

A Appendix

We provide a details of the dataset and experiment settings in the technical appendices. In the section A.1, we include a detailed statistical analysis of the dataset as well as the construction process of the dataset. In the section A.2, we provide a proof of equivalence of the proposed Joint-Code-Visual Reward to the combination of process reward and output reward. Section A.3 provides detailed implementation across experimental settings. Section A.6 provides user study results across three tasks and correlation analysis of the QA-based score with human preference trend. Finally, we provide qualitative results across different categories of animation and task types in section A.7.

A.1 Dataset

A.1.1 Construction Details

In this section, we provide a more detailed version of the dataset construction pipeline. As mentioned in 3, the construction pipeline go through following four steps.

Seed Code Set Construction We crawled a set of JavaScript chart templates from ECharts Team (2025). Among all chart types provided by ECharts, we excluded certain types that either require complex 3D modeling(e.g. 3D global map chart)—beyond the capability of MLLMs to generate code for—or involve excessively long data sequences(e.g. scatter chart that covers thousands of data point) that exceed the context and generation length limitation of MLLMs, leading to rendering failures. After manually filtering for appropriate chart types, we converted the original ECharts JavaScript samples into standardized HTML code to ensure consistent rendering as shown in Figure 11. We removed some interactive properties and retained only dynamic elements suitable for video display. The JavaScript code was embedded within an HTML template, with a fixed size setting (1920x1080) to ensure stable rendering results. This process yielded 175 seed templates spanning 20 chart categories as shown in 7.

Full Code Set Generation After obtaining the seed templates, we defined a pool of animation modifications m covering various aspects including *data elements*, *layout*, *visual style*, *animation speed*, and *animation effects*, as shown in 12. For each c_{seed} , we randomly sampled one to ten modifications and prompted Gemini-2.5-Pro Team et al. (2023) to apply them using prompt 13. For example, if modifications “Modify data content aspect of the provided HTML code” and “Modify line width aspect of the provided HTML code” were sampled, Gemini would modify the data to visualize and the line width of the original dynamic chart code accordingly. We filtered out non-renderable code patches. In the expansion process, Gemini2.5-pro demonstrated strong capabilities in modifying the provided code based on instructions. Over 90% of the generated code was renderable and adhered to the modification instructions, demonstrating the significant potential of Gemini as a key component in a data generation engine. We recorded the applied modifications for each expanded sample relative to its original seed for quality control purposes.

Annotation Generation After successfully obtaining the expanded code set, each valid code sample c was rendered into a dynamic chart animation video v using a headless browser with Timesnap Tung (2022) and FFmpeg team (2025). The HTML code was first rendered in a headless Chrome browser. Using the gtimesnap library, we controlled the animation playback speed, ensuring synchronization between Chrome display and screenshot capture tools. We captured 48 frames under a 24 FPS and 2-second setting and then merged these frames into an animation video using the FFmpeg library. Subsequently, Gemini-2.5-Pro was prompted to generate a detailed textual description t_d of the code using 14, which was then summarized into a concise description t_s . Additionally, the data sequence visualized in the expanded code was extracted by Gemini-2.5-Pro.

QA pair Generation After acquiring the annotations and code–video pairs, we generated 10 question–answer pairs for both code and visual modalities to assess whether the generated results met the specified requirements using prompt 15 and 17. An example is illustrated in Figure 10. The video and code QA pairs cover different aspects of the generated dynamic chart, providing both process supervision and outcome verification as demonstrated in A.2.

For each generated question, we prompted Gemini-2.5-Pro to answer based on the code and video. As shown in Table 3, Gemini achieved over 95% accuracy on video-based questions and over a 90% affirmative response rate on code-based questions when given reference code and video, validating the quality of the QA pairs and the robustness of Gemini as an evaluation proxy.

Model	CodeQA	VideoQA
Gemini2.5-Flash	87.79%	96.16%
Gemini2.5-Pro	91.93%	96.27%

Table 3: Performance Metrics of Gemini 2.5 Models on QA accuracy on ground truth input

A.1.2 Statistically Analysis

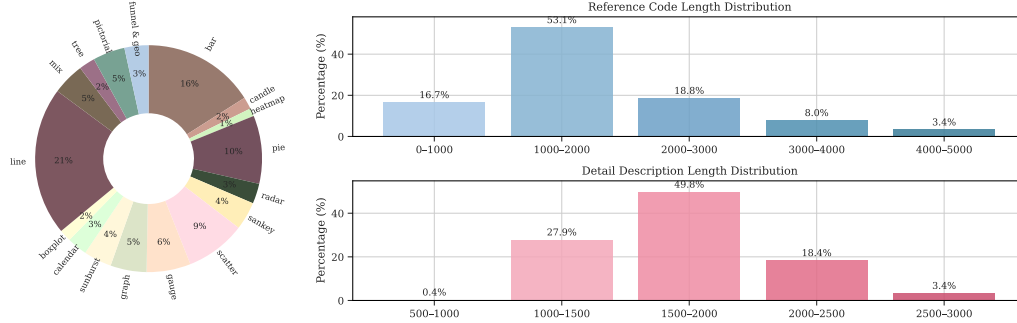


Figure 7: Dataset distribution of DCG-8K. **Left:** Chart category included in the DCG-8K. **Right:** Code length and detailed description length distribution of DCG-8K.

As shown in Figure 7, DCG-8K includes 18 categories of chart types, each covering several variants. Line, bar, and pie charts still play a major role, while the remaining types contribute to a diverse range of visualizations, from Sankey diagrams to heatmaps. For detailed chart descriptions, the token length distribution exceeds 1500 on average, and the required output length is often more than 2000 tokens, posing a significant challenge to the language backbone of MLLMs. Meanwhile, the video token length is fixed during processing and is shorter than the average language description length, in order to maintain higher data efficiency. However, the poorer performance shown in the main experiment (Table 1) indicates potential information loss during the vision modeling process.

A.2 Joint Code-Visual Reward as a Combined Process-Outcome Reward

Our proposed Joint Code-Visual Reward (JCVR) exhibits conceptual alignment with the original GRPO framework proposed in Shao et al. (2024). We establish the following two correspondences. **First**, the generated code c_g , is analogous to the intermediate thinking process t in GRPO. **Second**, the rendered video, v_g , obtained via a deterministic rendering function R such that $v_g = R(c_g)$, corresponds to the final answer a in the former setting for math or code tasks. The generative sequences can be formalized as:

$$\text{GRPO: } q \xrightarrow{\pi_\theta(t|q)} t \xrightarrow{\pi_\theta(a|q,t)} a \quad (4)$$

$$\text{JCVR-GRPO: } q \xrightarrow{\pi_\theta(c_g|q)} c_g \xrightarrow{R(\cdot)} v_g \quad (5)$$

where q is the initial instruction, and π_θ is the policy parameterized by the model parameters θ . A notable distinction arises in the generation of the final output: a in GRPO (Eq. 4) is typically a probabilistic sample from $\pi_\theta(a|q, t)$, whereas v_g is a deterministic output of $R(c_g)$.

As introduced in Section 3, $s_{\text{code}}(q, c)$ for code quality and $s_{\text{video}}(q, v)$ for video quality, both scores are evaluated with respect to the input query q . These scores directly map to the concepts of process rewards and outcome rewards in the original GRPO setting Guo et al. (2025) respectively:

$$s_{\text{code}}(q, c_g) \equiv R_{\text{process}}(t) \quad (6)$$

$$s_{\text{video}}(q, v_g) \equiv R_{\text{outcome}}(A) \quad (7)$$

Here, the thinking process t (represented by our generated code c_g) and the final answer A (represented by our rendered video $v_g = R(c_g)$) are derived in response to the query q . Therefore, the Joint Code-Visual Reward, r_{jcv} , as defined in Eq. 3, is an instantiation of the combined process and outcome reward used in the former reinforcement learning framework:

$$R_{\text{GRPO}}(t, A) = \lambda_t \cdot R_{\text{process}}(t) + \lambda_A \cdot R_{\text{outcome}}(A) \quad (8)$$

Given the established equivalences $t \equiv c_g$ (code as process) and $A \equiv R(c_g)$ (rendered video as outcome), and by setting the weighting coefficients $\lambda_t = w_{\text{code}}$ and $\lambda_A = w_{\text{video}}$, it follows that our specific reward formulation aligns with the standard reward structure combining process reward and **outcome reward**:

$$r_{\text{jcv}}(q, c_g, R(c_g)) \equiv R_{\text{GRPO}}(c_g, R(c_g)) \quad (9)$$

Thus, our approach provides a combination reward from both process and outcome supervision.

A.3 Implementation Details

A.4 Prompt Sets

We provide all prompt sets employed for dataset construction, training, and evaluation. These include: the rule pool for seed code modification (12); prompts for generating expanded code sets (13) and extracting data from them (16); prompts for generating detailed code descriptions (14), code QA pairs (15), and video QA pairs (17); and finally, prompts for evaluating the generation results for both code (18) and video (19).

A.5 training configuration

Argument	Value
train type	lora
torch dtype	bfloat16
train epochs	1
train batch size	64
learning rate	1e-4
lora rank	64
lora alpha	128
max length	16384
warmup ratio	0.05
gpu number	4

Table 4: Training hyperparameters for SFT.

Argument	Value
train type	lora
torch dtype	bfloat16
train epochs	1
train batch size	32
learning rate	1e-5
lora rank	32
lora alpha	64
max length	8192
warmup ratio	0.05
num generations	8
temperature	0.9
grpo β	0.04
grpo ϵ	0.2
gpu number	8

Table 5: GRPO Training hyperparameters.

MS-SWIFT ? framework is applied for training and inference. We provide the implementation configuration for supervised finetuning in Table 4, and reinforcement learning in Table 5. The rest configuration hyperparameters remain the same as the default setting in

ms-swift official training script. For reward scoring and inference performance evaluation on DCG-Bench, we utilize Gemini-2.5-pro-03-26 Team et al. (2023) with temperature set to 0 to guarantee stabilized output. For GRPO training, we utilize async architecture ?, using four gpus to inference and four gpus to optimize model parameters. For ablation study, all the configurations remain the same with only data partition changed, except for generalization experiment in ?? the training epoch is raise to 8 to match the computational budget.

A.6 User Study and correlation analysis

For the user study setting, we built a Gradio webpage, as shown in Figure 20, which randomly displays two videos generated by two models, along with either a textual description from S2C or D2C task or a reference video from V2C task. We utilize the 175 seed samples to conduct the user study. In addition to the V2C task results shown in Figure ??, we present user study results for the language-grounded S2C and D2C tasks. As illustrated in Figure 8, on both the D2C and S2C tasks, our model achieves competitive performance compared to the best-performing open-sourced MLLMs, while still showing a certain gap in code generation quality compared to commercial models. According to the Pearson correlation between the QA score and human preferences shown in Figure 9, human ratings exhibit a similar trend to Gemini-evaluated QA scores, with an average Pearson coefficient exceeding 0.75, further demonstrating the effectiveness of our evaluation system.

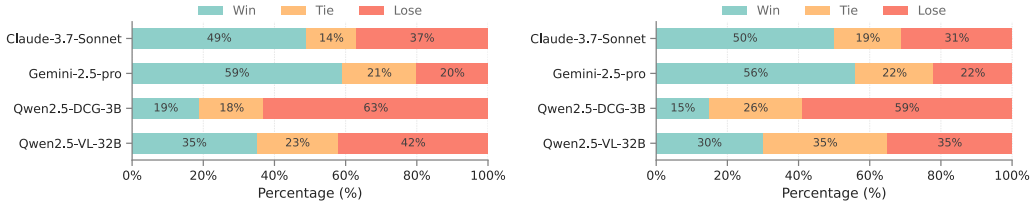


Figure 8: Left: user study result on D2C. Right: user study result on S2C.

A.7 Qualitative Result

We presents representative generation results. Under the same descriptions, Qwen2.5-DCG-3B generates well-structured dynamic chart animations that more closely align with the reference videos compared to other open-source models. Additional examples and animation videos are provided in the supplementary material.

We present more qualitative generation result of our model in comaprison with the reference video in Figure 21, Figure 22, Figure 23 and Figure 24.

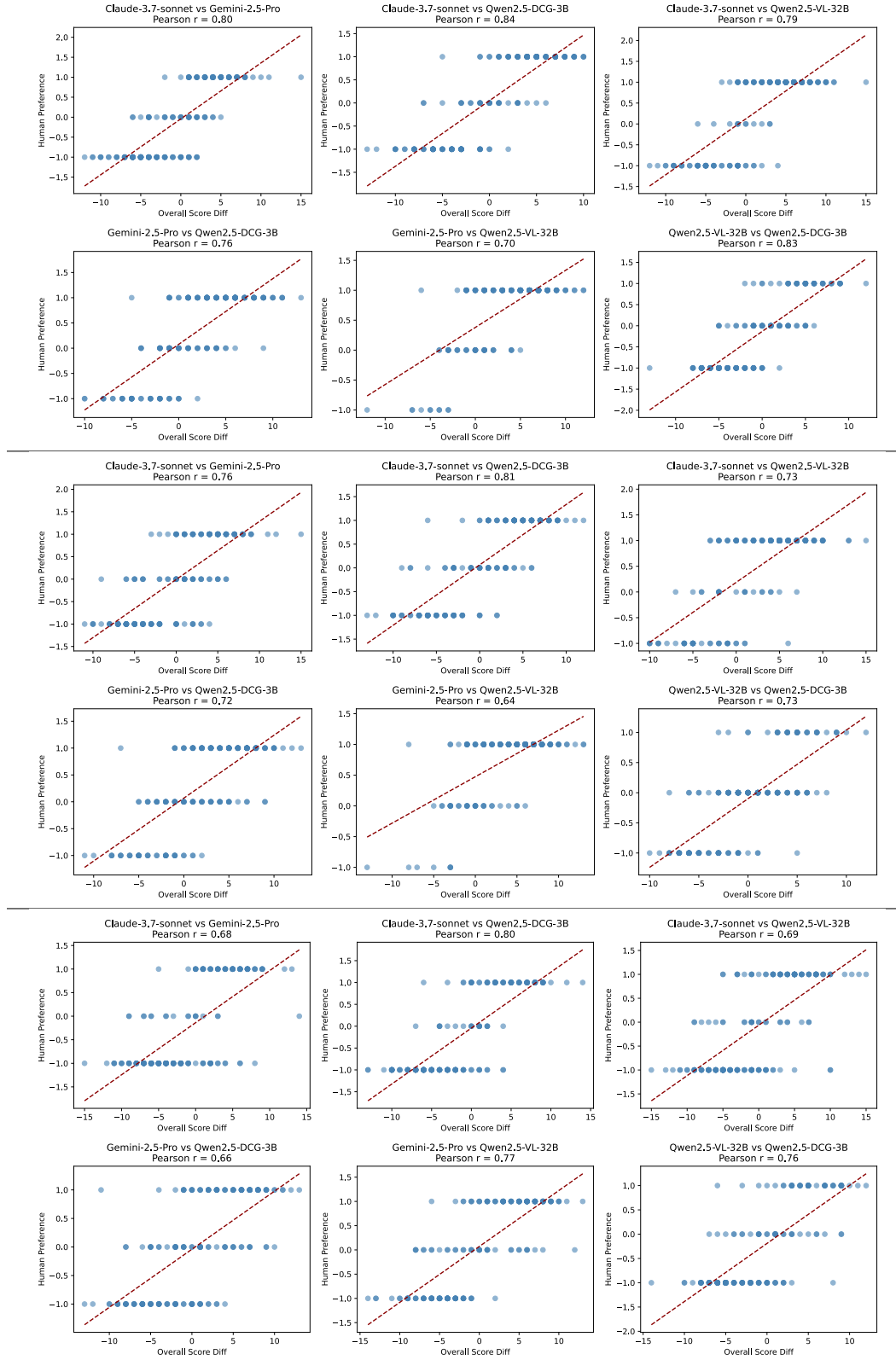


Figure 9: **Top:** D2C Pearson correlation between QA score and human preference. **Middle:** S2C Pearson correlation between QA score and human preference. **Bottom:** V2C Pearson correlation between QA score and human preference.



Figure 10: An example of Video QA and Code QA for a dynamic chart generation quality evaluation. **Left:** Code QA. **Middle:** Dynamic Chart Animation. **Right:** Video QA.

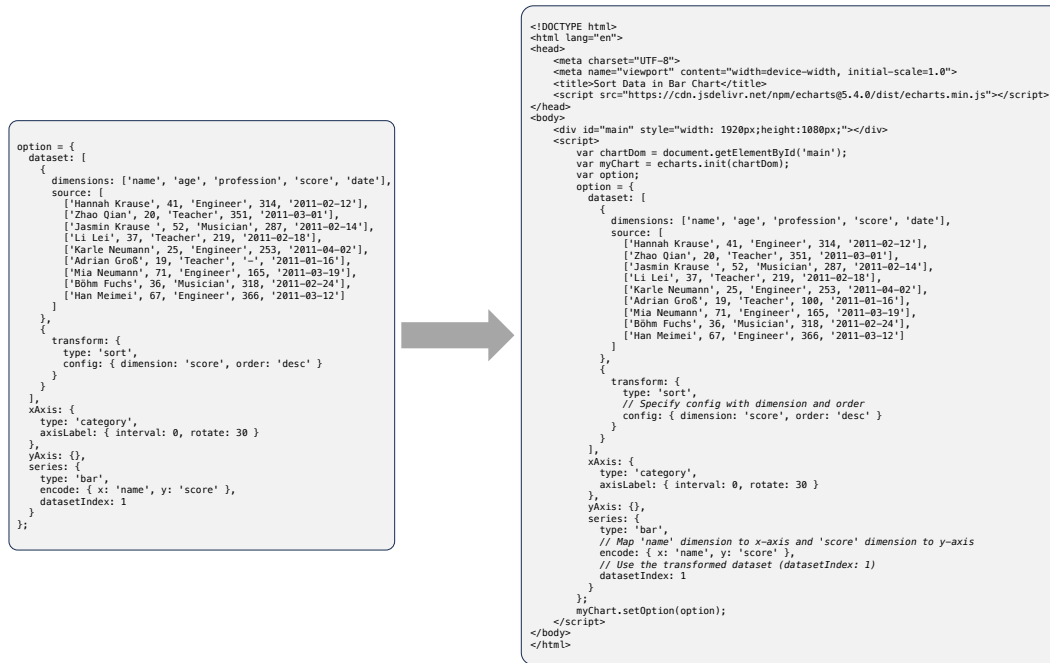


Figure 11: Raw javascript code transformed to HTML code

```

"Modify data content aspect of the provided HTML code.",
"Modify data format aspect of the provided HTML code.",
"Modify series color aspect of the provided HTML code.",
"Modify background color aspect of the provided HTML code.",
"Modify axis color (lines, ticks, labels) aspect of the provided HTML code.",
"Modify grid line/area color aspect of the provided HTML code.",
"Modify tooltip style (background color, border color, text color) aspect of the provided HTML code.",
"Modify legend text color aspect of the provided HTML code.",
"Modify title text color aspect of the provided HTML code.",
"Modify line type (solid, dashed, dotted) aspect of the provided HTML code.",
"Modify line width aspect of the provided HTML code.",
"Modify marker symbol shape aspect of the provided HTML code.",
"Modify marker symbol size aspect of the provided HTML code.",
"Modify chart style (border, corner radius) aspect of the provided HTML code.",
"Modify fill style (color, opacity for area charts) aspect of the provided HTML code.",
"Modify font (family, size, weight, style) aspect of the provided HTML code.",
"Modify label content and formatting aspect of the provided HTML code.",
"Modify label position aspect of the provided HTML code.",
"Modify label rotation angle aspect of the provided HTML code.",
"Modify chart position and margin (grid settings) aspect of the provided HTML code.",
"Modify axis position (top/bottom/left/right) aspect of the provided HTML code.",
"Modify axis type (value, category, time, log) aspect of the provided HTML code.",
"Modify axis range (min/max) aspect of the provided HTML code.",
"Modify axis scale (tick count, visibility of ticks/labels) aspect of the provided HTML code.",
"Modify series graph layout (force, circular) aspect of the provided HTML code.",
"Modify series stacking (for bar/line charts) aspect of the provided HTML code.",
"Modify initial animation settings (enable, duration, easing, delay) aspect of the provided HTML code.",
"Modify update animation settings (duration, easing) aspect of the provided HTML code.",
"Modify specific animation effects (e.g., graph node force layout process) aspect of the provided HTML code."

```

Figure 12: Modification dimensions used to modify the seed code.

```

### Task
You will receive a piece of seed HTML that creates a dynamic, interactive ECharts visualization, together
with the raw data that the chart is based on.
Generate a new but runnable HTML file that keeps the same chart type yet only changes the items explicitly
listed under "Elements that must be changed".

### Note
* The HTML code should be pure code without any comments or explanations.
* If the modification requirement doesn't make sense for provided chart type, ignore it.
* The modification should keep the harmony of the chart and make it modern, coherent and visually appealing, do
not use random combination of colors, shapes, etc.
* Make the font size larger in display of 1920 * 1080 so that it is more readable.
* Do not add or remove external libraries.
* Apart from the listed items, the structure, chart type and semantics of the original code must stay intact.
* Animate, style, and update the data coherently.
* Return a full, self-contained HTML document wrapped in a ```html ...``` code block.

### Seed HTML
```html
{seed_html}
```

### Seed data
```json
{seed_data}
```

### Elements change requirement
{elements}

### Output format
```html
<!-- place the complete modified HTML here -->
```

```

Figure 13: prompt for seed code expansion to full code set

Analyze the chart animation shown in the provided HTML code. Based on your analysis, generate a JSON object containing a detailed description of the animation. This JSON object should provide comprehensive information about the following elements, detailed enough that someone could potentially reproduce a similar animation programmatically.

The JSON object should follow this structure:

```
{
  "chartType": "String or Array of Strings: Identify the specific type(s) of chart presented (e.g., 'Bar Chart', 'Line Chart', 'Pie Chart', ['Combination Chart', 'Bar', 'Line']).",
  "animatedObjects": [
    "Strings: List all visual elements that undergo animation (e.g., 'Bars', 'Lines', 'Points', 'Areas', 'Pie Slices', 'X-Axis', 'Y-Axis', 'Gridlines', 'Data Labels', 'Title', 'Legend').",
  ],
  "animationSequence": [
    {
      "step": "Integer: A sequential number for the animation step (e.g., 1, 2).",
      "description": "String: A brief description of what happens in this step.",
      "elements": "String: List the specific animated object(s) involved in this step (e.g., 'Bars', 'Y-Axis Labels').",
      "action": "String: Describe the type of motion or change (e.g., 'Grow', 'Shrink', 'FadeIn', 'FadeOut', 'Appear', 'Disappear', 'Slide', 'Wipe', 'Rotate', ...).",
      "pathTarget": "String: Describe the trajectory or the final state/position/value reached (e.g., 'Grow vertically from zero to data value', 'Slide in from left'...).",
      "timing": {
        "speed": "String: Characterize the perceived speed (e.g., 'Fast', 'Slow', 'Moderate', 'Constant', 'Accelerating', 'Decelerating').",
        "easing": "String: Describe the easing function if apparent (e.g., 'Linear', 'Ease-In', 'Ease-Out', 'Ease-In-Out', 'Bounce').",
        "duration": "String: Estimated duration if possible (e.g., 'approx 0.5s', 'Short', 'Long').",
        "delay": "String: Describe any delay before this action starts (e.g., 'None', '0.2s after previous step', 'Staggered').",
        "order": "String: Describe if actions within the step are simultaneous or sequential (e.g., 'Simultaneous', 'Sequential', 'Staggered Start').",
      }
    }
  ],
  // Add more step objects as needed for the complete sequence
  "sizeChanges": [
    {
      "element": "String: The object whose size changes (e.g., 'Bars', 'Points').",
      "changeDescription": "String: Describe the size change (e.g., 'Grow from zero height/width', 'Increase radius from 0 to 5px', 'Area expands vertically').",
    }
  ],
  // Add more objects if multiple elements change size differently
  "colorDetails": {
    "staticColors": [
      {
        "element": "String: Chart element (e.g., 'Background', 'Axis Lines', 'Gridlines', 'Default Bar Color').",
        "color": "String: Description or hex/name if identifiable (e.g., 'White', '#F0F0F0', 'Dark Gray', 'Blue (#007bff)').",
      }
    ],
    "animatedColors": [
      {
        "element": "String: Element whose color animates (e.g., 'Bars', 'Points', 'Area Fill').",
        "trigger": "String: When the color change happens (e.g., 'On Animation Completion', 'On Hover', 'During Value Change', 'Based on Value').",
        "fromColor": "String: Initial color (if applicable).",
        "toColor": "String: Final color (description or hex/name).",
        "transition": "String: How the color changes (e.g., 'Instant', 'Smooth Fade', 'Wipe').",
      }
    ]
  },
  "shapeDetails": [
    {
      "element": "String: The object described (e.g., 'Bars', 'Points', 'Line Caps').",
      "initialShape": "String: The shape of the element.",
      "shapeChange": "String: Describe if and how the shape changes during animation (e.g., 'None', 'Morphs from circle to square'). Typically 'None' for standard charts.",
    }
  ],
  "labelsAndText": [
    {
      "elementType": "String: Type of text element (e.g., 'Title', 'Subtitle', 'X-Axis Label', 'Y-Axis Label', 'Data Label', 'Legend Text', 'Tooltip').",
      "text": "String: The actual text content, if static and readable, or a description (e.g., 'Category Names', 'Numerical Values', 'Chart Title Appears Here').",
      "animation": "String: Describe any animation applied to the text (e.g., 'Fade In with Bars', 'Appears Statically', 'Moves with Data Point', 'Appears on Hover', 'None').",
    }
  ],
}

Return *only* the JSON object itself, enclosed in triple backticks (```). Do not include any introductory or explanatory text before or after the JSON block. Here is the HTML code:\n
```

Figure 14: prompt for detailed description generation

```
**Role:** You are an AI assistant specialized in generating verification questions for evaluating visual animations against detailed, structured descriptions.

**Task:** Your goal is to create a set of specific questions designed for a Vision Language Model (VLM). These questions will help determine if a generated HTML animation accurately reflects a detailed description provided in a **structured JSON format**.

**Input:** You will receive a **JSON object** describing a chart animation. This JSON includes detailed information within keys such as:
  * 'chartType': The type(s) of chart displayed.
  * 'animatedObjects': Which elements undergo animation.
  * 'animationSequence': The specific step-by-step breakdown of the animation, including elements, actions, paths/targets, and detailed 'timing' (speed, easing, duration, delay, order).
  * 'sizeChanges': How element sizes change (or explicitly *don't* change).
  * 'colorDetails': Includes 'staticColors' for unchanging elements and 'animatedColors' detailing transitions (element, trigger, from/to colors, transition type).
  * 'shapeDetails': Information about element shapes and any changes.
  * 'labelsAndText': Details on text elements, their content, and whether they are static or animated.
  * 'dataRepresentation': How data is visualized and potentially example values/coordinates.
  * 'overallEffect' / 'overallDetailedCaption': Narrative summary reinforcing key aspects.

**Instructions:**
1. **Analyze the Input JSON:** Carefully parse the provided JSON caption. Extract key verifiable details from *all* relevant fields, paying close attention to:
   * The exact sequence of events ('animationSequence').
   * Which elements are animated vs. static ('animatedObjects', 'staticColors', 'labelsAndText').
   * Precise animation actions (e.g., fade-in, move, grow).
   * Specific timing parameters (duration, speed, easing, order from 'animationSequence.timing').
   * Exact color values (hex codes preferred if available) for both static elements ('staticColors') and animation transitions ('animatedColors').
   * Specified size or shape changes (or lack thereof) ('sizeChanges', 'shapeDetails').
   * The appearance and state of labels and text ('labelsAndText').
   * The final positions or visual representation of data ('dataRepresentation', 'pathTarget').
2. **Generate Questions:** Create exactly 10 distinct questions based on these extracted details.
3. **Question Format:**
   * Each question must be answerable with a simple "Yes" or "No".
   * *Crucially*, formulate each question so that "Yes" is the correct answer *if and only if* the animation perfectly matches the specific detail described in the JSON. This includes verifying both the specified animations *and* the specified static states of elements.
   * Questions must be clear, specific, and unambiguous. Refer to concrete elements (e.g., 'scatter plot points', 'histogram bars'), actions (e.g., 'fade in', 'remain static'), specific colors (e.g., 'solid blue #467484', 'light green #90EE60'), timings (e.g., 'simultaneously', 'within 0.1-0.2 seconds'), and states (e.g., 'fully opaque', 'present from the start').
   * Ensure the questions collectively cover the different aspects detailed in the JSON: sequence, timing, color (static & animated), size, shape, specific elements (animated & static), labels, and data representation.
4. **Output Format:** Present the 10 questions as a single JSON object. The keys should be "question1", "question2", ..., "question10", and the values should be the corresponding questions themselves, as follows:
{
  "question1": "Question text here",
  "question2": "Question text here",
  ...
  "question10": "Question text here"
}

**Provide the detailed animation caption as a JSON object and Data in CSV format below for analysis: {json}, {csv}
```

Figure 15: prompt for seed code expansion to full code set

```
Analyze the the provided HTML code, and extract the data sample used to construct the chart animation, as well as the chart type
and data type. If the data only has label and no data, return the label that construct the chart animation(for example, if only ).
Return the data in csv format with the first line as the header. If no specific data is used, return None.
Return in JSON format like:
{
  "data": "csv data",
  "chart_type": "chart_type",
  "data_type": "data_type",
}
Here is the HTML code:\n
```

Figure 16: prompt for data extraction from full code set

```
**Role:** You are an AI assistant specialized in generating verification questions for evaluating visual animations against dynamic chart
animation.

**Task:** Your goal is to create a set of specific questions designed for a Vision Language Model (VLM). These questions will help determine
if a provided HTML dynamic chart animation accurately reproduced the referring video .

**Input:** You will receive a chart animation video that include a dynamic chart animation.

**Instructions:**
1. **Analyze the Input Video:** Carefully analyze the input video. Extract key verifiable details from *all* relevant fields*, paying close
attention to:
* The exact sequence of events ('animationSequence').
* Which elements are animated vs. static ('animatedObjects', 'staticColors', 'labelsAndText').
* Precise animation actions (e.g., fade-in, move, grow).
* Specific timing parameters (duration, speed, easing, order from 'animationSequence.timing').
* Exact color values (hex codes preferred if available) for both static elements ('staticColors') and animation transitions
('animatedColors').
* Specified size or shape changes (or lack thereof) ('sizeChanges', 'shapeDetails').
* The appearance and state of labels and text ('labelsAndText').
* The final positions or visual representation of data ('dataRepresentation', 'pathTarget').

2. **Generate Questions:** Create exactly 10 distinct questions based on these extracted details.

3. **Question Format:**
* Each question must be answerable with a simple "Yes" or "No".
* **Crucially, formulate each question so that "Yes" is the correct answer *if and only if* the animation perfectly matches the specific
detail described in the JSON.** This includes verifying both the specified animations *and* the specified static states of elements.
* Questions must be clear, specific, and unambiguous. Refer to concrete elements (e.g., 'scatter plot points', 'histogram bars'), actions
(e.g., 'fade in', 'remain static'), specific colors (e.g., 'solid blue #4674B4', 'light green #90BE6D'), timings (e.g., 'simultaneously',
'within 0.1-0.2 seconds'), and states (e.g., 'fully opaque', 'present from the start').
* Ensure the questions collectively cover the different aspects displayed in the video: sequence, timing, color (static & animated), size,
shape, specific elements (animated & static), labels, and data representation.

4. **Output Format:** Present the 10 questions as a single JSON object. The keys should be "question1", "question2", ..., "question10", and
the values should be the corresponding questions themselves, as follows:
{
  "question1": "Question text here",
  "question2": "Question text here",
  ...
  "question10": "Question text here"
}

**Here is the video:**
```

Figure 17: prompt for video qa generation based on data and video

```
You will be provided with HTML code that can render a chart animation. And you will be provided with a list of 10 checking
questions. Based on the chart animation HTML code, please answer the check list questions with only yes or no. Be very strict when
answering the question, little mismatch you should answer no. return the answer in JSON format like:
{
  "question1": "answer1",
  "question2": "answer2",
  ...
  "question10": "answer10"
}

Here is the ten questions:
```

Figure 18: prompt for codeqa score evaluation based on generated code patch

```
You will be provided with a video that shows a chart animation. And you will be provided with a list of 10 checking questions.
Based on the chart animation video, please answer the check list questions with only yes or no. Be very strict when answering the
question, little mismatch you should answer no. Return the answer in JSON format like:
{
  "question1": "answer1",
  "question2": "answer2",
  ...
  "question10": "answer10"
}

Here is the ten questions:
```

Figure 19: prompt for videoqa score evaluation based on generated video

Dynamic Chart Generation Comparison Study

Choose Task Type
Simple Text to Chart

Reference Text

The chart animation begins with the static appearance of the title 'Scatter Plot with Marginal Histograms' and the axes/grid structure. Simultaneously, the three data series animate into view over approximately 0.5 seconds using a 'quinticInOut' easing function. Blue scatter points (#5470C6) appear at their respective (X,Y) coordinates in the bottom-left grid. In the top-left grid, green X-histogram bars (#91CC75) grow vertically from the bottom axis to heights representing data frequency within X-value bins; associated frequency count labels (#333) fade in above each bar. In the bottom-right grid, yellow Y-histogram bars (#FFA358) grow horizontally from the left axis to widths representing data frequency within Y-value bins; associated frequency count labels (#333) fade in to the right of each bar. Once the initial animation completes, the chart is static, with tooltips available on hover to show specific data point values or bin frequencies.

Model A Output

Model B Output

Evaluation Result for Left Model
☐ Win ☐ Loss ☐ Tie

Feedback

Dynamic Chart Generation Comparison Study

Choose Task Type
Video to Chart

Reference Video

Model A Output

Model B Output

Evaluation Result for Left Model
☐ Win ☐ Loss ☐ Tie

Feedback

Figure 20: User study gradio webpage example. **Up:** S2C task. **Down:** V2C task.

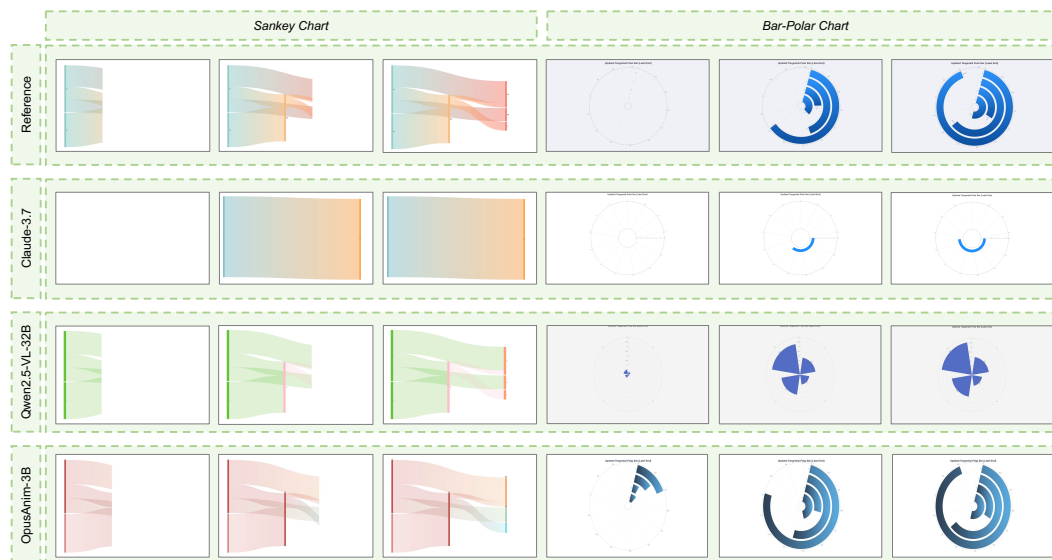


Figure 21: Qualitative Result (1).



Figure 22: Qualitative Result (2)

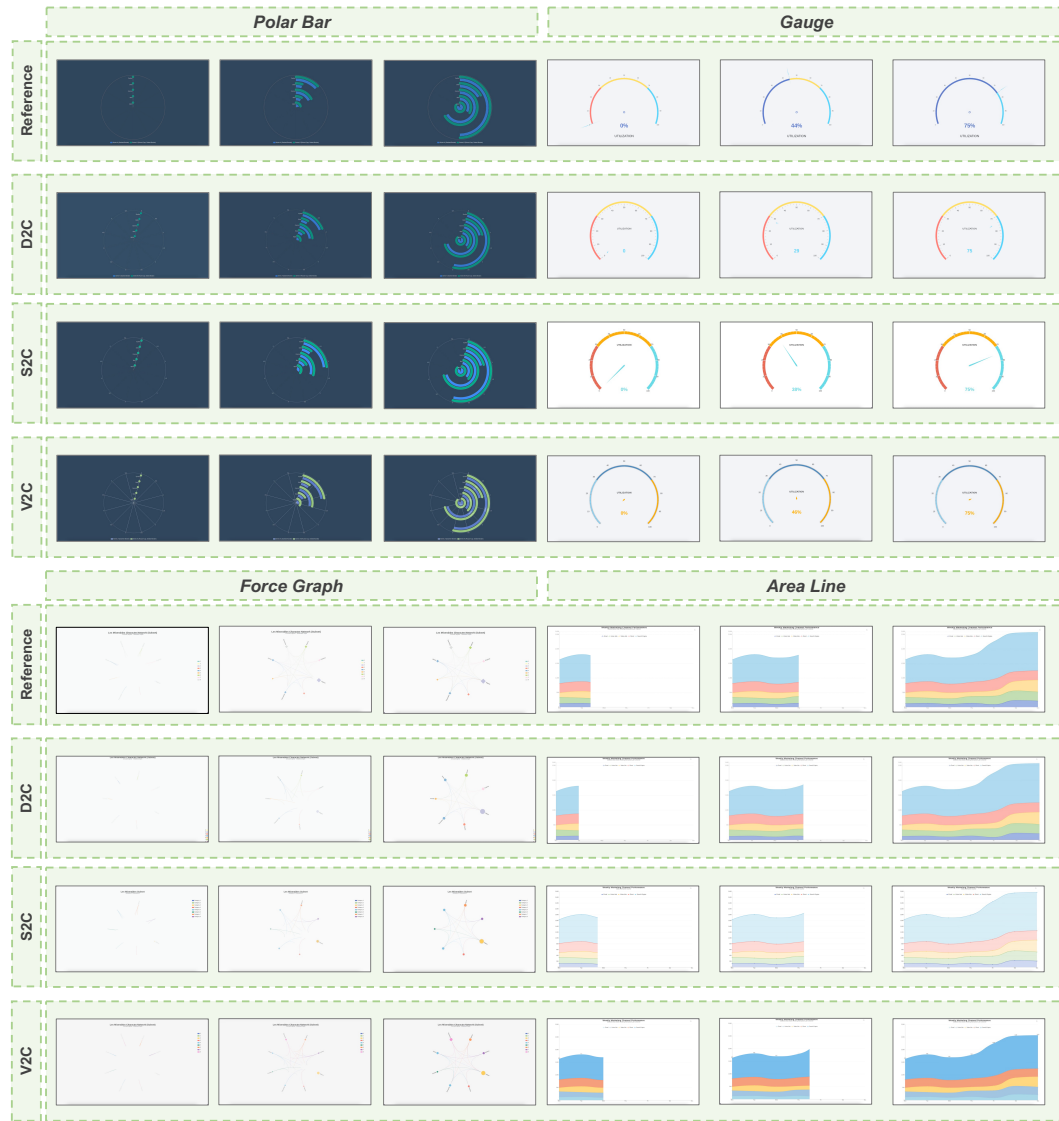


Figure 23: Qualitative Result (3)

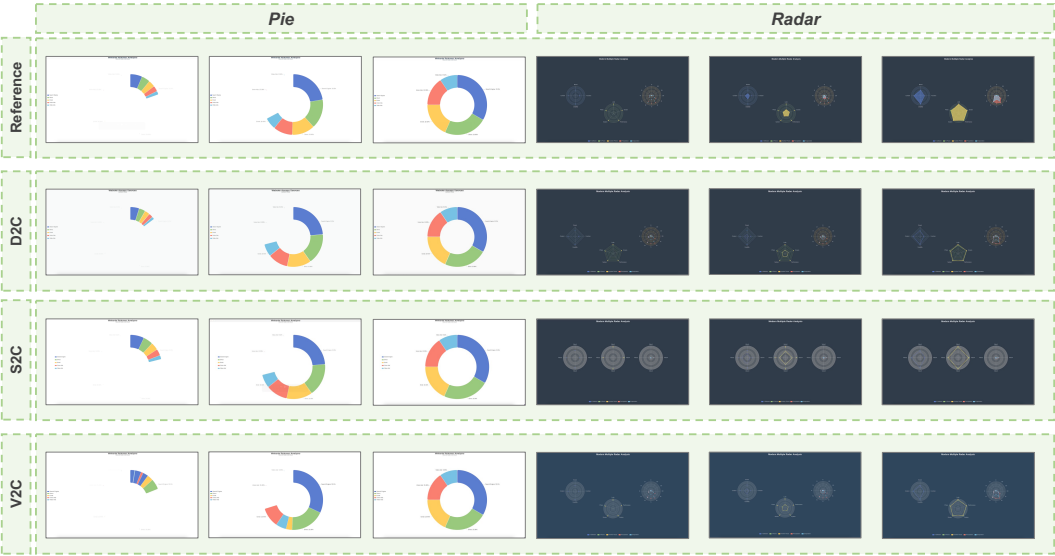


Figure 24: Qualitative Result (4)