Aligned Multi-View Scripts for Universal Chart-to-Code Generation

Anonymous ACL submission

Abstract

Chart-to-code generation converts a chart image into an executable plotting script, enabling faithful reproduction and editable visualizations. Existing methods are largely Python-centric, limiting practical use and overlooking a critical source of supervision: the same chart can be expressed by semantically equivalent scripts in different plotting languages. To fill this gap, we introduce **Chart2NCode**, a dataset of 176K charts paired with aligned scripts in Python, R, and LaTeX that render visually equivalent outputs, constructed via a metadata-to-template pipeline with rendering verification and human quality checks. Building on a LLaVA-style architecture, we further propose **CharLuMA**, a parameter-efficient adaptation module that augments the multimodal projector with a language-conditioned mixture of low-rank subspaces, allowing the model to share core chart understanding while specializing code generation to the target language through lightweight routing. Extensive experiments show consistent gains in executability and visual fidelity across all languages, outperforming strong open-source baselines and remaining competitive with proprietary systems. Further analyses reveal that balanced multi-language supervision benefits all languages and that the adapter allocates a compact shared core plus language-specific capacity [1].

1 Introduction

Charts serve as a compact and prevalent medium for communicating quantitative evidence in scientific literature, but they are frequently disseminated as static images, which impedes reproduction, editing, and reuse. Chart-to-code generation (Shi et al., 2025) bridges this gap by translating a chart image into an executable plotting script that faithfully reconstructs both the underlying data encodings and the visual design attributes.

[1]Codes and data are available at https://anonymous.4open.science/r/CharLuMA-DFF5.

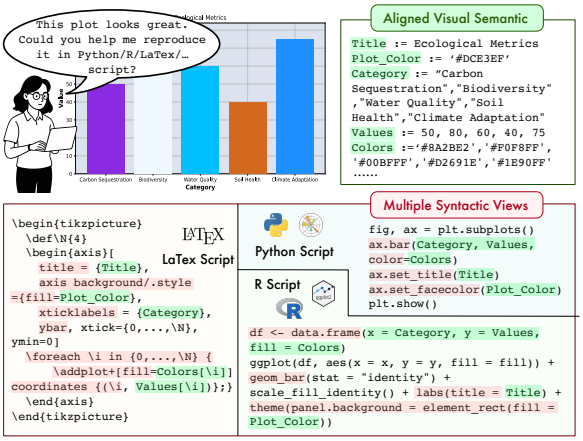


Figure 1: Illustration of aligned multi-view scripts for chart-to-code generation task.

While recent multimodal large language models (MLLMs) have demonstrated strong capabilities in general vision–language tasks (Yue et al., 2024; Lu et al., 2023; Zhang et al., 2025b), chart-to-code generation requires a substantially higher degree of precision: minor discrepancies in extracted data, axis specifications, or stylistic parameters can result in compilation failures or visually misleading outputs. Furthermore, the field remains constrained by a restrictive Python/matplotlib-centric bias (Wu et al., 2025; Zhao et al., 2025; Zhang et al., 2025a), neglecting the diverse ecosystem of plotting tools—such as R (ggplot2) and LaTeX (TikZ)—that serve as publication standards in many scientific disciplines (Mooney, 2022).

Beyond real-world applicability, the Python-centric paradigm overlooks a critical learning signal: the alignment of visual semantics across different plotting languages, as shown in Figure 1. While surface syntax diverges, scripts in multiple languages share a common latent visual semantic, encoding identical data tables, stylistic designs, and layout constraints. We frame these distinct scripts as complementary "views" of the same underlying chart semantics. We use the alignment across

languages to provide multi-target supervision for chart-to-code learning, where the same chart can be realized in multiple plotting syntaxes.

Realizing this multi-view idea necessitates a resource currently absent from the literature: aligned chart–code pairs across multiple languages that are visually isomorphic. The predominantly monolingual nature of existing datasets restricts models to a single target syntax, precluding the study of cross-language supervision (Shi et al., 2025; Zhao et al., 2025; Wu et al., 2025; Niu et al., 2025). To bridge this gap, we introduce **Chart2NCode**, a pioneering dataset of 176K chart images paired with aligned scripts in Python, R, and LaTeX. The dataset is constructed via an automated pipeline that synthesizes language-agnostic metadata into language-specific templates, ensuring high fidelity through rendering verification and human quality checking. This dataset supports both training and evaluation of multi-language chart-to-code models under consistent supervision.

A second challenge lies in the modeling: developing separate experts for each language is inefficient and discards the shared structure, while straightforward multi-language training can suffer from interference and uneven specialization. We propose **CharLuMA**, a parameter-efficient adaptation approach that preserves shared chart understanding while enabling language-specific code realization. Building on the LLaVA architecture (Liu et al., 2023), CharLuMA augments the multimodal projector with a language-conditioned mixture of low-rank subspaces (Figure 3). This module operates via a lightweight routing mechanism, which dynamically selects and combines a small subset of subspaces conditioned on the target language and visual features. This design enables the model to reuse a compact shared core while adapting its latent representations to specific syntactic conventions, offering an efficient alternative to the redundancy of independent experts or the interference of joint training. Experimental results demonstrate that balanced multi-language alignment yields consistent improvements across all languages and that the adapter allocates a compact shared core plus language-specific capacity.

To sum up, our contributions are three-fold:

- We formulate chart-to-code generation in a multi-language setting and propose CharLuMA that realizes multi-view script alignment via language-guided routing over low-rank subspaces.
- We present Chart2NCode, a dataset of 176K visually aligned chart–Python–R–LaTeX quadruples that enables the first systematic study of universal chart-to-code generation beyond Python.
- Extensive experiments validate our approach against open-source baselines and confirm that diverse, balanced multi-language supervision synergistically benefits all languages.

2 Related Work

Multimodal large language models employ multimodal projectors to bridge vision encoders with large language models, enabling reasoning across modalities. Existing works explores diverse strategies to optimize visual perception, ranging from compressing visual tokens (Li et al., 2022; Bai et al., 2023) to fusing multiple vision encoders (Tong et al., 2024; Lin et al., 2023). LLaVA (Liu et al., 2023, 2024) demonstrates that a simple MLP projector can effectively align modalities without discarding visual information. Some works employ sparsely gated MoE projectors (Li et al., 2025; Xu et al., 2025), which parallelize MLPs as experts at the cost of substantial parameter growth.

Chart-to-code generation task requires models to translate chart images into executable plotting scripts, challenging MLLMs with demands in visual understanding, code generation, and cross-modal reasoning. Prior efforts have primarily focused on chart-to-Python generation, spanning dataset construction (Shi et al., 2025; Zhao et al., 2025; Niu et al., 2025), multi-agent framework (Yang et al., 2024; Goswami et al., 2025) , and preference learning method (Zhang et al., 2025a). Other studies untilize chart-to-code generation for aligning multimodal projectors (Xu et al., 2025) or constructing question answering datasets (Zhang et al., 2024; He et al., 2025). These efforts remain restricted to single-language settings, which limits practical applicability and overlooks the learning signals in cross-language alignment.

Multi-view representation learning aims to construct comprehensive representations by integrating complementary information from multiple distinct views. It often enforces consistency constraints, extracting robust features that remain invariant across these views (Tian et al., 2020; Bachman et al., 2019). This principle has proven effective in both natural language processing (Conneau and Lample, 2019; Conneau et al., 2020) and code generation (Roziere et al., 2020; Guo et al., 2022), where mod-

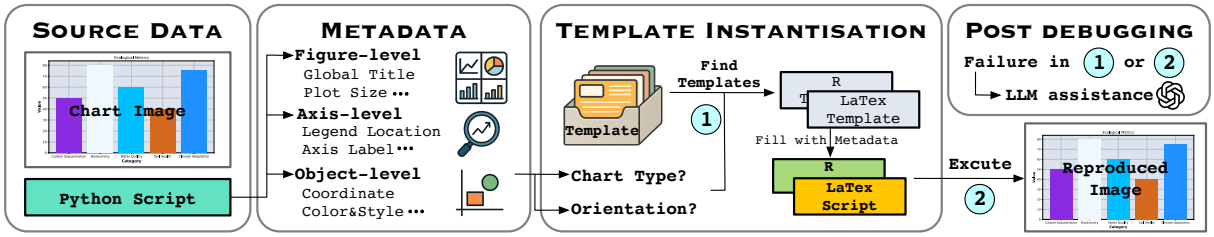


Figure 2: Overview of the automatic annotation pipeline of Chart2NCode.

els align diverse surface syntaxes into a shared semantic space. We extend this paradigm to scientific visualization, treating Python, R, and LaTeX scripts as complementary views of a single chart.

3 The Chart2NCode Dataset

We present Chart2NCode, the first large-scale dataset that aligns chart-code pairs across multiple programming languages. With 176K Chart-Python-R-LaTeX quadruples, Chart2NCode establishes a comprehensive resource for developing and evaluating multi-language chart-to-code models.

3.1 Automatic Annotation

We construct multi-language plotting scripts via an automatic annotation pipeline as shown in Figure 2. We first collect single-language source data from publicly available and open-source repositories. Specifically, we utilize Python scripts from ChartCoder (Zhao et al., 2025), a chart-specific subset of LaTeX scripts from DaTikZ (Belouadi et al., 2024a), and 40K newly curated R scripts from online communities, ensuring compliance with their respective open licenses and terms of use.

Metadata Extraction. We extract language-agnostic metadata from single-language plotting scripts at the figure, axis, and object levels. The figure level captures global attributes that determine the overall layout and presentation of the chart. The axis level records structural elements that define the coordinate system and its descriptive properties. The object level encodes graphical primitives together with their visual styles, ensuring precise representation of chart content. Metadata are obtained from plotting objects for Python (`matplotlib.axes`) and R (`ggplot_build()`), while LaTeX scripts are processed via regular-expression parsing. Collectively, these layers yield a comprehensive description of each chart, enabling faithful reconstruction using different languages in the following steps..

Template Instantiation. We synthesize multi-language scripts by identifying object-level metadata patterns to retrieve and instantiate language-specific templates. For instance, a horizontal bar chart is characterized at the object level by rectangles of equal height and varying width, which are organized into a data table and matched to the corresponding templates in different languages. Our library comprises 202 human-curated templates spanning over 30 chart subtypes in Python, R, and LaTeX, derived from systematic observations of the source data. Once the appropriate template is identified, it is instantiated with structured metadata such as titles, axis ticks, and data values. We also add an attribute-mapping process during instantiation to maintain cross-language consistency, such as mapping the `bold` font style in Python to the `bfseries` directive in LaTeX.

Post Debugging. In situations where template identifying is unsuccessful or script execution errors occur, we incorporate an LLM-assisted debugging module powered by GPT-4o (OpenAI, 2024b). If no suitable template exists, the module translates the available single-language script into the target languages; if an instantiated template fails, it applies error correction to restore executability. Scripts that remain invalid or produce deprecated figures are discarded to maintain dataset quality.

We detail the source data acquisition, annotation pipeline, and illustrative examples in Appendix A.1, Appendix A.2, and Appendix A.5, respectively.

3.2 Human Quality Checking

We conduct human evaluation to assess the cross-language fidelity of Chart2NCode using a random sample of 1,000 quadruples. We recruited three independent annotators from the university campus, requiring demonstrated proficiency in data visualization across all target languages. Annotators evaluated the samples across four dimensions on a 1–5 scale. The proportion of examples achieving an average score ≥ 4 demonstrates high quality across all dimensions: structural fidelity (98.2%), data in-

tegrity (95.2%), semantic consistency (97.4%), and stylistic coherence (95.8%). The inter-annotator agreement yields a Krippendorff's α (Krippendorff, 2011) of 0.81, indicating robust evaluation reliability. Further details and results are in Appendix A.3.

3.3 Data Statistics

Chart2NCode comprises 176k quadruples consisting of a chart image and aligned scripts in Python, R, and LaTeX, with 14.7% refined via LLM-assisted debugging. In terms of diversity, the dataset covers a wide spectrum of 20 distinct chart types, ranging from 18 regular types to 2 advanced categories featuring composite layouts and multiple coordinate systems. For benchmarking purpose, we construct a test set of 1,000 randomly sampled examples that achieve average scores of at least 4 across all quality aspects in Section 3.2. Using the Llama 3 tokenizer (Meta, 2024), we observe average token counts of 384.1 for Python, 591.8 for R, and 637.1 for LaTeX, with standard deviations of 189.7, 242.0, and 247.1, respectively. Detailed statistics are provided in Appendix A.4.

4 The CharLuMA Model

We propose CharLuMA, a chart-to-code MLLM that extends a LLaVA-style architecture with a novel low-rank subspace adapter for efficient multi-language adaptation. The model is optimized via a progressive training strategy that combines alignment pretraining with instruction tuning.

4.1 Architecture

CharLuMA is composed of a vision encoder and a LLM backbone, connected through a two-layer MLP projector augmented with a novel low-rank subspace adapter (see Figure 3). The adapter is governed by a language-guided routing policy that dynamically selects subspace experts based on both the chart's image features and the target language token, enabling language-specific specialization while maintaining shared visual understanding.

Vision Encoder. We adopt SigLIP (Zhai et al., 2023a) as the vision encoder, configured with an input resolution of 384×384. Pretrained on millions of image–text pairs, it provides strong priors for extracting semantically meaningful visual features. Formally, given a chart input $\mathbf{X}_v$, the vision encoder $g^v(\cdot)$ generates its corresponding representation $\mathbf{Z}_v$, *i.e.* $\mathbf{Z}_v = g^v(\mathbf{X}_v)$.

Multimodal Projector. The standard multimodal projector in LLaVA-style architectures (Liu et al., 2023) is a two-layer MLP block $\mathbf{W}$ that performs a one-to-one transformation, mapping visual features $\mathbf{Z}_v$ into the embedding space of the LLM backbone. The resulting output, $\mathbf{H}_{\text{base}} = \mathbf{W}\mathbf{Z}_v$, serves as a shared base representation across languages.

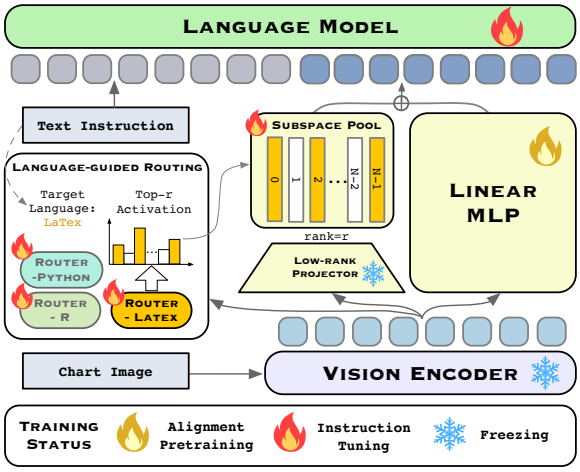


Figure 3: Overview of CharLuMA. The adapter employs language-conditioned routing to dynamically compose low-rank subspaces, exemplified here for a LaTeX target. The training strategy comprises alignment pretraining followed by instruction tuning.

To enable efficient language adaptation while preserving visual understanding, we augment this linear MLP block with a low-rank subspace adapter (Ding et al., 2025; Wu et al., 2024). The adapter consists of three components: a low-rank projector $\mathbf{A}$, a subspace pool $\{b_i\}_{i=1}^{N}$, and language-specific routers $\mathbf{W}^l$ ($l \in \{\text{Python}, \text{R}, \text{LaTeX}\}$) (Chen et al., 2023). Given the visual features $\mathbf{Z}_v$, the projector $\mathbf{A}$ maps them into a compact rank-r representation ($r < N$). The router then determines which subspaces to activate for the target language l, as specified in the text instruction. Concretely, the router $\mathbf{W}^l$ applies a language-specific transformation to the mean-pooled visual feature $\overline{\mathbf{Z}}_v$, yielding a probability distribution over the subspace pool. The top-r subspaces are then selected, $y^l = \text{top}_\text{r}(\text{softmax}(\mathbf{W}^l\overline{\mathbf{Z}}_v))$ where y^l denotes their indices, and concatenated to form the matrix $\mathbf{B} = \text{concat}_{i \in y^l} b_i$. The reconstruction matrix $\mathbf{B}$ is combined with the low-rank projector $\mathbf{A}$ to map the visual features into the LLM embedding space, yielding an language-adaptable representation. The final visual tokens injected into the LLM consists of visual tokens that merge the base and language-adaptable representations:

$$\mathbf{H}_v = \mathbf{H}_{\text{base}} + \mathbf{H}_{\text{adapt}} = \mathbf{W}\mathbf{Z}_v + \mathbf{A}\mathbf{B}\mathbf{Z}_v.$$

Large Language Model. We use DeepSeek-Coder

(Guo et al., 2024) as the LLM backbone, with 1.3B and 6.7B variants named CharLuMA-1.3B and CharLuMA-6.7B. The visual tokens $\mathbf{H}_v$ produced by the multimodal projector are concatenated with the text tokens $\mathbf{H}_t$ to construct the input sequence for the LLM $g^L(\cdot)$. The final output is then obtained as $g^L(\mathbf{H}_v; \mathbf{H}_t)$.

4.2 Training Strategy

Alignment Pretraining. We initialize the multimodal projector by pretraining the linear MLP block $\mathbf{W}$ on ChartMoE-Align (Xu et al., 2025), a dataset comprising 900k Chart–JSON pairs that encode structural representations. The vision encoder and LLM backbone remain frozen during this stage, ensuring that $\mathbf{W}$ learns to align visual features of charts with textual schema representations without altering pretrained components (Yan et al., 2024).
Instruction Tuning. We augment the multimodal projector with the proposed low-rank subspace adapter and finetune the model on Chart2NCode. We first warm up the language-specific routers $\mathbf{W}^l$ and the subspace pool $\{b_i\}_{i=1}^N$ over fixed steps, while keeping the MLP block, vision encoder, and LLM backbone frozen. The low-rank projector $\mathbf{A}$ is randomly initialized and kept frozen throughout training, ensuring that adaptation capacity is directed toward language-specific diversities rather than redundantly modeling visual commonalities (Ding et al., 2025; Tian et al., 2025). We then unfreeze the LLM backbone and continue training jointly with the routers and subspace pool, while keeping the MLP block, vision encoder, and $\mathbf{A}$ frozen. This progressive protocol stabilizes routing and subspace specialization in the early phase, and subsequently enables the LLM to effectively leverage language-adaptive visual tokens.

5 Experiment

We validate the efficacy of CharLuMA through extensive experiments across diverse benchmarks, establishing consistent gains over strong baselines in multi-language chart-to-code generation.

5.1 Implementation Details

During alignment pretraining, we train the MLP block for 1 epoch on 900k Chart–JSON pairs from ChartMoE-Align (Xu et al., 2025), with a learning rate of 2e-4. During instruction tuning, we warm up the subspace pool and language-specific routers for 274 steps, and then continue with full fine-tuning of the LLM backbone for 1 epoch on the Chart2NCode training set, which contains 175k Chart–Python–R–LaTeX quadruples. The learning rates are set to 2e-4 for the warm-up phase and 2e-5 for fine-tuning. We set the subspace size $N = 32$ and the rank $r = 16$. Detailed experimental settings are provided in Appendix B.1.

5.2 Evaluation Settings

Datasets. We evaluate CharLuMA and baselines on three chart-to-code datasets. The Chart2NCode test set provides 1,000 charts paired with scripts in Python, R, and LaTeX, enabling multi-language evaluation. ChartMimic (Shi et al., 2025) testmini set comprises 600 charts with human-curated matplotlib scripts in Python, spanning 22 chart types. Plot2Code (Wu et al., 2025) contains 132 high-quality matplotlib plots across 6 types.
Evaluation Metrics. We evaluate chart-to-code performance across two primary dimensions: executability and fidelity. Execution Rate (ER) measures the proportion of generated scripts that run successfully. To assess fidelity, we adopt DreamSim (DS) (Fu et al., 2023), a metric capturing perceptual similarity between generated and ground-truth images. Following the successful use of large foundation models for evaluation in computer vision (Shi et al., 2025; Zhao et al., 2025), we further employ an MLLM-as-Judge approach (MJ) to assess visual alignment of reproduced charts. We employ GPT-4o as the judge following criteria in Figure 11, with reproducibility validated against open-source alternatives with high correlation results (see Appendix B.2). For Python scripts specifically, we report an averaged F1 score covering text, layout, type, and color attributes (Shi et al., 2025). To ensure rigorous evaluation, unexecutable scripts are assigned a score of zero for DS, MJ and F1.

5.3 Baselines

General MLLMs. We evaluate both closed-source and open-source MLLMs as general-purpose baselines. The closed-source group includes GPT-4o (OpenAI, 2024b), GPT-4o-mini (OpenAI, 2024a), GPT-5-mini (OpenAI, 2025), Claude-3.5-Sonnet (Anthropic, 2024), and Claude-Sonnet-4 (Anthropic, 2025). The open-source group covers representative vision–language models including Qwen3-VL (Team, 2025), InternVL-3.5 (Wang et al., 2025), DeepSeek-VL2-Tiny (Lu et al., 2024), Phi-3.5-Vision (Abdin et al., 2024), and LLaVA-v1.6 (Liu et al., 2023).

Models	ChartMimic				Plot2Code				Chart2NCode									
	Chart2Python				Chart2Python				Chart2Python				Chart2R			Chart2LaTeX		
	ER	DS	MJ	F1	ER	DS	MJ	F1	ER	DS	MJ	F1	ER	DS	MJ	ER	DS	MJ
	Propriety Multimodal Large Language Models																	
GPT-5-mini	86.8	86.9	78.2	71.5	93.2	85.9	72.7	72.8	85.2	89.0	80.0	67.5	90.3	82.5	81.2	49.7	75.2	41.1
GPT-4o-mini	89.0	77.5	74.8	70.2	90.2	77.8	63.8	67.0	94.8	81.2	79.8	74.5	89.5	75.4	70.3	94.7	61.2	70.4
GPT-4o	93.2	83.5	83.5	79.0	92.4	83.6	76.7	75.4	98.5	85.0	87.4	80.9	94.5	78.8	78.3	88.4	72.4	69.8
Claude-Haiku-3.5	88.0	76.2	73.5	65.7	87.1	72.8	60.6	56.8	91.3	81.6	76.7	68.8	93.0	76.2	73.9	78.2	57.3	55.3
Claude-Sonnet-4	96.2	83.3	86.4	81.5	95.5	81.2	81.0	76.8	98.3	86.8	88.0	81.4	93.9	82.0	83.1	92.7	76.0	72.2
	Open-source Multimodal Large Language Models																	
Qwen3-VL-2B	56.3	68.4	39.7	39.2	68.9	64.2	41.2	50.1	74.0	78.0	59.6	61.0	56.5	52.4	42.0	56.0	60.8	37.4
Qwen3-VL-4B	72.5	71.9	58.4	55.2	77.3	66.4	54.4	55.4	87.6	83.2	77.2	76.1	75.4	66.4	60.9	62.4	68.6	45.2
Qwen3-VL-8B	78.7	72.5	65.2	61.9	78.8	68.1	57.3	56.9	91.1	83.7	80.8	80.6	73.6	72.7	57.2	77.3	66.8	57.1
InternVL3.5-2B	48.5	65.6	32.5	31.3	61.4	55.7	34.2	44.2	69.8	76.1	53.2	53.1	61.8	53.4	44.9	9.6	52.6	4.7
InternVL3.5-4B	62.8	69.5	44.7	43.0	62.1	58.8	38.0	42.7	77.9	78.4	63.4	63.0	66.8	56.4	51.5	25.7	55.5	14.7
InternVL3.5-8B	71.2	71.0	52.1	48.9	74.2	61.0	47.2	49.1	82.5	79.6	67.5	67.0	67.0	67.6	48.2	81.1	57.1	53.3
DeepSeek-VL2-3B	50.2	69.4	35.0	35.0	72.5	59.8	45.4	44.2	72.0	80.2	58.7	58.0	23.0	56.9	16.4	3.9	42.7	1.9
Phi-3.5-vision-4B	66.7	44.1	41.0	38.6	72.7	63.8	43.1	42.6	68.8	53.3	56.1	34.2	47.0	52.5	33.5	7.9	42.9	5.1
LLaVA-v1.6-7B	62.0	55.9	25.7	25.9	69.7	49.8	26.5	32.5	71.0	65.7	38.0	39.5	58.5	50.9	37.9	77.3	42.4	42.2
ChartLlama-13B	57.5	44.9	28.1	24.8	81.8	50.1	18.9	22.4	65.3	46.0	14.8	16.2	13.0	44.8	6.2	21.7	32.5	10.2
ChartMoE-8B	52.7	56.9	22.9	25.3	70.5	58.9	37.6	26.9	69.5	64.2	40.2	35.4	39.3	52.9	25.5	17.1	27.9	11.1
ChartCoder-7B	91.4	69.2	74.0	72.5	87.9	65.7	58.2	56.6	96.2	48.1	86.4	56.1	-	-	-	17.9	39.1	10.6
CharLuMA-1.3B	83.0	71.8	64.8	62.5	83.3	64.3	50.6	47.2	94.4	86.5	78.4	76.9	94.5	78.9	73.3	84.5	71.3	65.1
CharLuMA-6.7B	**92.3**	**77.4**	**75.1**	**73.3**	**96.2**	**68.3**	**62.2**	**60.5**	**98.0**	**88.7**	**88.1**	**83.5**	**96.5**	**81.8**	**80.9**	**89.0**	**72.5**	**74.2**

Table 1: Performance on ChartMimic, Plot2Code, and Chart2NCode test set. ER ⇑ denotes execution rate, DS ⇑ denotes the image-similarity score DreamSim, MJ ⇑ denotes the MLLM-as-Judge score, and F1 ⇑ denotes the heuristic F1 score for Python scripts. A "-" indicates that no executable script is generated.

Chart MLLMs. We also compare against chart-specialized MLLMs. ChartLlama (Han et al., 2023) adapts LLaVA to chart reasoning via instruction tuning. ChartMoE (Xu et al., 2025) advances chart understanding through a mixture-of-experts multimodal projector. ChartCoder (Zhao et al., 2025) directly targets chart-to-code generation by employing a code LLM as its language backbone.

5.4 Main Results

Existing MLLMs exhibit pronounced disparities in chart-to-code generation across different programming languages, as shown in Table 1. ChartCoder achieves 96.2 ER and 86.4 MJ on the Python subset of Chart2NCode, while its performance deteriorates significantly on other languages, yielding 17.9 ER on LaTeX and zero valid generations for R. We observe similar performance gaps in general-purpose open-source models on the Chart2NCode test set. DeepSeek-VL2-3B and Phi-3.5-Vision display acute imbalances, achieving around 70 ER on Python while falling below 10 ER on LaTeX. Qwen3-VL-8B suffers from severe fidelity degradation, showing markedly reduced visual alignment for R (57.2 MJ) and LaTeX (57.1 MJ) compared to Python (80.8 MJ). Proprietary models display this same tendency in a more moderate form; GPT-4o-mini and Claude-Haiku-3.5 maintain DS above 80 on Python, but drop to around 60 DS on LaTeX.

CharLuMA effectively addresses cross-language disparities, establishing itself as the open-source MLLM for universal chart-to-code generation. On established Python benchmarks, CharLuMA-6.7B delivers top-tier results with 92.3 ER and 77.4 DS on ChartMimic, along with 96.2 ER and 68.3 DS on Plot2Code. CharLuMA-1.3B shows high efficiency, securing 83.0 ER and 71.8 DS on ChartMimic. On the Chart2NCode test set, CharLuMA-6.7B sustains balanced performance across languages, achieving 88.7 DS and 83.5 F1 on Python, 81.8 DS and 80.9 MJ on R, and 72.5 DS and 74.2 MJ on LaTeX. Notably, it rivals Claude-Haiku-3.5 and GPT-4o-mini on all benchmarks, significantly narrowing the gap with proprietary systems.

6 Further Study

We conduct ablation studies and analyses to disentangle component contributions, demonstrating CharLuMA's robustness and interpretability.

Model Size	Projector Architecture	Chart2NCode		
		ER	DS	MJ
1.3B	Linear MLP	88.1	76.9	69.5
	Mixture-of-MLP	87.9	75.1	68.2
	Subspace Adapter	**91.1**	**78.9**	**72.3**
6.7B	Linear MLP	91.0	78.2	76.3
	Mixture-of-MLP	91.9	77.4	76.8
	Subspace Adapter	**94.5**	**81.0**	**81.1**

Table 2: Performance of alternative multimodal projector architectures during the instruction tuning stage of CharLuMA-1.3B and -6.7B on the Chart2NCode test set. Results are averaged over all three languages.

Language Model	Vision Encoder	Chart2NCode		
		ER	DS	MJ
DeepSeek-LLM-7B	SigLIP	88.6	77.1	74.3
DeepSeek-Coder-6.7B	CLIP	88.8	79.2	75.0
DeepSeek-Coder-6.7B	SigLIP	**94.5**	**81.0**	**81.1**

Table 3: Ablation study of backbone choices in CharLuMA on the Chart2NCode test set, with results averaged over all three languages.

Total Subspace	Activated Subspace	Total Router	Chart2NCode		
			ER	DS	MJ
16	8	3	88.9	77.6	70.5
32	8	3	89.4	77.8	71.3
64	32	3	87.8	75.6	68.5
32	16	1	86.1	75.1	67.0
32	32	0	85.8	73.2	66.3
32	16	3	**91.1**	**78.9**	**72.3**
w/o warming up before finetuning			87.1	75.6	67.9
w/o freezing A matrix of adapter			90.2	78.0	70.1

Table 4: Ablation study of subspace-router configurations in CharLuMA-1.3B on the Chart2NCode test set, with results averaged over all three languages.

6.1 Model Architecture Ablation

We conduct ablation studies on CharLuMA-1.3B with the Chart2NCode test set to examine alternative architectures, backbone choices, and subspace–router configurations.

Alternative Architecture. We compare our low-rank *subspace adapter* with two alternative projector designs in Table 2. The *linear MLP* baseline (Belouadi et al., 2024b,a; Zhao et al., 2025) yields modest improvements, resulting in 88.1 ER and 69.5 MJ for the 1.3B model. The *Mixture-of-MLP* design (Li et al., 2025; Xu et al., 2025) replaces the MLP block with a sparsely gated mixture-of-experts, each initialized from a pretrained MLP block, and we adapt it with a hard-routing policy that activates the language-specific and shared experts (see Appendix B.3). This achieves 91.9 ER and 76.8 MJ for the 6.7B model. In contrast, our low-rank *subspace adapter* achieves the strongest results across both model sizes with a far more compact design compared to Mixture-of-MLP.

Backbone Choices. We examine the effect of language and vision backbones in Table 3. Our default configuration with DeepSeek-Coder-6.7B and SigLIP yields the strongest results. Deviating from this setup leads to consistent performance drops: replacing the language model with DeepSeek-LLM-7B lowers scores to 88.6 ER and 74.3 MJ, while substituting the vision encoder with CLIP-Large reduces performance to 88.8 ER and 75.0 MJ.

Subspace–Router Configurations. We compare different subspace settings, routing strategies and training choices in CharLuMA-1.3B on the Chart2NCode test set in Table 4. Regarding subspace settings, we identify our default 32–16 configuration (Row 6) as the optimal balance; it outperforms both smaller and larger total subspaces (Rows 1 and 3) and surpasses the insufficient active capacity of the 32–8 setting (Row 2). We further confirm the necessity of language-guided routing in Rows 4–6, where replacing language-specific routers with a single shared router reduces DS from 78.9 to 75.1, and removing them entirely lowers it to 73.2. Finally, Rows 7–8 validate our training choices: removing the warming-up stage destabilizes specialization and lowers DS to 75.6, while unfreezing the A matrix compromises the low-rank representation, reducing DS to 78.0.

6.2 Language Structure Ablation

We examine the impact of language structure during training on Chart2NCode. To guarantee fairness, we enforce a fixed budget of training samples across all settings. Under the single-language setting, we give script supervision for each image from one single language. For the two-language setting, we randomly sample half of the charts and provide script supervision in two different languages (*e.g.*, pairing each image with both LaTeX and Python scripts). Similarly, in three-language setting, we randomly sample one-third of the charts and apply script supervision using all three languages. Additionally, we include an unaligned baseline using only the original source script per chart, which biases towards predominant languages and lacks multi-view alignment. Finally, the number of routers is configured to match the target lan-

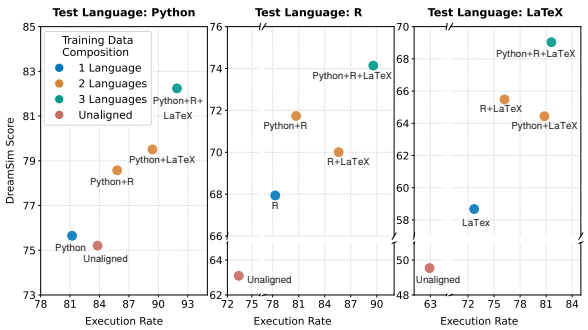


Figure 4: Ablation study of language structure using CharLuMA-1.3B on the Chart2NCode test set.

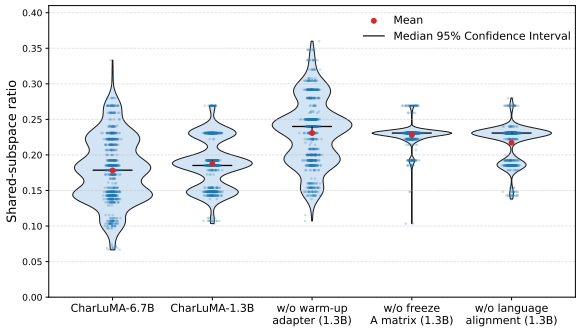


Figure 5: Distribution of shared-subspace ratios across CharLuMA and ablated models.

guage count (see Appendix B.3). Figure 4 shows that greater language diversity leads to substantial improvements on the Chart2NCode test set, effectively outweighing the reduction in unique visual exposure. The three-language model achieves the highest execution rates and visual fidelity scores across all languages, significantly outperforming single- and two-language models. Notably, training on unaligned source data induces systematic biases that impede universality, skewing the model toward the dominant language (Python).

6.3 Subspace Activation Analysis

We visualize the normalized activation frequency of 32 subspaces across CharLuMA's language-specific routers in Figure 7. These heatmaps reveal a hybrid allocation strategy, featuring compact shared clusters alongside broader language-specific zones. Specifically, CharLuMA-1.3B clearly delineates shared cores (*e.g.*, 21, 23, 30) from specific assignments (*e.g.*, 1 for Python, 18 for R). This structural organization persists in CharLuMA-6.7B, which exhibits a similarly balanced distribution with intermediate frequencies (e.g., 8, 20, 29), confirming that the architecture facilitates smooth cross-language integration across scales.

We introduce the *shared-subspace ratio* to quantify the cross-language allocation of subspaces, defined as the proportion of experts activated by all routers relative to the total activated set (Appendix B.3). Figure 5 reports the distribution of this ratio over a random sample of 1,000 Chart2NCode instances. CharLuMA-1.3B achieves a median ratio of 0.19, corresponding to roughly 5 shared experts out of 27. CharLuMA-6.7B shows a similar pattern with a median of 0.18, where about 4.9 experts are shared out of 27.5 on average. This indicates that scaling preserves a compact shared core, while allocating the increased capacity to expanded language-specific subspaces. In contrast, the ablated 1.3B variants exhibit inflated ratios (0.23–0.24), resulting from a contraction of the activation pool that compromises language-specific specialization.

6.4 Quantitative Analysis

We conduct a detailed error analysis of CharLuMA-6.7B on the Chart2NCode test set to reveal distinct language-specific failure dynamics. Execution errors in Python and R stem primarily from logic and data discrepancies, led by dimension mismatches (72.3% in Python, 56.1% in R) and undefined variables (11.9% in Python, 22.0% in R). LaTeX is uniquely prone to collapsing into unexecutable states due to rigid syntactic constraints (55.5%) such as missing braces. Beyond execution, successful generations often exhibit reproduction limitations relating to missing annotations, inaccurate subtypes, or stylistic inconsistencies (Appendix B.5). Case studies in Appendix B.6 further confirm CharLuMA's superior cross-language stability against GPT-4o and ChartCoder.

7 Conclusion

We leverage the visual semantic equivalence of scripts in different plotting languages to drive universal chart-to-code generation. We introduce Chart2NCode, a pioneering dataset of 176K visually aligned Chart–Python–R–LaTeX quadruples, and CharLuMA that realizes the multi-view supervision via language-conditioned routing over low-rank subspaces. Extensive experiments show that balanced multi-language supervision from aligned scripts provides complementary training signals that improve executability and visual fidelity over strong baselines. These contributions pave the way toward universal chart-to-code systems that reflect the diverse software ecosystems in practice.

Limitations

While this study offers a comprehensive analysis, we acknowledge specific limitations that merit future investigation. First, constrained by computational resources, we limited the CharLuMA architecture to 1.3B and 6.7B parameters. Although these scales demonstrate robust performance, scaling to larger backbones may yield further improvements in reasoning and generation quality. Second, the model remains constrained by the fixed input resolution of the visual encoder. While our choice of SigLIP represents a significant improvement over baselines like CLIP-Large, resolution bottlenecks may still limit performance on information-dense charts. Future work will focus on integrating high-resolution visual adapters to better handle these visually complex scenarios.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong Chen, and 110 others. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *Preprint*, arXiv:2404.14219.

Anthropic. 2024. Claude 3.5 sonnet.

Anthropic. 2025. Introducing claude 4.

Philip Bachman, R Devon Hjelm, and William Buchwalter. 2019. Learning representations by maximizing mutual information across views. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA. Curran Associates Inc.

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

Jonas Belouadi, Anne Lauscher, and Steffen Eger. 2024a. AutomaTikZ: Text-guided synthesis of scientific vector graphics with TikZ. In *The Twelfth International Conference on Learning Representations*.

Jonas Belouadi, Simone Paolo Ponzetto, and Steffen Eger. 2024b. Detikzify: Synthesizing graphics programs for scientific figures and sketches with tikz. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Tianlong Chen, Xuxi Chen, Xianzhi Du, Abdullah Rashwan, Fan Yang, Huizhong Chen, Zhangyang Wang, and Yeqing Li. 2023. Adamv-moe: Adaptive multi-task vision mixture-of-experts. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 17300–17311.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Alexis Conneau and Guillaume Lample. 2019. Cross-lingual language model pretraining. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA. Curran Associates Inc.

Chenhao Ding, Jiangyang Li, SongLin Dong, Xinyuan Gao, Yuhang He, and Yihong Gong. 2025. SuLoRA: Subspace low-rank adaptation for parameter-efficient fine-tuning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 5334–5349, Vienna, Austria. Association for Computational Linguistics.

Stephanie Fu, Netanel Tamir, Shobhita Sundaram, Lucy Chai, Richard Zhang, Tali Dekel, and Phillip Isola. 2023. Dreamsim: Learning new dimensions of human visual similarity using synthetic data. In *Advances in Neural Information Processing Systems*, volume 36, pages 50742–50768.

Kanika Goswami, Puneet Mathur, Ryan Rossi, and Franck Dernoncourt. 2025. Plotgen: Multi-agent llm-based scientific data visualization via multimodal retrieval feedback. In *Companion Proceedings of the ACM on Web Conference 2025*, WWW '25, page 1672–1676, New York, NY, USA. Association for Computing Machinery.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7212–7225, Dublin, Ireland. Association for Computational Linguistics.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *Preprint*, arXiv:2401.14196.

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

Wei He, Zhiheng Xi, Wanxu Zhao, Xiaoran Fan, Yiwen Ding, Zifei Shan, Tao Gui, Qi Zhang, and Xuanjing Huang. 2025. Distill visual chart reasoning ability from llms to mllms. *Preprint*, arXiv:2410.18798.

Klaus Krippendorff. 2011. Computing krippendorff's alpha-reliability. *Departmental Papers (ASC)*, page 43.

Jiachen Li, Xinyao Wang, Sijie Zhu, Chia-Wen Kuo, Lu Xu, Fan Chen, Jitesh Jain, Humphrey Shi, and Longyin Wen. 2025. Cumo: scaling multimodal llm with co-upcycled mixture-of-experts. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Junnan Li, Dongxu Li, Caiming Xiong, and Steven C. H. Hoi. 2022. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International Conference on Machine Learning*.

Ziyi Lin, Chris Liu, Renrui Zhang, Peng Gao, Longtian Qiu, Han Xiao, Han Qiu, Chen Lin, Wenqi Shao, Keqin Chen, Jiaming Han, Siyuan Huang, Yichi Zhang, Xuming He, Hongsheng Li, and Yu Qiao. 2023. Sphinx: The joint mixing of weights, tasks, and visual embeddings for multi-modal large language models. *Preprint*, arXiv:2311.07575.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2024. Improved baselines with visual instruction tuning. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 26286–26296.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Haoyu Lu, Wen Liu, Bo Zhang, Bingxuan Wang, Kai Dong, Bo Liu, Jingxiang Sun, Tongzheng Ren, Zhuoshu Li, Yaofeng Sun, Chengqi Deng, Hanwei Xu, Zhenda Xie, and Chong Ruan. 2024. Deepseek-vl: Towards real-world vision-language understanding. *Preprint*, arXiv:2403.05525.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chun yue Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. 2023. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *International Conference on Learning Representations*.

Meta. 2024. Introducing meta llama 3: The most capable openly available llm to date.

Paul Mooney. 2022. 2022 kaggle machine learning and data science survey.

Tianhao Niu, Yiming Cui, Baoxin Wang, Xiao Xu, Xin Yao, Qingfu Zhu, Dayong Wu, Shijin Wang, and Wanxiang Che. 2025. Chart2Code53: A large-scale diverse and complex dataset for enhancing chart-to-code generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 15839–15855, Suzhou, China. Association for Computational Linguistics.

OpenAI. 2024a. Gpt-4o mini: advancing cost-efficient intelligence.

OpenAI. 2024b. Hello gpt-4o.

OpenAI. 2025. Gpt-5 is here.

Baptiste Roziere, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised translation of programming languages. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Chufan Shi, Cheng Yang, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran Xu, Xinyu Zhu, Siheng Li, Yuxiang Zhang, Gongye Liu, Xiaomei Nie, Deng Cai, and Yujiu Yang. 2025. Chartmimic: Evaluating LMM's cross-modal reasoning capability via chart-to-code generation. In *The Thirteenth International Conference on Learning Representations*.

Qwen Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Chengzhong Xu. 2025. Hydralora: an asymmetric lora architecture for efficient fine-tuning. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Yonglong Tian, Dilip Krishnan, and Phillip Isola. 2020. Contrastive multiview coding. In *Computer Vision – ECCV 2020*, pages 776–794, Cham. Springer International Publishing.

Shengbang Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Manoj Middepogu, Sai Charitha Akula, Jihan Yang, Shusheng Yang, Adithya Iyer, Xichen Pan, Austin Wang, Rob Fergus, Yann LeCun, and Saining Xie. 2024. Cambrian-1: A fully open, vision-centric exploration of multimodal llms. In *Advances in Neural Information Processing Systems*, volume 37, pages 87310–87356. Curran Associates, Inc.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, Zhaokai Wang, Zhe Chen, Hongjie Zhang, Ganlin Yang, Haomin Wang, Qi Wei, Jinhui Yin, Wenhao Li, Erfei Cui, and 56 others. 2025. Internvl3.5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *Preprint*, arXiv:2508.18265.

Chengyue Wu, Zhixuan Liang, Yixiao Ge, Qiushan Guo, Zeyu Lu, Jiahao Wang, Ying Shan, and Ping Luo. 2025. Plot2Code: A comprehensive benchmark for evaluating multi-modal large language models

in code generation from scientific plots. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 3006–3028, Albuquerque, New Mexico. Association for Computational Linguistics.

Taiqiang Wu, Jiahao Wang, Zhe Zhao, and Ngai Wong. 2024. Mixture-of-subspaces in low-rank adaptation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7880–7899, Miami, Florida, USA. Association for Computational Linguistics.

Zhengzhuo Xu, Bowen Qu, Yiyan Qi, SiNan Du, Chengjin Xu, Chun Yuan, and Jian Guo. 2025. Chartmoe: Mixture of diversely aligned expert connector for chart understanding. In *The Thirteenth International Conference on Learning Representations*.

Pengyu Yan, Mahesh Bhosale, Jay Lal, Bikhyat Adhikari, and David Doermann. 2024. Chartreformer: Natural language-driven chart image editing. In *Document Analysis and Recognition - ICDAR 2024: 18th International Conference, Athens, Greece, August 30–September 4, 2024, Proceedings, Part I*, page 453–469, Berlin, Heidelberg. Springer-Verlag.

Zhiyu Yang, Zihan Zhou, Shuo Wang, Xin Cong, Xu Han, Yukun Yan, Zhenghao Liu, Zhixing Tan, Pengyuan Liu, Dong Yu, Zhiyuan Liu, Xiaodong Shi, and Maosong Sun. 2024. MatPlotAgent: Method and evaluation for LLM-based agentic scientific data visualization. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11789–11804, Bangkok, Thailand. Association for Computational Linguistics.

Xiang Yue, Yuansheng Ni, Tianyu Zheng, Kai Zhang, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, and 3 others. 2024. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9556–9567.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. 2023a. Sigmoid loss for language image pre-training. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11941–11952.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. 2023b. Sigmoid loss for language image pre-training. *Preprint*, arXiv:2303.15343.

Wenqi Zhang, Zhenglin Cheng, Yuanyu He, Mengna Wang, Yongliang Shen, Zeqi Tan, Guiyang Hou, Mingqian He, Yanna Ma, Weiming Lu, and Yueting Zhuang. 2024. Multimodal self-instruct: Synthetic abstract image and visual reasoning instruction using language model. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 19228–19252, Miami, Florida, USA. Association for Computational Linguistics.

Zhihan Zhang, Yixin Cao, and Lizi Liao. 2025a. Boosting chart-to-code generation in mllm via dual preference-guided refinement. In *Proceedings of the 33rd ACM International Conference on Multimedia*, MM '25, page 11032–11041, New York, NY, USA. Association for Computing Machinery.

Zhihan Zhang, Yixin Cao, and Lizi Liao. 2025b. XFinBench: Benchmarking LLMs in complex financial problem solving and reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8715–8758, Vienna, Austria. Association for Computational Linguistics.

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. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7333–7348, Vienna, Austria. Association for Computational Linguistics.

A Dataset

A.1 Data Acquisition

We collect single-language plotting scripts from established datasets and publicly available repositories governed by permissive licenses as our source data. ChartCoder (Zhao et al., 2025) contributes approximately 160k chart-to-Python scripts, while DaTikZ (Belouadi et al., 2024a) provides 49k vector-graphics-to-LaTex scripts, of which 8.8k correspond to charts with explicit axis structures. In addition, we curated 40k R plotting scripts from Stack Overflow [2], strictly adhering to the platform's attribution requirements and CC BY-SA data usage policy. To handle deprecated or non-executable scripts, we employed GPT-4o as an automated debugging assistant, guided by the prompt instructions in Figure 9, with a total API cost of 132.2 USD.

A.2 Annotation Pipeline

Metadata Structure and Extraction. We adopt a hierarchical metadata schema to capture chart information at three levels: figure, axis, and object. This structure provides a standardized representation of chart elements across languages while preserving both global properties and fine-grained graphical details. At the figure level, metadata records global properties such as the overall title, background color and legend, plot size (width, height, and units), twin-axis relationships, and subplot layout.

[2] Retrieved using StackAPI with keywords representative of R plotting functions and libraries, including `ggplot`, `plot_ly`, `geom`, `plot(`, `hist`, `boxplot` and so on.

For each axis, metadata focuses on type-agnostic attributes including axis titles, x- and y-axis labels, tick values and labels, legends, grids, panel boxes, background color, and annotations. At the object level, metadata captures fine-grained properties of graphical elements grouped into patches, lines, collections, and images. For each object, visual properties such as color, transparency, line width, marker style, and hatch patterns are recorded, together with precise geometric information such as rectangle bounds, circle centers and radii, polygon vertices, line coordinates, scatter offsets, and heatmap arrays. Cleaned labels are associated with color or stylish values where available, ensuring consistency with legends and categorical encodings.

Metadata is extracted by executing or parsing plotting scripts in their native environments. For Python plotting scripts, each script is executed in an isolated runtime, and the figure is inspected using `fig.get_axes()`. Axis-level attributes are gathered through standard APIs such as `ax.get_title()`, `ax.get_xlabel()`, and `ax.get_yticks()`. Object-level elements are obtained by iterating over `ax.patches`, `ax.lines`, `ax.collections` and so on. For R scripts based on `ggplot`, code is evaluated to collect the plotting object `p` built via `ggplot_build()`. We extract axis-level metadata from structures such as `p$labels$title`, `p$mapping$y`, and `p$theme$panel.border`, while object-level metadata is obtained by iterating over `p$layers`. For base R graphics, we wrap high-level functions like `barplot`, `hist`, and `boxplot`, as well as low-level commands such as `text`, `legend`, and `grid`, to capture metadata during execution. For LaTeX, we use the regular-expression parsing to detect `axis` environments while drawing commands are parsed to recover object geometries such as rectangles, circles, and paths.

Template Design. The templates are parameterized chart skeletons that translate extracted metadata into executable plotting code. Each template specifies placeholders for chart elements such as titles, axis labels, ticks, grids, legends, annotations, and objects, which are directly filled from metadata. The overall structure is consistent across languages, but implementation details differ. Taking the bar type for example, Python uses functions like `ax.bar` or `ax.barh` in matplotlib, R employs `geom_bar` in ggplot, and LaTeX relies on declarative PGFPlots options such as `xbar`, `ybar` and `addplot` using TikZ.

To maintain cross-language consistency during template instantiation, we employ an attribute-mapping process that normalizes visual properties across Python, R, and LaTeX. Legend locations are aligned so that values such as "upper right" in Python correspond to "right" in R and "north east" in LaTeX. Font styles are unified by mapping bold and italic settings into Python's weight and style fields, R's font face descriptors, or LaTeX commands like `bfseries` and `itshape`. Font sizes are standardized by converting numeric values in Python and R into LaTeX size categories such as `small` or `Large`. Annotation alignment is harmonized by translating Python's top, bottom, and center into equivalent justification values in R and LaTeX. Marker and line styles are also consolidated through shared dictionaries, ensuring that a logical style such as circle, dashed, or cross is rendered consistently across all languages. This mapping guarantees that semantic attributes are preserved even when the syntax differs, allowing metadata extracted in one language to be instantiated in another without loss of fidelity.

Metadata-Template Matching. A critical step in our automatic pipeline is to identify the correct template once the metadata of a chart has been extracted. We address this by assigning each chart a type and subtype based on patterns in the object-level metadata. Taking bar charts for example, we examine the geometry of rectangular patches: overlapping intervals reveal stacked bars, repeated clusters of equal size indicate grouped bars, with other cases default to base bars. For pie charts, subtype inference is based on patch geometry and offsets: the presence of an inner radius or nonzero x position signals a donut chart, displaced segment centers indicate exploded pies, and their combination yields donut–exploded pies. These inference rules allow the system to automatically select the most appropriate template across diverse chart variants.

LLM-assisted Debugging. We incorporate an LLM-assisted debugging module based on GPT-4o to handle cases where no suitable template is identified or when an instantiated template fails to execute. Instruction prompts for these two scenarios are provided in Figure 8 and Figure 9. The total expenditure on the OpenAI API is 316.6 USD.

A.3 Quality Assurance

We conduct a human evaluation to systematically assess the cross-language fidelity of Chart2NCode. We randomly sample 1,000

Dimension	Ann. 1	Ann. 2	Ann. 3	Avg.
Structural fidelity	98.7	97.8	98.1	98.2
Data integrity	94.5	95.8	95.3	95.2
Semantic consistency	97.9	96.6	97.7	97.4
Stylistic coherence	96.2	95.7	95.5	95.8

Table 5: Proportion (%) of examples with average rating ≥ 4 on 1,000 sampled quadruples, reported per annotator and averaged across annotators. Overall row averages the four dimensions.

chart–Python–R–LaTeX quadruples from the Chart2Ncode dataset, which are independently annotated by three annotators. All annotators were recruited on campus, with eligibility requiring prior experience in data visualization and programming in Python, R, and LaTeX. They were compensated in accordance with the institution's standard remuneration policies for academic work.

We conduct evaluations for each quadruple, comparing the reproduced charts in Python, R, and LaTeX against the original image, and annotators assess their fidelity across four dimensions. *Structural fidelity* measures whether the geometric arrangement of the chart is preserved, including the number and configuration of subplots and axis orientation. *Data integrity* evaluates whether the underlying quantitative values are reproduced exactly, meaning that the reconstructed chart reflects the same data table as the original. *Semantic consistency* assesses whether textual and categorical information is maintained, ensuring that titles, axis labels, legends, and annotations convey the same meaning without omissions, substitutions, or hallucinations. *Stylistic coherence* concerns the visual presentation, requiring that non-semantic design elements—such as color palettes, font attributes, line styles, and grid line visibility—remain consistent with the original chart. All dimensions are rated on a 1–5 scale, where 1 denotes severe mismatch and 5 denotes perfect alignment. A screenshot of the evaluation interface is available in Figure 14.

We compute the average per-dimension score across annotators for each example, and report the proportion of examples achieving an average score of at least 4. As shown in Table 5, the evaluation results confirm high fidelity across dimensions: 98.2% of examples exceed the threshold for structural fidelity, 95.2% for data integrity, 97.4% for semantic consistency, and 95.8% for stylistic coherence. To rigorously assess reliability on the full ordinal scale, we compute Krippendorff's α (Krippendorff, 2011). The resulting average α of 0.81 indicates substantial agreement beyond chance, representing a strong and practical level of consistency for human judgment in chart reproduction tasks.

A.4 Detailed Data Staistics

We report detailed statistics for the Chart2NCode dataset, spanning chart type distributions and code complexity. To ensure a robust training and evaluation source, the Chart2NCode dataset covers 20 distinct chart categories. Table 6 details the frequency and percentage of each type. The dataset explicitly challenges models with advanced composite visualizations, including Multidiff, which requires generating multiple heterogeneous subplots, and Combination, which involves overlaying distinct geometric types on a shared coordinate system.

Regarding code complexity on the Chart2NCode dataset, we utilized the Llama 3 tokenizer (Meta, 2024) to calculate token counts. The resulting statistics show a mean length of 384.1 tokens for Python ($\sigma = 189.7$, median 348.0), 591.8 tokens for R ($\sigma = 242.0$, median 545.0), and 637.1 tokens for LaTeX ($\sigma = 247.1$, median 595.0).

Type	Area	Bar	Box	Bubble
Percent	5.5%	11.6%	5.3%	2.0%
Type	Density	Donut	ErrorBar	ErrorPoint
Percent	1.9%	3.2%	2.9%	4.8%
Type	Heatmap	Histogram	Line	Lollipop
Percent	6.9%	1.3%	12.4%	0.5%
Type	Pie	Quiver	Radar	Scatter
Percent	7.6%	0.7%	6.7%	6.5%
Type	Violin	3D	Multidiff	Combination
Percent	6.3%	1.0%	9.1%	3.8%

Table 6: Distribution of chart types in Chart2NCode.

A.5 Case Study of Annotation Pipeline

We present two illustrative cases in Figure 12 and Figure 13 to demonstrate the functionality of our annotation pipeline.

B Experimental Settings and Results

B.1 Training and Evaluation Settings

We adopt SigLIP (Zhai et al., 2023b) as the vision encoder and DeepSeek-Coder (Guo et al., 2024) as the LLM backbone, yielding two variants of our model: CharLuMA-1.3B and CharLuMA-6.7B.

The multimodal projector is implemented as a standard two-layer MLP block augmented with our low-rank subspace adapter.

For alignment pretraining, we train the MLP block for one epoch on 900k chart–JSON pairs from ChartMoE-Align (Xu et al., 2025), while freezing both the vision encoder and LLM, with a learning rate of 2e-4. During instruction tuning, we first warm up the subspace pool and language-specific routers for 274 steps, and then perform full fine-tuning of the LLM backbone for one epoch on 175k chart–Python–R–LaTeX quadruples from Chart2NCode. In this stage, the vision encoder and MLP block remain frozen, the adapter is updated, and the learning rates are set to 2e-4 for warm-up and 2e-5 for fine-tuning. The low-rank projector $\mathbf{A}$ remains frozen throughout. Each training batch is constructed to include all three languages.

All training experiments are conducted with a global batch size of 128 on 8× NVIDIA L40S GPUs. The training cost for CharLuMA-1.3B is approximately 82 GPU hours, consisting of 35 GPU hours for pretraining, 6 GPU hours for warm-up, and 41 GPU hours for fine-tuning. For CharLuMA-6.7B, the total cost is about 321 GPU hours, including 109 GPU hours for pretraining, 18 GPU hours for warm-up, and 193 GPU hours for fine-tuning. More training hyperparameters are in Table 7.

For evaluation, we follow a standardized setup across all baselines, fixing the maximum token length to 2,048. The prompting format for the chart-to-code generation task is shown in Figure 10, adapted from Shi et al. (2025). Proprietary MLLMs evaluated include `gpt-4o-2024-08-06`, `gpt-4o-mini-2024-07-18`, `gpt-5-mini-2025-08-07`, `claude-3-5-haiku-20241022`, and `claude-sonnet-4-20250514`, all accessed through their official APIs. For open-source MLLMs, we directly run released checkpoints on NVIDIA L20 GPUs. Additionally, the total expenditure for MLLM-as-Judge metrics through the OpenAI API is 217.6 USD.

B.2 Detailed MLLM-as-Judge Metric

We employ an MLLM-as-Judge (MJ) approach to assess visual alignment of reproduced charts. Following Shi et al. (2025), we utilize GPT-4o (OpenAI, 2024b) to quantify the extent to which a generated chart corresponds to the ground truth. Specifically, the generated chart and the ground-truth chart are both input into GPT-4o. The model is instructed to evaluate their similarity across six dimensions—text, layout, chart type, data integrity, style, and clarity—according to the criteria detailed in Figure 11. Subsequently, GPT-4o assigns a final similarity score ranging from 0 to 100.

Hyperparameter	Alignment Pretraining	Warm-up	Instruction Tuning
Learning rate	2e-4	2e-4	2e-5
LR schedule	Cosine decay	Cosine decay	Cosine decay
Optimizer	AdamW	AdamW	AdamW
Max tokens	2,048	2,048	2,048
Vision encoder	Frozen	Frozen	Frozen
LLM	Frozen	Frozen	Trainable
MLP Block	Trainable	Frozen	Frozen
Adapter	Frozen	Trainable	Trainable

Table 7: Training hyperparameters for CharLuMA across stages in Section 5.1.

To validate the reliability of the MLLM-as-Judge metric, we analyze its correlation with human judgment. We select a subset of 100 examples from the Chart2NCode test set and gather the outputs of ChartLuMA-6.7B in Python, R, and LaTeX, resulting in a total of 300 figures for assessment. Three independent annotators were recruited based on their expertise in data visualization and proficiency in the relevant programming languages. They conducted the human evaluation using the interface in Figure 15, which strictly mirrors the criteria in Figure 11. The final human score for each chart is derived by averaging the ratings from the three annotators. We calculate the Pearson correlation coefficient (Shi et al., 2025) between the MLLM-as-Judge scores and the human evaluations, yielding a value of 0.7387. This strong correlation demonstrates that the MLLM-as-Judge approach serves as a reliable proxy for human visual assessment.

To verify the stability of the MLLM-as-Judge metric, we conduct five independent evaluation runs using the outputs of ChartLuMA-6.7B on the Chart2NCode test set. This yields consistent mean scores for Python (88.2), R (80.9), and LaTeX (74.3), with negligible standard deviations of 0.07, 0.06, and 0.08, respectively. These results confirm the high stability of MLLM-as-Judge metric.

To further address concerns regarding the reproducibility of closed-source APIs, we compare our primary judge (GPT-4o) against two leading open-source alternatives: Qwen3-VL-8B (Team, 2025) and InternVL3.5-8B (Wang et al., 2025). Using the same subset of 100 examples and identical scoring criteria (Figure 11), we observe high Pearson correlations between the open-source judges

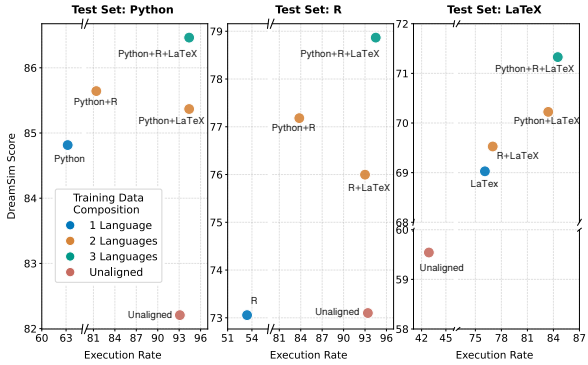


Figure 6: Ablation study of language structure using CharLuMA-1.3B on the Chart2NCode test set.

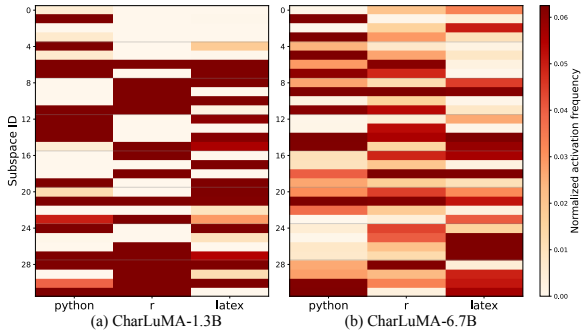


Figure 7: Heatmap of subspace activation frequency for CharLuMA.

and GPT-4o (0.8728 for Qwen3-VL-8B and 0.8540 for InternVL3.5-8B). Furthermore, both models demonstrate high alignment with human annotators (achieving correlations of 0.6975 and 0.6733 respectively). These results confirm that our evaluation protocol is robust and can be reliably reproduced using accessible open-source weights.

B.3 Detailed Analysis Setting

Alternative Architecture. We compare our language-guided low-rank subspace adapter with two alternative connector architectures: a linear MLP and a Mixture-of-MLP. In the linear MLP setting, the pretrained MLP block, initialized on chart–JSON pairs, is directly fine-tuned on Chart2NCode. In the Mixture-of-MLP setting, four experts are initialized from the pretrained MLP block, one of which is frozen as a shared expert, while the remaining three serve as language-specific experts. Hard routing is applied such that, in a Python generation task, the Python-specific expert is activated jointly with the shared expert. This setup mirrors the configuration with four experts in total, of which two are activated for each time, as reported in prior studies (Li et al., 2025; Xu et al., 2025). Warm-up training is also employed in this setting, followed by continued training with the LLM backbone.

Language Structure Ablation. We examine the impact of language diversity by restricting the number of plotting languages involved during training. For the language-controlled settings, we utilize strictly the target language scripts for each chart image. We further include an unaligned baseline where each chart image is paired with its original raw script from the source data described in Appendix A.1. To normalize the total training steps, we inversely scale the visual data: while the single-language models utilize the full 175K charts in the Chart2NCode training set, the two-language and three-language settings are restricted to random subsets of 1/2 (about 87.5K) and 1/3 (about 58.3K) of the chart images, respectively. The ensures a constant 175K chart-script pairs during training across all configurations. The training strategy is consistent with Section 4.2 for all configurations. We adjust the number of routers to match the target language count.

To validate that our findings are not artifacts of the downsampling strategy, we conduct a complementary ablation where the training budget is aligned to the three-language training without visual data reduction. In this regime, we allow all configurations to utilize the complete set of 175k unique chart images. To equate the computational cost with the three-language setting that trains on 525K chart-script pairs, we oversample the configurations with fewer targets: the single-language dataset is replicated three times, and the two-language dataset is augmented by randomly duplicating half of the available samples to reach the equivalent scale. Results in Figure 6 show that the three-language model remains the top performer across all evaluation languages.

Shared Subspace Ratio. We visualize the subspace activation pattern of language-specific routers in ChartLuMA in Figure 7. For quantification, we introduce the *shared-subspace ratio*, which measures how much different language-specific routers rely on the same experts when processing the same chart. Formally, for each chart example c, let $S_{c,l} \subseteq \{0, \ldots, N-1\}$ denote the set of activated experts chosen by the router for language l, with $N = 32$ in our standard setting. Each router activates a fixed number of experts (top–k, with $k = 16$ in our experiments). Given the set

Model	Prompt Version	Chart2NCode		
		ER	DS	MJ
Claude-Sonnet-4	Default	94.9	81.6	81.1
	Variant 1	94.9	81.7	81.2
	Variant 2	95.1	81.6	81.2
Qwen3-VL-8B	Default	80.7	74.4	65.0
	Variant 1	80.8	74.3	64.8
	Variant 2	80.5	74.3	65.1
CharLuMA-1.3B	Default	91.1	78.9	72.3
	Variant 1	91.0	79.1	72.5
	Variant 2	91.2	79.0	72.3

Table 8: Sensitivity study of evaluation prompt on the Chart2NCode test set.

of languages $\mathcal{L}_c$ available for chart c, we define $I_c = \bigcap_{l \in \mathcal{L}_c} S_{c,l}$ and $U_c = \bigcup_{l \in \mathcal{L}_c} S_{c,l}$, where I_c is the set of experts shared across all languages and U_c is the total set of experts activated by any language. The *shared-subspace ratio* for chart c is then $R_c = \frac{|I_c|}{|U_c|}$, which lies in $[0, 1]$. A higher value indicates a dense shared core and a lower value implies strong language-specific specialization.

B.4 Prompt Sensitivity Study

To ensure robustness to specific lexical cues, we compare the standard ChartMimic prompt (Shi et al., 2025) against two variants, while maintaining fixed system messages and output formats. The first variant strips contextual framing to retain only the core directive: *"Generate the <language> code to reproduce the chart in this image."* The second variant employs alternative wording: *"Create a script in <language> that renders the figure shown. Ensure the output matches the visual details of the provided image."* As shown in Table 8, the performance variance across these three settings is negligible, confirming that the model's capabilities are robust to instructional phrasing rather than being artifacts of a specific prompt template.

B.5 Error Analysis

We conduct an error analysis to identify the common sources of execution failures and reproduction limitations of CharLuMA-6.7B. Execution failures in Python and R stem primarily from logic and data discrepancies, led by dimension mismatches (72.3%, 56.1%) and undefined variables (11.9%, 22.0%). In contrast, LaTeX errors are predominantly syntactic, with syntax omissions (55.5%) significantly outweighing undefined variables (33.1%) and dimension mismatches (11.4%). For example, the Python case in Figure 16(a) produces incompatible x–y list lengths when calling the `ax.plot` function. The R case in Figure 16(b) invokes an undefined variable `angle` in a `geom_polygon` call. The LaTeX case in Figure 16(c) fails due to an omitted closing curly brace in the title and x-tick label definition. Regarding reproduction fidelity, our qualitative assessment identifies annotation gaps as the dominant failure mode, manifested as mislabeled groups in Figure 16(a) or hallucinated text annotations in Figure 16(d). We also observe chart subtypes inaccuracies, illustrated by the generation of stacked instead of grouped error bars in Figure 16(b). Finally, stylistic inconsistencies remain prevalent, ranging from malformed x-ticks and incorrect ordering in R (Figure 16(c)) to deviant color schemes in LaTeX (Figure 16(d)).

B.6 Case Study

We conduct a qualitative comparison of CharLuMA-6.7B against GPT-4o and ChartCoder using representative cases from Chart2NCode and ChartMimic. In the Chart2NCode examples (Figure 18, Figure 19, and Figure 20), CharLuMA-6.7B demonstrates robust cross-language consistency, successfully reproducing high-quality charts where GPT-4o exhibits reduced reliability and ChartCoder frequently fails to generate valid R or LaTeX scripts. Furthermore, through the four chart-to-Python examples from ChartMimic (Figure 21), we find that CharLuMA-6.7B matches the state-of-the-art performance of GPT-4o and ChartCoder, confirming its ability to handle advanced visual reasoning without compromising single-language proficiency.

C LLM Usage

Large Language Models (LLMs) were used solely for grammatical and stylistic refinement of text originally drafted by the authors. They did not contribute to the research conceptualization, design, or analysis. The authors retain full responsibility for the accuracy and integrity of the final content.

Instruction Prompt for Handling Missing Templates in Post-Debugging

You are provided with a `{original language}` plotting script as shown below. Your task is to transform it to `{target language}` language, starting with "'`{target language symbol}` and ending with "'.
`{original plotting script}`

Figure 8: Instruction prompt for handling missing templates in the post-debugging stage of the automatic annotation pipeline.

Instruction Prompt for Failed Template Execution in Post-Debugging

You are provided with two code snippets. The first is the original code, a `{original language}` plotting script serving as the reference implementation. The second is the transformed code, a version of the original script translated into `{target language}`, which is currently unexecutable due to syntax or logic errors.
Original Code: `{original plotting script}`
Transformed Code: `{failed template}`
Your task is to identify and correct all errors in the transformed code that prevent it from executing. The corrected script must produce a chart that is semantically equivalent to the one generated by the original code. High-level chart semantics such as axis labels, tick values, bar orientation, or grouping should remain unchanged unless modification is required for successful execution. You may reorder code lines, fix syntax issues, and adjust function arguments as needed. Please output only the corrected code, starting with "'`{target language symbol}` and ending with "'.

Figure 9: Instruction prompt for failed template execution in the post-debugging stage of the automatic annotation pipeline.

Prompt Template of Chart-to-code Generation Task

You are an expert `{target language}` developer who specializes in writing code based on a given picture. I found a very nice picture in a STEM paper, but there is no corresponding source code available. I need your help to generate the `{target language}` code that can reproduce the picture based on the picture I provide.
Now, please give me the code that reproduces the picture below, starting with "'`{target language symbol}` and ending with "'.

Figure 10: Prompt template of chart-to-code generation task (adapted from ChartMimic (Shi et al., 2025)).

Prompt Template of MLLM-as-Judge approach enhanced

You are an excellent judge at evaluating visualization chart plots. The first image (reference image) is created using ground truth matplotlib code, and the second image (AI-generated image) is created using matplotlib code generated by an AI assistant. Your task is to score how well the AI-generated plot matches the ground truth plot.
Scoring Methodology:
The AI-generated image's score is based on the following criteria, totaling a score out of 100 points:
1. Chart Types (20 points): Does the AI-generated image include all chart types present in the reference image (e.g., line charts, bar charts, etc.)?
2. Layout (10 points): Does the arrangement of subplots in the AI-generated image match the reference image (e.g., number of rows and columns)?
3. Text Content (20 points): Does the AI-generated image include all text from the reference image (e.g., titles, annotations, axis labels), excluding axis tick labels?
4. Data (20 points): How accurately do the data trends in the AI-generated image resemble those in the original image and is the number of data groups the same as in the reference image?
5. Style (20 points): Does the AI-generated image match the original in terms of colors (line colors, fill colors, etc.), marker types (point shapes, line styles, etc.), legends, grids, and other stylistic details?
6. Clarity (10 points): Is the AI-generated image clear and free of overlapping elements?
Evaluation:
Compare the two images head to head and provide a detailed assessment. Use the following format for your response:
— Comments:
- Chart Types: ${your comment and subscore}
- Layout: ${your comment and subscore}
- Text Content: $your comment and subscore
- Data: ${your comment and subscore}
- Style: ${your comment and subscore}
- Clarity: ${your comment and subscore}
Score: ${your final score out of 100} —
Please use the above format to ensure the evaluation is clear and comprehensive.

Figure 11: MLLLM-as-Judge prompt template for chart-to-code generation evaluation (adapted from ChartMimic (Shi et al., 2025))

Source Script in Python

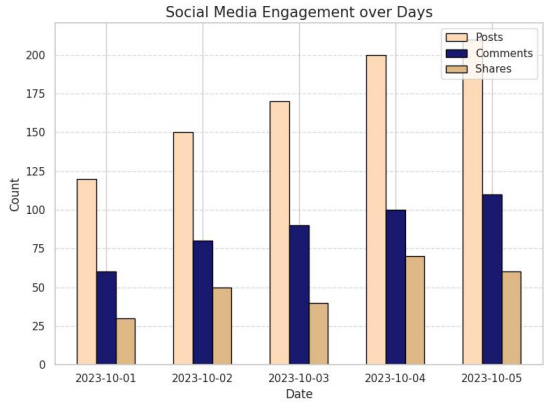
```python
import matplotlib.pyplot as plt
import numpy as np

dates = ['2023-10-01', '2023-10-02', '2023-10-03', '2023-10-04', '2023-10-05']
posts = [120, 150, 170, 200, 210]
comments = [60, 80, 90, 100, 110]
shares = [30, 50, 40, 70, 60]

fig, ax = plt.subplots(figsize=(8, 6))

bar_width = 0.2
x = np.arange(len(dates))
palette = ['#FFDAB9', '#191970', '#DEB887']
edge_color = 'black'
bars1 = ax.bar(x - bar_width, posts, width=bar_width, color=palette[0],
edgecolor=edge_color, label='Posts')
bars2 = ax.bar(x, comments, width=bar_width, color=palette[1], edgecolor=edge_color,
label='Comments')
bars3 = ax.bar(x + bar_width, shares, width=bar_width, color=palette[2],
edgecolor=edge_color, label='Shares')

ax.set_title('Social Media Engagement over Days', fontsize=15)
ax.set_xticks(x)
ax.set_xticklabels(dates)
ax.set_ylabel('Count', fontsize=12)
ax.set_xlabel('Date', fontsize=12)
ax.grid(True, which='both', axis='y', linestyle='--', alpha=0.7)
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, labels, loc='upper right', bbox_to_anchor=(1, 1), ncol=1)
plt.tight_layout()
plt.show()
```



Metadata


```
{   "plot_size": {"width": 8.0, "height": 6.0, "unit": "inch"},
    "twin_axes": {},
    "axes_layout": { "n_row": 1, "n_col": 1},
    "facecolor": "#ffffff",
    "ax_0": {
        "type_agnostic": {
            "axis": {"type": "rectilinear", "aspect": "auto"},
            "title": {"content": "Social Media Engagement over Days",
                "size": 15.0, "style": "normal,normal"},
            "x_label": {"content": "Date", "size": 12.0,
                "style": "normal,normal"},
            "y_label": {"content": "Count", "size": 12.0,
                "style": "normal,normal"
            },
            "x_ticks": [{"text": "2023-10-01", "position": ["0",0]},
                {"text": "2023-10-02", "position": ["1",0]},
                {"text": "2023-10-03", "position": ["2",0]},
                {"text": "2023-10-04", "position": ["3",0]},
                {"text": "2023-10-05", "position": ["4",0]}],
            "y_ticks": [{"text": "0", "position": [0,0.0]},
                {"text": "25", "position": [0,25.0]},
                {"text": "50", "position": [0,50.0]},
                {"text": "75", "position": [0,75.0]},
                {"text": "100", "position": [0,100.0]},
                {"text": "125", "position": [0,125.0]},
                {"text": "150", "position": [0,150.0]},
                {"text": "175", "position": [0,175.0]},
                {"text": "200", "position": [0,200.0]},
                {"text": "225", "position": [0,225.0]},],
            "legend": {"exist": true, "loc": 1,"ncol": 1},
            "grid": {"x": true, "y": true},
            "panel_box": true,
            "background_color": "#ffffff",
            "annotation": [],
            "label_to_color": {"Posts": "#ffdab9",
                "Comments": "#191970","Shares": "#deb887"},
            "container_type": [
                "BarContainer","BarContainer","BarContainer"]
        },
```



```
"type_specific": {
    "type": ["bar"], "sub_type": "grouped-bar", "orientation": "vertical",
    "template": ["bar_grouped_vertical_r.jinja", "bar_grouped_vertical_latex.jinja"]}
```


Metadata-Template Matching


```
"object": {
"patches": [
{"object_type": "Rectangle", "facecolor": "#ffdab9", "linewidth": 1.0, "linestyle": "solid",
"hatch": null, "geometry": {"x": -0.3, "y": 0.0, "width": 0.2, "height": 120.0}},
{"object_type": "Rectangle", "facecolor": "#ffdab9", "linewidth": 1.0, "linestyle": "solid",
"hatch": null, "geometry": {"x": 0.7, "y": 0.0,"width": 0.2, "height": 150.0}},
{"object_type": "Rectangle", "facecolor": "#ffdab9", "linewidth": 1.0, "linestyle": "solid",
"hatch": null, "geometry": {"x": 1.7, "y": 0.0, "width": 0.2, "height": 170.0}},
{"object_type": "Rectangle", "facecolor": "#ffdab9", "linewidth": 1.0, "linestyle": "solid",
"hatch": null, "geometry": {"x": 2.7, "y": 0.0, "width": 0.2, "height": 200.0}},
{"object_type": "Rectangle", "facecolor": "#ffdab9", "linewidth": 1.0, "linestyle": "solid",
"hatch": null, "geometry": {"x": 3.7, "y": 0.0, "width": 0.2, "height": 210.0}},
{"object_type": "Rectangle", "facecolor": "#191970", "linewidth": 1.0, "linestyle": "solid",
"hatch": null, "geometry": {"x": -0.1, "y": 0.0, "width": 0.2, "height": 60.0}},
{"object_type": "Rectangle", "facecolor": "#191970", "linewidth": 1.0, "linestyle": "solid",
"hatch": null, "geometry": {"x": 0.9, "y": 0.0, "width": 0.2, "height": 80.0}},
{"object_type": "Rectangle", "facecolor": "#191970", "linewidth": 1.0, "linestyle": "solid",
"hatch": null, "geometry": {"x": 1.9, "y": 0.0, "width": 0.2, "height": 90.0}},
{"object_type": "Rectangle", "facecolor": "#191970", "linewidth": 1.0, "linestyle": "solid",
"hatch": null, "geometry": {"x": 2.9, "y": 0.0, "width": 0.2, "height": 100.0}},
{"object_type": "Rectangle", "facecolor": "#191970", "linewidth": 1.0, "linestyle": "solid",
"hatch": null, "geometry": {"x": 3.9, "y": 0.0, "width": 0.2, "height": 110.0}},
{"object_type": "Rectangle", "facecolor": "#deb887", "linewidth": 1.0, "linestyle": "solid",
"hatch": null, "geometry": {"x": 0.1, "y": 0.0, "width": 0.2, "height": 30.0}},
{"object_type": "Rectangle", "facecolor": "#deb887", "linewidth": 1.0, "linestyle": "solid",
"hatch": null, "geometry": {"x": 1.1, "y": 0.0, "width": 0.2, "height": 50.0}},
{"object_type": "Rectangle", "facecolor": "#deb887", "linewidth": 1.0, "linestyle": "solid",
"hatch": null, "geometry": {"x": 2.1, "y": 0.0, "width": 0.2, "height": 40.0}},
{"object_type": "Rectangle", "facecolor": "#deb887", "linewidth": 1.0, "linestyle": "solid",
"hatch": null, "geometry": {"x": 3.1, "y": 0.0, "width": 0.2, "height": 70.0}},
{"object_type": "Rectangle", "facecolor": "#deb887", "linewidth": 1.0, "linestyle": "solid",
"hatch": null, "geometry": {"x": 4.1, "y": 0.0, "width": 0.2, "height": 60.0}}],
"lines": [], "collections": [], "images": []}}}
```


Script in R

```r
library(ggplot2)

tick_labels  <- c('2023-10-01','2023-10-02','2023-10-03','2023-10-04','2023-10-05')
group_labels <- c('Posts','Comments','Shares')
values <- list(c(120,150,170,200,210), c( 60, 80, 90,100,110), c( 30, 50, 40, 70,
60))
colors <- c('#ffdab9','#191970','#deb887')
df <- data.frame(
  Category = factor(rep(tick_labels, times = length(group_labels)), levels =
tick_labels),
  Group    = factor(rep(group_labels, each  = length(tick_labels)), levels =
group_labels),
  Value    = unlist(values)
)

p <- ggplot(df, aes(x = Category, y = Value, fill = Group)) +
  geom_bar(stat = "identity", position = position_dodge(width = 0.8), width = 0.8,
show.legend = TRUE) +
  scale_fill_manual(values = colors, breaks = group_labels, labels = group_labels) +
  labs(title = "Social Media Engagement over Days", x = "Date", y = "Count", fill =
"Group") +
  theme(
    plot.title   = element_text(size = 15, hjust = 0.5, face = "plain"),
    axis.title.x = element_text(size = 12, face = "plain"),
    axis.title.y = element_text(size = 12, face = "plain"),
    panel.background = element_rect(fill = "#ffffff"),
    panel.grid.major = element_line(color = "grey"),
    panel.border     = element_rect(colour = "black", fill = NA, size = 0.5),
    legend.position  = "right"
  )
p <- p + scale_y_continuous(breaks = c(0.0, 25.0, 50.0, 75.0, 100.0, 125.0, 150.0,
175.0, 200.0, 225.0), labels = c('0', '25', '50', '75', '100', '125', '150', '175',
'200', '225'))
p <- p + guides(fill = guide_legend(ncol = 1))
print(p)
```

Script in LaTex

```latex
\documentclass{standalone}
\usepackage{pgfplots}
\pgfplotsset{compat=1.18}
\usepgfplotslibrary{groupplots}
\usepackage[x11names, rgb]{xcolor}
\definecolor{c00}{HTML}{FFDAB9}
\definecolor{c01}{HTML}{191970}
\definecolor{c02}{HTML}{DEB887}
\definecolor{cb}{HTML}{FFFFFF}

\begin{document}
\begin{tikzpicture}
\begin{axis}{
    ybar, bar width=0.2, width=8.0in, height=6.0in,
    title=Social Media Engagement over Days, title style={font=\large, align=center},
    xlabel=Date, x tick label style={font=\small, align=center},
    ylabel=Count, y tick label style={font=\small, align=center},
    xtick={0, 1, 2, 3, 4}, xticklabels={{2023-10-01}, {2023-10-02}, {2023-10-03},
{2023-10-04}, {2023-10-05}},
    xtick align=center, enlarge x limits=0.2, ymin=0, grid=major, axis lines=box,
    legend style={legend pos=north east, legend columns=1}, axis
background/.style={fill=cb}
]

\addplot+[ ybar, fill=c00, bar shift=-0.180 ] coordinates {
    (0, 120.0) (1, 150.0) (2, 170.0) (3, 200.0) (4, 210.0)};
\addplot+[ ybar, fill=c01, bar shift=0.000 ] coordinates {
    (0, 60.0) (1, 80.0) (2, 90.0) (3, 100.0) (4, 110.0)};
\addplot+[ ybar, fill=c02, bar shift=0.180 ] coordinates {
    (0, 30.0) (1, 50.0) (2, 40.0) (3, 70.0) (4, 60.0)};
\legend{ {Posts}, {Comments}, {Shares} }

\end{axis}
\end{tikzpicture}
\end{document}
```

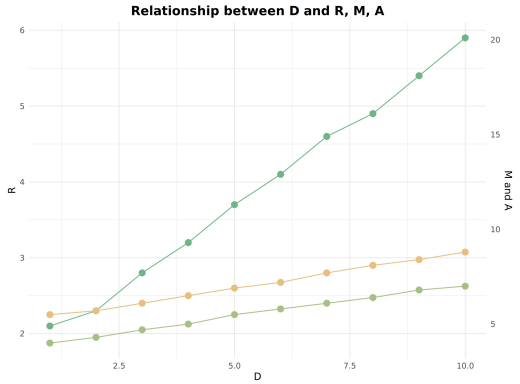
Figure 12: Case study of annotation pipeline in a vertical grouped bar chart.

Source Script in R

```
library(ggplot2)

data <- data.frame(
  D = 1:10,
  R = c(2.1, 2.3, 2.8, 3.2, 3.7, 4.1, 4.6, 4.9, 5.4, 5.9),
  M = c(5.5, 5.7, 6.1, 6.5, 6.9, 7.2, 7.7, 8.1, 8.4, 8.8),
  A = c(4.0, 4.3, 4.7, 5.0, 5.5, 5.8, 6.1, 6.4, 6.8, 7.0)
)

ggplot(data, aes(x = D)) +
  geom_line(aes(y = R), color = "#6FB585") +
  geom_point(aes(y = R), size = 3, color = "#6FB585") +
  geom_line(aes(y = M * 0.25 + 0.875), color = "#E8BF80") +
  geom_point(aes(y = M * 0.25 + 0.875), size = 3, color = "#E8BF80") +
  geom_line(aes(y = A * 0.25 + 0.875), color = "#A8BF85") +
  geom_point(aes(y = A * 0.25 + 0.875), size = 3, color = "#A8BF85") +
  scale_y_continuous(name = 'R', sec.axis = sec_axis(~(.-0.875)/0.25, name = 'M and
A')) +
  labs(
    title = "Relationship between D and R, M, A",
    x = "D",
    y = "R"
  ) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, size = 14, face = "bold"))
dev.off()
```




```
"type_specific": {
    "type": ["line"], "sub_type": "dotted-line", "orientation": "vertical",,
    "template": ["line_multi_marker_python.jinja", "line_multi_marker_latex.jinja"]}
```

Metadata

```
{   "plot_size": {"width": 7, "height": 7, "unit": "inch"},
    "twin_axes": [],
    "axes_layout": { "n_row": 1, "n_col": 1},
    "facecolor": "#ffffff",
    "ax_0": {
        "type_agnostic": {
        "axis": {"position": null, "type": "cartesian", "aspect": null},
        "title": {"content": "Relationship between D and R, M, A",
                "size": 14, "style": "bold"},
        "x_label": {"content": "D", "size": "NA", "style": null},
        "y_label": {"content": "R", "size": "NA", "style": null},
        "x_ticks": [{"text": "2.5", "position": [2, 0]},
            {"text": "5", "position": [3, 0]},
            {"text": "7.5", "position": [4, 0]},
            {"text": "10", "position": [5, 0]}], "
        y_ticks": [ {"text": "2", "position": [0, 1]},
            {"text": "3", "position": [0, 2]},
            {"text": "4", "position": [0, 3]},
            {"text": "5", "position": [0, 4]},
            {"text": "6", "position": [0, 5]}],
        "legend": {"exist": false, "loc": null, "ncol": null},
        "grid": {"x": true, "y": true},
        "panel_box": false,
        "background_color": "#ffffff",
        "annotation": [],
        "label_to_color": []
    },
```

Metadata-Template Matching

```
{
  "object": {
    "patches": [],
    "lines": [
      {"object_type": "GeomLine", "color": "#6FB585", "linewidth": 0.5, "linestyle": 1,
       "geometry": {"x": [1,2,3,4,5,6,7,8,9,10], "y": [2.1,2.3,2.8,3.2,3.7,4.1,4.6,4.9,5.4,5.9]}},
      {"object_type": "GeomLine", "color": "#E8BF80", "linewidth": 0.5, "linestyle": 1,
       "geometry": {"x": [1,2,3,4,5,6,7,8,9,10], "y":
[2.25,2.3,2.4,2.5,2.6,2.675,2.8,2.9,2.975,3.075]}},
      {"object_type": "GeomLine", "color": "#A8BF85", "linewidth": 0.5, "linestyle": 1,
       "geometry": {"x": [1,2,3,4,5,6,7,8,9,10], "y":
[1.875,1.95,2.05,2.125,2.25,2.325,2.4,2.475,2.575,2.625]}}
    ],
    "collections": [
      {"object_type": "GeomPoint", "facecolors": [None]*10, "edgecolors": ["#6FB585"]*10,
       "linewidths": [0.5]*10, "sizes": [3]*10, "shape": [19]*10,
       "geometry": [[1,2.1],[2,2.3],[3,2.8],[4,3.2],[5,3.7],[6,4.1],[7,4.6],[8,4.9],[9,5.4],
[10,5.9]]},
      {"object_type": "GeomPoint", "facecolors": [None]*10, "edgecolors": ["#E8BF80"]*10,
       "linewidths": [0.5]*10, "sizes": [3]*10, "shape": [19]*10,
       "geometry": [[1,2.25],[2,2.3],[3,2.4],[4,2.5],[5,2.6],[6,2.675],[7,2.8],[8,2.9],[9,2.975],
[10,3.075]]},
      {"object_type": "GeomPoint", "facecolors": [None]*10, "edgecolors": ["#A8BF85"]*10,
       "linewidths": [0.5]*10, "sizes": [3]*10, "shape": [19]*10,
       "geometry": [[1,1.875],[2,1.95],[3,2.05],[4,2.125],[5,2.25],[6,2.325],[7,2.4],[8,2.475],
[9,2.575],[10,2.625]]}
    ],
    "images": []
  }
}
```

Script in Python

```
import matplotlib.pyplot as plt

num_group = 3
x_values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y_values = [[2.1, 2.3, 2.8, 3.2, 3.7, 4.1, 4.6, 4.9, 5.4, 5.9], [2.25, 2.3, 2.4,
2.5, 2.6, 2.675, 2.8, 2.9, 2.975, 3.075], [1.875, 1.95, 2.05, 2.125, 2.25, 2.325,
2.4, 2.475, 2.575, 2.625]]
line_color = ['#6fb585', '#e8bf80', '#a8bf85']
line_style = ['-', '-', '-']
line_width = [1.0, 1.0, 1.0]
marker_color = ['#6fb585', '#e8bf80', '#a8bf85']
marker_style = ['o', 'o', 'o']
marker_size = [8, 8, 8]

fig, ax = plt.subplots(figsize=(7, 7))

for i in range(num_group):
    ax.plot(
        x_values,
        y_values[i],
        color=line_color[i],
        linestyle=line_style[i],
        linewidth=line_width[i],
        marker=marker_style[i],
        markersize=marker_size[i],
        markerfacecolor=marker_color[i]
    )

ax.set_xlabel("D", fontsize=12, fontweight="normal", fontstyle="normal")
ax.set_ylabel("R", fontsize=12, fontweight="normal", fontstyle="normal")
ax.set_title("Relationship between D and R, M, A", fontsize=14, fontweight="bold",
fontstyle="normal")
ax.set_yticks([2.0, 3.0, 4.0, 5.0, 6.0])
ax.set_yticklabels(['2', '3', '4', '5', '6'])
ax.grid(True)
for spine in ax.spines.values():
    spine.set_visible(False)
ax.set_facecolor("#ffffff")

plt.tight_layout()
plt.show()
```

Script in LaTex

```
\documentclass{standalone}
\usepackage{pgfplots}
\pgfplotsset{compat=1.18}
\usepgfplotslibrary{fillbetween}
\usepackage[x11names, rgb]{xcolor}
\definecolor{c00}{HTML}{6FB585}
\definecolor{c01}{HTML}{E8BF80}
\definecolor{c02}{HTML}{A8BF85}
\definecolor{cb}{HTML}{FFFFFF}
\begin{document}
\begin{tikzpicture}
\begin{axis}[
    width=7in, height=7in,
    title=Relationship between D and R, M, A, title style={font=\normalsize\bfseries},
    xlabel=D, x tick label style={font=\normalsize, align=center},
    ylabel=R, y tick label style={font=\normalsize, align=center},
    ytick={2.0, 3.0, 4.0, 5.0, 6.0}, yticklabels={{2},{3},{4},{5},{6},},
    enlarge x limits=0.05, enlarge y limits=0.05,
    grid=major, axis lines=none, axis background/.style={fill=cb}
]
\addplot+[
    color=c00, mark=o, mark options={fill=c00, scale=3pt},
    line width=0.2pt, style=solid
] coordinates {
    (1, 2.1) (2, 2.3) (3, 2.8) (4, 3.2) (5, 3.7) (6, 4.1) (7, 4.6) (8, 4.9) (9, 5.4) (10,
5.9)};
\addplot+[
    color=c01, mark=o, mark options={fill=c01, scale=3pt},
    line width=0.2pt, style=solid
] coordinates {
    (1, 2.25) (2, 2.3) (3, 2.4) (4, 2.5) (5, 2.6) (6, 2.675) (7, 2.8) (8, 2.9) (9, 2.975)
(10, 3.075)};
\addplot+[
    color=c02, mark=o, mark options={fill=c02, scale=3pt},
    line width=0.2pt, style=solid
] coordinates {
    (1, 1.875) (2, 1.95) (3, 2.05) (4, 2.125) (5, 2.25) (6, 2.325) (7, 2.4) (8, 2.475) (9,
2.575) (10, 2.625)};
\end{axis}
\end{tikzpicture}
\end{document}
```

Figure 13: Case study of annotation pipeline in a dotted line chart.

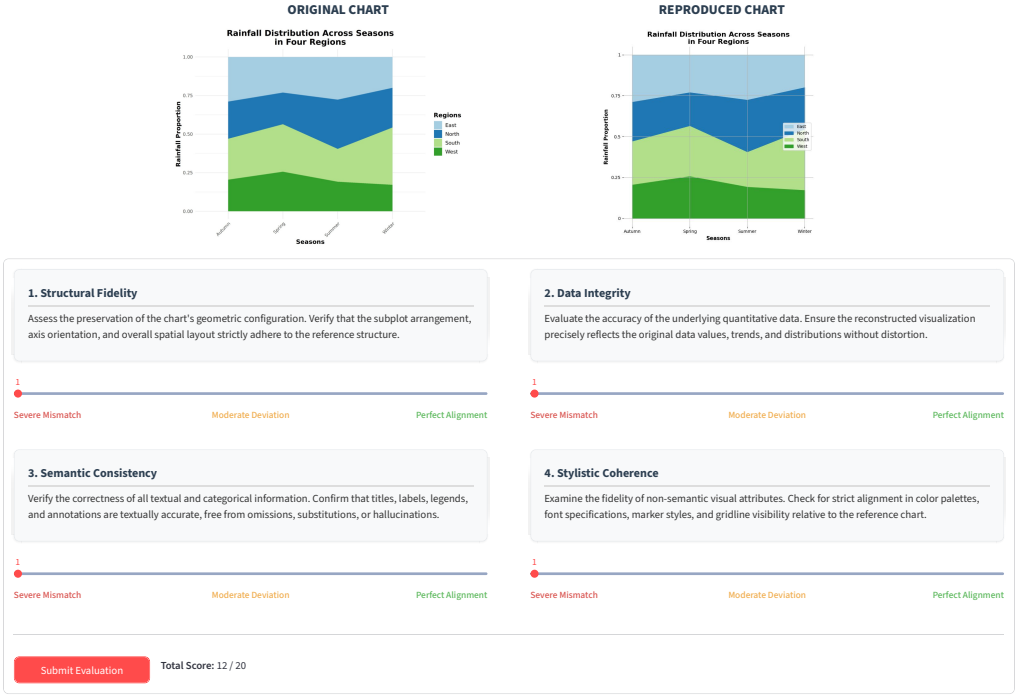


Figure 14: Screenshot of the human quality checking questionnaire.

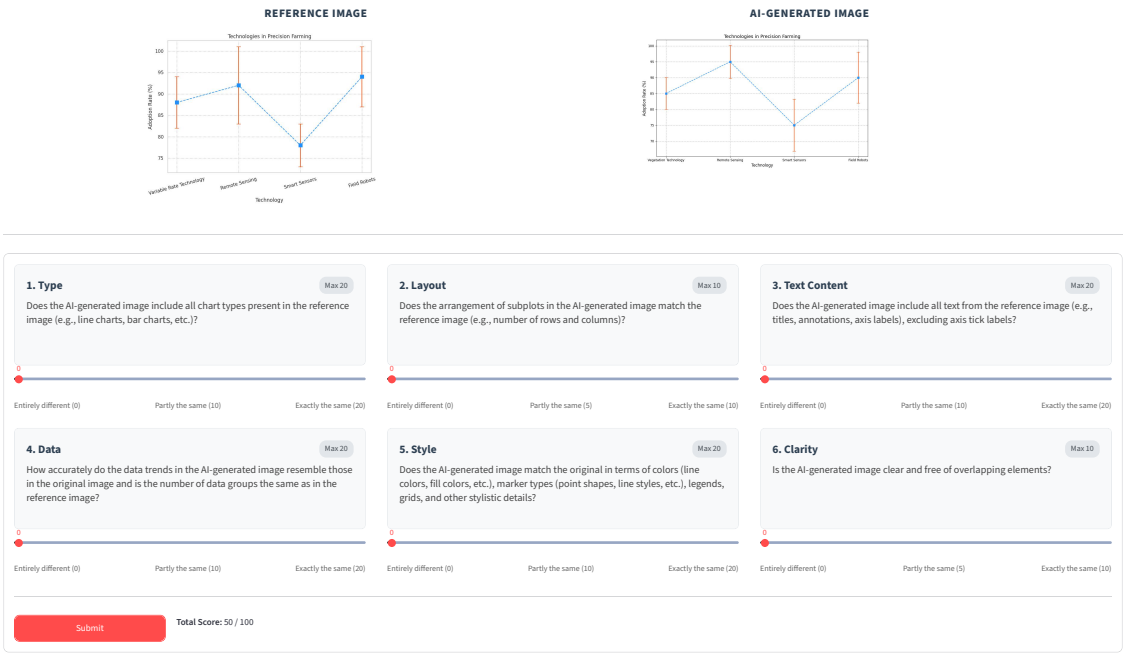
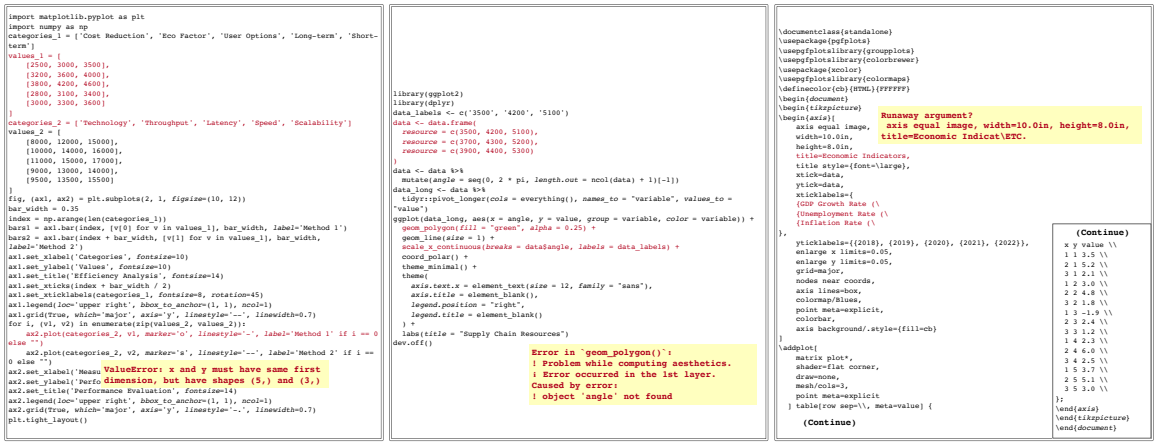


Figure 15: Screenshot of the human evaluation questionnaire for MLLM-as-judge metrics.



(a) Error Case for Python (b) Error Case for R (c) Error Case for LaTex

Figure 16: Case study of execution errors in generated code for CharLuMA-6.7B.

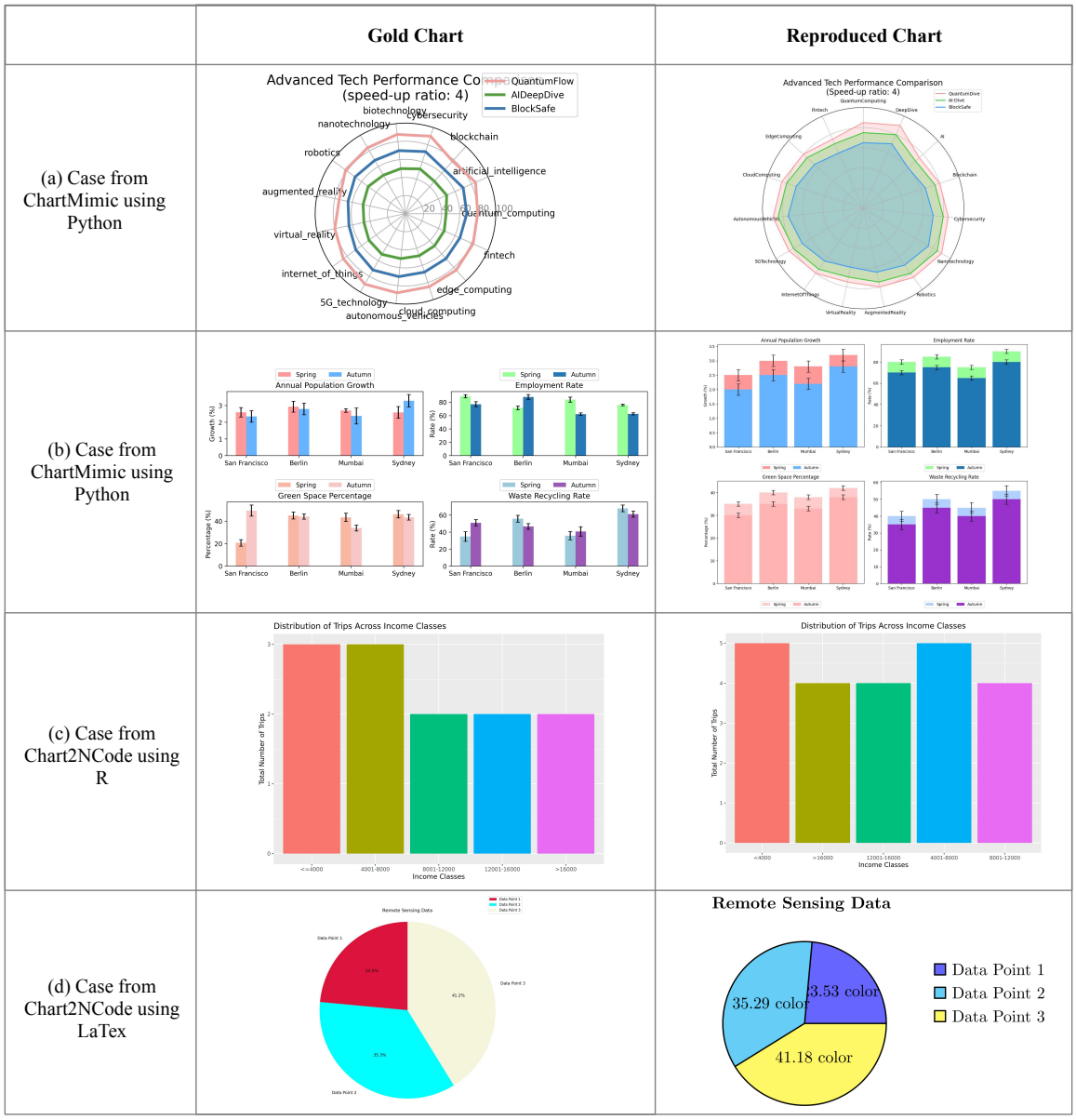


Figure 17: Case study of reproduction errors in generated charts for CharLuMA-6.7B.

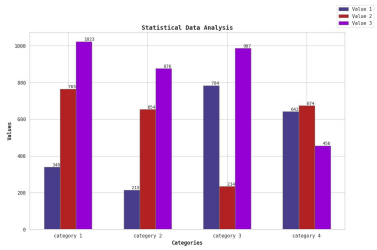
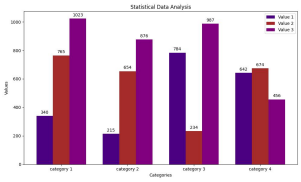
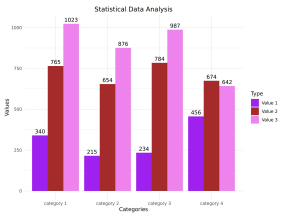
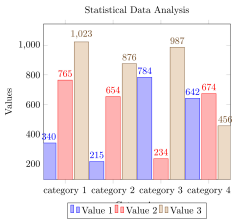
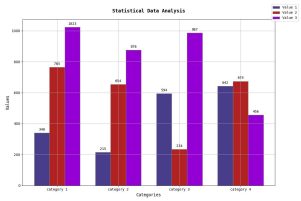
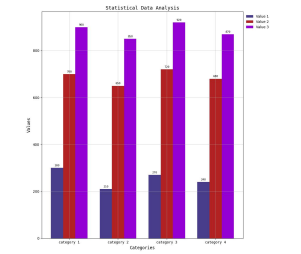
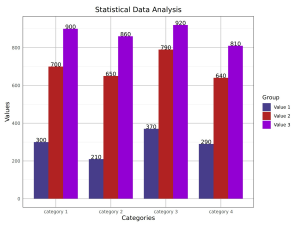
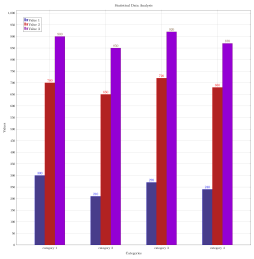
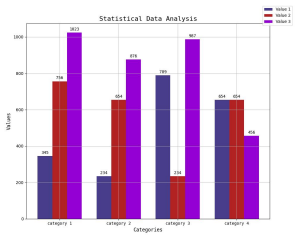
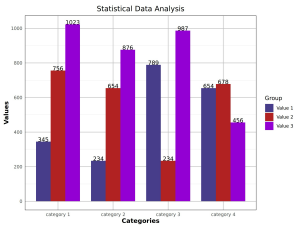
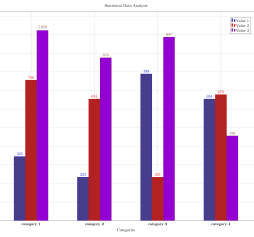
Gold Chart			
	Python	**R**	**Latex**
GPT-4o			
ChartCoder		Fail to Execute	Statistical Data Analysis Data **1 Introduction** Figure 1: Bar Chart **2 Conclusion**
CharLuMA-1.3B			
CharLuMA-6.7B			

Figure 18: Case study of a grouped bar chart input and generated outputs from the Chart2NCode test set across three plotting languages.

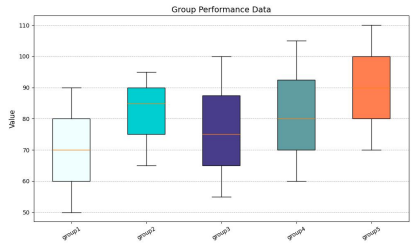
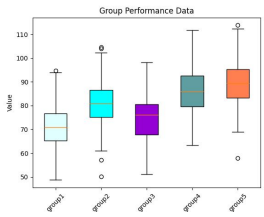
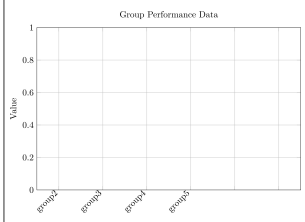
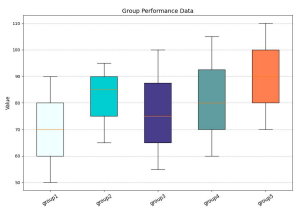
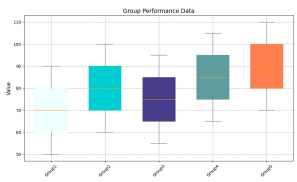
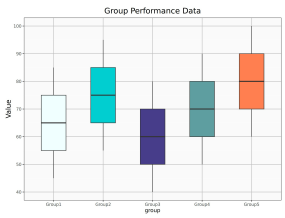
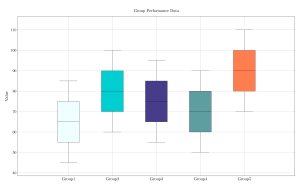
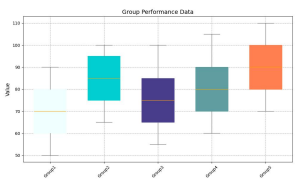
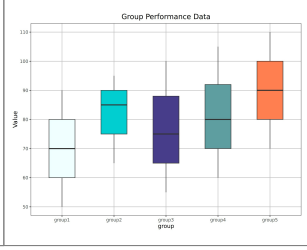
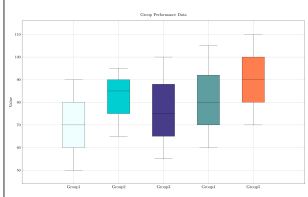
<table>
<tr><td>Gold Chart</td><td colspan="3"> 
</td></tr>
<tr><td></td><td>Python</td><td>R</td><td>Latex</td></tr>
<tr><td>GPT-4o</td><td> 
</td><td>
</td><td> 
</td></tr>
<tr><td>ChartCoder</td><td> 
</td><td>Fail to Execute</td><td>Fail to Execute</td></tr>
<tr><td>CharLuMA-1.3B</td><td> 
</td><td> 
</td><td>  </td></tr>
<tr><td>CharLuMA-6.7B</td><td> 
</td><td> 
</td><td>  </td></tr>
</table>

Figure 19: Case study of a box chart input and generated outputs from the Chart2NCode test set across three plotting languages.

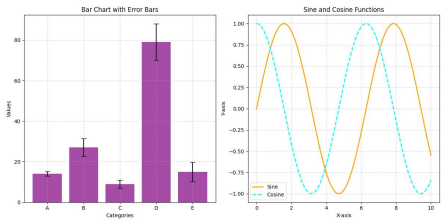
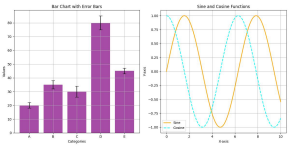
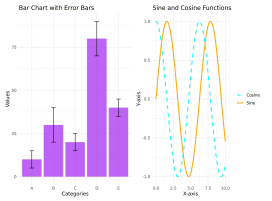
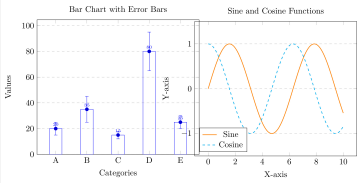
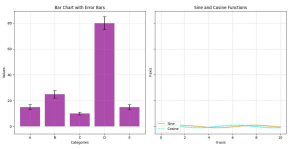
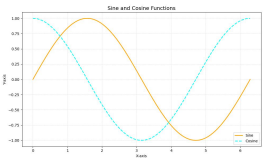
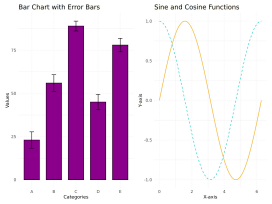
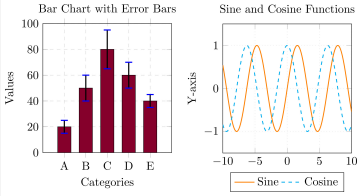
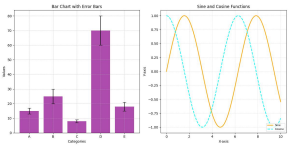
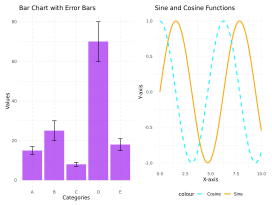
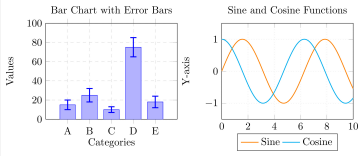
Gold Chart			
	Python	**R**	**Latex**
GPT-4o			
ChartCoder		Fail to Execute	Fail to Execute
CharLuMA-1.3B			
CharLuMA-6.7B			

Figure 20: Case study of a two-subplot chart input and generated outputs from the Chart2NCode test set across three plotting languages.

Figure 21: Case study of model inputs and generated outputs from ChartMimic in Python.