et al., 2025b; Cao et al., 2026; Team, 2025), which have solved coding challenges in domains such as software engineering (Jimenez et al., 2023; Zhang et al., 2025b; Pan et al., 2025; Shum et al., 2025), code completion (Ding et al., 2023; Yang et al., 2024; Gong et al., 2024; Zhang et al., 2026a), and algorithmic problem-solving (Chen et al., 2021a; Zhuo et al., 2025; Jain et al., 2024). Chart-to-code generation is another prominent application area, where the goal is to reproduce the visualization code from an image. This capability fulfills a frequent and practical user need by enabling users to recover the underlying visualization logic from static images, which is especially valuable when the original code is unavailable and the chart needs to be edited, extended, or reused in different contexts. However, while current VLMs excel at creating simple, single-panel charts, they struggle to generate plots with multiple subplots and intricate composite layouts, especially when derived from large, complex structured data. As illustrated in Figure 1, a state-of-the-art model fails to accurately replicate the intended multi-plot structure.

Benchmark	Real Data	Complex	Interactive	Multi Tasks
Plot2Code	✗	✗	✗	✗
Design2Code	✗	✗	✗	✗
ChartMimic	✗	✗	✗	✓
Ours	✓	✓	✓	✓

Table 1: Comparison of RealChart2Code with existing chart-to-code benchmarks.

Prior benchmarks for chart-to-code generation have primarily focused on simple chart types and single-panel layouts. They often rely on either pre-existing chart-code pairs from the internet, which pose a risk of data leakage, or on synthetic data created to replicate figures from scientific papers (e.g., Plot2Code (Wu et al., 2024), ChartMimic (Yang et al., 2025)). Furthermore, they lack metrics for evaluating a model's ability to refine code in multi-turn conversation. With the rapid advancement of LLMs, such benchmarks are no longer sufficient for evaluating a model's ability to handle chart-to-code tasks involving complex, real-world data and intricate plot structures.

To systematically evaluate these capabilities,

*Equal contribution.
†Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 41995–42032
July 2-7, 2026

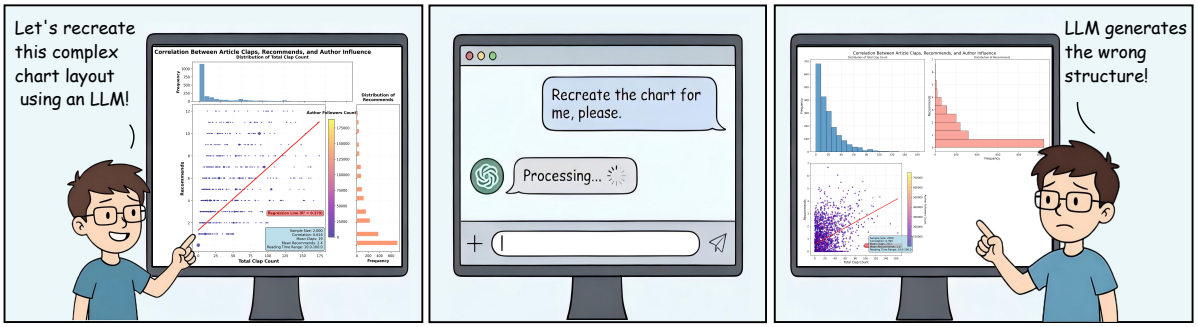

Figure 1: A real-world example illustrating the limitations of LLMs on complex chart-to-code tasks. When presented with a sophisticated request, the model generates a low-quality output and struggles to make effective improvements during the subsequent refinement process.

we introduce `RealChart2Code`, a new large-scale benchmark comprising 2896 instances. `RealChart2Code` is distinguished from prior work in four key aspects, as illustrated in Table 1. ❶ First, it is grounded in realistic visualization scenarios and utilizes authentic datasets, in contrast to benchmarks that rely on synthetic data or arbitrary constructions. ❷ Second, it introduces a significantly higher level of complexity by incorporating intricate chart structures and a diverse range of chart types. ❸ Third, it features an interactive chart-to-code framework that simulates real-world development workflows. ❹ Finally, it incorporates three challenging tasks designed to comprehensively evaluate model capabilities in generating complex visualizations, understanding chart semantics, and modifying plots. Specifically, we construct the benchmark by rigorously filtering high-quality datasets from Kaggle (Kaggle, 2025), manually designing complex visualization tasks, and implementing the corresponding ground-truth code. Furthermore, we construct realistic chart refinement contexts by manually designing errors and correction instructions, ultimately yielding a comprehensive benchmark grounded in authentic development scenarios.

We evaluate 14 prominent VLMs on the `RealChart2Code` benchmark, including 5 proprietary and 9 open-weight models. We observe that most models that perform well on simple benchmarks fail to achieve comparable performance on `RealChart2Code`, primarily due to difficulties in handling complex chart structures and large-scale, authentic data. To validate our quantitative results, we conduct a human evaluation that manually inspects the correctness and fidelity of the generated visualizations. A subsequent correlation analysis (§ 5.1) demonstrates a strong correlation between our multi-level metrics and human judgments. Finally, we perform extensive quantitative analysis and qualitative case studies (§5.2) on model performance across multiple benchmarks (§5.3). This analysis reveals key similarities and differences in model capabilities across tasks of varying difficulty and types, providing valuable insights to guide future research.

2 Related Works

2.1 Code Generation

Recent advances in Large Language Models (LLMs), including general-purpose models (e.g., GPT (OpenAI, 2023), Claude (Anthropic, 2023), Gemini (Team, 2024)) and specialized code models (e.g., Qwen-Coder (Hui et al., 2024), DeepSeek-Coder (Guo et al., 2024), Codestral (MistralAI, 2024)), have demonstrated powerful coding capabilities (DeepSeek-AI and etc., 2024; Rozière et al., 2023; Team et al., 2025b,a). While conventional tasks like algorithmic problem-solving (Chen et al., 2021a; Zhuo et al., 2025) and software engineering (Jimenez et al., 2023; Zhang et al., 2025b) are evaluated on functional correctness (Chen et al., 2021b), this paper focuses on data visualization, a domain where generated code must produce a visually accurate output, a requirement shared by front-end design (Xu et al., 2025; Lu et al., 2025; Chen et al., 2025) and SVG generation (Xing et al., 2025).

2.2 Data Visualization

Prior LLM-based data visualization research spans three main areas. The first, chart understanding, focuses on interpreting visual information from plots for tasks like question answering or summary generation (Li et al., 2024; Zeng et al., 2024;

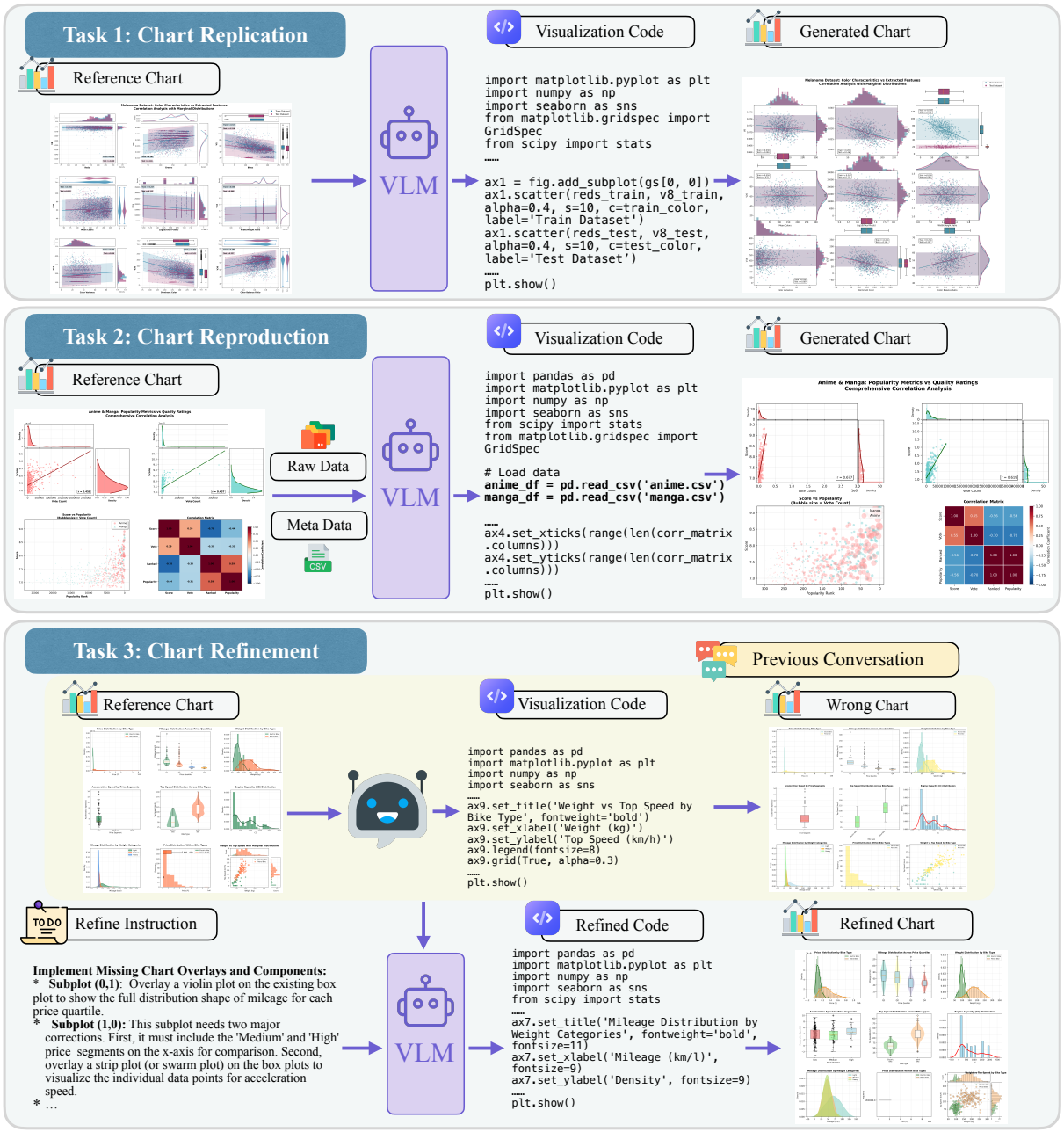

Figure 2: The three core tasks of the RealChart2Code benchmark. **Chart Replication** is the fundamental chart-to-code task. **Chart Reproduction** adds the challenge of using provided raw data files. **Chart Refinement** introduces a conversational component, where the model must debug and modify code to fix errors according to user feedback.

Rahman et al., 2023; Kantharaj et al., 2022; Jia et al., 2025; Zhang et al., 2025c; Ma et al., 2024; Zheng et al., 2026; Li et al., 2025; Zhang et al., 2026b,c; Xin et al., 2025; Hui et al., 2025; Chang et al., 2025; Jiang et al., 2026). The second, Text-to-Visualization (Text2Vis), concerns generating visualization specifications or code from natural language descriptions (Luo et al., 2025; Galimzyanov et al., 2025; Ni et al., 2025; Zhang et al., 2025a; Fang et al., 2026; Yan et al., 2026b). The third, Chart-to-Code (Chart2Code), involves reverse-engineering a visualization by generating the code required to replicate it (Wu et al., 2024; Yang et al., 2025; Zhao et al., 2025; Luo et al., 2026; Su et al., 2026, 2025; Yan et al., 2026a). However, existing benchmarks in this domain predominantly feature simple, single-panel plots, which are insufficient for evaluating an LLM's ability to handle complex layouts and high information density. To address this critical gap in chart-to-code evaluation, we introduce RealChart2Code, a benchmark specifically designed to assess performance on intricate,

multi-panel charts derived from real-world data.

3 RealChart2Code

3.1 Task Definition

We define the chart-to-code task as a conditional code generation problem. Formally, given a source chart image V and an accompanying prompt P, a LLM, denoted by $\mathcal{F}(\cdot)$, must generate an executable code snippet C. This code must render a visualization that accurately reproduces the visual and structural elements of V while adhering to any requirements in P. The task is formulated as $C = \mathcal{F}(V, P)$. The RealChart2Code benchmark evaluates models on three distinct variants of this core task, illustrated in Figure 2: **(1) Chart Replication**, is the fundamental chart-to-code task where the model must reverse-engineer the visualization from the image alone, measuring its core visual-to-code translation ability. **(2) Chart Reproduction**, provides the model with the chart image, raw data, and metadata, assessing its capability to generate the correct plot using large-scale, real-world data sources. **(3) Chart Refinement**, which requires the model to correct a chart with predefined errors through a multi-turn dialogue, assessing its ability to perform iterative debugging based on user instructions.

3.2 Benchmark Coverage Analysis

Chart Types The chart data in RealChart2Code can be classified from two perspectives: visualization intent and chart type. Our taxonomy includes seven high-level intent categories and 50 distinct plot types. Crucially, all visualizations in the benchmark are designed to be complex, featuring composite charts or intricate multi-panel layouts. As a result, a single plot type label is often insufficient to describe a given instance. Detailed examples of these categories and complex layouts are provided in Appendix A.1.

Dataset Distribution RealChart2Code covers diverse thematic topics across eight high-level domains: Finance, Industry, Health, Research, Society, Media, Technology, and Environment. These domains are further divided into 35 fine-grained sub-topics, ensuring broad applicability to real-world scenarios. Figure 3 illustrates the distribution of chart images and CSV data across all three tasks using CLIP (Radford et al., 2021) and t-SNE (Maaten and Hinton, 2008). As shown in Figure 3(a) and (b), both distributions are widely dispersed across the feature space, indicating substantial diversity in visual styles, layouts, and data characteristics across Chart Replication, Chart Reproduction, and Chart Refinement tasks. Figure 3(c) shows the data length distribution, reflecting the complexity of real-world datasets. This comprehensive coverage ensures that RealChart2Code challenges models with diverse chart types and data patterns.

3.3 Data Curation Process

The construction of RealChart2Code follows a four-stage pipeline: (1) Data Collection and Filtering, (2) Visualization Task Design, (3) Code Implementation, and (4) Error Injection. We describe each stage in detail below. Further details and strict quality control measures are provided in Appendix B.

Data Collection and Filtering We collect open-source datasets from Kaggle. Our use of these datasets is strictly for scientific research; other uses fall under their original licenses. Our process involved a two-stage filtering pipeline. First, we performed an initial screening of over 8,000 datasets, which collectively contained more than 100,000 files and 30 billion data rows. This screening was based on community metrics such as vote counts, download counts, and usability ratings. From this initial pool, we conducted a second, more rigorous filtering stage to select 1,036 high-quality datasets suitable for our benchmark's task and chart construction. The final curated collection for RealChart2Code contains 3,271 raw data files, with approximately 860 million rows in total.

Visualization Task Design Using the curated datasets, we designed 1,016 unique and complex visualizations. Each of these visualizations serves as the basis for two distinct tasks: (1) Chart Replication, where the model receives only the chart image, and (2) Chart Reproduction, where the model is also provided with the corresponding raw data. This dual-task structure results in 2,032 instances across these two categories. Every visualization was designed to be contextually relevant to its source dataset, ensuring practical, real-world meaning. To guarantee task diversity and complexity, the design process was guided by a taxonomy of 7 high-level visualization intents and 50 distinct chart types.

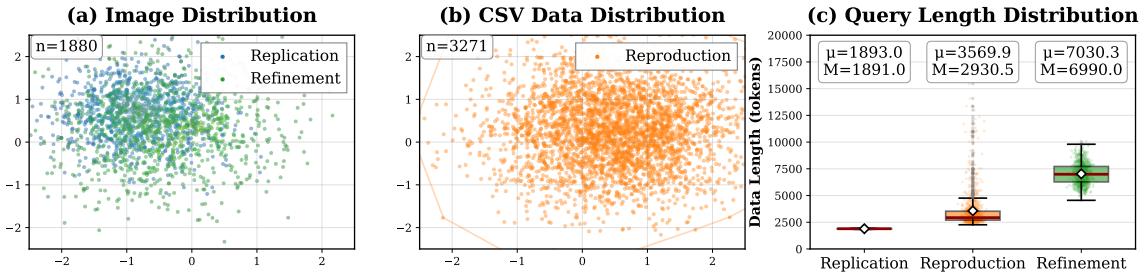

Figure 3: Dataset distribution across tasks. (a) Image distribution. (b) CSV data distribution. (c) Data length distribution with median (M) and mean (μ) in tokens.

Ground-Truth Code Implementation For each task, our in-house team of five expert Python developers implemented the ground-truth code using Matplotlib and its associated libraries. This code serves as the reference solution that models must replicate. The sandboxed execution environment used for evaluation is detailed in Appendix A.2.

Error Injection To create the Chart Refinement tasks, we manually injected errors into a subset of the ground-truth charts. These intentionally flawed charts serve as the starting point for a multi-turn dialogue. The types of errors are diverse, including incorrect chart types, data mapping errors, element overlap and other common errors. Through this process, we constructed 864 Chart Refinement tasks. In total, the `RealChart2Code` benchmark consists of 2,896 instances, comprising 1,016 for Chart Replication, 1,016 for Chart Reproduction, and the 864 for Chart Refinement.

3.4 Evaluation Metrics

Our evaluation first assesses functional correctness, which we measure as the Pass Rate: the percentage of generated code snippets that execute successfully in our sandbox environment without errors. Submissions that fail this check are automatically assigned a score of zero. For all valid outputs, we deploy a multi-agent judging panel that uses a voting system to score visual accuracy. Each chart is assessed on a 3-point scale (0, 1, or 2) across eight key criteria: chart type, spatial layout, text elements, axis configuration, color scheme, style, component completeness, and data pattern consistency. Notably, for the Chart Reproduction task, Data Pattern Consistency is evaluated programmatically. We perform a code-level comparison to ensure the model's data handling is identical to the reference implementation, rather than relying on visual inspection. Beyond these core accuracy metrics, our evaluation also includes a qualitative assessment of the chart's design, scored on Visual Clarity, Compositional Balance, and Typographic Quality. The complete scoring rubrics and prompts used for our automated evaluation are detailed in Appendix C.1.

4 Experiments

4.1 Experiments Setup

Baseline Models We evaluate 14 widely-used proprietary and open-source Large Language Models. For proprietary models, we evaluate five leading models: Anthropic's flagship models, Claude-4.5-Sonnet and Claude-4.5-Opus (Anthropic, 2023); OpenAI's advanced model, GPT-5.1 (OpenAI, 2025); and Google's Gemini 3 Pro Preview and Gemini 2.5 Flash (Team, 2024). For open-weight models, we select 9 competitive models with parameter sizes ranging from 7B to 241B.

Benchmark Setup To ensure a comprehensive evaluation, in addition to our primary `RealChart2Code` benchmark, we also evaluated model performance on two established chart-to-code benchmarks: Plot2Code (Wu et al., 2024) and ChartMimic (Yang et al., 2025).

Evaluation Details All experiments were conducted using the standard OpenAI API format, with a chat structure compliant with ChatML (OpenAI, 2022). We employed a greedy decoding strategy and set the maximum output token limit to 32,768. If a model did not support this context length, its own maximum limit was used instead. The final reported results are an average of three independent runs. Proprietary models were queried via their official APIs, while open-weight models were served using the SGLang framework. Additional details on prompts and evaluation are provided in Appendix D.

Model	Chart Replication		Chart Reproduction		Chart Refinement		AVG score
	Pass (%)	Score	Pass (%)	Score	Pass (%)	Score	
Closed-source LLMs							
Claude-4.5-Sonnet (Anthropic, 2023)	85.8	7.1	73.7	5.7	89.0	8.0	7.0
Claude-4.5-Opus (Anthropic, 2023)	**87.7**	7.8	**86.1**	**7.4**	**91.2**	**9.4**	**8.2**
Gemini-2.5-Flash (Team, 2024)	63.2	5.2	57.6	4.8	69.0	6.0	5.3
Gemini-3-Pro-Preview (Team, 2024)	82.1	**9.0**	68.7	6.7	83.6	8.5	8.1
GPT-5.1 (OpenAI, 2025)	71.2	5.7	64.8	4.8	73.0	5.8	5.4
Open-source LLMs							
DeepSeek-VL-7B (Lu et al., 2024)	9.7	0.4	5.9	0.3	31.4	1.4	0.7
Intern-VL-3.5-241B (Wang et al., 2025)	**54.3**	**3.5**	**49.2**	**2.5**	60.2	4.3	3.4
Intern-VL-3.5-30B (Wang et al., 2025)	20.3	0.8	15.3	0.7	41.6	2.0	1.2
Qwen3-VL-235B (Bai et al., 2025)	49.2	3.3	38.2	2.4	**65.1**	**5.1**	**3.6**
Qwen3-VL-30B (Bai et al., 2025)	17.3	0.8	19.0	0.9	45.2	2.2	1.3
GLM-4.5V-106B (Hong et al., 2025)	38.4	2.8	39.1	2.1	44.3	2.0	2.3
GLM-4.1V-9B (Hong et al., 2025)	12.8	0.7	16.2	0.9	39.9	1.8	1.1
MiMo-VL-7B-RL (Team et al., 2025c)	11.0	0.4	10.7	0.5	34.8	1.5	0.8
ChartCoder (Zhao et al., 2025)	48.0	**3.5**	40.5	2.3	54.7	3.9	3.2

Table 2: **Quantitative results on RealChart2Code 14 primary LLMs across three tasks: Chart Replication, Chart Reproduction, and Chart Refinement.** For each task, we report Pass Rate (%) and Score. AVG score is the average of scores across all three tasks. Within each model category, the best score is **bolded** and the second-best is underlined.

Model	ChartMimic				Plot2Code		
	Direct Mimic		Customized Mimic				
	Pass (%)	Score	Pass (%)	Score	Pass (%)	Text	Rating
Closed-source LLMs							
Claude-4.5-Sonnet (Anthropic, 2023)	**100.0**	90.1	99.5	91.5	93.9	73.3	8.8
Claude-4.5-Opus (Anthropic, 2023)	98.5	92.3	**100.0**	95.1	**99.6**	75.4	9.0
Gemini-2.5-Flash (Team, 2024)	88.5	69.8	89.1	74.4	87.9	72.8	8.6
Gemini-3-Pro-Preview (Team, 2024)	97.3	**96.0**	**100.0**	**96.8**	90.9	79.5	**9.6**
GPT-5.1 (OpenAI, 2025)	97.8	89.8	98.5	91.2	98.5	**82.6**	8.8
Open-source LLMs							
DeepSeek-VL-7B (Lu et al., 2024)	41.3	19.7	59.3	37.6	64.4	32.6	2.3
Intern-VL-3.5-241B (Wang et al., 2025)	90.5	76.9	90.7	78.4	**90.2**	69.0	7.6
Intern-VL-3.5-30B (Wang et al., 2025)	85.6	68.1	87.2	68.5	88.6	66.3	6.7
Qwen3-VL-235B (Bai et al., 2025)	**91.5**	**80.4**	**92.7**	**81.8**	89.8	**69.8**	**8.9**
Qwen3-VL-30B (Bai et al., 2025)	89.5	68.6	89.1	69.9	88.9	65.0	7.9
GLM-4.5V-106B (Hong et al., 2025)	87.4	72.8	86.7	71.2	84.1	61.7	5.8
GLM-4.1V-9B (Hong et al., 2025)	86.6	70.2	87.1	70.1	74.1	59.1	4.9
MiMo-VL-7B-RL (Team et al., 2025c)	83.2	58.5	87.7	64.7	73.5	55.5	4.3
ChartCoder (Zhao et al., 2025)	90.9	75.3	92.1	76.7	87.9	54.5	4.7

Table 3: Quantitative results for 14 LLMs across two benchmarks, ChartMimic and Plot2Code.

4.2 Main Results

This section presents the evaluation results for 14 leading LLMs on our `RealChart2Code` benchmark, as well as on the existing ChartMimic and Plot2Code benchmarks. The primary performance on `RealChart2Code` is detailed in Table 2, while results on the other benchmarks are shown in Table 3. For a granular analysis, Figure 4 breaks down the scores by task and sub-metrics, including three quality assessment criteria. The key findings are as follows.

❶ Claude-4.5-Opus leads proprietary models, but a significant performance gap exists between proprietary and open-source models. Among proprietary models, Claude-4.5-Opus achieves the highest overall score of 8.2, demonstrating strong and consistent performance across all three tasks. Gemini-3-Pro-Preview follows closely with a score of 8.1, even achieving the top score on the funda-

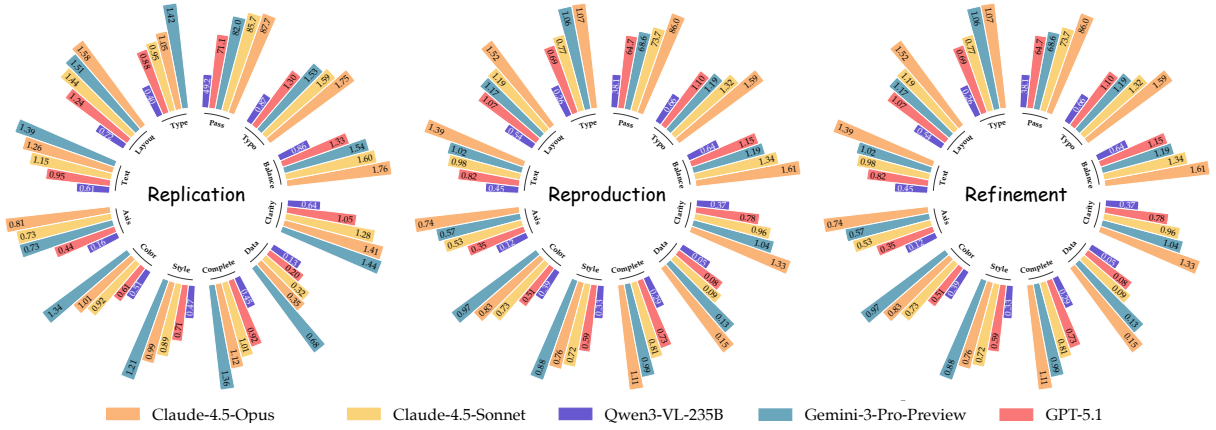

Figure 4: Performance breakdown across tasks and metrics for top models. Each radial chart shows scores for eight visual accuracy metrics (Type: chart type consistency, Layout: spatial layout, Text: text elements, Axis: axis configuration, Color: color scheme, Style: style and format, Complete: component completeness, Data: data alignment), execution pass rate (Pass), and three quality metrics (Clarity: visual clarity, Balance: compositional balance, Typo: typographic quality).

mental Chart Replication task at 9.0. In the open-source category, performance is considerably lower. The top-performing models are Qwen3-VL-235B and Intern-VL-3.5-241B, with scores of 3.6 and 3.4, respectively, which are less than half of those from the leading proprietary models. This highlights a significant capability gap on the complex, real-world tasks featured in RealChart2Code.

❷ Strong performance on simpler benchmarks does not guarantee success on RealChart2Code. We observe a significant performance disparity for models across different benchmarks, indicating that RealChart2Code provides a much stronger differentiation of model capabilities. For instance, models like Qwen3-VL-235B and Intern-VL-3.5-241B achieve excellent scores above 75 on ChartMimic but experience a drastic performance degradation on RealChart2Code, where their scores drop to 3.6 and 3.4, respectively. This suggests that while previous benchmarks can identify basic competency, their lack of complexity fails to distinguish between models with truly advanced visual reasoning and code generation abilities. In contrast, our benchmark effectively separates the performance of even top-tier models, with Claude-4.5-Opus scoring 8.2 and GPT-5.1 scoring 5.4. Additionally, we find that different models exhibit specific error patterns, which are analyzed in detail in Section 5.2.

Metric	Fleiss' κ (Inter-Agent)	Cohen's κ (Agent-Human)
Type	0.921	0.91
Layout	0.998	0.99
Text	0.896	0.79
Axis	0.892	0.86
Color	0.879	0.72
Style	0.829	0.78
Data	0.813	0.82
Completeness	0.781	0.74
Average	0.824	0.83

Table 4: Reliability analysis: Fleiss' κ for inter-agent agreement and Cohen's κ for agreement between multi-agent judge and human evaluations across eight metrics.

5 Discussion

5.1 Reliability Analysis

We validate the robustness of our automated evaluation framework through internal consistency and alignment with human judgment (Table 4, Figure 5). To assess the consensus among agents, we calculated Fleiss' κ across the entire RealChart2Code benchmark. The resulting average Inter-Agent κ score of 0.8239 indicates that the multi-agent framework maintains high stability throughout the evaluation process.

We further examined the correlation between our judge and human experts using 600 tasks sampled from Claude-4.5-Sonnet results. By computing Cohen's κ, we observed an average agreement score of 0.83. This strong correlation demonstrates that our automated multi-agent judge effectively captures human preference. This reliability is visually

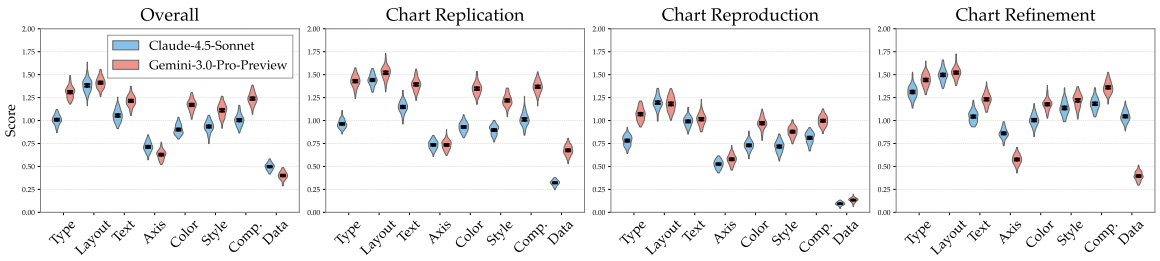

Figure 5: Score distributions with 95% confidence intervals across evaluation metrics for Claude-4.5-Sonnet and Gemini-3.0-Pro-Preview on chart replication, reproduction, and refinement tasks.

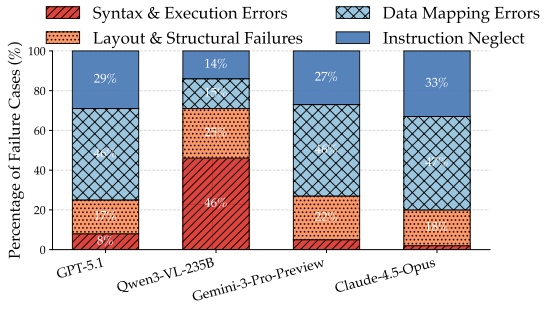

Figure 6: Error type distribution across models. Percentages indicate the proportion of each error category in failure cases.

corroborated by Figure 5, which presents the score distributions and 95% confidence intervals. The distinct distributions combined with narrow intervals confirm that the judge provides a discriminatory and precise assessment of visualization quality.

5.2 Error Analysis

To gain deeper insights into the limitations of current LLMs, we conducted a comprehensive error analysis covering all failure instances in the `RealChart2Code` benchmark. This analysis aims to reveal systematic weaknesses in chart-to-code generation and to better understand where existing models break down in practice. We categorize the observed errors into four primary types: (1) Syntax and Execution Errors, (2) Layout and Structural Failures, (3) Data Mapping Errors, and (4) Instruction Neglect, which together capture the major sources of performance degradation.

Divergent Failure Patterns. We observed a significant disparity in the types of errors exhibited by proprietary versus open-weight models, as illustrated in Figure 6: **❶ Open-weight models (e.g., Qwen3-VL, InternVL)** are particularly susceptible to Syntax and Execution Errors. These models frequently hallucinate non-existent libraries or invoke invalid functions, leading to immediate code execution failures. Furthermore, when the code does execute, they often struggle with spatial reasoning, resulting in Layout and Structural Failures, such as overlapping subplots or incorrect grid definitions. **❷ Proprietary models (e.g., Claude-4.5, GPT-5.1)**, in contrast, demonstrate robust coding capabilities with minimal syntax errors. Their failures are predominantly Data Mapping Errors, where the visual structure is correct, but specific data series are mapped to the wrong axes or visual attributes do not match the prompt requirements.

Challenges in Iterative Refinement. The Chart Refinement task reveals a critical weakness in maintaining context. We identified a frequent error mode termed "Regressive Editing." When users request a specific modification, models often successfully apply the fix but inadvertently introduce new errors in previously correct parts of the code. This phenomenon indicates that even state-of-the-art models struggle to balance local code updates with global consistency during multi-turn conversations. Detailed case studies illustrating these error types and specific failure modes are provided in Appendix F.

5.3 Performance Analysis Across Benchmarks

Figure 7 compares normalized model performance (0-100%) on `RealChart2Code` against `ChartMimic` and `Plot2Code`. We observe that while proprietary models like Gemini-3-Pro-Preview saturate existing benchmarks with scores exceeding 91%, their performance declines to approximately 50% on our dataset. This gap widens for open-weight models (e.g., Qwen3-VL), where competitive scores ($\sim$85%) degrade to below 25%. These results suggest that capabilities demonstrated in simplified synthetic environments fail to transfer effectively to the complex, data-driven scenarios inherent in real-world visualization tasks.

6 Conclusion and Future Works

In this work, we addressed the lack of systematic evaluation for Vision Language Models (VLMs) on realistic and complex data visualization tasks. We introduced RealChart2Code, a large-scale benchmark grounded in authentic datasets that assesses capabilities across chart replication, reproduction from raw data, and iterative refinement. Our comprehensive evaluation of 14 leading models demonstrates that while current LLMs are capable of simple plotting, they suffer significant performance degradation when handling complex multi-panel layouts and real-world data. Specifically, we identified a distinct capability gap: proprietary models demonstrate superior visual reasoning, whereas open-weight models frequently struggle with syntax and spatial logic. In future research, we aim to develop automated pipelines for generating high-quality synthetic data to address the scarcity of complex training examples and improve model generalization on intricate layouts.

7 Limitations

However, we acknowledge limitations in scope and evaluation. First, our code implementations are currently confined to Matplotlib. While this scope is specific, the granular and imperative nature of Matplotlib code effectively reflects a model's fundamental visualization understanding and logical reasoning capabilities, serving as a robust proxy for general plotting skills. Second, although our MLLM-based judges show strong correlation with human experts, they may still fail to detect subtle visual artifacts, such as minor element overlaps or precise color nuances. These areas offer promising directions for future research in more fine-grained visual evaluation.

8 Acknowledgments

This work is supported by National Key Research and Development Program (2023YFC3305203), National Natural Science Foundation of China (92570204, 62576339)

References

Anthropic. 2023. Introducing Claude.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, and 1 others. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Ruisheng Cao, Mouxiang Chen, Jiawei Chen, Zeyu Cui, Yunlong Feng, Binyuan Hui, Yuheng Jing, Kaixin Li, Mingze Li, Junyang Lin, and 1 others. 2026. Qwen3-coder-next technical report. *arXiv preprint arXiv:2603.00729*.

Ge Chang, Jinbo Su, Jiacheng Liu, Pengfei Yang, Yuhao Shang, Huiwen Zheng, Hongli Ma, Yan Liang, Yuanchun Li, and Yunxin Liu. 2025. Grail: Learning to interact with large knowledge graphs for retrieval augmented reasoning. *arXiv preprint arXiv:2508.05498*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021a. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021b. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Qiaosheng Chen, Yang Liu, Lei Li, Kai Chen, Qipeng Guo, Gong Cheng, and Fei Yuan. 2025. Interactscience: Programmatic and visually-grounded evaluation of interactive scientific demonstration code generation. *arXiv preprint arXiv:2510.09724*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

DeepSeek-AI and etc. 2024. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Yangruibo Ding, Zijian Wang, Wasi Uddin Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. 2023. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Chengyu Fang, Heng Guo, Zheng Jiang, Chunming He, Xiu Li, and Minfeng Xu. 2026. Photon: Speedup volume understanding with efficient multimodal large language models. *arXiv preprint arXiv:2603.25155*.

Timur Galimzyanov, Sergey Titov, Yaroslav Golubev, and Egor Bogomolov. 2025. Drawing pandas: A benchmark for llms in generating plotting code. *Preprint*, arXiv:2412.02764.

Linyuan Gong, Sida Wang, Mostafa Elhoushi, and Alvin Cheung. 2024. Evaluation of llms on syntax-aware code fill-in-the-middle tasks. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, and 1 others. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, and 1 others. 2025. Glm-4.1 v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *arXiv preprint arXiv:2507.01006*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Tingfeng Hui, Pengyu Zhu, Bowen Ping, Ling Tang, Guanting Dong, Yaqi Zhang, and Sen Su. 2025. Decif: Improving instruction-following through meta-decomposition. *arXiv preprint arXiv:2505.13990*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *Preprint*, arXiv:2403.07974.

Caijun Jia, Nan Xu, Jingxuan Wei, Qingli Wang, Lei Wang, Bihui Yu, and Junnan Zhu. 2025. Chartreasoner: Code-driven modality bridging for long-chain reasoning in chart question answering. *Preprint*, arXiv:2506.10116.

Zheng Jiang, Heng Guo, Chengyu Fang, Changchen Xiao, Xinyang Hu, Lifeng Sun, and Minfeng Xu. 2026. Medvr: Annotation-free medical visual reasoning via agentic reinforcement learning. In *The Fourteenth International Conference on Learning Representations*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Kaggle. 2025. Kaggle. Accessed: 2025-12-30.

Shankar Kantharaj, Rixie Tiffany Ko Leong, Xiang Lin, and 1 others. 2022. Chart-to-text: A large-scale benchmark for chart summarization. *arXiv preprint arXiv:2203.06486*.

Yuying Li, Siyi Qian, Hao Liang, Leqi Zheng, Ruichuan An, Yongzhen Guo, and Wentao Zhang. 2025. Capgeo: A caption-assisted approach to geometric reasoning. *arXiv preprint arXiv:2510.09302*.

Zekun Li, Xianjun Yang, Kyuri Choi, Wanrong Zhu, Ryan Hsieh, HyeonJung Kim, Jin Hyuk Lim, Sungyoung Ji, Byungju Lee, Xifeng Yan, and 1 others. 2024. Mmsci: A multimodal multi-discipline dataset for phd-level scientific comprehension. *arXiv preprint arXiv:2407.04903*.

Haoyu Lu, Wen Liu, Bo Zhang, Bingxuan Wang, Kai Dong, Bo Liu, Jingxiang Sun, Tongzheng Ren, Zhuoshu Li, Hao Yang, and 1 others. 2024. Deepseek-vl: towards real-world vision-language understanding. *arXiv preprint arXiv:2403.05525*.

Zimu Lu, Yunqiao Yang, Houxing Ren, Haotian Hou, Han Xiao, Ke Wang, Weikang Shi, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. 2025. Webgen-bench: Evaluating llms on generating interactive and functional websites from scratch. *Preprint*, arXiv:2505.03733.

Sijia Luo, Xiaokang Zhang, Yuxuan Hu, Bohan Zhang, Ke Wang, Jinbo Su, Mengshu Sun, Lei Liang, and Jing Zhang. 2026. Sparse-rl: Breaking the memory wall in llm reinforcement learning via stable sparse rollouts. *arXiv preprint arXiv:2601.10079*.

Tianqi Luo, Chuhan Huang, Leixian Shen, Boyan Li, Shuyu Shen, Wei Zeng, Nan Tang, and Yuyu Luo. 2025. nvbench 2.0: Resolving ambiguity in text-to-visualization through stepwise reasoning. *Preprint*, arXiv:2503.12880.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. 2024. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems*, 37:94871–94908.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

MistralAI. 2024. Codestral. `https://mistral.ai/news/codestral`. 2024.05.29.

Yuansheng Ni, Ping Nie, Kai Zou, Xiang Yue, and Wenhu Chen. 2025. Viscoder: Fine-tuning llms for executable python visualization code generation. *Preprint*, arXiv:2506.03930.

OpenAI. 2022. ChatML.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

OpenAI. 2025. Gpt-5 system card. Technical report.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2025. Training software engineering agents and verifiers with swe-gym. *Preprint*, arXiv:2412.21139.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, and 1 others. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR.

Raian Rahman, Rizvi Hasan, Abdullah Al Farhad, and 1 others. 2023. Chartsumm: A comprehensive benchmark for automatic chart summarization of long and short summaries. *arXiv preprint arXiv:2304.13620*.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, and 1 others. 2023. Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

KaShun Shum, Binyuan Hui, Jiawei Chen, Lei Zhang, Jiaxi Yang, Yuzhen Huang, Junyang Lin, Junxian He, and 1 others. 2025. Swe-rm: Execution-free feedback for software engineering agents. *arXiv preprint arXiv:2512.21919*.

Jinbo Su, Lingzhe Gao, Wei Li, Shihao Liu, Haojie Lei, Xinyi Wang, Yuanzhao Guo, Ke Wang, Daiting Shi, and Dawei Yin. 2025. Racqc: Advanced retrieval-augmented generation for chinese query correction. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 675–689.

Jinbo Su, Yuxuan Hu, Cuiping Li, Hong Chen, Jia Li, Lintao Ma, and Jing Zhang. 2026. Tablecache: Primary foreign key guided kv cache precomputation for low latency text-to-sql. *arXiv preprint arXiv:2601.08743*.

5 Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, and 152 others. 2025a. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *Preprint*, arXiv:2508.06471.

Gemini Team. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *Preprint*, arXiv:2403.05530.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, and 150 others. 2025b. Kimi k2: Open agentic intelligence. *Preprint*, arXiv:2507.20534.

Kimi Team, Angang Du, Bohong Yin, Bowei Xing, Bowen Qu, Bowen Wang, Cheng Chen, Chenlin Zhang, Chenzhuang Du, Chu Wei, and 1 others. 2025c. Kimi-vl technical report. *arXiv preprint arXiv:2504.07491*.

Qwen Team. 2025. Qwen3-coder: Agentic coding in the world. *Blog post*.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, and 1 others. 2025. Internvl3. 5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *arXiv preprint arXiv:2508.18265*.

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*.

Yuan Xin, Dingfan Chen, Linyi Yang, Michael Backes, and Xiao Zhang. 2025. Jailbreaking attacks vs. content safety filters: How far are we in the llm safety arms race? *arXiv preprint arXiv:2512.24044*.

Ximing Xing, Juncheng Hu, Guotao Liang, Jing Zhang, Dong Xu, and Qian Yu. 2025. Empowering llms to understand and generate complex vector graphics. *Preprint*, arXiv:2412.11102.

Kai Xu, YiWei Mao, XinYi Guan, and ZiLong Feng. 2025. Web-bench: A llm code benchmark based on web standards and frameworks. *Preprint*, arXiv:2505.07473.

Shannan Yan, Jingchen Ni, Leqi Zheng, Jiajun Zhang, Peixi Wu, Dacheng Yin, Jing Lyu, Chun Yuan, and Fengyun Rao. 2026a. Adamem: Adaptive user-centric memory for long-horizon dialogue agents. *arXiv preprint arXiv:2603.16496*.

Shannan Yan, Leqi Zheng, Keyu Lv, Jingchen Ni, Hongyang Wei, Jiajun Zhang, Guangting Wang, Jing Lyu, Chun Yuan, and Fengyun Rao. 2026b. Learning cross-view object correspondence via cycle-consistent mask prediction. *arXiv preprint arXiv:2602.18996*.

Cheng Yang, Chufan Shi, 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. 2025. Chartmimic: Evaluating lmm's cross-modal reasoning capability via chart-to-code generation. *Preprint*, arXiv:2406.09961.

Jian Yang, Jiajun Zhang, Jiaxi Yang, Ke Jin, Lei Zhang, Qiyao Peng, Ken Deng, Yibo Miao, Tianyu Liu, Zeyu Cui, and 1 others. 2024. Execrepobench: Multi-level executable code completion evaluation. *arXiv preprint arXiv:2412.11990*.

Xingchen Zeng, Haichuan Lin, Yilin Ye, and Wei Zeng. 2024. Advancing multimodal large language models in chart question answering with visualization-referenced instruction tuning. *IEEE Transactions on Visualization and Computer Graphics*.

Jiajun Zhang, Zeyu Cui, Jiaxi Yang, Lei Zhang, Yuheng Jing, Zeyao Ma, Tianyi Bai, Zilei Wang, Qiang Liu, Liang Wang, and 1 others. 2026a. From completion to editing: Unlocking context-aware code infilling via search-and-replace instruction tuning. *arXiv preprint arXiv:2601.13384*.

Jiajun Zhang, Jianke Zhang, Zeyu Cui, Jiaxi Yang, Lei Zhang, Binyuan Hui, Qiang Liu, Zilei Wang, Liang Wang, and Junyang Lin. 2025a. Plotcraft: Pushing the limits of llms for complex and interactive data visualization. *arXiv preprint arXiv:2511.00010*.

Lei Zhang, Jiaxi Yang, Min Yang, Jian Yang, Mouxiang Chen, Jiajun Zhang, Zeyu Cui, Binyuan Hui, and Junyang Lin. 2025b. Swe-flow: Synthesizing software engineering data in a test-driven manner. *arXiv preprint arXiv:2506.09003*.

Qianchi Zhang, Hainan Zhang, Liang Pang, Yongxin Tong, Hongwei Zheng, and Zhiming Zheng. 2026b. Less is more: Compact clue selection for efficient retrieval-augmented generation reasoning. In *Proceedings of the ACM Web Conference 2026*, pages 1971–1982.

Qianchi Zhang, Hainan Zhang, Liang Pang, Hongwei Zheng, and Zhiming Zheng. 2026c. Stable-rag: Mitigating retrieval-permutation-induced hallucinations in retrieval-augmented generation. *arXiv preprint arXiv:2601.02993*.

Xiaokang Zhang, Sijia Luo, Bohan Zhang, Zeyao Ma, Jing Zhang, Yang Li, Guanlin Li, Zijun Yao, Kangli Xu, Jinchang Zhou, and 1 others. 2025c. Tablellm: Enabling tabular data manipulation by llms in real office usage scenarios. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 10315–10344.

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

Leqi Zheng, Jiajun Zhang, Canzhi Chen, Chaokun Wang, Hongwei Li, Yuying Li, Yaoxin Mao, Shannan Yan, Zixin Song, Zhiyuan Feng, and 1 others. 2026. What should i cite? a rag benchmark for academic citation prediction. *arXiv preprint arXiv:2601.14949*.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, and 14 others. 2025. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *Preprint*, arXiv:2406.15877.

Appendix

A Benchmark Details

A.1 Chart Tyeps

RealChart2Code encompasses a diverse collection of 50 chart types to ensure comprehensive coverage of real-world visualization scenarios. Table 5 presents the frequency distribution of these chart types across all tasks in the benchmark. To avoid inflated counts from task repetition, each unique visualization task is counted only once, as the same task set is used across both single-turn and multi-turn interactions. The distribution reflects the natural occurrence of chart types in practical applications, with fundamental visualization types (e.g., line charts, scatter plots, bar charts) appearing more frequently, while specialized techniques (e.g., parallel coordinates, dendrograms, sankey diagrams) are represented in proportion to their real-world usage. This balanced yet realistic distribution ensures that models are evaluated on both common and advanced visualization capabilities.

A.2 Render Envirnment

To ensure the safe, consistent, and reproducible execution of all model-generated code, we constructed a dedicated sandboxed environment containerized using Docker. This environment, based on Python 3.13, provides an isolated and standardized platform pre-installed with a comprehensive suite of libraries for data analysis and visualization. Foundational libraries include Pandas for data manipulation and NumPy for numerical operations. For visualization, the environment is equipped with Matplotlib as the primary plotting library, complemented by specialized libraries such as Seaborn for high-level statistical graphics, Plotly for interactive charts, Squarify for treemaps, and others like scikit-learn and statsmodels to support diverse charting requirements. All evaluations are conducted on a server with 128 CPU cores and 1024 GB of RAM, where each code execution runs within its container without network access and is subject to a 120 seconds timeout. This fully-specified setup eliminates system variability and ensures a fair and secure assessment of each model's capabilities.

B Benchmark Data Curation Details

B.1 Design Principles

The design of `RealChart2Code` is driven by the necessity to evaluate Vision Language Models (VLMs) not just on their ability to generate code, but on their capacity to perceive, interpret, and reconstruct complex visual data. Our curation process is guided by four core principles:

- **Grounded in Authentic Data**: Unlike benchmarks that rely on synthetic or simplified data, `RealChart2Code` is constructed entirely from real-world datasets sourced from Kaggle. This ensures that models are tested against the noise, scale, and irregularity inherent in actual data science workflows. The visualizations reflect genuine analytical intent, moving beyond arbitrary plotting to meaningful data storytelling.

- **Visual-Grounded Reverse Engineering**: Distinct from text-to-visualization tasks where models generate charts from open-ended prompts, our benchmark focuses on *Chart Replication* and *Reproduction*. This requires the model to strictly adhere to a visual reference, testing its ability to translate pixel-level visual information (layout, styling, color mapping) into executable matplotlib code with high fidelity.

- **Complexity and Diversity**: We explicitly target the "complexity gap" in current evaluations. The benchmark includes a wide spectrum of chart types (over 50 distinct types) and complex composite layouts (e.g., multi-panel figures, dual-axis plots). This compositional complexity probes the model's spatial reasoning and logical understanding of library constraints.

- **Iterative Refinement Capability**: Real-world coding is rarely a one-shot process. By incorporating a dedicated *Chart Refinement* module based on error injection, we assess the model's ability to diagnose issues, understand user feedback, and modify existing code without breaking its functionality—a critical skill for collaborative AI assistants.

B.2 Data Collection and Filtering

The foundation of `RealChart2Code` is a rigorous data selection pipeline designed to filter massive open-source repositories into a high-quality curation. We utilized Kaggle as our primary source, strictly adhering to scientific research licensing terms. The process was executed in two phases:

Table 5: Frequency distribution of 50 chart types in RealChart2Code. Each task is counted once to prevent inflation from multi-turn repetitions. The distribution reflects natural occurrence patterns in real-world visualization applications.

Type	Count	Type	Count	Type	Count	Type	Count
Line Chart	845	Scatter Plot	782	Bar Chart	723	Time Series Plot	689
Area Chart	521	Histogram	487	Box Plot	456	Violin Plot	398
Heatmap	374	Stacked Bar Chart	342	Grouped Bar Chart	318	Stacked Area Chart	295
Bubble Chart	267	Pie Chart	243	Error Bar Plot	231	Density Plot	218
Contour Plot	195	Radar Chart	187	Waterfall Chart	176	Network Graph	164
Parallel Coordinates	152	Sankey Diagram	143	Treemap	138	Sunburst Chart	127
Ridgeline Plot	119	Slope Chart	112	Stream Graph	105	Dendrogram	98
Funnel Chart	91	Candlestick Chart	87	Gantt Chart	82	Chord Diagram	76
Alluvial Diagram	71	Population Pyramid	67	Marimekko Chart	63	Lollipop Chart	58
Dumbbell Plot	54	Pair Plot	49	Dot Plot	45	Word Cloud	41
Waffle Chart	38	2D Histogram	34	Hexbin Plot	31	Jitter Plot	27
Joy Plot	24	Autocorrelation Plot	21	Ternary Plot	18	Circular Barplot	15
Polar Chart	12	Stripplot	9				

Phase 1: Automated Large-Scale Screening We initiated the process with a pool of over 8,000 candidate datasets, comprising more than 100,000 files and roughly 30 billion data rows. To ensure data quality and relevance, we applied a quantitative filtering mechanism based on community engagement metrics. Datasets were ranked and filtered according to vote counts, download frequency, and official usability ratings. This step eliminated incomplete, poorly documented, or trivial datasets.

Phase 2: Expert Curation From the pre-screened pool, we conducted a manual review to select datasets suitable for complex visualization tasks. The selection criteria focused on data richness (e.g., sufficient columns for multi-dimensional plotting) and domain diversity. This resulted in a final selection of 1,036 high-quality datasets. The final curated collection comprises 3,271 raw data files with approximately 860 million rows, providing a robust testbed for data-intensive generation tasks.

B.3 Visualization Task Design

Using the 1,036 curated datasets, we constructed 1,016 unique visualization scenarios. To ensure comprehensive coverage of analytical tasks, the design process was guided by a taxonomy of 7 high-level visualization intents and over 50 distinct chart types.

The taxonomy of intents includes:

- **Correlation**: Tasks involving Scatter Plots, Bubble Charts, Pairwise Plots, and Heatmaps to show relationships between variables.
- **Deviation**: Visualizations such as Diverging Bar Charts and Area Charts to highlight variations from a baseline.
- **Ranking**: Ordered Bar Charts, Lollipop Charts, and Slope Charts to display comparative rankings.
- **Distribution**: Histograms, Box Plots, Violin Plots, and Density Plots to analyze data spread.
- **Composition**: Stacked Bar Charts, Treemaps, and Pie Charts to show part-to-whole relationships.
- **Change**: Line Charts, Time Series Decompositions, and Area Charts to visualize trends over time.
- **Groups**: Cluster Plots and Parallel Coordinates to reveal cluster structures within data.

These 1,016 visualizations form the basis for the *Chart Replication* (Image $\rightarrow$ Code) and *Chart Reproduction* (Image + Data $\rightarrow$ Code) tasks, totaling 2,032 instances.

B.4 Ground-Truth Code Implementation

To establish a high-quality "Gold Standard," we avoided using model-generated code as ground truth. Instead, an in-house team of five expert Python developers manually implemented the solution code for all 1,016 visualizations. The implementation strictly utilizes `matplotlib` and standard data processing libraries (e.g., `pandas`, `numpy`). The code was rigorously tested to ensure it

is: ❶ Executable: Runs error-free in the evaluation sandbox. ❷ Reproducible: Accurately regenerates the target visualization from the provided data. ❸ Idiomatic: Follows standard coding practices, ensuring the benchmark evaluates the generation of maintainable and clean code.

B.5 Error Injection and Refinement Tasks

To construct the *Chart Refinement* subset, we manually introduced errors into a selected subset of the ground-truth implementations. This process generated 864 distinct refinement tasks. The injected errors cover a diverse range of categories to simulate real-world debugging scenarios:

- **Visual/Stylistic Errors**: Overlapping elements, incorrect color schemes, missing legends, or illegible labels.
- **Data Mapping Errors**: Assigning incorrect columns to axes, wrong aggregation methods, or incorrect data filtering.
- **Chart Type Errors**: Using a suboptimal or incorrect chart type for the given data type (e.g., using a bar chart for continuous time-series data).

In these tasks, the model is presented with the "flawed" code and the resulting incorrect chart, along with a natural language instruction to correct the issue, requiring multi-turn reasoning and code editing capabilities.

B.6 Quality Control

To guarantee the robustness and reliability of `RealChart2Code`, we implemented a strict multi-stage quality control protocol involving peer review, automated execution checks, and consensus-based adjudication.

Ground-Truth Code Verification Given that our benchmark relies on human-authored code as the gold standard, ensuring the correctness and quality of these scripts is paramount. We established a **Cross-Validation Peer Review Workflow**:

1. **Execution Sanity Check**: All ground-truth scripts were first run in our standardized sandbox environment. Scripts that failed to execute, produced warnings, or timed out were automatically flagged for revision.
2. **Visual Fidelity Review**: Each successful visualization was inspected by a secondary expert who did not author the code. This reviewer verified the chart against a three-point checklist: (1) *Data Accuracy* (does the chart correctly represent the underlying raw data?), (2) *Visual Clarity* (are labels, legends, and layouts legible and overlap-free?), and (3) *Idiomatic Coding* (does the implementation follow standard Matplotlib best practices?).
3. **Adjudication**: Any code flagged as suboptimal during peer review was returned to the original author for correction. If a disagreement persisted regarding the implementation style, a senior lead made the final decision to ensure consistency across the benchmark.

Multi-turn Refinement Verification For the Chart Refinement tasks, ensuring the logical consistency between the intentionally flawed code, the rendered image, and the correction instruction was critical. We employed a **Triple-Verification Strategy** similar to the data curation phase:

1. **Triplet Validation**: Each refinement instance consists of a triplet: (Flawed Code/Image, User Instruction, Corrected Ground Truth). These triplets were reviewed by independent annotators.
2. **Logic & Consistency Check**: Annotators verified two specific conditions: First, *Error Visibility*—confirming that the injected error (e.g., "incorrect axis range") was clearly distinguishable in the rendered image. Second, *Solvability*—ensuring that the user instruction provided sufficient information for a model to deduce the correct fix without ambiguity.
3. **Majority Vote & Consensus**: A task was only accepted if it received a unanimous "Pass" vote. In cases where the error was deemed too subtle or the instruction too vague (partial agreement), the task underwent a consensus discussion phase to polish the prompt. Triplets failing to reach consensus were discarded to prevent model confusion.

This rigorous verification loop ensures that `RealChart2Code` provides a fair, noise-free, and logically sound evaluation environment for multi-modal code generation.

C Evaluation Details

C.1 Evaluation Metrics

This section details the comprehensive scoring rubric used to evaluate the generated visualizations across all three tasks. Our evaluation framework comprises two primary dimensions: **Visual Structure Alignment** for assessing fidelity to reference charts, and **Data Alignment** for verifying computational correctness in code reproduction.

C.1.1 Chart Replication and Chart Refinement Tasks

For Chart Replication (Task 1) and Chart Refinement (Task 3), evaluation focuses on visual structure alignment with the reference chart. Each metric uses a 3-level scoring system (0/1/2).

Visual Structure Alignment (3-Level Scoring: 0/1/2) Assess whether the generated chart replicates the structural elements of the reference chart with precision.

1. **Chart Type Consistency**
   - *Question:* Does the generated chart use identical chart types as the reference for all visualizations (e.g., line, bar, scatter, pie, heatmap)?
   - *Scoring Criteria:*

     Score 2 All chart types match exactly across all subplots, including primary chart types and any overlaid secondary elements (e.g., dual-axis charts, combination charts).

     Score 1 Primary chart types match for all subplots, but minor secondary elements differ (e.g., missing auxiliary bar overlay on a line chart, absent trendline, missing error bars on primary plot).

     Score 0 Any primary chart type differs (e.g., bar chart used where line chart required), or multiple subplots have type mismatches, or fundamental chart category is wrong (e.g., scatter instead of bar).

2. **Spatial Layout Consistency**
   - *Question:* Does the generated chart replicate the reference chart's exact component arrangement (subplot grid dimensions, relative positioning of all visual elements)?
   - *Scoring Criteria:*

     Score 2 Exact grid structure (e.g., 2×3 layout), identical subplot positioning, precise relative placement of all elements (legends, annotations, insets) within ±2% of reference dimensions.

     Score 1 Correct grid structure and subplot count, but minor positioning deviations (legend slightly displaced, annotation shifted but still in correct subplot, spacing variations within 5-15% of reference).

     Score 0 Incorrect grid dimensions (e.g., 1×4 instead of 2×2), wrong number of subplots, major element misplacement (legend in wrong subplot, annotations in incorrect panels), or spatial relationships fundamentally altered.

3. **Text Element Consistency**
   - *Question:* Does the generated chart contain all textual content from the reference (titles, subtitles, annotations, axis labels) with identical wording, excluding axis tick values?
   - *Scoring Criteria:*

     Score 2 All text content matches exactly in wording, placement, and hierarchy (main title, subplot titles, axis labels, annotations, data labels).

     Score 1 All critical text present (main title, axis labels, subplot titles) with exact wording, but minor discrepancies in secondary elements (1-2 missing annotations, slightly reworded footnotes, minor label omissions that do not affect core interpretation).

     Score 0 Missing or altered critical text (wrong/missing main title, incorrect axis labels, absent subplot titles, 3+ missing annotations), or significant semantic changes to any primary text element.

4. **Axis Configuration Consistency**
   - *Question:* Does the generated chart replicate all axis properties from the reference (variable names, units, scale types, ranges, tick intervals) and legend specifications for each subplot?
   - *Scoring Criteria:*

Score 2 Perfect match for all axis properties (scale type, range, units, label text, tick intervals) and legend specifications (position, order, labels) across all subplots.

Score 1 Axis variable names and scale types correct, but minor deviations (axis range extends ±10-20% beyond reference, tick intervals slightly different but reasonable, legend order swapped but all entries present, units abbreviated differently but semantically equivalent).

Score 0 Incorrect axis scale type (log vs. linear), wrong variable assignments, axis range drastically different (>30% deviation or missing critical data region), missing axis labels, incorrect units, or legend missing critical entries.

5. **Color Scheme Consistency**

- *Question:* Does the generated chart apply the same color mappings as the reference for all data series, categories, and visual elements?
- *Scoring Criteria:*

Score 2 Identical color mappings for all elements (data series, categorical encodings, heatmap scales, background, grid, annotations) with exact hex/RGB matches or perceptually indistinguishable equivalents.

Score 1 Correct color palette family and mapping logic, but minor shade variations (slightly lighter/darker versions of reference colors, saturation differences within 15%, 1-2 secondary elements use similar but non-exact colors).

Score 0 Different color palette applied, incorrect categorical color assignments (e.g., Category A colored red instead of blue), reversed color scale (e.g., heatmap scale inverted), or 3+ elements with significantly different colors.

6. **Style and Format Consistency**

- *Question:* Does the generated chart match the reference chart's stylistic attributes (background color, grid styles, marker shapes, line patterns, font families, border styles)?
- *Scoring Criteria:*

Score 2 All stylistic attributes match (grid style [solid/dashed/dotted], marker shapes, line widths, line patterns, font families, background colors, border styles, opacity levels).

Score 1 Core visual style maintained with minor deviations (grid style slightly different [dashed vs. dotted], marker shapes similar but not exact [circle vs. hexagon], line width within ±1px, font family same category [all sans-serif] but different face).

Score 0 Multiple style mismatches (wrong grid style + wrong markers + wrong font category), fundamentally different aesthetic (dark theme vs. light theme), or critical style elements missing (no gridlines when reference has them, no borders when required).

7. **Component Completeness**

- *Question:* Does the generated chart include all visual components present in the reference chart?
- *Scoring Criteria:*

Score 2 Chart contains all visual components from the reference including all data series, markers, annotations, grid elements, legends, statistical overlays (trendlines, confidence intervals, reference lines), decorative features, and auxiliary elements.

Score 1 All primary components present (all main data series, essential legends, critical annotations, primary axis elements, key statistical markers) but missing 1-3 minor elements (optional secondary annotations, decorative styling features, non-essential reference lines, auxiliary data labels on individual points).

Score 0 Missing critical components (1+ major data series absent, essential legend missing, key annotations

omitted, primary axis elements incomplete, important statistical overlays missing) that significantly affect information completeness or interpretation accuracy.

8. **Data Pattern Consistency**

   - *Question:* Does the generated chart visually replicate the reference chart's data patterns (data point positions, trend shapes, distribution profiles, statistical markers)?
   - *Scoring Criteria:*

     Score 2 Data patterns match reference with >95% positional accuracy, all trend shapes replicated, distribution profiles visually identical, all statistical markers (means, medians, confidence intervals) present and correct.

     Score 1 Core data trends and patterns recognizable (correlation directions correct, major peaks/troughs present, distribution shapes similar), but minor deviations (5-15% positional variance, slight smoothing differences, 1-2 missing minor statistical markers, interpolation artifacts in small regions).

     Score 0 Data patterns significantly diverged (>15% positional errors, trend directions incorrect, distribution shapes fundamentally different, major statistical markers missing/wrong), or substantial data missing/added.

C.1.2 Chart Reproduction Task

For Chart Reproduction (Task 2), the **Data Pattern Consistency** metric is replaced with **Data Alignment**, which performs code-level verification to ensure computational correctness rather than visual similarity.

Data Alignment (3-Level Scoring: 0/1/2) Verify that the generated code produces computationally equivalent data transformations and mappings as the reference implementation.

1. **Data Source Matching**

   - Both codes must load data from the same file(s).
   - Column selections must access the same fields (even if using different syntax).
   - *Example:* `df['col']` ≡ `df.col` ≡ `df.loc[:, 'col']`

2. **Data Transformation Equivalence**

   - For each subplot, check if the **final data arrays** passed to plotting functions are equivalent.
   - *Accept as equivalent:*
     - Different groupby syntax producing same result: `df.groupby('A')['B'].sum()` ≡ `df.groupby('A').agg({'B':'sum'})['B']`
     - Different filtering methods: `df[df['x'] > 0]` ≡ `df.query('x > 0')` ≡ `df.loc[df['x'] > 0]`
     - Reordered operations that do not affect output: `df.sort_values('A').reset_index(drop=True)` ≡ `df.reset_index(drop=True).sort_values('A')`
     - Different intermediate variable names
     - Different but equivalent aggregation functions on same data
   - *Reject as different:*
     - Using different columns: `df['A']` vs. `df['B']`
     - Different aggregation types: `.sum()` vs. `.mean()`
     - Different filters: `df[df['x'] > 0]` vs. `df[df['x'] > 5]`
     - Different data subsets for same visual element
     - Missing or extra data transformations that change values

3. **Visual Element Data Mapping**

   - For each subplot, verify corresponding visual elements use equivalent data.
   - *Scoring Criteria:*

     Score 2 All data transformations are computationally equivalent, and all visual elements map to identical data arrays.

Score 1 Core data transformations correct with minor acceptable variations (different but equivalent filtering logic, reordered operations producing same output, 1-2 minor data processing steps differ but final arrays match).

Score 0 Data transformations produce different results (different columns used, different aggregations applied, filters produce different subsets), or visual elements map to non-equivalent data arrays.

Note: All other Visual Structure Alignment metrics (Chart Type Consistency, Spatial Layout Consistency, Text Element Consistency, Axis Configuration Consistency, Color Scheme Consistency, Style and Format Consistency, and Component Completeness) apply identically to the Chart Reproduction task as described above.

C.2 Model Details

D Evaluation Prompts

Generation Prompt for Chart Replication

Role:
You are an expert Python data visualization engineer specializing in matplotlib and seaborn.
Task:
Generate executable Python code that precisely replicates the provided reference chart image with pixel-level fidelity.
Requirements:
Your code must exactly replicate:

1. **Chart Structure:**
   - Chart types
   - Subplot layout
2. **Data Representation:**
   - Axis properties (labels, units, ranges, scales, ticks)
   - Data values and trends
   - Legends (content, position, style)
3. **Visual Styling:**
   - Colors (all elements)
   - Typography (fonts, sizes)
   - Line/marker styles (patterns, shapes, widths)
   - Grids and backgrounds
4. **Text Content:**
   - Titles, annotations, labels (exact wording and placement)

Code Standards:

- Use matplotlib and/or seaborn exclusively
- Ensure code is self-contained and executable
- Generate synthetic data matching reference characteristics

Output Format:
Provide only the Python code block:

```
```python
Your code here
```
```

Evaluation Prompt for Chart Replication

Role and Goal:
You are an expert data visualization quality assessor. Your task is to rigorously evaluate Chart B (generated output) against Chart A (reference chart) based on the original task requirements. The evaluation must assess fidelity in visual structure, data representation, and design execution with strict adherence to standards.
Evaluation Categories:
1. Visual Structure Alignment (3-Level Scoring: 0/1/2)
Assess whether Chart B replicates the structural elements of Chart A with precision.

1. **Chart Type Consistency**
   - **Question**: Does Chart B use identical chart types as Chart A for all visualizations?
   - **Scoring Criteria**:
     - **Score 2**: All chart types match exactly
     - **Score 1**: Primary chart types match

Model	Version/HF Checkpoint	Release Time	Source
Proprietary Multimodal Large Language Models			
GPT-5.1 (OpenAI, 2025)	gpt-5-1-2025-11-13	2025-11-13	https://openai.com/zh-Hans-CN/index/gpt-5-1/
Claude 4.5 Sonnet (Anthropic, 2023)	claude-4-5-sonnet-20250929	2025-09-29	https://www.anthropic.com/news/claude-sonnet-4-5
Claude 4.5 Opus (Anthropic, 2023)	claude-4-5-Opus-20251101	2025-11-01	https://www.anthropic.com/news/claude-opus-4-5
Gemini-2.5-Flash (Comanici et al., 2025)	gemini-2-5-flash-20250617	2025-06-17	https://deepmind.google/models/gemini/flash/
Gemini-3-Pro-Preivew (Comanici et al., 2025)	gemini-3-pro-preview	2025-11-18	https://deepmind.google/models/gemini/pro/
Open-Source Multimodal Large Language Models			
Qwen3-VL-30B (Bai et al., 2025)	Qwen/Qwen3-VL-30B-A3B-Instruct	2025-10-10	https://huggingface.co/Qwen/Qwen3-VL-30B-A3B-Instruct
Qwen3-VL-235B (Bai et al., 2025)	Qwen/Qwen3-VL-235B-A22B-Instruct	2025-10-10	https://huggingface.co/Qwen/Qwen3-VL-235B-A22B-Instruct
Deepseek-VL-7B (Lu et al., 2024)	deepseek-ai/deepseek-vl-7b-base	2024-03-09	https://huggingface.co/deepseek-ai/deepseek-vl-7b-base
MiMo-VL-7B-RL (Team et al., 2025c)	XiaomiMiMo/MiMo-VL-7B-RL-2508	2025-08-10	https://huggingface.co/XiaomiMiMo/MiMo-VL-7B-RL-2508
GLM-4.1V-9B (Hong et al., 2025)	zai-org/GLM-4.1V-9B-Thinking	2025-07-30	https://huggingface.co/zai-org/GLM-4.1V-9B-Thinking
GLM-4.5V-106B (Hong et al., 2025)	zai-org/GLM-4.5V	2025-07-30	https://huggingface.co/zai-org/GLM-4.5V
Intern-VL 3.5 30B (Wang et al., 2025)	OpenGVLab/InternVL3_5-30B-A3B	2025-08-25	https://huggingface.co/OpenGVLab/InternVL3_5-30B-A3B
Intern-VL 3.5 241B (Wang et al., 2025)	OpenGVLab/InternVL3_5-241B-A28B	2025-08-25	https://huggingface.co/OpenGVLab/InternVL3_5-241B-A28B

Table 6: Model details for all models. The release time and model source information are provided for reference.

  - **Score 0**: Any primary chart type differs

2. **Spatial Layout Consistency**
   - **Question**: Does Chart B replicate Chart A's exact component arrangement?
   - **Scoring Criteria**:
     - **Score 2**: Exact grid structure and positioning
     - **Score 1**: Correct grid structure with minor deviations
     - **Score 0**: Incorrect grid dimensions or major misplacement

3. **Text Element Consistency**
   - **Question**: Does Chart B contain all textual content from Chart A?
   - **Scoring Criteria**:
     - **Score 2**: All text content matches exactly
     - **Score 1**: Critical text present with exact wording
     - **Score 0**: Missing or altered critical text

4. **Axis Configuration Consistency**
   - **Question**: Does Chart B replicate all axis properties from Chart A?
   - **Scoring Criteria**:
     - **Score 2**: Perfect match for all axis properties
     - **Score 1**: Axis variable names and scale types correct
     - **Score 0**: Incorrect axis scale type or range

5. **Color Scheme Consistency**
   - **Question**: Does Chart B apply the same color mappings as Chart A?
   - **Scoring Criteria**:
     - **Score 2**: Identical color mappings for all elements
     - **Score 1**: Correct palette family with minor variations
     - **Score 0**: Different color palette applied

6. **Style and Format Consistency**
   - **Question**: Does Chart B match Chart A's stylistic attributes?
   - **Scoring Criteria**:
     - **Score 2**: All stylistic attributes match
     - **Score 1**: Core visual style maintained
     - **Score 0**: Multiple style mismatches

7. **Data Pattern Consistency**

- **Question**: Does Chart B visually replicate Chart A's data patterns?
- **Scoring Criteria**:
  - **Score 2**: Data patterns match with >95% accuracy
  - **Score 1**: Core data trends recognizable
  - **Score 0**: Data patterns significantly diverged

8. **Component Completeness**
   - **Scoring Criteria**:
     - **Score 2**: All visual components present
     - **Score 1**: All primary components present
     - **Score 0**: Missing critical components

2. Execution Quality (3-Level Scoring: 0/1/2)
Evaluate the technical execution quality of Chart B independently.

1. **Visual Clarity**
   - **Scoring Criteria**:
     - **Score 2**: Zero overlap affecting readability
     - **Score 1**: Minor overlap limited to non-critical elements
     - **Score 0**: Critical overlap affecting primary elements
2. **Compositional Balance**
   - **Scoring Criteria**:
     - **Score 2**: Optimal spatial distribution
     - **Score 1**: Adequate composition with minor issues
     - **Score 0**: Poor composition with disproportionate elements
3. **Typographic Quality**
   - **Scoring Criteria**:
     - **Score 2**: All text appropriately sized and legible
     - **Score 1**: Text mostly appropriate with minor issues
     - **Score 0**: Text severely undersized or mispositioned

Output Format:
Your response **MUST** be a single, valid JSON object with no additional text.

```json
{
  "visual_structure_alignment": {
    "chart_type_consistency": {
      "score": 2,
    "reason": "Detailed explanation"
    },
    "spatial_layout_consistency": {
      "score": 2,
    "reason": "Detailed explanation"
    },
    "text_element_consistency": {
      "score": 2,
    "reason": "Detailed explanation"
    },
   "axis_configuration_consistency":
    { "score": 2,
    "reason": "Detailed explanation"
    },
    "color_scheme_consistency": {
      "score": 2,
    "reason": "Detailed explanation"
    },
   "style_and_format_consistency": {
      "score": 2,
    "reason": "Detailed explanation"
    },
    "data_pattern_consistency": {
      "score": 2,
    "reason": "Detailed explanation"
    },
    "component_completeness": {
      "score": 2,
    "reason": "Detailed explanation"
    }
  },
  "execution_quality": {
    "visual_clarity": {
      "score": 2,
    "reason": "Detailed explanation"
    },
    "compositional_balance": {
      "score": 2,
    "reason": "Detailed explanation"
    },
```

```
    "typographic_quality": {
      "score": 2,
    "reason": "Detailed explanation"
    }
  },
  "improvement_recommendations":
 "Priority fixes with explanations"
}
```

Generation Prompt for Chart Reproduction

Role:
You are an expert Python data visualization engineer specializing in matplotlib and seaborn.
Task:
Analyze the provided data information and reference chart image to identify the data relationships and patterns being visualized. Generate executable Python code that precisely replicates the chart with pixel-level fidelity using the actual data structure described.
Inputs:

1. **Data Information:**
   - Dataset descriptions including shape, columns, data types, and sample rows
   - Multiple CSV/XLSX files may be provided with their respective schemas
2. **Reference Chart:**
   - Image showing the target visualization to replicate

Analysis Requirements:
Before generating code, identify:

- Which dataset(s) and columns are used in the visualization
- Data transformations applied (aggregations, calculations, filtering, etc.)
- Relationships between multiple datasets (if applicable)
- Time series patterns, trends, or statistical relationships displayed

Code Requirements:
Your code must exactly replicate:

1. **Data Loading:**
   - **MANDATORY:** Read data from the provided CSV or XLSX files using pandas (pd.read_csv() or pd.read_excel())
   - Use the exact file paths/names mentioned in the data information
   - Apply necessary data type conversions (dates, numeric values, categories)
   - Handle multiple files if multiple datasets are referenced
2. **Data Mapping:**
   - **CRITICAL:** Ensure each data column is correctly mapped to its corresponding visual element in the chart
   - X-axis must display the correct column data in the proper order
   - Y-axis must display the correct column data with accurate values
   - Multiple series/groups must use the correct data columns with proper labels
   - Verify that data values in the visualization match the source data numerically and categorically
   - Apply correct aggregations (sum, mean, count, etc.) to match the reference chart
   - Ensure data filtering and grouping operations produce the exact subsets shown in the visualization
3. **Chart Structure:**
   - Chart types (line, bar, scatter, etc.)
   - Subplot layout and arrangement
   - Figure dimensions and aspect ratios
4. **Data Representation:**
   - Axis properties (labels, units, ranges, scales, ticks)

- Data values, trends, and patterns from the input datasets
- Legends (content, position, style)
- Statistical elements (means, medians, trend lines, etc.)

5. **Visual Styling:**
   - Colors for all elements (lines, markers, backgrounds)
   - Typography (fonts, sizes, weights)
   - Line/marker styles (patterns, shapes, widths, transparency)
   - Grids, spines, and backgrounds
6. **Text Content:**
   - Titles, axis labels, annotations (exact wording and placement)
   - Value labels or data point annotations

Code Standards:

- Use matplotlib and/or seaborn exclusively
- Load and process data matching the provided data_info structure
- Include necessary data preprocessing (parsing dates, filtering, calculations)
- Ensure code is self-contained and executable
- Use the actual column names and data types from the input datasets
- **DO NOT generate synthetic data**

Output Format:
Provide only the Python code block:

```
```python
# Your code here
```

Data Evaluation Prompt for Chart Reproducti

Task:
Compare two Python visualization codes and determine if they use **functionally equivalent data** for each corresponding subplot.

Evaluation Criteria:

1. Data Source Matching

- Both codes must load data from the same file(s)
- Column selections must access the same fields (even if using different syntax)
- Example: `df['col']` ≡ `df.col` ≡ `df.loc[:, 'col']`

2. Data Transformation Equivalence
For each subplot, check if the **final data arrays** passed to plotting functions are equivalent:
ACCEPT as equivalent:

- Different groupby syntax producing same result:
  - `df.groupby('A')['B'].sum()` ≡ `df.groupby('A').agg({'B': 'sum'})['B']`
- Different filtering methods:
  - `df[df['x'] > 0]` ≡ `df.query('x > 0')` ≡ `df.loc[df['x'] > 0]`
- Reordered operations that don't affect output:
  - `df.sort_values('A').reset_index(drop=True)` ≡ `df.reset_index(drop=True).sort_values('A')`
- Different intermediate variable names
- Different but equivalent aggregation functions on same data

REJECT as different:

- Using different columns: `df['A']` vs `df['B']`
- Different aggregation types: `.sum()` vs `.mean()`
- Different filters: `df[df['x'] > 0]` vs `df[df['x'] > 5]`
```

- Different data subsets for same visual element
- Missing or extra data transformations that change values

3. Visual Element Data Mapping
For each subplot, verify corresponding visual elements use equivalent data:
Must match:

- X-axis data source and values
- Y-axis data source and values
- Color/hue grouping variables
- Size/marker data sources
- Facet/subplot grouping variables
- Filter conditions (if applied to specific elements)

May differ:

- Variable names (`x_data` vs `x_vals`)
- Intermediate calculation steps (as long as final result is same)
- Code organization (functions vs inline)

Verification Steps:

1. **Identify corresponding subplots** between two codes (by position or title)
2. **For each subplot pair**, trace data flow:
   - What columns are loaded?
   - What transformations applied? (group, filter, aggregate, calculate)
   - What data arrays enter `plt.plot()`, `plt.bar()`, `sns.lineplot()`, etc.?
3. **Compare final data semantics**, not code syntax
4. **Check ALL subplots** - if any subplot uses different data, return NO

Edge Cases:

- **Hardcoded values**: If one code uses `plt.bar([1,2,3], [4,5,6])` and another loads same values from CSV, consider **EQUIVALENT**
- **Derived columns**: `df['ratio'] = df['A']/df['B']` must use same columns in both codes
- **Statistical calculations**: `df['A'].mean()` vs `np.mean(df['A'])` are **EQUIVALENT**

Output Format:

```
<reasoning>
[Brief analysis of each subplot's data usage comparison]
</reasoning>

<result>YES</result> <!-- If all subplots use equivalent data -->
or
<result>NO</result>  <!-- If ANY subplot uses different data -->
```

Examples:
Example 1: Should return YES

```
# Code A
data = pd.read_csv('data.csv')
train = data[data['split'] == 'Train']
plt.bar(train['category'],
train['count'])

# Code B
df = pd.read_csv('data.csv')
filtered = df.query("split == 'Train'")
plt.bar(filtered['category'].values,
filtered['count'].values)
```

Evaluation Prompt for Chart Reproduction

Role and Goal:
You are an expert data visualization quality assessor. Your task is to rigorously evaluate Chart B (generated output) against Chart A (reference chart) based on the original task requirements. The evaluation must assess fidelity in visual structure, data representation, and design execution with strict adherence to standards.

Evaluation Categories:

1. Visual Structure Alignment (3-Level Scoring: 0/1/2)
Assess whether Chart B replicates the structural elements of Chart A with precision.

1. **Chart Type Consistency**
   - **Question**: Does Chart B use identical chart types as Chart A for all visualizations?
   - **Scoring Criteria**:
     - **Score 2**: All chart types match exactly across all subplots
     - **Score 1**: Primary chart types match for all subplots
     - **Score 0**: Any primary chart type differs
2. **Spatial Layout Consistency**
   - **Question**: Does Chart B replicate Chart A's exact component arrangement?
   - **Scoring Criteria**:
     - **Score 2**: Exact grid structure, identical subplot positioning
     - **Score 1**: Correct grid structure and subplot count, but minor positioning deviations
     - **Score 0**: Incorrect grid dimensions, wrong number of subplots, major element misplacement
3. **Text Element Consistency**
   - **Question**: Does Chart B contain all textual content from Chart A?
   - **Scoring Criteria**:
     - **Score 2**: All text content matches exactly in wording, placement, and hierarchy
     - **Score 1**: All critical text present with exact wording, but minor discrepancies in secondary elements
     - **Score 0**: Missing or altered critical text
4. **Axis Configuration Consistency**
   - **Question**: Does Chart B replicate all axis properties from Chart A?
   - **Scoring Criteria**:
     - **Score 2**: Perfect match for all axis properties and legend specifications
     - **Score 1**: Axis variable names and scale types correct, but minor deviations
     - **Score 0**: Incorrect axis scale type, wrong variable assignments, axis range drastically different
5. **Color Scheme Consistency**
   - **Question**: Does Chart B apply the same color mappings as Chart A?
   - **Scoring Criteria**:
     - **Score 2**: Identical color mappings for all elements with exact hex/RGB matches
     - **Score 1**: Correct color palette family and mapping logic, but minor shade variations
     - **Score 0**: Different color palette applied, incorrect categorical color assignments
6. **Style and Format Consistency**
   - **Question**: Does Chart B match Chart A's stylistic attributes?
   - **Scoring Criteria**:
     - **Score 2**: All stylistic attributes match
     - **Score 1**: Core visual style maintained with minor deviations
     - **Score 0**: Multiple style mismatches or fundamentally different aesthetic
7. **Component Completeness**
   - **Scoring Criteria**:
     - **Score 2**: Chart B contains all visual components from Chart A
     - **Score 1**: All primary components present but missing 1-3 minor elements

- **Score 0**: Missing critical components that significantly affect information completeness

2. Execution Quality (3-Level Scoring: 0/1/2)
Evaluate the technical execution quality of Chart B independently.

1. **Visual Clarity**
   - **Scoring Criteria**:
     - **Score 2**: Zero overlap or occlusion affecting readability
     - **Score 1**: Minor overlap limited to 1-2 non-critical elements
     - **Score 0**: Critical overlap affecting primary elements that severely impairs interpretation
2. **Compositional Balance**
   - **Scoring Criteria**:
     - **Score 2**: Optimal spatial distribution with proportional element sizing
     - **Score 1**: Adequate composition with acceptable proportions and spacing
     - **Score 0**: Poor composition with disproportionate elements
3. **Typographic Quality**
   - **Scoring Criteria**:
     - **Score 2**: All text appropriately sized relative to chart dimensions
     - **Score 1**: Text mostly appropriate with minor issues
     - **Score 0**: Text severely undersized or oversized, illegible elements

Output Format:
Your response **MUST** be a single, valid JSON object with no additional text. Use this exact structure:

```
{
  "visual_structure_alignment": {
    "chart_type_consistency": {
      "score": 1,
    "reason": "Detailed explanation"
    },
   "spatial_layout_consistency": {
      "score": 2,
    "reason": "Detailed explanation"
    },
    "text_element_consistency": {
      "score": 1,
    "reason": "Detailed explanation"
    },
   "axis_configuration_consistency":
    { "score": 1,
    "reason": "Detailed explanation"
    },
    "color_scheme_consistency": {
      "score": 2,
    "reason": "Detailed explanation"
    },
   "style_and_format_consistency": {
      "score": 1,
    "reason": "Detailed explanation"
    },
    "data_pattern_consistency": {
      "score": 2,
    "reason": "Detailed explanation"
    },
    "component_completeness": {
      "score": 1,
    "reason": "Detailed explanation"
    }
  },
  "execution_quality": {
    "visual_clarity": {
      "score": 1,
    "reason": "Detailed explanation"
    },
    "compositional_balance": {
      "score": 2,
    "reason": "Detailed explanation"
    },
    "typographic_quality": {
      "score": 2,
    "reason": "Detailed explanation"
    }
  },
  "improvement_recommendations": "
```

```
  Priority fixes: (1) Add missing
  auxiliary bar overlay..."
}
```

Generation Prompt for Chart Refinement

Role:
You are an expert Python data visualization engineer specializing in matplotlib and seaborn, with strong skills in chart debugging and correction.
Task:
You will receive:

1. An **image of a chart** that contains visual errors or issues
2. An **instruction** describing what needs to be corrected

Your goal is to generate corrected, executable Python code that fixes the issues identified in the instruction while maintaining other aspects of the original chart.
Instruction Input Format:
The instruction will be provided in natural language describing the required corrections, such as:

- "Fix the incorrect color scheme - bars should be blue instead of red"
- "Correct the y-axis scale which is currently inverted"
- "Fix the overlapping labels on the x-axis"
- "Correct the legend positioning - it's currently obscuring the data"
- "Fix the incorrect data values in the second subplot"
- "Adjust the title alignment and font size"
- "Fix the misaligned text labels"
- "Correct the vertical alignment of annotations"

Your Responsibilities:

1. **Analyze the Chart Image:**
   - Identify the chart structure, type, and layout
   - Understand the current state (including the errors)
   - Note all visual elements and styling
2. **Interpret the Instruction:**
   - Understand what specific corrections are needed
   - Determine which elements should be modified
   - Preserve all elements not mentioned in the instruction
3. **Generate Corrected Code that Must Exactly Replicate:**
   - **Chart Structure:**
     - Chart types
     - Subplot layout
   - **Data Representation:**
     - Axis properties (labels, units, ranges, scales, ticks)
     - Data values and trends
     - Legends (content, position, style)
   - **Visual Styling:**
     - Colors (all elements)
     - Typography (fonts, sizes)
     - Line/marker styles (patterns, shapes, widths)
     - Grids and backgrounds
   - **Text Content & Alignment:**
     - Titles, annotations, labels (exact wording and placement)
     - Text alignment (horizontal and vertical: left/center/right, top/center/bottom)
     - Text rotation and orientation
     - Spacing and positioning relative to chart elements

Code Standards:

- Use matplotlib and/or seaborn exclusively
- Ensure code is self-contained and executable

- Generate synthetic data that matches the reference characteristics
- Apply corrections precisely as instructed
- Maintain all other visual properties of the original chart
- Pay special attention to alignment properties (ha, va, align parameters)

Output Format:
Provide only the corrected Python code block:

```
```python
# Your corrected code here
```

Evaluation Prompt for Chart Refinement

Role and Goal:
You are an expert data visualization quality assessor. Your task is to rigorously evaluate Chart B (generated output) against Chart A (reference chart) based on the original task requirements. The evaluation must assess fidelity in visual structure, data representation, and design execution with strict adherence to standards.

Evaluation Categories:

1. Visual Structure Alignment (3-Level Scoring: 0/1/2)
Assess whether Chart B replicates the structural elements of Chart A with precision.

1. **Chart Type Consistency**
   - **Question**: Does Chart B use identical chart types as Chart A for all visualizations (e.g., line, bar, scatter, pie, heatmap)?
   - **Scoring Criteria**:
     - **Score 2**: All chart types match exactly across all subplots, including primary chart types and any overlaid secondary elements (e.g., dual-axis charts, combination charts)
     - **Score 1**: Primary chart types match for all subplots, but minor secondary elements differ (e.g., missing auxiliary bar overlay on a line chart, absent trendline, missing error bars on primary plot)
     - **Score 0**: Any primary chart type differs (e.g., bar chart used where line chart required), or multiple subplots have type mismatches, or fundamental chart category is wrong (e.g., scatter instead of bar)
2. **Spatial Layout Consistency**
   - **Question**: Does Chart B replicate Chart A's exact component arrangement (subplot grid dimensions, relative positioning of all visual elements)?
   - **Scoring Criteria**:
     - **Score 2**: Exact grid structure (e.g., 2×3 layout), identical subplot positioning, precise relative placement of all elements (legends, annotations, insets) within ±2% of reference dimensions
     - **Score 1**: Correct grid structure and subplot count, but minor positioning deviations (legend slightly displaced, annotation shifted but still in correct subplot, spacing variations within 5-15% of reference)
     - **Score 0**: Incorrect grid dimensions (e.g., 1×4 instead of 2×2), wrong number of subplots, major element misplacement (legend in wrong subplot, annotations in incorrect panels), or spatial relationships fundamentally altered
3. **Text Element Consistency**
   - **Question**: Does Chart B contain all textual content from Chart A (titles, subtitles, annotations, axis
```

labels) with identical wording, excluding axis tick values?

- **Scoring Criteria**:
  - **Score 2**: All text content matches exactly in wording, placement, and hierarchy (main title, subplot titles, axis labels, annotations, data labels)
  - **Score 1**: All critical text present (main title, axis labels, subplot titles) with exact wording, but minor discrepancies in secondary elements (1-2 missing annotations, slightly reworded footnotes, minor label omissions that don't affect core interpretation)
  - **Score 0**: Missing or altered critical text (wrong/missing main title, incorrect axis labels, absent subplot titles, 3+ missing annotations), or significant semantic changes to any primary text element

4. **Axis Configuration Consistency**

- **Question**: Does Chart B replicate all axis properties from Chart A (variable names, units, scale types, ranges, tick intervals) and legend specifications for each subplot?
- **Scoring Criteria**:
  - **Score 2**: Perfect match for all axis properties (scale type, range, units, label text, tick intervals) and legend specifications (position, order, labels) across all subplots
  - **Score 1**: Axis variable names and scale types correct, but minor deviations (axis range extends ±10-20% beyond reference, tick intervals slightly different but reasonable, legend order swapped but all entries present, units abbreviated differently but semantically equivalent)
  - **Score 0**: Incorrect axis scale type (log vs linear), wrong variable assignments, axis range drastically different (>30% deviation or missing critical data region), missing axis labels, incorrect units, or legend missing critical entries

5. **Color Scheme Consistency**

- **Question**: Does Chart B apply the same color mappings as Chart A for all data series, categories, and visual elements?
- **Scoring Criteria**:
  - **Score 2**: Identical color mappings for all elements (data series, categorical encodings, heatmap scales, background, grid, annotations) with exact hex/RGB matches or perceptually indistinguishable equivalents
  - **Score 1**: Correct color palette family and mapping logic, but minor shade variations (slightly lighter/darker versions of reference colors, saturation differences within 15%, 1-2 secondary elements use similar but non-exact colors)
  - **Score 0**: Different color palette applied, incorrect categorical color assignments (e.g., Category A colored red instead of blue), reversed color scale (e.g., heatmap scale inverted), or 3+ elements with significantly different colors

6. **Style and Format Consistency**

- **Question**: Does Chart B match Chart A's stylistic attributes (background color, grid styles, marker shapes, line patterns, font families, border styles)?

- **Scoring Criteria**:
  - **Score 2**: All stylistic attributes match (grid style [solid/dashed/dotted], marker shapes, line widths, line patterns, font families, background colors, border styles, opacity levels)
  - **Score 1**: Core visual style maintained with minor deviations (grid style slightly different [dashed vs dotted], marker shapes similar but not exact [circle vs hexagon], line width within ±1px, font family same category [all sans-serif] but different face)
  - **Score 0**: Multiple style mismatches (wrong grid style + wrong markers + wrong font category), fundamentally different aesthetic (dark theme vs light theme), or critical style elements missing (no gridlines when reference has them, no borders when required)

7. **Data Pattern Consistency**

- **Question**: Does Chart B visually replicate Chart A's data patterns (data point positions, trend shapes, distribution profiles, statistical markers)?
- **Scoring Criteria**:
  - **Score 2**: Data patterns match reference with >95% positional accuracy, all trend shapes replicated, distribution profiles visually identical, all statistical markers (means, medians, confidence intervals) present and correct
  - **Score 1**: Core data trends and patterns recognizable (correlation directions correct, major peaks/troughs present, distribution shapes similar), but minor deviations (5-15% positional variance, slight smoothing differences, 1-2 missing minor statistical markers, interpolation artifacts in small regions)
  - **Score 0**: Data patterns significantly diverged (>15% positional errors, trend directions incorrect, distribution shapes fundamentally different, major statistical markers missing/wrong), or substantial data missing/added

8. **Component Completeness**

- **Scoring Criteria**:
  - **Score 2**: Chart B contains all visual components from Chart A including all data series, markers, annotations, grid elements, legends, statistical overlays (trendlines, confidence intervals, reference lines), decorative features, and auxiliary elements
  - **Score 1**: All primary components present (all main data series, essential legends, critical annotations, primary axis elements, key statistical markers) but missing 1-3 minor elements (optional secondary annotations, decorative styling features, non-essential reference lines, auxiliary data labels on individual points)
  - **Score 0**: Missing critical components (1+ major data series absent, essential legend missing, key annotations omitted, primary axis elements incomplete, important statistical overlays missing) that significantly affect information completeness or interpretation accuracy

2. Execution Quality (3-Level Scoring: 0/1/2)

Evaluate the technical execution quality of Chart B independently.

1. **Visual Clarity**
   - **Scoring Criteria**:
     - **Score 2**: Zero overlap or occlusion affecting readability; all elements (subplots, titles, axis labels, legends, annotations, tick marks, data markers) are fully legible with adequate separation (minimum 3-5px spacing between interactive elements)
     - **Score 1**: Minor overlap limited to 1-2 non-critical elements (legend box edge touching subplot border, secondary annotation slightly overlapping gridline, axis tick label near but not touching neighbor) that marginally affects aesthetics but not comprehension
     - **Score 0**: Critical overlap affecting primary elements (title obscuring data, axis labels overlapping each other, legend blocking data points, tick marks colliding, data series indistinguishable due to occlusion) that severely impairs interpretation
2. **Compositional Balance**
   - **Scoring Criteria**:
     - **Score 2**: Optimal spatial distribution with proportional element sizing (subplots balanced, margins uniform), harmonious spacing (consistent gaps between subplots, appropriate padding), effective whitespace utilization, professional aesthetic quality with clear visual hierarchy
     - **Score 1**: Adequate composition with acceptable proportions and spacing, minor imbalances (one subplot slightly larger than others, margins vary by 5-10%, spacing somewhat inconsistent) that don't significantly detract from usability or professionalism
     - **Score 0**: Poor composition with disproportionate elements (subplots drastically different sizes without justification, margins vary by >20%, cramped or excessive spacing, wasted whitespace, visual imbalance creating confusion about hierarchy or importance, unprofessional appearance
3. **Typographic Quality**
   - **Scoring Criteria**:
     - **Score 2**: All text appropriately sized relative to chart dimensions (titles 14-18pt, axis labels 10-12pt, tick labels 8-10pt for standard charts), fully legible at standard viewing distance (arm's length for digital, 2-3ft for print), correctly spelled, semantically accurate, properly positioned without overlap
     - **Score 1**: Text mostly appropriate with minor issues (1-2 labels slightly undersized [7-8pt where 10pt ideal] but still readable, minor positioning suboptimality [axis label slightly too far from axis], abbreviations used where full words preferred but meaning clear)
     - **Score 0**: Text severely undersized (<7pt for body text) or oversized (>20pt for labels), illegible elements, spelling errors, semantic inaccuracies (wrong variable names, incorrect units), severely mispositioned labels (rotated when horizontal preferred, outside visible area)

Output Format:
Your response **MUST** be a single, valid JSON object with no additional text. Use this exact structure:

```
{
  "visual_structure_alignment": {
    "chart_type_consistency": {
      "score": 1,
     "reason": "Detail Explanation"
    },
    "spatial_layout_consistency": {
      "score": 2,
     "reason": "Detail Explanation"
    },
    "text_element_consistency": {
      "score": 1,
     "reason": "Detail Explanation"
    },
   "axis_configuration_consistency":
    { "score": 1,
     "reason": "Detail Explanation"
    },
    "color_scheme_consistency": {
      "score": 2,
     "reason": "Detail Explanation"
    },
   "style_and_format_consistency": {
      "score": 1,
     "reason": "Detail Explanation"
    },
    "data_pattern_consistency": {
      "score": 2,
     "reason": "Detail Explanation"
    },
    "component_completeness": {
      "score": 1,
     "reason": "Detail Explanation"
    }
  },
  "execution_quality": {
    "visual_clarity": {
      "score": 1,
     "reason": "Detail Explanation"
    },
    "compositional_balance": {
      "score": 2,
     "reason": "Detail Explanation"
    },
    "typographic_quality": {
      "score": 2,
      "reason": "Detail Explanation"
    }
  },
  "improvement_recommendations":
  "Priority fixes: (1) Add missing
  auxiliary bar overlay..."
}
```

E Additional Discussion

E.1 Performance Analysis Across Benchmarks

To contextualize the difficulty of RealChart2Code, we compare the performance of the evaluated models against two existing state-of-the-art benchmarks: ChartMimic (Yang et al., 2025) and Plot2Code (Wu et al., 2024). To ensure a fair comparison, we normalize all scores to a 0-100% scale. As shown in Figure 7, we observe a dramatic performance collapse when models transition from existing datasets to our proposed benchmark.

The "Complexity Gap". Existing benchmarks appear to have reached a saturation point for top-tier proprietary models. For instance, Gemini-3-Pro-Preview and Claude-4.5-Opus achieve near-perfect scores on existing tasks, averaging **96.0%** and **91.2%** respectively. However, their performance practically *halves* on RealChart2Code, dropping to **50.6%** and **51.3%**. This vast disparity—visually represented by the "Complexity Gap" below the diagonal in Figure 7—demonstrates that current SOTA models, while proficient at simple syntax, are far from solving the challenge of generating complex, multi-panel visualizations from authentic raw data.

Open-Weight Models Struggle. The gap is even more pronounced for open-weight models. While leading models like Qwen3-VL-235B perform admirably on standard benchmarks (averaging **84.7%**), they fail to generalize to the rigor of real-world data science, scoring only **22.5%** on RealChart2Code. This indicates that while open-weight models have memorized chart patterns, they lack the robust reasoning capabilities required for the iterative refinement and complex logic handling demanded by our benchmark.

F Case Study

Error Case 1 As shown in Figure 8, the model demonstrates a strong capability in code genera-

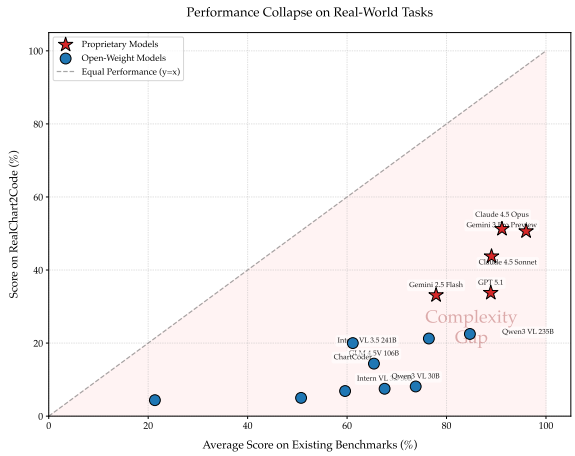

Figure 7: **Performance Collapse on Real-World Tasks.** We compare the normalized average scores (0-100%) of models on existing benchmarks (x-axis) versus `RealChart2Code` (y-axis). The dashed line represents equal performance ($y = x$). The significant distance of all models from this line highlights the **Complexity Gap**: even state-of-the-art models that master simple plotting tasks see their performance drop by nearly 50% when facing the realistic challenges in `RealChart2Code`.

tion for individual subplots, accurately replicating the visual style and data distribution of the reference. However, the synthesis of the multi-panel figure is structurally deficient. The generated script relies on default subplot positioning rather than explicit layout management tools such as `matplotlib.gridspec`. Consequently, the final output suffers from severe overcrowding, particularly in the bottom row where the text labels and plot elements overlap. This indicates that while the model captures local plotting semantics, it lacks the global spatial planning reasoning required for dense composite visualizations.

Error Case 2 As illustrated in Figure 9, this case pushes the boundaries of the model's spatial reasoning capabilities. The reference image requires nesting complex sub-layouts (e.g., a scatter matrix and a joint plot) within a larger dashboard grid. The model struggles significantly with this hierarchical composition. Specifically, the pairwise relationship matrix (bottom right) is rendered as disjointed, individual plots rather than a coherent grid, and the marginal joint plot (top left) fails to render the central scatter distribution entirely. This suggests that while the model can identify distinct plot types, it lacks the ability to orchestrate nested layout containers and coordinate systems required for high-density dashboards.

Error Case 3 As demonstrated in Figure 10, the model exhibits a severe failure in global canvas utilization. While the code attempts to construct the multi-panel grid, it lacks the necessary logic to balance figure size (`figsize`) with element scaling and resolution. The generated plot suffers from a critical aspect ratio mismatch, where the visualization is rendered at a "thumbnail" scale within a disproportionately large canvas. This results in excessive whitespace on the right and top margins, indicating that the model struggles to align code parameters with standard visual output requirements for readability.

Error Case 4 Figure 11 illustrates a fundamental failure in parsing composite chart structures. The reference image features advanced "JointGrid" style layouts, where central axes are spatially locked with marginal axes to display bivariate distributions. The model, however, interprets these marginal components as standalone visualizations. Consequently, it "flattens" the hierarchical design, rendering the marginal histograms as separate, strictly grid-aligned panels rather than integrated overlays. This error demonstrates a lack of understanding regarding semantic grouping in data visualization, leading to a fragmented layout that fails to reconstruct the cohesive narrative of the original image.

Error Case 5 Qwen3-VL-235B exhibits a recurring tendency to generate plausible-looking but functionally invalid API calls, affecting approximately 20% of the generated solutions. Specifically, the model frequently executes `plt.style.use('seaborn-v0_11')`, triggering a `ValueError` since this specific style parameter does not exist in standard Matplotlib or Seaborn libraries. This specific hallucination suggests a contamination or noise issue within the model's code-domain training data, where it has memorized incorrect versioning strings or deprecated syntax. Consequently, a significant portion of the outputs fail not due to visual reasoning deficits, but due to fundamental library knowledge errors.

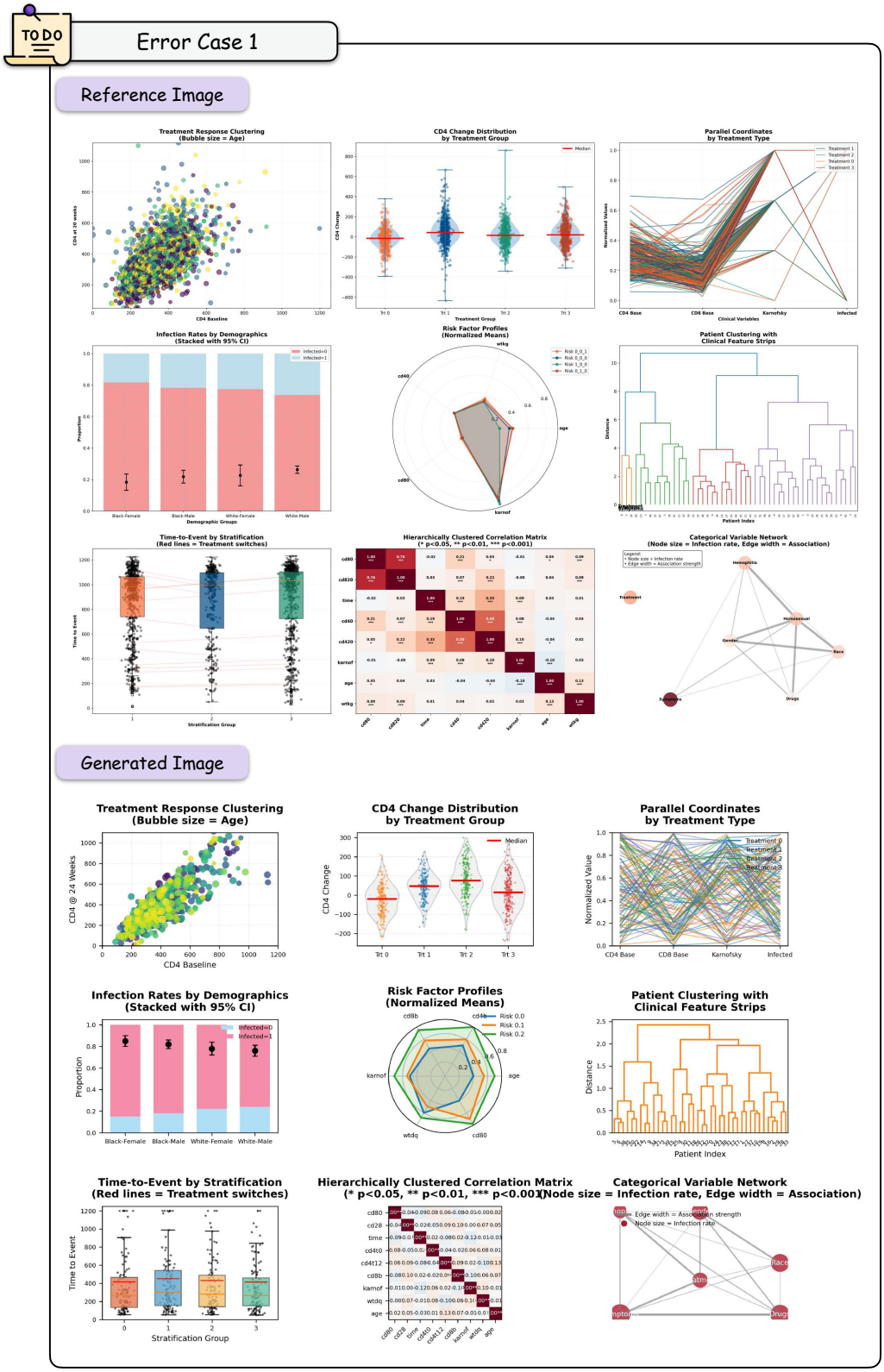

Figure 8: **Error Case 1: Layout Optimization Failure.** Comparison between the reference visualization (top) and the model-generated output (bottom). While the generated code correctly implements the individual plotting logic for diverse chart types, it fails to utilize advanced layout managers (e.g., `GridSpec`), resulting in overcrowded subplots and overlapping elements.

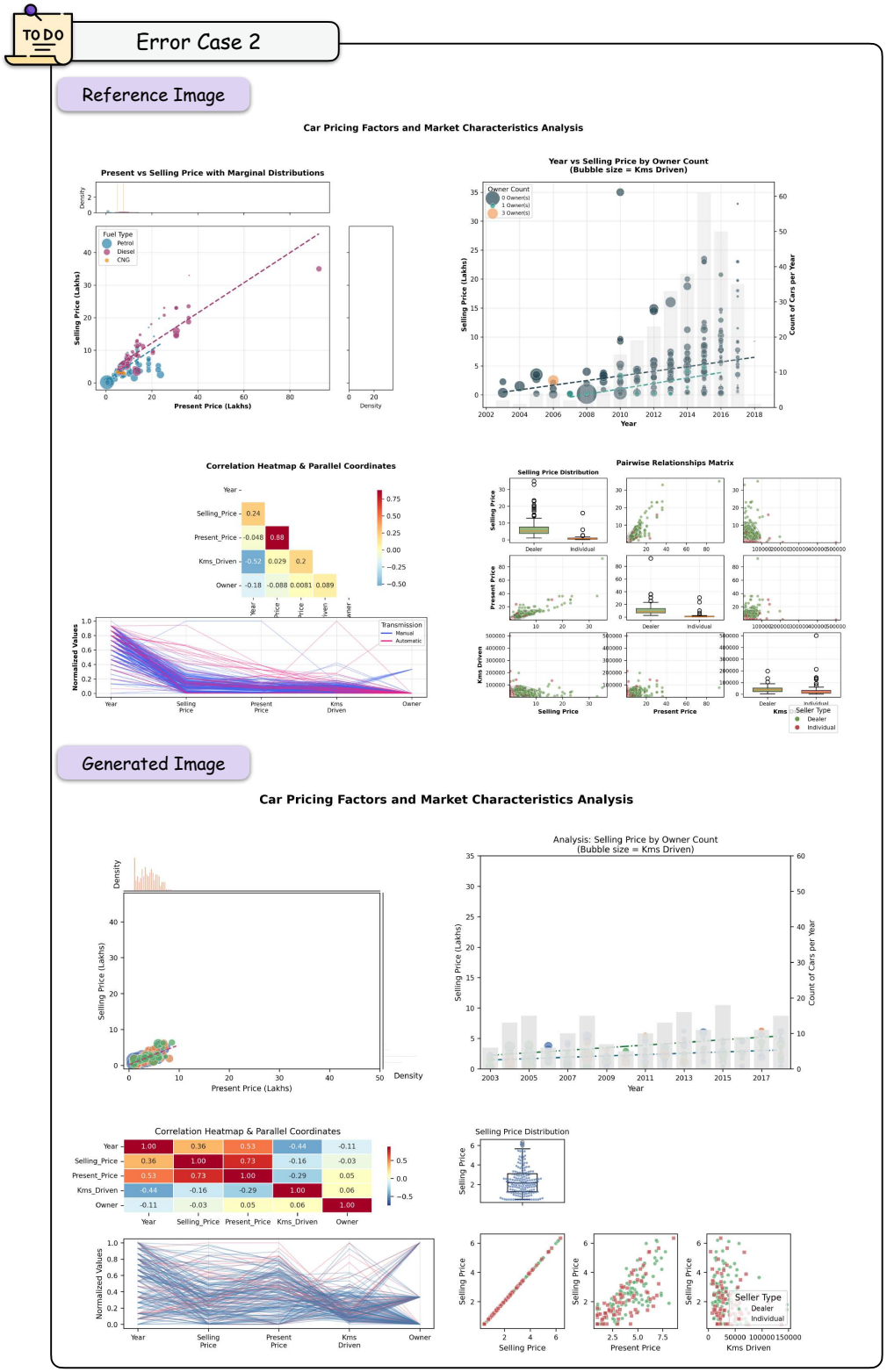

Figure 9: **Error Case 2: Complex Composite Layout Failure.** Comparison between the reference dashboard (top) and the model prediction (bottom). The target visualization features a highly intricate arrangement of heterogeneous charts, including marginal distributions, dual-axis overlays, and pairwise matrices. The model fails to recover the hierarchical structure, resulting in fragmented subplots, missing central data elements, and a collapse of the global layout.

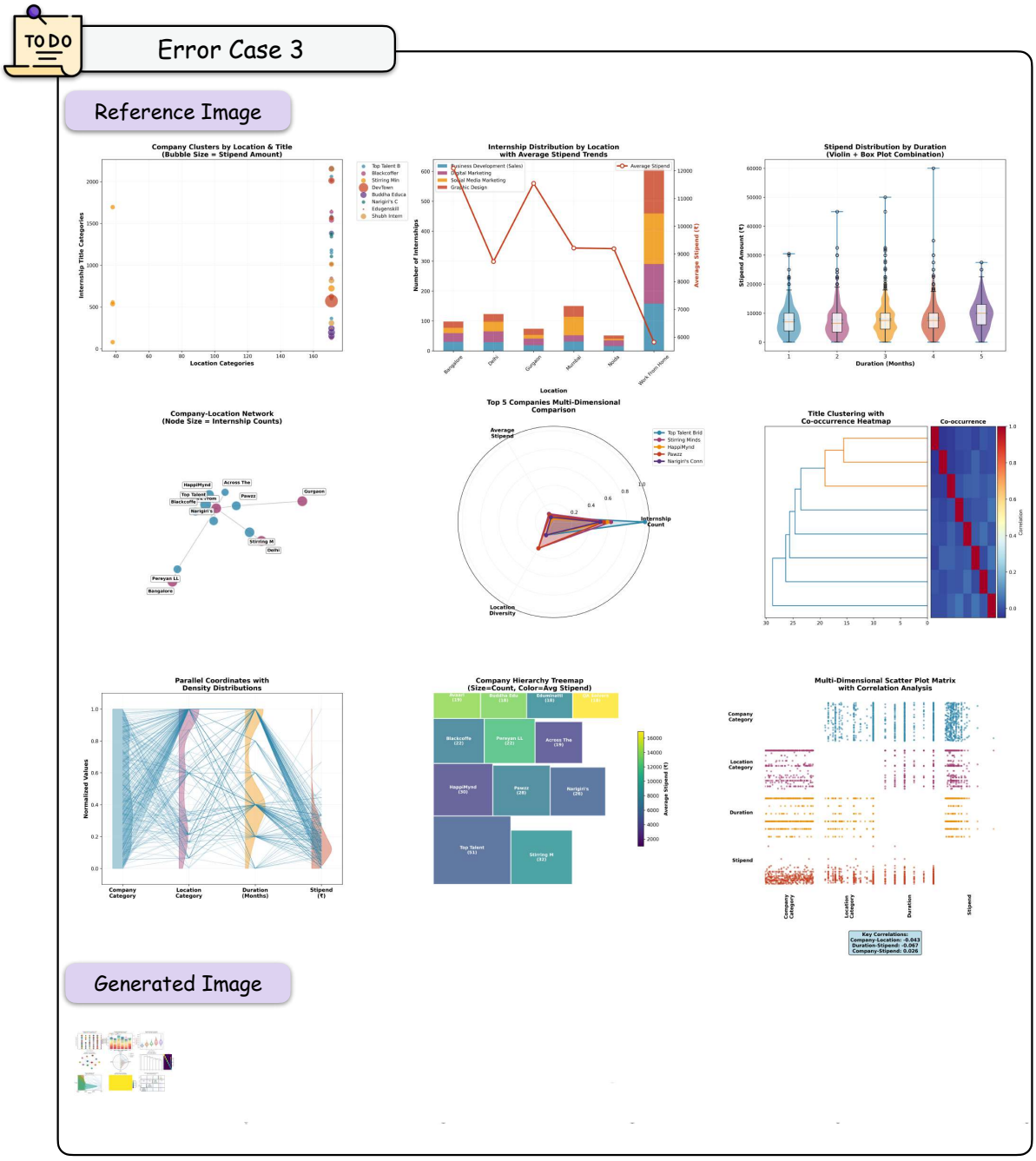

Figure 10: **Error Case 3: Canvas Scaling and Dimensioning Failure.** Comparison between the reference visualization (top) and the model-generated output (bottom). The model fails to correctly parametrize the global figure dimensions relative to the content, resulting in the entire subplot grid being compressed into a minute area. This scaling mismatch leaves the majority of the canvas as empty whitespace, rendering the visualization illegible.

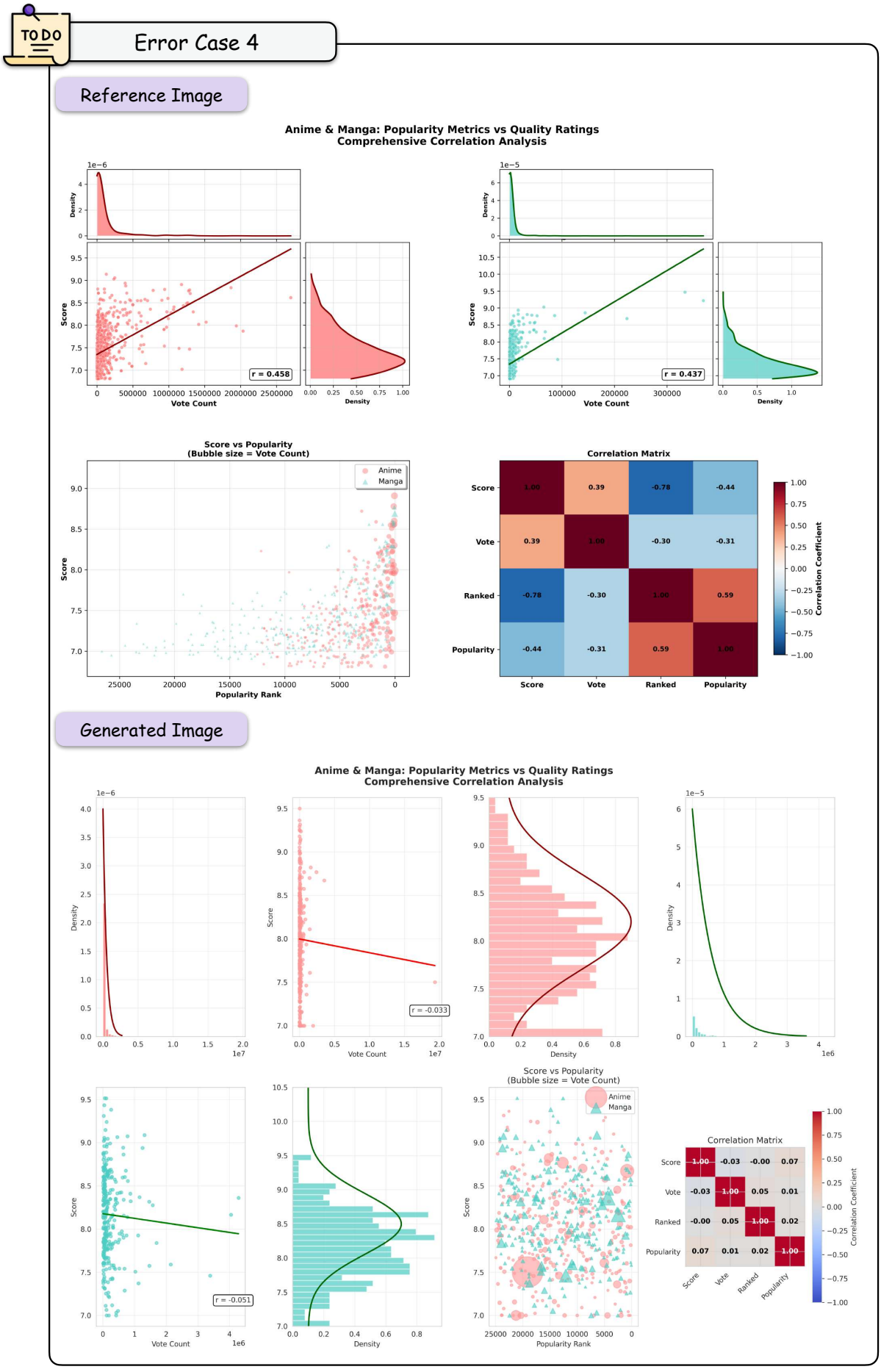

Figure 11: **Error Case 4: Semantic Layout Decomposition.** Comparison between the reference dashboard (top) and the model output (bottom). The reference relies on composite visualizations, specifically joint plots that integrate scatter plots with marginal density distributions. The model fails to perceive these grouped elements as unified semantic entities, instead breaking them down into disjointed, independent subplots. This results in a complete loss of the original grid structure and intended visual hierarchy.