CharLuMA: Efficient Multi-Language Chart-to-Code Generation with Low-Rank Subspace Adaptation

Anonymous authors
Paper under double-blind review

Abstract

Chart-to-code generation involves translating a chart image into an executable plotting script. However, prior work has largely focused on Python-only solutions, limiting real-world applicability and leaving the learning signals inherent in cross-language equivalences untapped. We argue that aligned multi-language scripts serve as complementary "views" of the same chart, providing mutual guidance to regularize the visual-to-code mapping. As an instantiation of this idea, we introduce CharLuMA – a multimodal large language model (MLLM) that integrates a language-guided mixture of low-rank subspaces into its multimodal projector. This architecture enables parameter-efficient adaptation via dynamic routing to language-specific subspaces, while preserving shared visual-semantic representations of charts. To facilitate training and evaluation at scale, we present Chart2NCode, a dataset of 176k Chart–Python–R–LaTeX quadruples that maintain consistent visual equivalence across languages. Experiments on multiple benchmarks demonstrate that CharLuMA achieves state-of-the-art performance among open-source MLLMs and even surpasses some proprietary systems. Critically, training with more diverse and balanced language sets yields consistent and substantial improvements across all languages by leveraging the rich supervisory signals embedded in cross-language equivalences. Subspace activation analysis further reveals a hybrid allocation pattern, with compact shared cores complemented by broader language-specific zones, while stronger models exhibit smoother and more balanced allocations. Taken together, these results establish multi-language alignment as an effective supervision paradigm for achieving universal chart-to-code generation[1].

1 Introduction

Chart-to-code generation is the task of translating charts into executable plotting scripts that accurately reconstruct the underlying data and visual design, positioned at the intersection of visual understanding, code generation, and cross-modal reasoning (Shi et al., 2025). The demand for automated chart reproduction is increasing in various domains such as science, finance, and biology. Recent advances in multimodal large language models (MLLMs) have demonstrated impressive performance across a wide range of vision–language tasks, even approaching human-level capability (Yue et al., 2024; Lu et al., 2023; Wang et al., 2025b; Zhang et al., 2025). Nevertheless, chart-to-code generation remains a particularly demanding problem, requiring models to recover structured data, interpret intricate visual encodings, and produce precise, executable code with strict fidelity.

Existing works focus on translating charts into single-language codes, predominantly using matplotlib in Python (Shi et al., 2025; Zhao et al., 2025; Wu et al., 2025; Belouadi et al., 2024b;a). For example, ChartMimic (Shi et al., 2025) introduced a benchmark with human-curated matplotlib scripts for chart reconstruction, and ChartCoder (Zhao et al., 2025) trained a code-focused large language model (LLM) on large-scale chart–Python pairs. While effective within the Python ecosystem, this line of research overlooks the diversity of plotting libraries and languages used in practice—analysts in many fields rely on R (ggplot2) or LaTeX (TikZ), among others, to create charts.

[1] Codes and data are available at `https://anonymous.4open.science/r/CharLuMA-226D`.

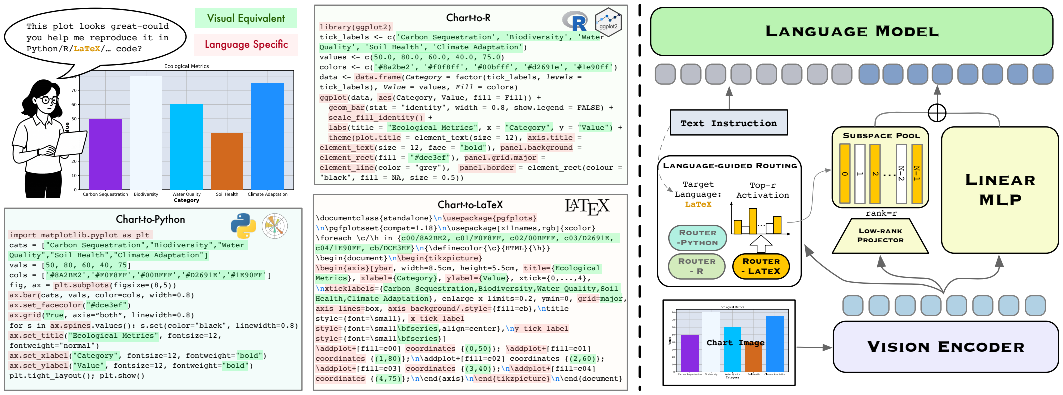


Chart-to-R

```
library(ggplot2)
tick_labels <- c('Carbon Sequestration', 'Biodiversity', 'Water
Quality', 'Soil Health', 'Climate Adaptation')
values <- c(50.0, 80.0, 60.0, 40.0, 75.0)
colors <- c('#8a2be2', '#f0f8ff', '#00bfff', '#d2691e', '#1e90ff')
data <- data.frame(Category = factor(tick_labels, levels =
tick_labels), Value = values, Fill = colors)
ggplot(data, aes(Category, Value, fill = Fill)) +
  geom_bar(stat = "identity", width = 0.8, show.legend = FALSE) +
  scale_fill_identity() +
  labs(title = "Ecological Metrics", x = "Category", y = "Value") +
  theme(plot.title = element_text(size = 12), axis.title =
element_text(size = 12, face = "bold"), panel.background =
element_rect(fill = "#dce3ef"), panel.grid.major =
element_line(color = "grey"),  panel.border = element_rect(colour =
"black", fill = NA, size = 0.5))
```

Chart-to-Python

```
import matplotlib.pyplot as plt
cats = ["Carbon Sequestration","Biodiversity","Water
Quality","Soil Health","Climate Adaptation"]
vals = [50, 80, 60, 40, 75]
cols = ['#8A2BE2','#F0F8FF','#00BFFF','#D2691E','#1E90FF']
fig, ax = plt.subplots(figsize=(8,5))
ax.bar(cats, vals, color=cols, width=0.8)
ax.set_facecolor("#dce3ef")
ax.grid(True, axis="both", linewidth=0.8)
for s in ax.spines.values(): s.set(color="black", linewidth=0.8)
ax.set_title("Ecological Metrics", fontsize=12,
fontweight="normal")
ax.set_xlabel("Category", fontsize=12, fontweight="bold")
ax.set_ylabel("Value", fontsize=12, fontweight="bold")
plt.tight_layout(); plt.show()
```

Chart-to-LaTeX

```
\documentclass{standalone}\n\usepackage{pgfplots}
\n\pgfplotsset{compat=1.18}\n\usepackage[x11names,rgb]{xcolor}
\foreach \c/\h in {c00/8A2BE2, c01/F0F8FF, c02/00BFFF, c03/D2691E,
c04/1E90FF, cb/DCE3EF}\n{\definecolor{\c}{HTML}{\h}}
\begin{document}\n\begin{tikzpicture}
\begin{axis}[ybar, width=8.5cm, height=5.5cm, title={Ecological
Metrics}, xlabel={Category}, ylabel={Value}, xtick={0,...,4},
\nxticklabels={Carbon Sequestration,Biodiversity,Water Quality,Soil
Health,Climate Adaptation}, enlarge x limits=0.2, ymin=0, grid=major,
axis lines=box, axis background/.style={fill=cb},\ntitle
style={font=\small}, x tick label
style={font=\small\bfseries,align=center},\ny tick label
style={font=\small\bfseries}]
\addplot+[fill=c00] coordinates {(0,50)}; \addplot+[fill=c01]
coordinates {(1,80)};\n\addplot+[fill=c02] coordinates {(2,60)};
\addplot+[fill=c03] coordinates {(3,40)};\n\addplot+[fill=c04]
coordinates {(4,75)};\n\end{axis}\n\end{tikzpicture}\n\end{document}
```

Figure 1: Overview of chart-to-code generation task and CharLuMA architecture. We introduce CharLuMA, a multimodal large language model for chart-to-code generation that augments the multimodal projector with a language-guided mixture of low-rank subspaces.

This single-language focus limits real-world applicability and overlooks a key opportunity: *cross-language alignment*. If a chart is faithfully expressed in different programming languages, those parallel code snippets essentially offer different "views" of the same visualization. Prior works have left the rich learning signal from such cross-language equivalence untapped.

To address these limitations, we introduce **CharLuMA**, a MLLM for chart-to-code generation that is explicitly trained across multiple programming languages. CharLuMA extends a LLaVA-style architecture (Liu et al., 2023) by enhancing its multimodal projector with a language-guided mixture of low-rank subspaces (Figure 1). In essence, we equip the multimodal projector with a low-rank adapter composed of lightweight subspaces, together with a routing mechanism that activates the appropriate combination according to the target language. This design enables parameter-efficient cross-language adaptation: the model learns to share core visual–semantic representations of charts while dynamically adjusting its internal representation to the syntax and conventions of each specific language. By routing through language-specialized subspaces, CharLuMA strikes a balance between cross-language generality and fine-grained language-specific fidelity in code generation. Moreover, training and evaluating a multi-language chart-to-code model requires data that are precisely aligned across languages. Hence, we present **Chart2NCode**, a large-scale dataset of chart images paired with visually equivalent plotting scripts in Python, R, and LaTeX. Chart2NCode contains 176k examples, including a 1k test set, with each example provided as a quadruple: one chart image and three parallel code scripts that faithfully render the same visualization. By enforcing visual consistency across languages, these aligned quadruples offer a rich supervisory signal for learning robust cross-language chart representations.

We demonstrate the effectiveness of our approach through extensive experiments on multiple chart-to-code benchmarks, where CharLuMA achieves state-of-the-art results among open-source MLLMs and even surpasses Claude-Haiku-3.5 and GPT-4o-mini on most metrics. Notably, training with more diverse language sets consistently improves chart-to-code performance, as shown in Figure 4, not only by enhancing cross-language generalization but also by strengthening in-language robustness. When training data are imbalanced across languages, however, the model develops a clear bias toward dominant languages, which limits its universality. This underscores that the learning signals inherent in cross-language equivalences are essential for chart-to-code generation. Subspace activation analysis further shows that CharLuMA allocates subspace capacity in a hybrid manner, with compact clusters shared across languages and broader regions devoted to language-specific specialization, while stronger models exhibit smoother and more balanced allocations.

In summary, our work establishes multi-language alignment as a powerful new paradigm for chart-to-code generation. CharLuMA, together with the Chart2NCode dataset, represents a significant step toward more ***universal and adaptable chart-to-code models***. By leveraging complementary views of the same chart across languages, we show that it is possible to regularize the visual-to-code mapping and achieve more robust, accurate results than ever before. These contributions open the door to chart-to-code systems that can serve a broader range of users and software ecosystems, moving beyond Python-only solutions toward truly language-flexible chart generation.

2 RELATED WORK

Multimodal Large Language Models. MLLMs employ multimodal projectors to bridge vision encoders with large language models, enabling reasoning across modalities. Models such as BLIP-2 (Li et al., 2022), Flamingo (Alayrac et al., 2022), mPLUG-Owl (Ye et al., 2024), and Qwen-VL (Bai et al., 2023) adopt Q-Formers or resamplers to compress visual tokens for efficient alignment on large-scale image–text corpora. LLaVA (Liu et al., 2023; 2024) extends the instruction-tuning paradigm to the visual domain, demonstrating that a simple MLP projector with one-to-one mapping can effectively align modalities without discarding visual information. Some works (Tong et al., 2024; Lin et al., 2023) explore the combination of various vision encoder to enhance visual representations. More recent work has scaled MLLMs by substituting dense MLP projectors with sparsely gated mixture-of-experts architectures (Xu et al., 2025; Li et al., 2025), which parallelize multiple MLP blocks but incur significant parameter overhead.

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. Shi et al. (2025) introduced a benchmark of manually curated matplotlib scripts, while Zhao et al. (2025) released a large-scale training corpus. Yang et al. (2024) and Goswami et al. (2025) incorporate user instructions and agent-based methods to enhenace the faithful code synthesis. Other studies untilize chart-to-Python generation for aligning multimodal projectors (Xu et al., 2025) or constructing chart question answering datasets (Zhang et al., 2024b; He et al., 2025). Beyond chart, Belouadi et al. (2024a) and Belouadi et al. (2024b) have developed datasets for image-to-LaTeX generation towards vector graphics. Nevertheless, these efforts remain restricted to single-language settings, which limits practical applicability and overlooks the learning signals in cross-language equivalences.

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 through an automatic annotation pipeline consisting of metadata extraction, template instantiation, and post-debugging (Figure 2). We start by collecting single-language plotting scripts as the source data. ChartCoder (Zhao et al., 2025) provides large-scale Python plotting scripts, while DaTikZ (Belouadi et al., 2024a) contributes extensive TikZ-based codes of scientific vector graphics in LaTeX, from which we extract only the subset corresponding to charts. We further complement these resources by curating 40k R plotting scripts from online platforms and chart galleries (see Appendix B.1).

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 in each language (e.g., `matplotlib.axes` in Python), while LaTeX scripts are processed via regular-expression parsing. Collectively, these layers yield a comprehensive and lossless description of each chart, enabling faithful reconstruction across multiple languages.

Template Instantiation. We synthesize multi-language plotting scripts by identifying and filling language-specific templates based on object-level patterns in the metadata. 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 33 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

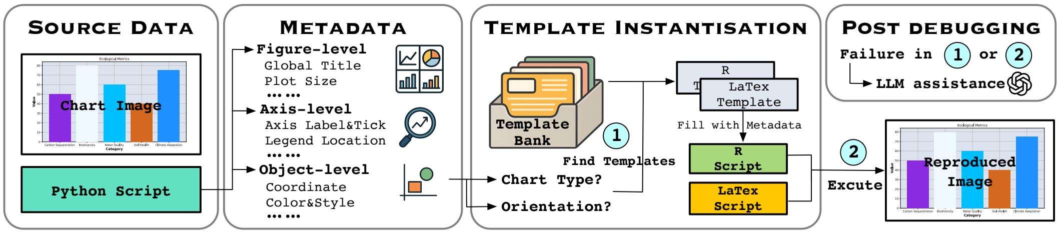


Figure 2: Overview of the automatic annotation pipeline of Chart2NCode.

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.

3.2 Human Quality Checking

We conduct human evaluation to assess the cross-language fidelity of Chart2NCode. A random sample of 1,000 quadruples is independently evaluated by three annotators across four dimensions—structural fidelity, data integrity, semantic consistency, and stylistic coherence—with each dimension rated on a 1–5 scale. The proportion of examples with an average score of at least 4 reaches 97.6% for structural fidelity, 91.6% for data integrity, 95.7% for semantic consistency, and 95.6% for stylistic coherence (see Appendix B.3). These results highlight the robust cross-language consistency of Chart2NCode and its reliability for advancing chart-to-code generation research.

3.3 Data Statistics

Chart2NCode encompasses a total of 176k Chart–Python–R–LaTeX quadruples through our automatic pipeline, with 14.7% are refined via LLM-assisted debugging. The dataset spans 15 standard chart types, including bar (18.8%), line (17.1%), scatter (13.2%), radar (5.73%), histogram (4.59%), and box (4.43%). We constructed a test set of 1,000 randomly sampled examples that achieved average scores of at least 4 across all quality aspects in Section 3.2. The average code lengths are 3,998.5 and 4,229.3 characters for the training and test sets, respectively. Comprehensive statistics and details regarding the annotation pipeline are provided in Appendix B.2 and Appendix B.4.

4 The ChartLuMA 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 through 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. 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, as illustrated in Figure 3.

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. $\mathbf{Z}_v = g^v(\mathbf{X}_v)$.

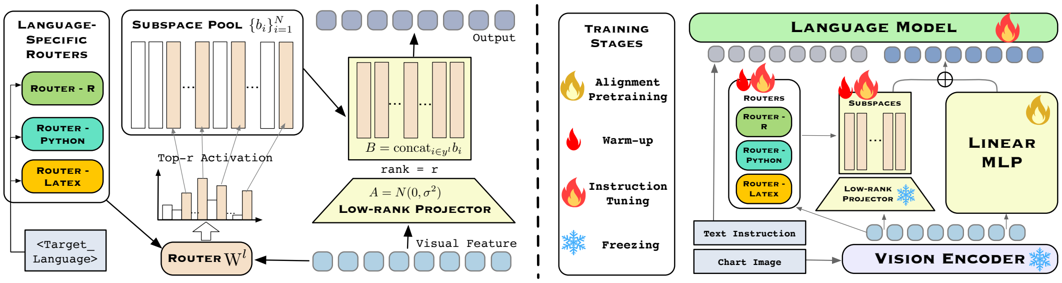


Figure 3: Overview of the CharLuMA architecture and training strategy. The adapter leverages a language-guided routing policy combined with a mixture of low-rank subspaces. The training proceeds through three stages: alignment pretraining, warm-up, and instruction tuning.

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.

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; Chen et al., 2023). The adapter comprises three components: a low-rank projector $\mathbf{A}$, a language-specific router $\mathbf{W}^l$, and a subspace pool $\{b_i\}_{i=1}^N$. 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 $l \in \{\text{Python}, \text{R}, \text{LaTeX}\}$, 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 covering 900k Chart–JSON pairs that capture structural elements such as tables, annotations, and styles. 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 fine-tune the model on Chart2NCode. We first warm up the language-specific routers $\mathbf{W}^l$ ($l \in \{\text{Python}, \text{R}, \text{LaTeX}\}$) 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 demonstrate the effectiveness of CharLuMA through comprehensive experiments, which achieve consistent improvements in multi-language chart-to-code generation across diverse benchmarks and surpass competitive baselines.

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), while keeping the vision encoder and LLM frozen, 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, while keeping the MLP block and vision encoder frozen. 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 $r = 16$, with detailed analysis provided in Section 6.1. All experiments are conducted with a global batch size of 128 on 8× NVIDIA L40S GPUs. The total training cost is approximately 82 GPU hours for CharLuMA-1.3B and 321 GPU hours for CharLuMA-6.7B. More details are provided in Appendix C.1.

5.2 Evaluation Settings

Datasets. We evaluate CharLuMA and the baselines on three chart-to-code datasets. The Chart2NCode test set provides 1,000 charts paired with plotting scripts in Python, R, and LaTeX, enabling multi-language evaluation. ChartMimic (Shi et al., 2025) includes 2,400 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 plot types.

Evaluation Metrics. We assess chart-to-code generation performance from three perspectives: executability, code similarity, and image fidelity. Execution Rate (ER) measures the proportion of generated scripts that run successfully. CrystalBLEU (CB) (Eghbali & Pradel, 2022), a BLEU variant tailored for code, assesses code-level similarity. For image-level fidelity, we adopt DreamSim (DS) (Fu et al., 2023), a fine-tuned metric for perceptual similarity. For Python scripts, we report the averaged F1 score across text, layout, type, and color attributes (Shi et al., 2025), where unexecutable scripts are assigned with zero values. To avoid code similarity inflation for models trained on in-distribution data, we employ image-side GPT-4o scoring (GS) (Shi et al., 2025) on the Chart2NCode test set, where unexecutable scripts are assigned with zero values as well.

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., 2025a), GLM-4.5v (Team et al., 2025), DeepSeek-VL (Lu et al., 2024), Phi-3.5-Vision (Abdin et al., 2024), and LLaVA-1.5 (Liu et al., 2023).

Chart MLLMs. We also compare against chart-specialized MLLMs tailored for chart reasoning and chart-to-code generation. ChartLlama (Han et al., 2023) extends the LLaVA-v1.5 framework with instruction tuning on multiple chart reasoning tasks. TinyChart (Zhang et al., 2024a) is built on TinyLLaVA (Zhou et al., 2024) for efficient chart understanding. ChartMoE (Xu et al., 2025) advances chart understanding through a mixture-of-experts multimodal projector, integrating chart-to-code generation as a core modality alignment task. 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, the state of the art among open-source

Table 1: Performance on ChartMimic, Plot2Code, and Chart2NCode test set. ER ⇑ denotes execution rate, CB ⇑ denotes the code-similarity score CrystalBLEU, DS ⇑ denotes the image-similarity score DreamSim, F1 ⇑ denotes the heuristic F1 score for Python scripts, and GS ⇑ denotes the image-similarity GPT-4o scoring. A "-" indicates that no executable script is generated.

Models	**ChartMimic**				**Plot2Code**				**Chart2NCode**									
	Chart2Python				Chart2Python				Chart2Python				Chart2R			Chart2LaTeX		
	ER	CB	DS	F1	ER	CB	DS	F1	ER	GS	DS	F1	ER	GS	DS	ER	GS	DS
	Propriety Multimodal Large Language Models																	
GPT-5-mini	86.8	13.6	86.9	71.5	93.2	8.9	85.9	72.8	85.2	80.0	89.0	67.5	90.3	81.2	82.5	49.7	41.1	75.2
GPT-4o-mini	89.0	9.0	77.5	70.2	90.2	20.7	77.8	67.0	94.8	79.8	81.2	74.5	89.5	70.3	75.4	94.7	70.4	61.2
GPT-4o	93.2	10.2	83.5	79.0	92.4	24.2	83.6	75.4	98.5	87.4	85.0	80.9	94.5	78.3	78.8	88.4	69.8	72.4
Claude-Haiku-3.5	88.0	7.5	76.2	65.7	87.1	16.2	72.8	56.8	91.3	76.7	81.6	68.8	93.0	73.9	76.2	78.2	55.3	57.3
Claude-Sonnet-4	96.2	13.7	83.3	79.5	95.5	12.9	81.2	76.8	98.3	88.0	86.8	81.4	93.9	83.1	82.0	92.7	72.2	76.0
	Open-source Multimodal Large Language Models																	
Qwen3-VL-2B	59,0	6.2	68.9	40.4	68.9	13.0	64.2	50.1	74.0	59.6	78.0	61.0	56.5	42.0	52.4	56.0	37.4	60.8
Qwen3-VL-4B	78.8	7.6	71.9	59.7	77.3	12.9	66.4	55.4	87.6	77.2	83.2	76.1	75.4	60.9	66.4	62.4	45.2	68.6
Qwen3-VL-8B	81.8	7.9	72.5	64.0	78.8	14.2	68.1	56.9	91.1	80.8	83.7	80.6	73.6	57.2	72.7	77.3	57.1	66.8
InternVL3.5-2B	51.2	4.4	67.0	32.3	61.4	12.2	55.7	44.2	69.8	53.2	76.1	53.1	61.8	44.9	53.4	9.6	4.7	52.6
InternVL3.5-4B	66.6	7.7	70.1	46.0	62.1	13.3	58.8	42.7	77.9	63.4	78.4	63.0	66.8	51.5	56.4	25.7	14.7	55.5
InternVL3.5-8B	74.0	8.1	70.9	51.7	74.2	13.9	61.0	49.1	82.5	67.5	79.6	67.0	67.0	48.2	67.6	81.1	53.3	57.1
DeepSeek-VL-7B	41.3	4.7	67.8	19.0	64.4	13.3	59.4	47.0	65.9	52.5	74.2	44.6	58.8	40.6	57.0	17.5	12.3	49.8
Phi-3.5-vision-4B	66.7	6.9	44.1	38.6	72.7	14.9	63.8	42.6	68.8	56.1	53.3	34.2	47.0	33.5	52.5	7.9	5.1	42.9
LLaVA-v1.5-7B	33.0	0.7	49.6	6.7	34.9	7.1	52.1	10.4	32.9	40.2	51.9	8.9	41.4	31.0	50.7	19.4	11.7	41.0
GLM-4.5v-108B	88.4	8.7	73.3	67.6	83.3	13.3	80.8	56.2	85.0	79.5	85.6	77.3	85.3	70.3	77.2	80.8	63.8	62.6
ChartLlama-13B	70.8	0.0	45.0	15.9	81.8	4.1	50.1	22.4	65.3	14.8	46.0	16.2	13.0	6.2	44.8	81.7	49.2	32.5
TinyChart-3B	84.1	8.1	60.8	53.9	81.1	12.1	64.0	54.0	92.1	86.3	46.5	55.2	-	-	-	-	-	-
ChartMoE-8B	55.0	1.3	56.9	25.7	70.5	6.7	58.9	26.9	69.5	40.2	64.2	35.4	39.3	25.5	52.9	17.1	11.1	27.9
ChartCoder-7B	88.9	**8.8**	61.3	59.3	87.9	13.9	65.7	56.6	96.2	86.4	48.1	56.1	-	-	-	17.9	10.6	39.1
CharLuMA-1.3B	84.8	7.3	75.1	57.5	83.3	14.5	64.3	47.2	94.4	78.4	86.5	76.9	94.5	73.3	78.9	84.5	65.1	71.3
CharLuMA-6.7B	**91.8**	8.6	**79.2**	**70.3**	**96.2**	**15.8**	**68.3**	**60.5**	**98.0**	**88.1**	**88.7**	**83.5**	**96.5**	**80.9**	**81.8**	**89.0**	**74.2**	**72.5**

systems for chart-to-Python generation, achieves 86.4 GS and 48.1 DS on the Python subset of Chart2NCode, while its performance deteriorates significantly on other languages, with the execution rate dropping to 17.9 on the LaTeX subset and failing to generate valid R scripts. General-purpose open-source models such as DeepSeek-VL-7B and Phi-3.5-Vision show larger imbalances on Chart2NCode, achieving execution rates above 65 on Python but falling below 20 on LaTeX. DeepSeek-VL-7B further exhibits sharp degradation in chart quality, with DreamSim dropping from 74.2 in Python to 57.0 in R and 54.2 in LaTeX. Proprietary models display the same tendency in a more moderate form, as GPT-5-mini and Claude-Haiku-3.5 achieve execution rates above 85 and heuristic F1 scores above 65 on Python, while their performance declines when extended to LaTeX.

CharLuMA effectively addresses the cross-language disparity and establishes itself as the most capable open-source MLLM for general chart-to-code generation. CharLuMA-6.7B delivers the strongest results on well-established chart-to-Python benchmarks among open-source models, achieving 79.2 DS and 70.3 F1 on ChartMimic, and 68.3 DS and 60.5 F1 on Plot2Code. The smaller CharLuMA-1.3B also performs competitively, with 75.1 DS and 57.5 F1 on ChartMimic, and 64.3 DS and 47.2 F1 on Plot2Code, indicating its parameter efficiency. On the multi-language Chart2NCode test set, both models sustain robust and balanced performance across Python, R, and LaTeX. CharLuMA-6.7B achieves 88.7 DS and 83.5 F1 on Python, 81.8 DS and 80.9 GS on R, and 72.5 DS and 74.2 GS on LaTeX, demonstrating consistent generalization beyond Python. Notably, CharLuMA-6.7B outperforms Claude-Haiku-3.5 on most metrics across all benchmarks and delivers performance comparable to GPT-4o-mini on ChartMimic and Chart2NCode. These results underscore CharLuMA's ability to advance open-source chart-to-code generation beyond single-language dominance, narrowing the gap with proprietary systems.

6 Further Study

We conduct ablation studies and in-depth analyses to disentangle the contributions of different components in CharLuMA, demonstrating its robustness and interpretability.

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.

Model Size	Projector Architecture	**Chart2NCode**		
		ER	CB	DS
1.3B	Linear MLP	88.1	14.8	76.9
	Mixture-of-MLP	87.9	13.8	75.1
	Subspace Adapter	**91.1**	**23.2**	**78.9**
6.7B	Linear MLP	91.0	20.3	78.2
	Mixture-of-MLP	91.9	19.3	77.4
	Subspace Adapter	**94.5**	**24.5**	**81.0**

Table 3: Ablation study of subspace settings, router configurations, and training choices in CharLuMA-1.3B on the Chart2NCode test set, with results averaged over all three languages.

Total Subspace	Activated Subspace	Total Router	**Chart2NCode**		
			ER	CB	DS
16	8	3	88.9	21.4	77.6
32	8	3	89.4	22.1	77.8
64	32	3	87.8	19.6	75.6
32	16	1	86.1	17.1	75.1
32	32	0	85.8	16.6	73.2
32	16	3	**91.1**	**23.2**	**78.9**
w/o warming up before finetuning			87.1	18.8	75.6
w/o freezing A matrix of adapter			90.2	21.9	78.0

6.1 Model Architecture Ablation

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

Alternative Architecture. We compare our low-rank *subspace adapter* with two alternative projector designs in Table 2. The *linear MLP* block serves as a standard baseline (Belouadi et al., 2024b;a; Zhao et al., 2025) but yields modest improvements, with the 1.3B model staying 88.1 ER and 14.8 CB. 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 always activates the language-specific and shared experts (see Appendix C.2). This raises the execution rate to 91.9 but leads to reduced code and image similarity on the 6.7B model. In contrast, our low-rank *subspace adapter* achieves the strongest results across both model sizes, combining language-aware specialization with parameter efficiency.

Effect of Subspace Number. We compare CharLuMA-1.3B under different total and activated subspace settings. In Table 3, rows 1–3 demonstrate that moderate scaling from 16 to 32 subspaces improves diversity and performance, while further expansion to 64 leads to degradation in code accuracy and visual similarity. These results suggest that the 32–16 configuration provides the best balance between expressiveness and efficiency for subspace specialization.

Effect of Routing Policy. We compare different routing strategies for activating subspaces in CharLuMA-1.3B. In Table 3, rows 4–5 show that replacing the three language-specific routers with a single shared router reduces CrystalBLEU from 23.2 to 17.1, while removing routers altogether lowers it further to 16.6. These results confirm the importance of language-guided routing policy for maintaining code fidelity and cross-language alignment.

Effect of Training Choices. In Table 3, row 7 shows that removing the warming-up stage lowers CrystalBLEU from 23.2 to 18.8 and DreamSim from 78.9 to 75.6, underscoring its role in stabilizing subspace and router specialization. Row 8 shows that unfreezing the A matrix reduces CrystalBLEU to 21.9 and DreamSim to 78.0, indicating that freezing A helps maintain a compact low-rank representation while supporting effective language-specific specialization.

Effect of Backbone Choices. We examine the effect of backbone choices in CharLuMA by modifying the language model and vision encoder separately. First, we replace DeepSeek-Coder-6.7B with the general-purpose DeepSeek-LLM-7B while keeping the vision encoder fixed. Second, we replace SigLIP with CLIP-Large with an input resolution of 336 × 336 while retaining the original language model. As reported in Table 4, the default configuration with DeepSeek-Coder-6.7B

Table 4: Ablation study of backbone choices in CharLuMA on the Chart2NCode test set, with results averaged over all three languages.

Language Model	Vision Encoder	**Chart2NCode**		
		ER	CB	DS
DeepSeek-LLM-7B	SigLIP	88.6	21.8	77.1
DeepSeek-Coder-6.7B	CLIP	88.8	22.0	79.2
DeepSeek-Coder-6.7B	SigLIP	**94.5**	**24.5**	**81.0**

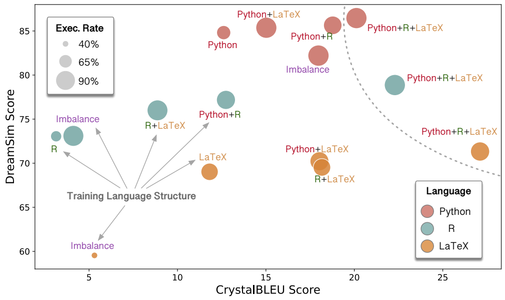


Figure 4: Ablation study of language structure using CharLuMA-1.3B on Chart2NCode.

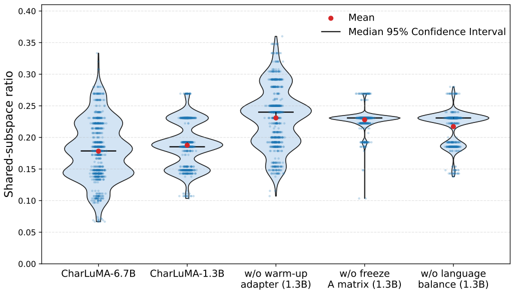


Figure 5: Distribution of shared-subspace ratios across CharLuMA and ablated models.

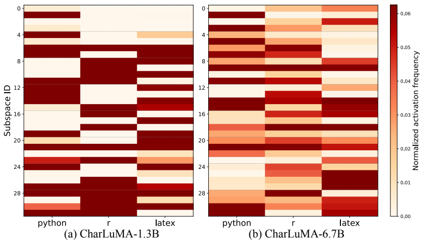
and SigLIP achieves the strongest execution rate, CrystalBLEU, and DreamSim scores, whereas each alternative substitution leads to a consistent drop in performance across all metrics.

6.2 Language Structure Ablation

We study the effect of language diversity and balance on the general chart-to-code generation capability of MLLMs by varying the number of programming languages during training, with the number of routers matched to the number of languages. All language configurations are trained with the same strategy and number of steps as CharLuMA-1.3B (see Section 5.1 and Appendix C.2) and evaluated on the Chart2NCode test set restricted to the languages included in training. As shown in Figure 4, greater language diversity leads to substantial improvements. Models trained on three languages achieve the highest execution rates and the strongest code- and image-level similarity scores across all target languages, whereas two-language and single-language settings fall behind by large margins. Moreover, training on the diverse but imbalanced source distribution (76.6% Python, 19.2% R and 4.2% LaTeX) further skews the model toward the dominant language and degrades its performance on the other languages. These results demonstrate that *language diversity enhances both cross-language generalization and in-language robustness by leveraging the learning signals inherent in cross-language equivalences.* At the same time, balanced supervision is critical, as imbalances in the training data introduce systematic biases that undermine universality. Together, these findings underscore the importance of Chart2NCode as the first balanced multi-language dataset for enabling robust and equitable chart-to-code generation.

6.3 Subspace Activation Analysis

We visualize the subspace activation patterns of language-specific routers in CharLuMA-1.3B and CharLuMA-6.7B in Figure 6. The heatmaps display the normalized activation frequency of 32 subspaces for each language and reveal a hybrid allocation of the subspace pool, with compact shared clusters alongside broader language-specific zones. In CharLuMA-1.3B, subspaces 21, 23, and 30 are frequently activated across all languages, while subspace 1 is used primarily for Python, 18 for R, and 17 for LaTeX. In contrast, CharLuMA-6.7B shows a more balanced distribution, with most subspaces—such as 8, 20, and 29—exhibiting intermediate activation frequencies across the three languages, indicating smoother cross-language integration.



(a) CharLuMA-1.3B
(b) CharLuMA-6.7B

Figure 6: Heatmap of subspace activation frequency for CharLuMA.

We compute the *shared-subspace ratio* to quantify the cross-language allocation of experts. For each chart, it is defined as the proportion of experts simultaneously activated by all language-specific routers relative to the total set of experts activated (see Appendix C.2). Figure 5 reports the distribution of this ratio over a random 1k sample from Chart2NCode. CharLuMA-1.3B achieves a median ratio of 0.19, corresponding to roughly 5 experts shared within a total activation pool of about 27, indicating a compact shared core complemented by broad language-private allocation. 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, suggesting that scaling preserves and slightly reinforces this balanced allocation. In contrast, the ablated 1.3B variants exhibit inflated ratios (0.23–0.24), where more experts are pulled into shared use while the overall activated pool shrinks, indicating weakened specialization.

6.4 Qualitative Analysis

We conduct a qualitative analysis that combines error diagnosis of CharLuMA-6.7B and comparisons with GPT-4o and ChartCoder across multiple benchmarks. For error analysis, we find that execution failures often stem from mismatched data dimensions or undefined variables in Python and R, and from syntax issues such as missing braces in LaTeX, while successful runs may still diverge due to missing annotations, misclassified chart subtypes, or stylistic inconsistencies. For model comparison, case studies from Chart2NCode and ChartMimic demonstrate that CharLuMA consistently produces faithful outputs across Python, R, and LaTeX, whereas GPT-4o shows reduced reliability in R and LaTeX, and ChartCoder frequently fails to produce valid code in these two languages. More details are provided in Appendix C.4 and Appendix C.5.

7 Conclusion

We introduced CharLuMA, a multimodal LLM for chart-to-code generation with a language-guided mixture of low-rank subspaces in its multimodal projector, and Chart2NCode, a dataset of 176k visually aligned Chart–Python–R–LaTeX quadruples. CharLuMA achieves state-of-the-art performance among open-source MLLMs, with ablation studies showing that balanced multi-language training enhances cross-language generalization and mitigates bias toward dominant languages. Subspace analyses further reveal a hybrid allocation of shared and language-specific regions that supports both transfer and fidelity. By leveraging parallel code views of the same chart across languages, we show that cross-language alignment provides a powerful supervisory signal for robust and accurate code generation. These contributions pave the way toward universal, language-flexible chart-to-code systems that better reflect the diverse software ecosystems in practice.

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, Weizhu Chen, Yen-Chun Chen, Yi-Ling Chen, Hao Cheng, Parul Chopra, Xiyang Dai, Matthew Dixon, Ronen Eldan, Victor Fragoso, Jianfeng Gao, Mei Gao, Min Gao, Amit Garg, Allie Del Giorno, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Wenxiang Hu, Jamie Huynh, Dan Iter, Sam Ade Jacobs, Mojan Javaheripi, Xin Jin, Nikos Karampatziakis, Piero Kauffmann, Mahoud Khademi, Dongwoo Kim, Young Jin Kim, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Yunsheng Li, Chen Liang, Lars Liden, Xihui Lin, Zeqi Lin, Ce Liu, Liyuan Liu, Mengchen Liu, Weishung Liu, Xiaodong Liu, Chong Luo, Piyush Madan, Ali Mahmoudzadeh, David Majercak, Matt Mazzola, Caio César Teodoro Mendes, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Liliang Ren, Gustavo de Rosa, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Yelong Shen, Swadheen Shukla, Xia Song, Masahiro Tanaka, Andrea Tupini, Praneetha Vaddamanu, Chunyu Wang, Guanhua Wang, Lijuan Wang, Shuohang Wang, Xin Wang, Yu Wang, Rachel Ward, Wen Wen, Philipp Witte, Haiping Wu, Xiaoxia Wu, Michael Wyatt, Bin Xiao, Can Xu, Jiahang Xu, Weijian Xu, Jilong Xue, Sonali Yadav, Fan Yang, Jianwei Yang, Yifan Yang, Ziyi Yang, Donghan Yu, Lu Yuan, Chenruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. Phi-3 technical report: A highly capable language model locally on your phone, 2024. URL `https://arxiv.org/abs/2404.14219`.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza

Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob L Menick, Sebastian Borgeaud, Andy Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikoł aj Bińkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karén Simonyan. Flamingo: a visual language model for few-shot learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 23716–23736. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/960a172bc7fbf0177ccccbb411a7d800-Paper-Conference.pdf.

Anthropic. Claude 3.5 sonnet, 2024. URL https://www.anthropic.com/news/claude-3-5-sonnet.

Anthropic. Introducing claude 4, 2025. URL https://www.anthropic.com/news/claude-4.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A frontier large vision-language model with versatile abilities. *ArXiv*, abs/2308.12966, 2023. URL https://api.semanticscholar.org/CorpusID:263875678.

Jonas Belouadi, Anne Lauscher, and Steffen Eger. AutomaTikZ: Text-guided synthesis of scientific vector graphics with TikZ. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=v3K5TVP8kZ.

Jonas Belouadi, Simone Paolo Ponzetto, and Steffen Eger. Detikzify: Synthesizing graphics programs for scientific figures and sketches with tikz. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b. URL https://openreview.net/forum?id=bcVLFQCOjc.

Tianlong Chen, Xuxi Chen, Xianzhi Du, Abdullah Rashwan, Fan Yang, Huizhong Chen, Zhangyang Wang, and Yeqing Li. Adamv-moe: Adaptive multi-task vision mixture-of-experts. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17300–17311, 2023. doi: 10.1109/ICCV51070.2023.01591.

Chenhao Ding, Jiangyang Li, SongLin Dong, Xinyuan Gao, Yuhang He, and Yihong Gong. SuLoRA: Subspace low-rank adaptation for parameter-efficient fine-tuning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 5334–5349, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.278. URL https://aclanthology.org/2025.findings-acl.278/.

Aryaz Eghbali and Michael Pradel. Crystalbleu: precisely and efficiently measuring the similarity of code. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, ICSE '22, pp. 341–342, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392235. doi: 10.1145/3510454.3528648. URL https://doi.org/10.1145/3510454.3528648.

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

Kanika Goswami, Puneet Mathur, Ryan Rossi, and Franck Dernoncourt. Plotgen: Multi-agent llm-based scientific data visualization via multimodal retrieval feedback. In *Companion Proceedings of the ACM on Web Conference 2025*, WWW '25, pp. 1672–1676, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713316. doi: 10.1145/3701716.3716888. URL https://doi.org/10.1145/3701716.3716888.

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. Deepseek-coder: When the large language model meets programming – the rise of code intelligence, 2024. URL https://arxiv.org/abs/2401.14196.

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

Wei He, Zhiheng Xi, Wanxu Zhao, Xiaoran Fan, Yiwen Ding, Zifei Shan, Tao Gui, Qi Zhang, and Xuanjing Huang. Distill visual chart reasoning ability from llms to mllms, 2025. URL https://arxiv.org/abs/2410.18798.

Jiachen Li, Xinyao Wang, Sijie Zhu, Chia-Wen Kuo, Lu Xu, Fan Chen, Jitesh Jain, Humphrey Shi, and Longyin Wen. 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, 2025. Curran Associates Inc. ISBN 9798331314385.

Junnan Li, Dongxu Li, Caiming Xiong, and Steven C. H. Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International Conference on Machine Learning*, 2022. URL https://api.semanticscholar.org/CorpusID:246411402.

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. Sphinx: The joint mixing of weights, tasks, and visual embeddings for multi-modal large language models, 2023. URL https://arxiv.org/abs/2311.07575.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=w0H2xGHlkw.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 26286–26296, 2024. doi: 10.1109/CVPR52733.2024.02484.

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. Deepseek-vl: Towards real-world vision-language understanding, 2024.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chun yue Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *International Conference on Learning Representations*, 2023. URL https://api.semanticscholar.org/CorpusID:264491155.

OpenAI. Gpt-4o mini: advancing cost-efficient intelligence, 2024a. URL https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence.

OpenAI. Hello gpt-4o, 2024b. URL https://openai.com/index/hello-gpt-4o.

OpenAI. Gpt-5 is here, 2025. URL https://openai.com/gpt-5/.

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. Chartmimic: Evaluating LMM's cross-modal reasoning capability via chart-to-code generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=sGpCzsfd1K.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

V Team, Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, Shuaiqi Duan, Weihan Wang, Yan Wang, Yean Cheng, Zehai He, Zhe Su, Zhen Yang, Ziyang Pan, Aohan Zeng, Baoxu Wang, Bin Chen, Boyan Shi, Changyu Pang, Chenhui Zhang, Da Yin, Fan Yang, Guoqing Chen, Jiazheng Xu, Jiale Zhu, Jiali Chen, Jing Chen, Jinhao Chen, Jinghao Lin, Jinjiang Wang, Junjie Chen, Leqi Lei, Letian Gong, Leyi Pan, Mingdao Liu, Mingde Xu, Mingzhi Zhang, Qinkai Zheng, Sheng Yang, Shi Zhong, Shiyu Huang, Shuyuan Zhao, Siyan Xue, Shangqin Tu, Shengbiao Meng, Tianshu Zhang, Tianwei Luo, Tianxiang Hao, Tianyu Tong, Wenkai Li, Wei Jia, Xiao Liu, Xiaohan Zhang, Xin Lyu, Xinyue Fan, Xuancheng Huang, Yanling Wang, Yadong Xue, Yanfeng Wang, Yanzi Wang, Yifan

An, Yifan Du, Yiming Shi, Yiheng Huang, Yilin Niu, Yuan Wang, Yuanchang Yue, Yuchen Li, Yutao Zhang, Yuting Wang, Yu Wang, Yuxuan Zhang, Zhao Xue, Zhenyu Hou, Zhengxiao Du, Zihan Wang, Peng Zhang, Debing Liu, Bin Xu, Juanzi Li, Minlie Huang, Yuxiao Dong, and Jie Tang. Glm-4.5v and glm-4.1v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning, 2025. URL `https://arxiv.org/abs/2507.01006`.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Chengzhong Xu. 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, 2025. Curran Associates Inc. ISBN 9798331314385.

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. Cambrian-1: A fully open, vision-centric exploration of multimodal llms. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 87310–87356. Curran Associates, Inc., 2024. URL `https://proceedings.neurips.cc/paper_files/paper/2024/file/9ee3a664ccfeabc0da16ac6f1f1cfe59-Paper-Conference.pdf`.

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, Guanzhou Chen, Zichen Ding, Changyao Tian, Zhenyu Wu, Jingjing Xie, Zehao Li, Bowen Yang, Yuchen Duan, Xuehui Wang, Zhi Hou, Haoran Hao, Tianyi Zhang, Songze Li, Xiangyu Zhao, Haodong Duan, Nianchen Deng, Bin Fu, Yinan He, Yi Wang, Conghui He, Botian Shi, Junjun He, Yingtong Xiong, Han Lv, Lijun Wu, Wenqi Shao, Kaipeng Zhang, Huipeng Deng, Biqing Qi, Jiaye Ge, Qipeng Guo, Wenwei Zhang, Songyang Zhang, Maosong Cao, Junyao Lin, Kexian Tang, Jianfei Gao, Haian Huang, Yuzhe Gu, Chengqi Lyu, Huanze Tang, Rui Wang, Haijun Lv, Wanli Ouyang, Limin Wang, Min Dou, Xizhou Zhu, Tong Lu, Dahua Lin, Jifeng Dai, Weijie Su, Bowen Zhou, Kai Chen, Yu Qiao, Wenhai Wang, and Gen Luo. Internvl3.5: Advancing open-source multimodal models in versatility, reasoning, and efficiency, 2025a. URL `https://arxiv.org/abs/2508.18265`.

Zirui Wang, Mengzhou Xia, Luxi He, Howard Chen, Yitao Liu, Richard Zhu, Kaiqu Liang, Xindi Wu, Haotian Liu, Sadhika Malladi, Alexis Chevalier, Sanjeev Arora, and Danqi Chen. Charxiv: charting gaps in realistic chart understanding in multimodal llms. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2025b. Curran Associates Inc. ISBN 9798331314385.

Chengyue Wu, Zhixuan Liang, Yixiao Ge, Qiushan Guo, Zeyu Lu, Jiahao Wang, Ying Shan, and Ping Luo. Plot2Code: A comprehensive benchmark for evaluating multi-modal large language models in code generation from scientific plots. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 3006–3028, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.164. URL `https://aclanthology.org/2025.findings-naacl.164/`.

Taiqiang Wu, Jiahao Wang, Zhe Zhao, and Ngai Wong. Mixture-of-subspaces in low-rank adaptation. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7880–7899, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.450. URL `https://aclanthology.org/2024.emnlp-main.450/`.

Zhengzhuo Xu, Bowen Qu, Yiyan Qi, SiNan Du, Chengjin Xu, Chun Yuan, and Jian Guo. Chartmoe: Mixture of diversely aligned expert connector for chart understanding. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=o5TsWTUSeF`.

Pengyu Yan, Mahesh Bhosale, Jay Lal, Bikhyat Adhikari, and David Doermann. 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*, pp. 453–469, Berlin, Heidelberg, 2024. Springer-Verlag. ISBN 978-3-031-70532-8. doi: 10.1007/978-3-031-70533-5_26. URL https://doi.org/10.1007/978-3-031-70533-5_26.

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. MatPlotAgent: Method and evaluation for LLM-based agentic scientific data visualization. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11789–11804, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.701. URL https://aclanthology.org/2024.findings-acl.701/.

Qinghao Ye, Haiyang Xu, Guohai Xu, Jiabo Ye, Ming Yan, Yiyang Zhou, Junyang Wang, Anwen Hu, Pengcheng Shi, Yaya Shi, Chenliang Li, Yuanhong Xu, Hehong Chen, Junfeng Tian, Qi Qian, Ji Zhang, Fei Huang, and Jingren Zhou. mplug-owl: Modularization empowers large language models with multimodality, 2024. URL https://arxiv.org/abs/2304.14178.

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, Huan Sun, Yu Su, and Wenhu Chen. 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)*, pp. 9556–9567, 2024. doi: 10.1109/CVPR52733.2024.00913.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11941–11952, 2023a. doi: 10.1109/ICCV51070.2023.01100.

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

Liang Zhang, Anwen Hu, Haiyang Xu, Ming Yan, Yichen Xu, Qin Jin, Ji Zhang, and Fei Huang. TinyChart: Efficient chart understanding with program-of-thoughts learning and visual token merging. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 1882–1898, Miami, Florida, USA, November 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.112. URL https://aclanthology.org/2024.emnlp-main.112/.

Wenqi Zhang, Zhenglin Cheng, Yuanyu He, Mengna Wang, Yongliang Shen, Zeqi Tan, Guiyang Hou, Mingqian He, Yanna Ma, Weiming Lu, and Yueting Zhuang. Multimodal self-instruct: Synthetic abstract image and visual reasoning instruction using language model. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 19228–19252, Miami, Florida, USA, November 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1072. URL https://aclanthology.org/2024.emnlp-main.1072/.

Zhihan Zhang, Yixin Cao, and Lizi Liao. XFinBench: Benchmarking LLMs in complex financial problem solving and reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 8715–8758, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.457. URL https://aclanthology.org/2025.findings-acl.457/.

Xuanle Zhao, Xianzhen Luo, Qi Shi, Chi Chen, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ChartCoder: Advancing multimodal large language model for chart-to-code generation. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7333–7348, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.363. URL https://aclanthology.org/2025.acl-long.363/.

Baichuan Zhou, Ying Hu, Xi Weng, Junlong Jia, Jie Luo, Xien Liu, Ji Wu, and Lei Huang. Tinyllava: A framework of small-scale large multimodal models, 2024. URL `https://arxiv.org/abs/2402.14289`.

A LLM Usage

Large Language Models (LLMs) were utilized in this work solely as auxiliary tools for linguistic refinement. Their function was restricted to enhancing grammar, clarity, and stylistic consistency of text that had been originally drafted by the authors. At no stage did LLMs contribute to research ideation, methodological design, data collection, analysis, or interpretation of results. All intellectual contributions, scientific content, and conclusions presented in this paper are entirely attributable to the authors. The authors accept full responsibility for the accuracy, originality, and integrity of the submission, including sections of text that may have been refined with the assistance of LLMs.

B Dataset

B.1 Data Acquisition

We collect single-language plotting scripts from established datasets and publicly available repositories 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-Python scripts, of which 8.8k correspond to charts with explicit axis structures. In addition, we curated 40k R plotting scripts from widely used online resources including R gallery [2] and stack overflow [3]. To handle deprecated or non-executable scripts encountered during crawling, we employed GPT-4o as an automated debugging assistant, guided by the prompt instructions in Figure 8, with a total API cost of 132.2 USD.

B.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. 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

[2] https://r-graph-gallery.com/all-graphs.html

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

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 7: Instruction Prompt for Handling Missing Templates in Post-Debugging

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 fontface 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 chart variants without manual intervention.

LLM-assisted Debugging. We incorporate an LLM-assisted debugging module based on GPT-4o to handle cases where no suitable template can be identified or when an instantiated template fails to execute. Instruction prompts for these two scenarios are provided in Figure 7 and Table 8. The total expenditure on the OpenAI API amounts to 316.6 USD.

Our automatic pipeline finally generates 176K Chart-Python-R-LaTex quadruples, with 14.7% are refined via LLM-assisted debugging. A randomly sampled set of 1,000 examples is reserved as the test set. The average code lengths are 3998.5 and 4229.3 characters for the training and test sets, respectively. The dataset covers a broad range of chart types, including bar (18.8%), line (17.1%), scatter (13.2%), pie (7.3%), ring (5.1%), radar (5.73%), histogram (4.59%), box (4.43%), heatmap (3.56%), violin (3.13%), error point (2.94%), area (2.81%), density (2.79 %), error bar (2.68%), bubble (2.2%), and others (3.64 %).

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 8: Instruction Prompt for Failed Template Execution in Post-Debugging

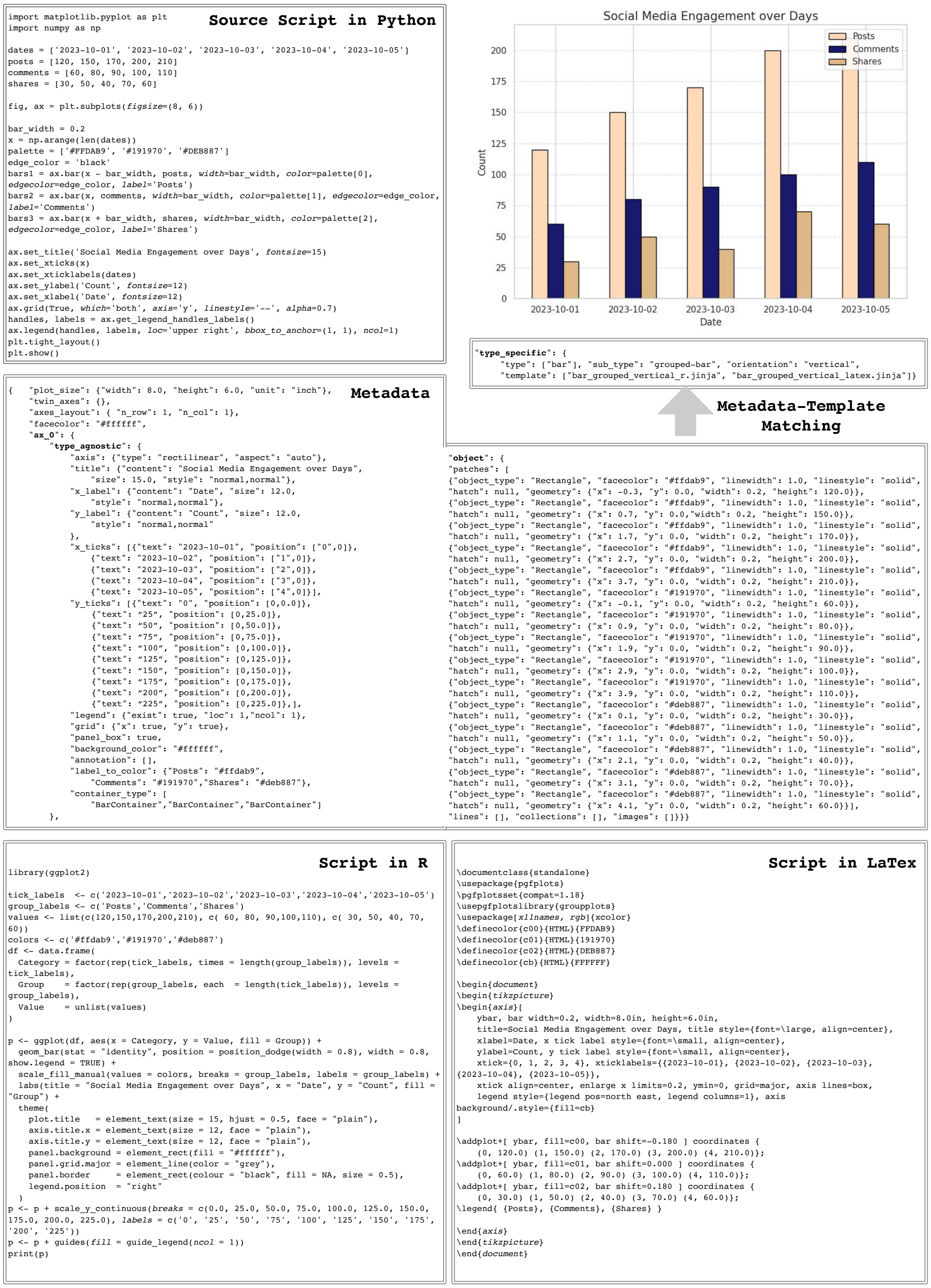
B.3 Quality Assurance

We conduct a human evaluation to systematically assess the cross-language fidelity of Chart2NCode. We randomly sample 1,000 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 pairwise 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 as well as 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, gridline visibility, and panel borders—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.

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: 97.6% of examples exceed the threshold for structural fidelity, 91.6% for data integrity, 95.7% for semantic consistency, and 95.6% for stylistic coherence. To further assess reliability, we compute Fleiss' κ on binarized labels (rating ≥ 4 vs. < 4). The resulting average Fleiss' κ of 0.83 indicates substantial agreement beyond chance, representing a strong and practical level of consistency for human judgment in chart reproduction tasks.

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.

Dimension	**Ann. 1**	**Ann. 2**	**Ann. 3**	**Avg.**
Structural fidelity	98.3	97.1	97.5	97.6
Data integrity	90.5	91.5	92.8	91.6
Semantic consistency	94.9	96.6	95.7	95.7
Stylistic coherence	96.2	95.0	95.5	95.6
Overall	95.0	94.3	94.6	94.6



```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()
```

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

```
{   "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"]
        },
```

```
"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": []}}}
```

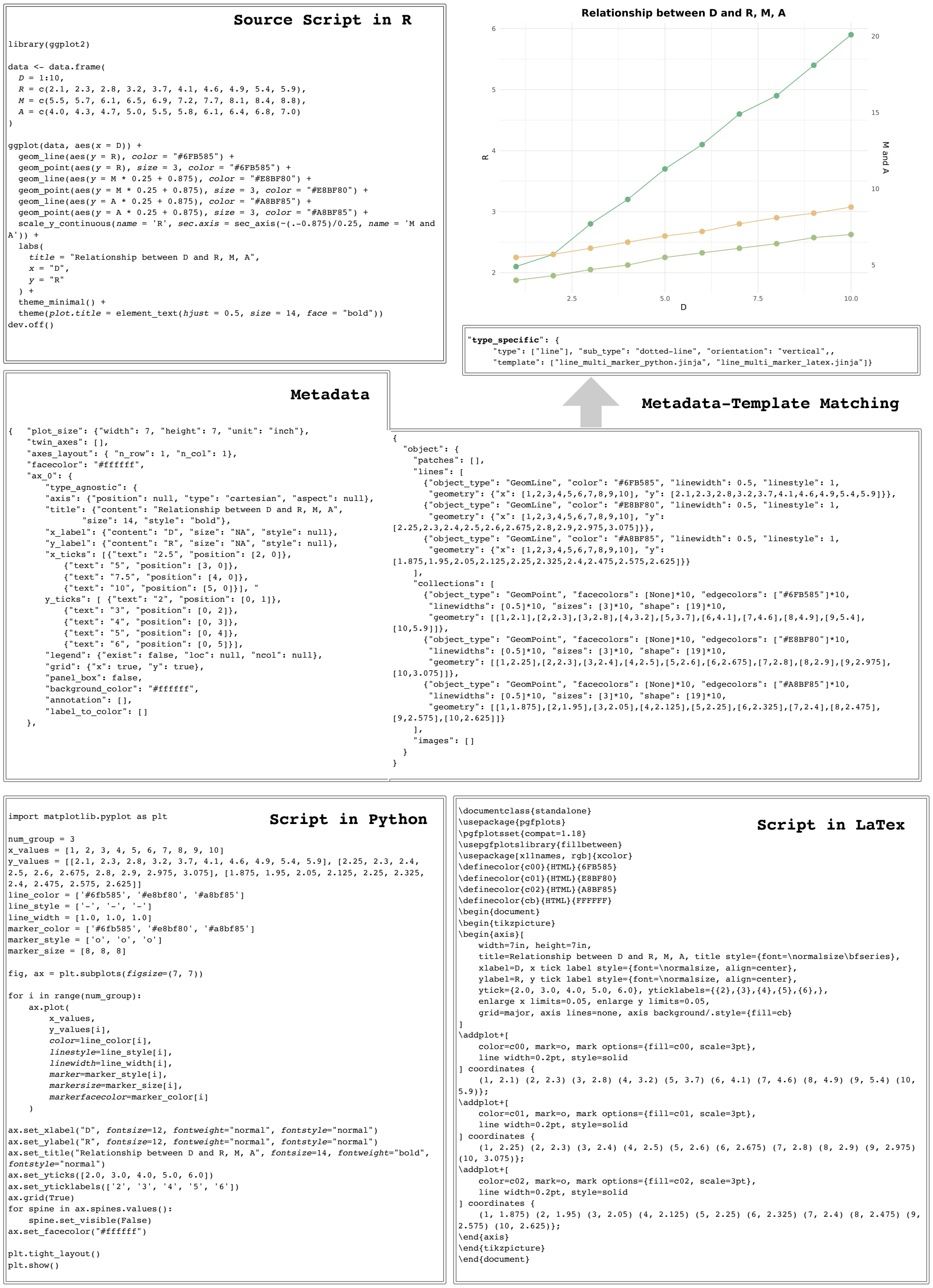
```r
library(ggplot2)

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)
```

```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 9: Case study of annotation pipeline in a vertical grouped bar chart.

B.4 Case Study

We present two illustrative cases in Figure 9 and Figure 10 to demonstrate the functionality of our annotation pipeline.



```
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"]}
```

```
{   "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": []
    },
```

```
{
  "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": []
  }
}
```

```
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()
```

```
\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 10: Case study of annotation pipeline in a dotted line chart.

C Experimental Settings and Results

C.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 connector 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 within the adapter 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 total 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 6.

Table 6: Training hyperparameters for CharLuMA across stages in Section 5.1.

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

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 11, 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.

C.2 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 conduct a language structure ablation to examine the impact of varying the number of plotting languages and corresponding routers during training, while strictly controlling the total number of training steps. In the full three-language configuration, the model is trained on 175k chart images paired with $175 \times 3 = 525$k plotting scripts, evenly split across

Python, R, and LaTeX. For the single-language setting, we keep the dataset size constant by training on 175k chart–script pairs but increase the number of epochs to three. For the two-language setting, we preserve the same number of training steps by randomly duplicating half of the available plotting scripts to reach the equivalent scale. This ensures that all configurations—one, two, or three languages—are trained under comparable conditions. For the imbalanced configuration, we train on the source data described in Appendix B.1, and maintain the same number of training steps by randomly duplicating samples, as in the two-language setting. The training strategy for all language configurations are the same in Section 4.2.

Shared Subspace Ratio. The shared-subspace ratio is a statistic we design to quantify how much different language-specific routers rely on the same experts when processing the same chart. Formally, for each chart example c, let $S_{c,l} \subseteq \{0, \dots, N-1\}$ 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 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 high value indicates that most experts are shared across languages, while a low value indicates that only a few experts are shared and the rest are language-specific.

C.3 Prompt Sensitivity Study

We adopt the prompt used in ChartMimic (Shi et al., 2025) to maintain experimental consistency, as illustrated in Figure 11. To ensure that the inclusion of the phrase "a STEM paper" does not introduce unintended bias, we conduct a targeted prompt sensitivity analysis. Specifically, we evaluate two variants: (i) removing only the phrase "a STEM paper," and (ii) removing the entire sentence in which it appears. Both ChartLuMA-1.3B and Phi-3.5-vision are assessed on the Chart2NCode test set under these modified prompts. The results in Table 7 indicate that these variations yield no substantive differences in performance, confirming the robustness of our evaluation prompt.

Table 7: Sensitivity study of evaluation prompt on Chart2NCode test set using Phi-3.5-vision and CharLuMA-1.3B.

Model	Prompt Version	Chart2NCode		
		ER	CB	DS
Phi-3.5-vision	Default	41.2	7.7	49.6
	Version 1	41.4	7.5	49.9
	Version 2	41.0	7.5	49.6
CharLuMA-1.3B	Default	91.1	23.2	78.9
	Version 1	91.0	23.1	79.1
	Version 2	91.2	22.9	79.0

C.4 Error Analysis

We conduct an error analysis to identify the common sources of execution failures and reproduction limitations of CharLuMA-6.7B. In terms of execution failures, Python and R scripts most frequently break due to mismatched data dimensions or the use of undefined variables, whereas LaTeX scripts typically fail because of syntax omissions, such as missing braces. For example, the Python case in Figure 13(a) produces incompatible x–y list lengths when calling the `ax.plot` function. The R case in Figure 13(b) invokes an undefined variable `angle` in a `geom_polygon` call. The LaTeX case in Figure 13(c) fails due to an omitted closing curly brace in the title and x-tick label definition.

For reproduction limitations, the generated code executes but yields charts that diverge from the reference in various ways. We observe three recurring patterns: (i) annotation errors, such as missing legends or hallucinated axis labels; (ii) chart type errors, where the model misclassifies the intended chart subtype; and (iii) stylistic errors, including incorrect color palettes, font settings, or line styles. For instance, the reproduced chart in Figure 13(a) from ChartMimic mislabels a group name ("AI-Dive" instead of "AIDeepDive") and incorrectly overlays an additional filled area in the radar plot that does not exist in the gold chart. Figure 13(b), also from ChartMimic, shows a subtype recognition error, where stacked error bars are generated in place of grouped error bars. The case in Figure 13(c) from Chart2NCode using R demonstrates malformed x-tick labels (a missing "=") and an ordering of bars inconsistent with the gold chart. Finally, the LaTeX example in Figure 13(d)

Prompt Template of Chart-to-code Generation Task enhanced

You are an expert {`target language`} developer who specializes in writing matplotlib 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 matplotlib code that reproduces the picture below, starting with ”““{`target language symbol`}” and ending with ”““”.

Figure 11: Prompt Template of Chart-to-code Generation Task

Prompt Template of GPT-4o Scoring enhanced

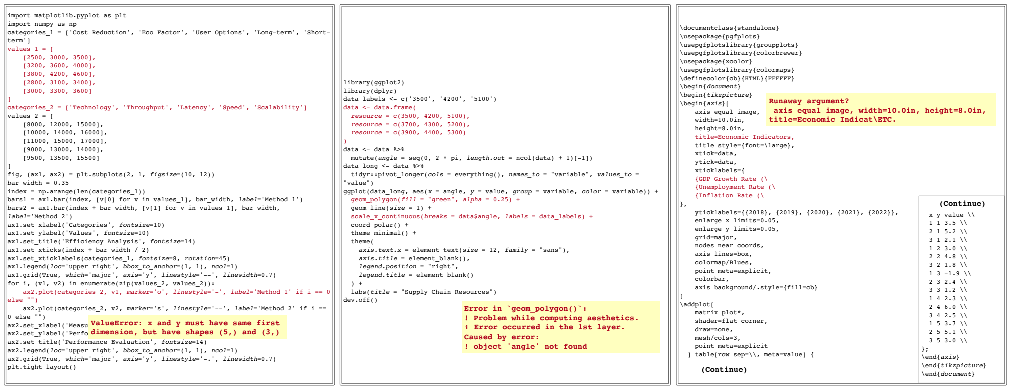
You are an excellent judge at evaluating visualization chart plots. The first image (reference image) is created using ground truth matplotlib code, and the second image (AI-generated image) is created using matplotlib code generated by an AI assistant. Your task is to score how well the AI-generated 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 12: Prompt Template of Chart-to-code Generation Task

from Chart2NCode exhibits an incorrect color scheme and hallucinates additional text annotations within a pie chart.

C.5 Examples

We qualitatively compare CharLuMA-6.7B with GPT-4o and ChartCoder on representative cases drawn from both the Chart2NCode test set and ChartMimic. In the Chart2NCode examples (Figure 15, Figure 16, and Figure 17), CharLuMA-6.7B consistently reproduces high-quality charts across Python, R, and LaTeX, whereas GPT-4o exhibits reduced reliability in R and LaTeX, and ChartCoder frequently fails to generate valid scripts in these languages. We also present four chart-to-Python examples from ChartMimic (Figure 18), which highlight CharLuMA-6.7B's strong chart reproduction ability in Python, performing on par with GPT-4o and ChartCoder, the current state-of-the-art among open-source MLLMs for chart-to-Python generation.



```python
import matplotlib.pyplot as plt
import numpy as np
categories_1 = ['Cost Reduction', 'Eco Factor', 'User Options', 'Long-term', 'Short-term']
values_1 = [
    [2500, 3000, 3500],
    [3200, 3600, 4000],
    [3800, 4200, 4600],
    [2800, 3100, 3400],
    [3000, 3300, 3600]
]
categories_2 = ['Technology', 'Throughput', 'Latency', 'Speed', 'Scalability']
values_2 = [
    [8000, 12000, 15000],
    [10000, 14000, 16000],
    [11000, 15000, 17000],
    [9000, 13000, 14000],
    [9500, 13500, 15500]
]
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 12))
bar_width = 0.35
index = np.arange(len(categories_1))
bars1 = ax1.bar(index, [v[0] for v in values_1], bar_width, label='Method 1')
bars2 = ax1.bar(index + bar_width, [v[1] for v in values_1], bar_width,
label='Method 2')
ax1.set_xlabel('Categories', fontsize=10)
ax1.set_ylabel('Values', fontsize=10)
ax1.set_title('Efficiency Analysis', fontsize=14)
ax1.set_xticks(index + bar_width / 2)
ax1.set_xticklabels(categories_1, fontsize=8, rotation=45)
ax1.legend(loc='upper right', bbox_to_anchor=(1, 1), ncol=1)
ax1.grid(True, which='major', axis='y', linestyle='--', linewidth=0.7)
for i, (v1, v2) in enumerate(zip(values_2, values_2)):
    ax2.plot(categories_2, v1, marker='o', linestyle='-', label='Method 1' if i == 0
else "")
    ax2.plot(categories_2, v2, marker='s', linestyle='--', label='Method 2' if i ==
0 else "")
ax2.set_xlabel('Measu
ax2.set_ylabel('Perfo
ax2.set_title('Performance Evaluation', fontsize=14)
ax2.legend(loc='upper right', bbox_to_anchor=(1, 1), ncol=1)
ax2.grid(True, which='major', axis='y', linestyle='-.', linewidth=0.7)
plt.tight_layout()
```

```
ValueError: x and y must have same first
dimension, but have shapes (5,) and (3,)
```

(a) Error Case for Python

```r
library(ggplot2)
library(dplyr)
data_labels <- c('3500', '4200', '5100')
data <- data.frame(
  resource = c(3500, 4200, 5100),
  resource = c(3700, 4300, 5200),
  resource = c(3900, 4400, 5300)
)
data <- data %>%
  mutate(angle = seq(0, 2 * pi, length.out = ncol(data) + 1)[-1])
data_long <- data %>%
  tidyr::pivot_longer(cols = everything(), names_to = "variable", values_to =
"value")
ggplot(data_long, aes(x = angle, y = value, group = variable, color = variable)) +
  geom_polygon(fill = "green", alpha = 0.25) +
  geom_line(size = 1) +
  scale_x_continuous(breaks = data$angle, labels = data_labels) +
  coord_polar() +
  theme_minimal() +
  theme(
    axis.text.x = element_text(size = 12, family = "sans"),
    axis.title = element_blank(),
    legend.position = "right",
    legend.title = element_blank()
  ) +
  labs(title = "Supply Chain Resources")
dev.off()
```

```
Error in `geom_polygon()`:
! Problem while computing aesthetics.
i Error occurred in the 1st layer.
Caused by error:
! object 'angle' not found
```

(b) Error Case for R

```latex
\documentclass{standalone}
\usepackage{pgfplots}
\usepgfplotslibrary{groupplots}
\usepgfplotslibrary{colorbrewer}
\usepackage{xcolor}
\usepgfplotslibrary{colormaps}
\definecolor{cb}{HTML}{FFFFFF}
\begin{document}
\begin{tikzpicture}
\begin{axis}[
    axis equal image,
    width=10.0in,
    height=8.0in,
    title=Economic Indicators,
    title style={font=\large},
    xtick=data,
    ytick=data,
    xticklabels={
    {GDP Growth Rate (\
    {Unemployment Rate (\
    {Inflation Rate (\
},
    yticklabels={{2018}, {2019}, {2020}, {2021}, {2022}},
    enlarge x limits=0.05,
    enlarge y limits=0.05,
    grid=major,
    nodes near coords,
    axis lines=box,
    colormap/Blues,
    point meta=explicit,
    colorbar,
    axis background/.style={fill=cb}
]
\addplot[
    matrix plot*,
    shader=flat corner,
    draw=none,
    mesh/cols=3,
    point meta=explicit
  ] table[row sep=\\, meta=value] {

     (Continue)
```

```latex
(Continue)
 x y value \\
 1 1 3.5 \\
 2 1 5.2 \\
 3 1 2.1 \\
 1 2 3.0 \\
 2 2 4.8 \\
 3 2 1.8 \\
 1 3 -1.9 \\
 2 3 2.4 \\
 3 3 1.2 \\
 1 4 2.3 \\
 2 4 6.0 \\
 3 4 2.5 \\
 1 5 3.7 \\
 2 5 5.1 \\
 3 5 3.0 \\
};
\end{axis}
\end{tikzpicture}
\end{document}
```

```
Runaway argument?
 axis equal image, width=10.0in, height=8.0in,
title=Economic Indicat\ETC.
```

(c) Error Case for LaTex

Figure 13: Case study of execution errors in generated code for CharLuMA-6.7B.

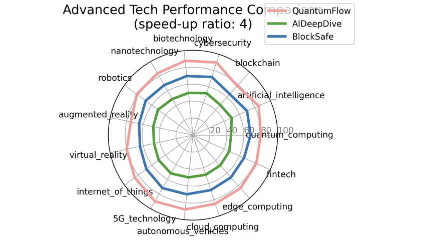
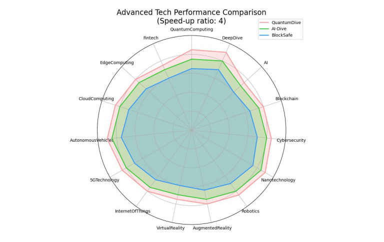
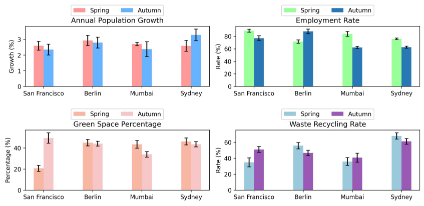
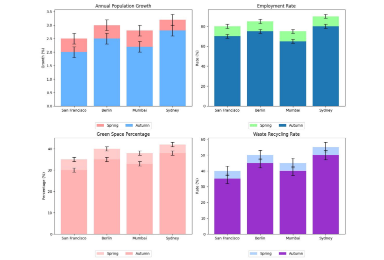
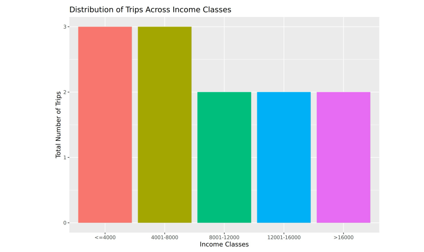
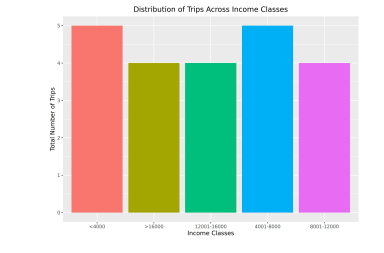
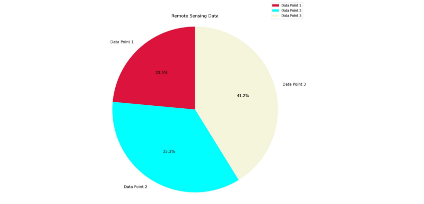
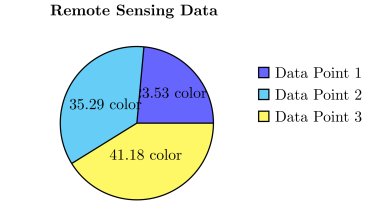
	Gold Chart	**Reproduced Chart**
(a) Case from ChartMimic using Python		
(b) Case from ChartMimic using Python		
(c) Case from Chart2NCode using R		
(d) Case from Chart2NCode using LaTex		

Figure 14: Case study of reproduction errors in generated charts for CharLuMA-6.7B.

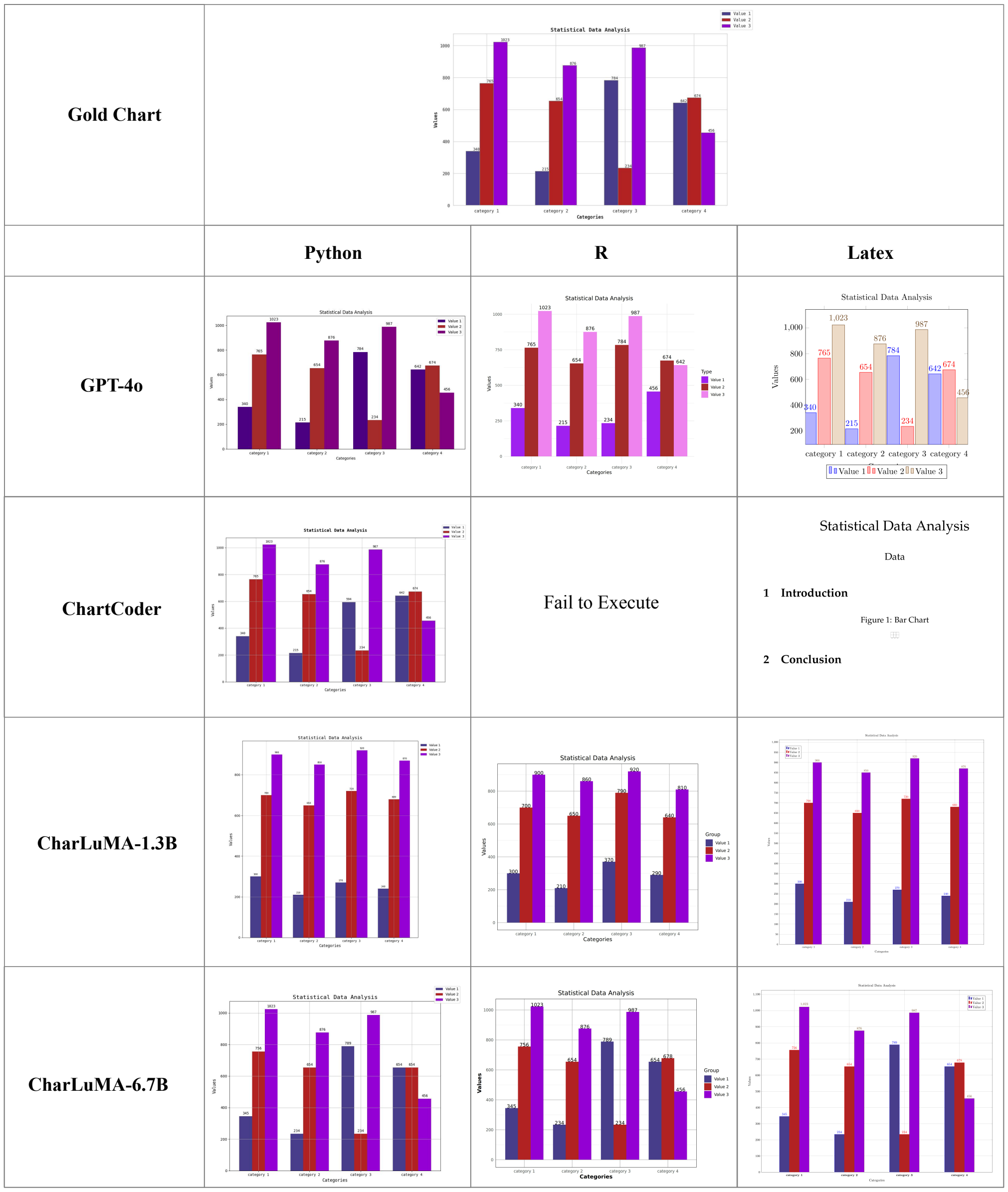


Figure 15: Case study of a grouped bar chart input and generated outputs from the Chart2NCode test set across three plotting languages.

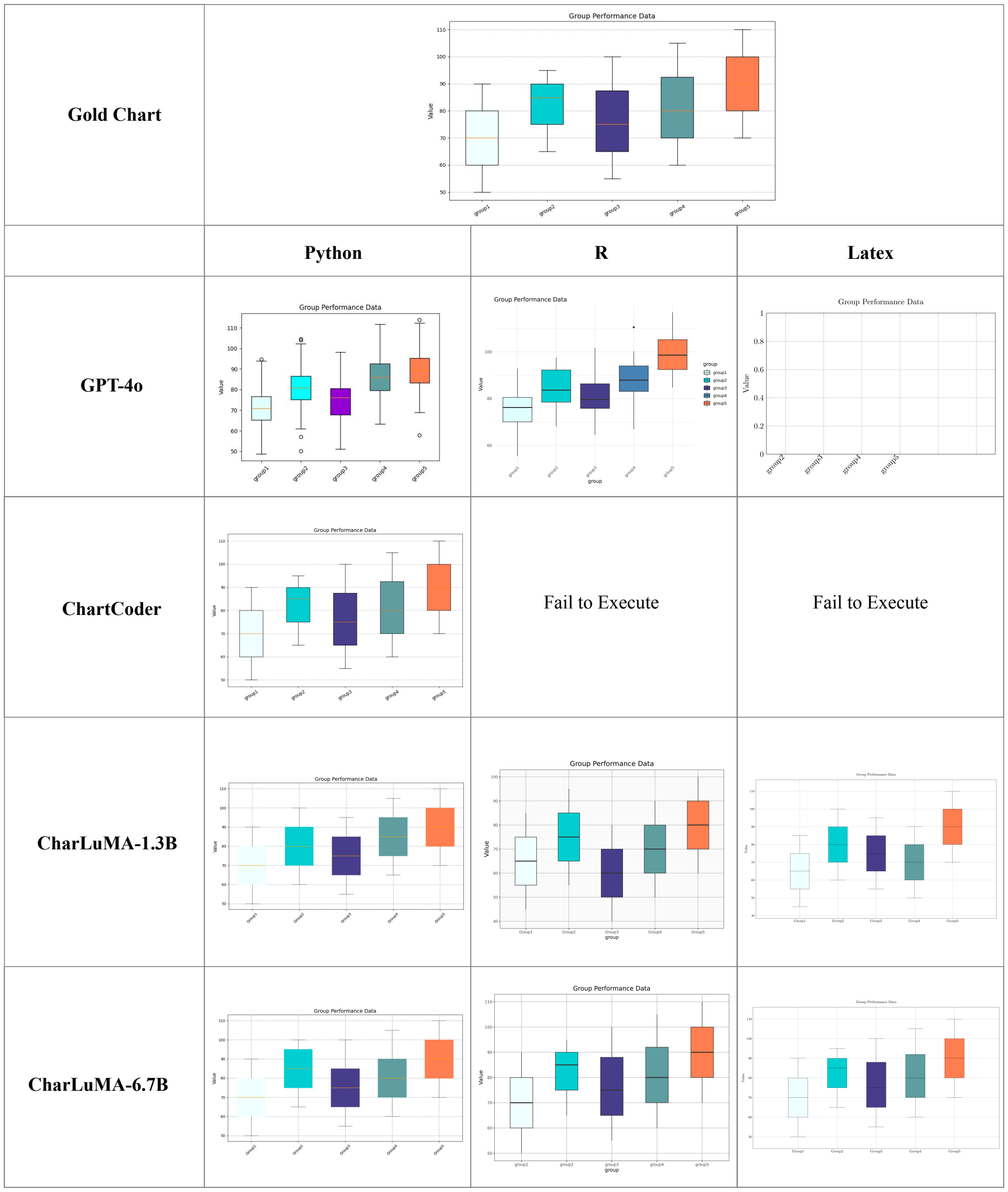


Figure 16: Case study of a box chart input and generated outputs from the Chart2NCode test set across three plotting languages.

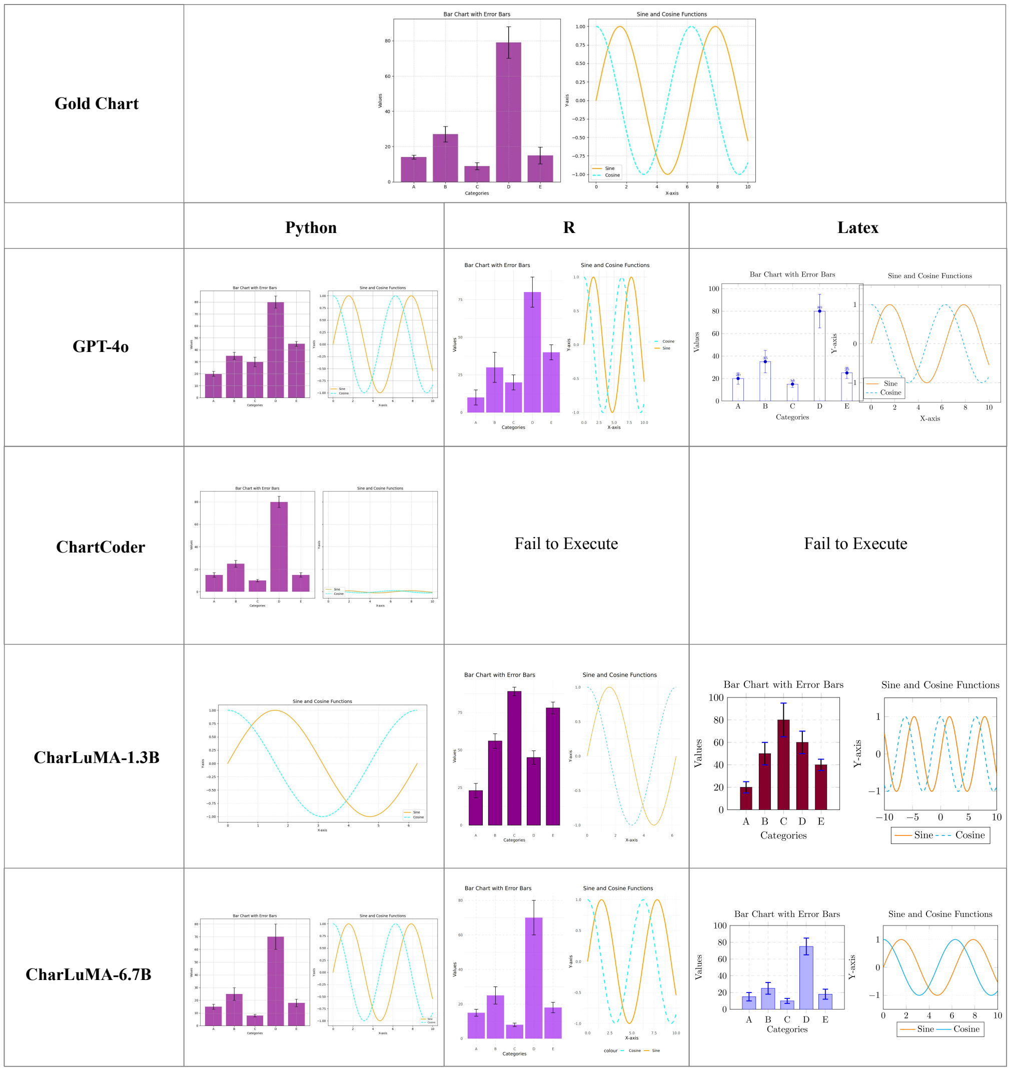


Figure 17: Case study of a two-subplot chart input and generated outputs from the Chart2NCode test set across three plotting languages.

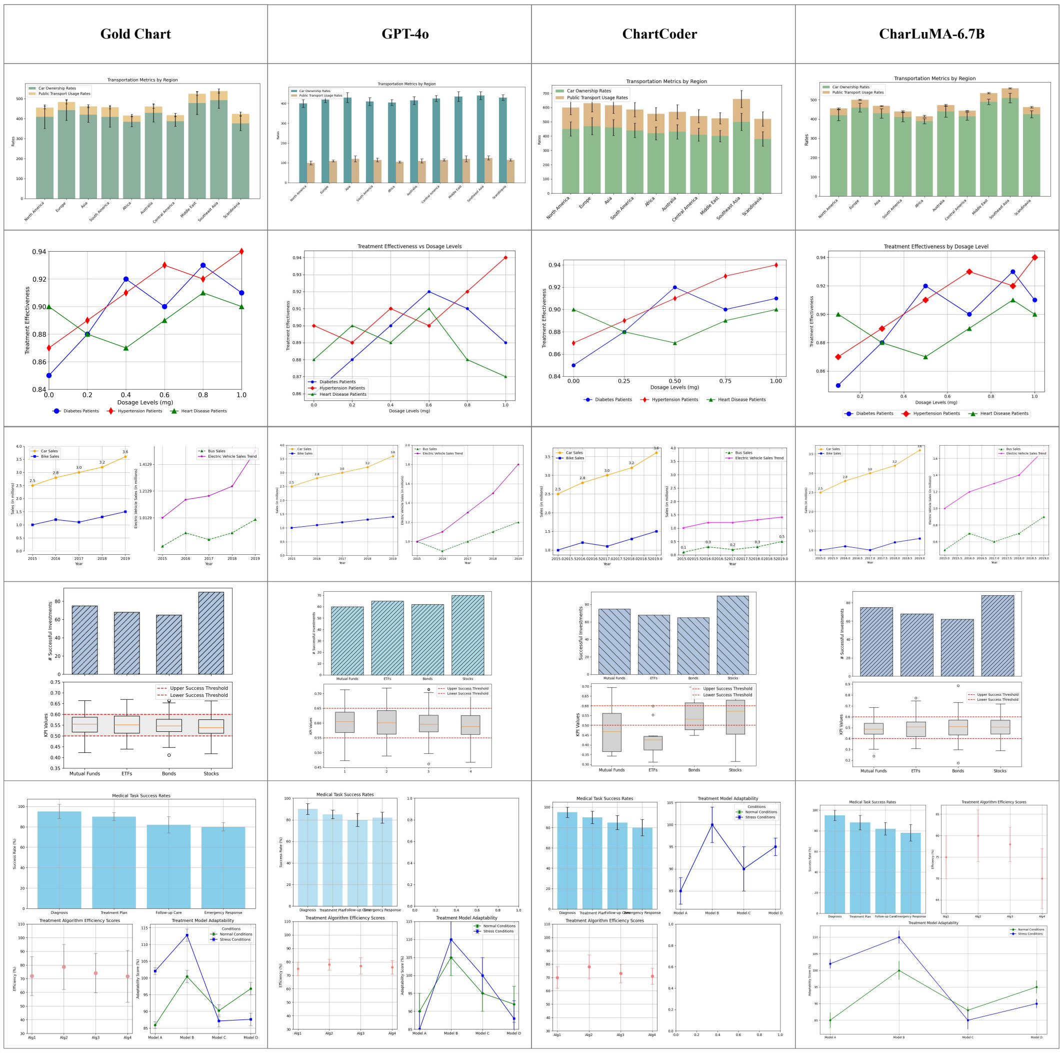


Figure 18: Case study of model inputs and generated outputs from ChartMimic in Python.