

MM-ReCoder: Advancing Chart-to-Code Generation with Reinforcement Learning and Self-Correction

Supplementary Material

We elaborate on the details of our rewards and MM-ReCoder prompts in Section A. Section B shows our RL training curves. In Section C, we conduct ablation studies on the RL training strategy, cold start, and the model used in the model-based reward. We further compare MM-ReCoder with a few baselines through human evaluation in Section D. Finally, qualitative results are in Section E.

A. Implementation Details

In this section, we first provide a detailed description of the prompt used for the model-based reward. Then we elaborate on the prompt we use to guide MM-ReCoder in self-correction.

A.1. Reward Implementation

In the rule-based reward, we evaluate text score, color score, chart type score, and layout score of the generated chart. The implementation of all these scores is borrowed from ChartMimic [39]. Please refer to their paper and code for implementation.

To measure the model-based reward, we use vLLM to launch a Qwen2.5-VL-72B server and set the sampling temperature to be 0.3. If the generated code encounters runtime error, its model-based reward is 0. Otherwise, we use the following prompt to ask Qwen2.5-VL-72B to score the generated chart:

<ground truth image>

<generated image>

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:

—
Detailed Assessment: \${your assessment for each point}

—
Scores:

- **Chart Types:** \${your score out of 20}

- **Layout:** \${your score out of 10}

- **Text Content:** \${your score out of 20}

- **Data:** \${your score out of 20}

- **Style:** \${your score out of 20}

- **Clarity:** \${your score out of 10}

Total Score: \${your total score out of 100}

—
Please use the above format to ensure the evaluation is clear and comprehensive.

A.2. Model Prompt of MM-ReCoder

In the first turn of the conversation, we give MM-ReCoder a chart image and ask it to reproduce it using Python. We also include an instruction to follow the template requirement of our format reward described in Section 3.4.1. Specifically, our prompt is:

<image>

You are an expert developer specializing in writing Python matplotlib code based on a given picture. I need your help to generate the Python code that can reproduce the picture based on the picture I provided.

You **FIRST** think about the reasoning process as an internal monologue and then provide the final code. The reasoning process **MUST BE** enclosed within <think></think> tags. The final code **MUST BE** put in

```
```python
Your code
```
at the end.
```

Notice that the specific task prompt, which is the black text in the prompt above, may differ from dataset to dataset. The one we show here is an example from Chart2Code-160k [47]. During training and evaluation, we use the task prompt provided by each dataset.

After the model outputs a first-turn response, we extract and execute the code inside the ````python...```` block. If the execution encounters a runtime error, we get the error information from the Python interpreter. Otherwise, we derive an image rendered from the code.

In the second turn, if the feedback is a runtime error, we instruct the model to revise its code as the following:

<error information>

Your code encountered a runtime error. Please continue modifying the code.

You **FIRST** think about the reasoning process as an internal monologue and then provide the final code. The reasoning process **MUST BE** enclosed within <think></think> tags. The final code **MUST BE** put in

```
```python
Your code
```
at the end.
```

If the chart is successfully rendered, we use the following prompt:

<image>

Here is the image rendered by your code. Please continue modifying the code so that the reproduced chart looks more like the given image.

You **FIRST** think about the reasoning process as an internal monologue and then provide the final code.

The reasoning process **MUST BE** enclosed within <think></think> tags. The final code **MUST BE** put in

```
```python
Your code
```
at the end.
```

When we employ MM-ReCoder for more than two turns, we repeat this process and the conversations in the previous turns are all kept as contexts.

B. RL Training Curves

In Figure A1, we show the training reward curves and the length of thinking trace/code of MM-ReCoder. We also evaluate the model every 40 steps on ChartMimic during training.

We find that the training rewards reach a plateau at the end of the first RL stage — shared first turn. But they begin to rise again in the second stage, full-trajectory optimization. On the contrary, the evaluation rewards on ChartMimic keep rising during training. However, the evaluation rewards can only hit 85% while the training rewards reach 95%. This shows that there is a distribution shift between the training data and the evaluation benchmarks.

One interesting observation is that, during the shared-first-turn stage, the length of the model’s thinking trace first grows from ~ 200 tokens to ~ 600 tokens, and then gradually drops below 100 tokens. During the first half of the shared-first-turn stage, we find the model learns to propose a long item list of code revision. At the 500th step, the model includes roughly 30 revision items in the thinking trace, leading to an average length of 600 tokens. After this point, the model gradually discards incorrect revision items and finally converges to around four items per chart.

C. Ablation Studies

In this section, we conduct ablation studies on the self-correction RL training strategy, cold start, and the model used in the model-based reward.

C.1. RL Training Strategy

Hyperparameters in full-trajectory optimization. In our full-trajectory optimization method, we use a hyperparameter γ to balance the rewards of the two turns and a hyperparameter η to encourage the model for successful self-correction. In Section 4.4, we claim that $\gamma = 0.5$ and $\eta = 0.1$ work best when only using the full-trajectory optimization. We perform ablation studies on this design choice in Table A1.

Firstly, when the reward of the first turn is not utilized ($\gamma = 0$), the model always lowers the quality of its first-

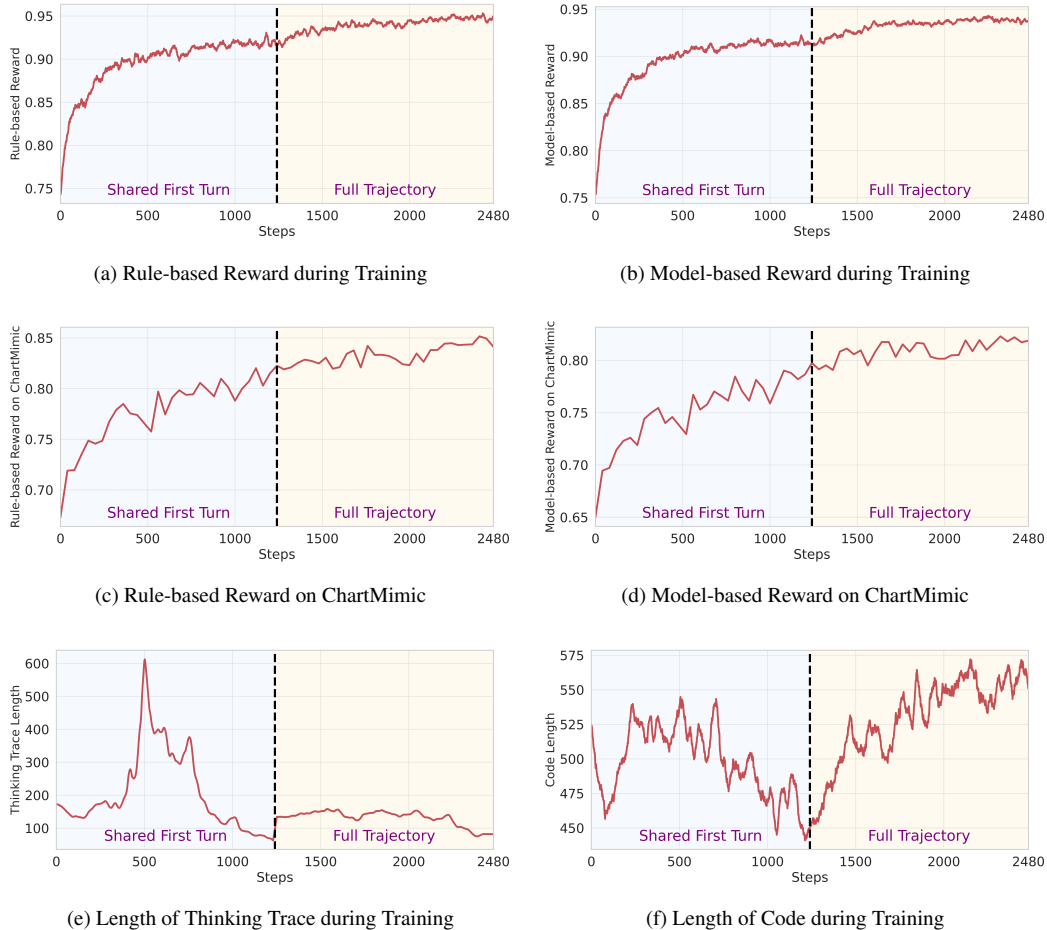


Figure A1. Training curves of MM-ReCoder. Curves on the training set are smoothed with a window size of 20 steps. We evaluate the model on ChartMimic every 40 steps. Note that the value of the rule-based reward can be slightly different from the reported model performance on low-level score because the latter is evaluated under the official evaluation codebase of ChartMimic.

| Strategy | γ | η | First turn | | Second turn | | Avg. low-level improvement on rendered charts | Percentage of improved samples | Percentage of degraded samples | Percentage of repeated code |
|-----------------|----------|--------|------------|-----------|-------------|-----------|---|--------------------------------|--------------------------------|-----------------------------|
| | | | Exec.Rate | Low-Level | Exec.Rate | Low-Level | | | | |
| Full-trajectory | 0 | 0 | 91.8 | 81.8 | 95.2 | 83.9 | 0.21 | 3.4 | 2.6 | 46.9 |
| Full-trajectory | 0 | 0.1 | 64.5 | 66.7 | 96.0 | 84.3 | 10.11 | 87.5 | 2.3 | 0.3 |
| | 0 | 0.3 | 79.0 | 73.7 | 94.5 | 83.2 | 4.57 | 77.1 | 8.3 | 0.0 |
| | 0 | 0.5 | 71.8 | 72.5 | 96.2 | 77.7 | 3.13 | 77.7 | 10.2 | 0.0 |
| Full-trajectory | 0.5 | 0.1 | 96.5 | 85.0 | 97.5 | 85.5 | -0.03 | 2.2 | 2.2 | 55.6 |
| | 0.8 | 0.1 | 95.2 | 83.7 | 96.3 | 84.9 | 0.01 | 3.2 | 3.9 | 45.9 |
| | 1 | 0.1 | 95.5 | 84.7 | 96.2 | 84.2 | 0.00 | 1.4 | 1.4 | 5.1 |

Table A1. Ablation on the hyperparameters in full-trajectory optimization. All experiments are conducted **without model-based reward**.

round output to hack the self-correction bonus η . And higher η leads to worse second-turn performance. Secondly, when $\eta = 0.1$, the existence of γ can prevent the model from lowering the first-turn quality and achieve better performance. However, it also hinders the model from learning self-correction, indicated by the nearly-zero low-level score improvement on rendered charts. A γ of 0.5 shows the best overall performance across all the choices we have tried.

RL strategy combinations. While our RL strategy in MM-

ReCoder is shared-first-turn optimization followed by full-trajectory optimization, we try out several other possible combinations of the RL strategies in Table A2.

As our method is composed of two stages, which trains the model for two epochs on the training set in total, we first try employing the full-trajectory and shared-first-turn optimizations for two epochs for fair comparison. For the full-trajectory optimization, however, training with it for two epochs causes over-fitting, achieving lower performance

| Strategy | γ η | | First turn | | Second turn | | Avg. low-level improvement on rendered charts | Percentage of improved samples | Percentage of degraded samples | Percentage of repeated code |
|-------------------------------------|-----------------|-----------|------------|-----------|-------------|-------------|---|--------------------------------|--------------------------------|-----------------------------|
| | Exec.Rate | Low-Level | Exec.Rate | Low-Level | | | | | | |
| Full-trajectory | 0 | 0 | 91.8 | 81.8 | 95.2 | 83.9 | 0.21 | 3.4 | 2.6 | 46.9 |
| | 0.5 | 0.1 | 96.5 | 85.0 | 97.5 | 85.5 | -0.03 | 2.2 | 2.2 | 55.6 |
| Full-trajectory for 2 epochs | 0 | 0 | 92.3 | 79.0 | 97.0 | 81.8 | 0.16 | 4.5 | 2.5 | 54.0 |
| | 0.5 | 0.1 | 94.3 | 82.2 | 94.2 | 82.3 | 0.07 | 2.7 | 1.6 | 87.7 |
| Shared-first-turn | - | - | 91.7 | 79.8 | 95.8 | 82.6 | 0.72 | 14.4 | 8.4 | 16.8 |
| Shared-first-turn for 2 epochs | - | - | 92.3 | 82.3 | 97.2 | 84.8 | 0.35 | 10.5 | 4.7 | 12.3 |
| Shared-first-turn + Full-trajectory | 0 | 0 | 94.3 | 83.7 | 97.7 | 86.0 | 0.55 | 12.1 | 8.2 | 21.6 |
| Full-trajectory + Shared-first-turn | 0 | 0 | 93.3 | 83.6 | 97.2 | 85.0 | 0.02 | 2.9 | 2.7 | 46.7 |
| Single-turn + Full-trajectory | 0 | 0 | 94.7 | 84.5 | 96.5 | 85.2 | 0.03 | 2.7 | 4.0 | 58.4 |

Table A2. Ablation on the RL stage combinations. All experiments are conducted **without model-based reward**.

| Cold Start | Performance on ChartMimic after RL | | | |
|--|------------------------------------|-----------------------|--|-----------------------------|
| | First turn low-level | Second turn low-level | Average improvement on rendered charts | Percentage of repeated code |
| None | 83.0 | 84.9 | 0.53 | 36.3 |
| Single-turn cold start | 82.3 | 84.8 | 0.02 | 13.1 |
| Single-turn cold start + multi-turn cold start | 83.7 | 86.0 | 0.55 | 21.6 |

Table A3. Ablation studies on cold start. We train the base model with different cold start recipes and the same RL strategy (full-trajectory + shared-first-turn). We evaluate their performance on ChartMimic after RL. Experiments are conducted **without model-based reward**.

than one epoch. On the other hand, shared-first-turn for two epochs can further boost the model performance compared to one epoch, but underperforms the other strategies.

We explore strategy combinations besides the one we use in our main method. For example, we can switch the order of the two stages, performing full-trajectory optimization followed by shared-first-turn. Moreover, we can also train a model under the single-turn RL setting first and then perform full-trajectory optimization to enable self-correction. However, as shows in Table A2, these two strategies cannot elicit the self-correction capability of the model and result in suboptimal performance. They repeat their first-turn code in the second turn with a frequency of nearly 50%. We attribute this to their weak self-correction ability after full-trajectory optimization and single-turn RL. In this case, the model’s output entropy is significantly reduced during the first-stage RL, making it unable to explore self-correction effectively in the second stage.

C.2. Cold Start

To investigate the effect of cold start, we perform our two-stage RL on two checkpoints: (a) Qwen2.5-VL-7B without any cold start and (b) the checkpoint that has only undergone the single-turn cold start. The results are in Table A3 and the RL stage is performed without model-based reward.

We find that our self-correction RL training strategy can enable Qwen2.5-VL-7B to self-correct without any cold start, indicating that the teacher model used to generate the code start data is not the key to elicit self-correction.

However, the performance without cold start is around 1% lower than that of the cold-started model. While cold start on the single-turn data, Chart2Code-160k, can improve the base model’s coding capability as shown in Table 5, directly applying RL on this checkpoint does not realize self-correction because of its weak capability in multi-turn conversation. After the multi-turn cold start stage, the model is able to benefit from both the cold start and our multi-turn self-correction RL method.

We also look into the qualitative results to investigate the effect of cold start. We notice that the model without cold start generates very short and abstract thinking traces, while the one with cold start learns to describe the image differences in detail and outlines an action list before coding, which is important to boost the model’s performance.

C.3. Model-based Reward

In the RL reward of MM-ReCoder, we use Qwen2.5-VL-72B judge as a model-based reward to score the generated charts. While Qwen2.5-VL-72B is larger than our base model, Qwen2.5-VL-7B, we show in Table A4 that using Qwen2.5-VL-7B itself as the reward model can also guide the learning of MM-ReCoder.

In this experiment, we train MM-ReCoder in the single-turn setting without self-correction. We find that using Qwen2.5-VL-7B as the reward model can improve the high-level score by 4% compared to training without model-based reward. However, the model’s low-level score is degraded by 3.8%. We hypothesize that the occasional in-

| Reward Model | ChartMimic | | |
|----------------|------------|-----------|------------|
| | Exec.Rate | Low-level | High-level |
| None | 95.0 | 84.8 | 78.6 |
| Qwen2.5-VL-7B | 95.3 | 81.0 | 82.6 |
| Qwen2.5-VL-72B | 95.0 | 84.3 | 83.7 |

Table A4. Ablation on the reward model **under the single-turn RL setting**. Qwen2.5-VL-7B as the reward model is able to improve the high-level score on ChartMimic, but replacing Qwen2.5-VL-7B with Qwen2.5-VL-72B can further boost the model.

accuracies of the Qwen2.5-VL-7B judge can mislead the learning of the low-level score. Moreover, replacing the 7B reward model with the 72B version can better improve the model’s capability, so we use Qwen2.5-VL-72B in our experiments to optimize the performance of MM-ReCoder.

D. Human Evaluation

D.1. A/B Testing

We conduct human evaluation on the charts generated by MM-ReCoder and the baseline models. We compare two models in each study by evaluating human annotators’ preference on their generated charts.

For a pair of models, we randomly select 100 test cases from ChartMimic and form a triplet consisting of the two model-generated charts and the ground truth image. For each triplet, the model-generated charts are anonymized and presented in a random order. Annotators are asked to choose which chart aligns better with the ground truth. If the two charts are roughly equal, annotators can select “tie.” In each study, we have three annotators reviewing all the selected test cases. For each test case, if at least two annotators prefer a particular chart, it is counted as a win for that chart; otherwise, it is counted as a tie. Finally, we report the win/tie/loss rates of our model.

We report the results in Table A5. Our model wins 37% of the time and loses 20% of the time against ChartCoder, and wins 40% and loses 23% against Qwen2.5-VL-72B. However, our model only wins 19% but loses 48% against Qwen3-VL-235B-A22B. The human evaluation results demonstrate that MM-ReCoder outperforms ChartCoder and Qwen2.5-VL-72B, but underperforms Qwen3-VL-235B-A22B, which is in line with the high-level score evaluation in Table 1.

D.2. Score Improvement Reflects Self-correction

In our experiments, we use the improvement in the evaluation scores to indicate the success of self-correction. To check how faithfully the score improvement can reflect chart improvement, we manually check all ChartMimic test samples with improved scores in the second turn. We find that 76.5% of them exhibit improvements that are easily discernible by human eyes. This shows that the score improve-

| | Win | Tie | Loss |
|------------------------------|-----|-----|------|
| Ours v.s. ChartCoder | 37% | 43% | 20% |
| Ours v.s. Qwen2.5-VL-72B | 40% | 37% | 23% |
| Ours v.s. Qwen3-VL-235B-A22B | 19% | 33% | 48% |

Table A5. Human evaluation between MM-ReCoder and baselines on ChartMimic. The results are in line with the high-level score.

| Model | Diagnosis error | Coding error | Regression |
|--------------------|-----------------|--------------|------------|
| MM-ReCoder | 8 (30.8%) | 13 (50.0%) | 5 (19.2%) |
| Qwen3-VL-8B | 22 (37.9%) | 28 (48.3%) | 8 (13.8%) |
| Qwen3-VL-235B-A22B | 15 (20.5%) | 33 (45.2%) | 25 (34.3%) |

Table A6. Frequencies of self-correction failure modes.

ment can indicate chart improvement in most cases.

E. Qualitative Results

In this section, we show examples reflecting the self-correction capability of MM-ReCoder. We also qualitatively compare MM-ReCoder with baseline models.

E.1. Self-correction Examples

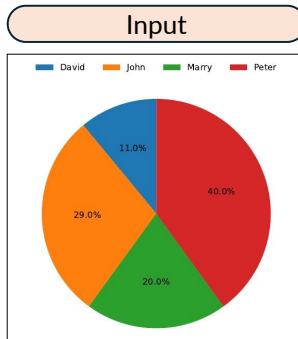
We showcase qualitative results of successful self-correction in Figures A2 to A8. MM-ReCoder is able to self-correct the labels, axis ranges, hatches, category orders, and data points in the charts and handle runtime errors. However, although MM-ReCoder tends to propose multiple revision items in its thinking trace, we notice that not all the items are reflected in the revised code. This demonstrates a limitation of MM-ReCoder that the thinking trace and generated code are not always perfectly aligned.

E.2. Comparison with Baselines

We qualitatively compare MM-ReCoder with baseline models in Figure A9. While all the models can faithfully reproduce the chart types, MM-ReCoder is better at using the correct colors, texts, chart styles, and data points. We further demonstrate failure cases in Figure A10 of MM-ReCoder and baselines. We find that the models tend to fail when the given chart type is rare, whose code functions are possibly not covered by the training data.

E.3. Analysis of Self-correction Failures

To summarize typical failure modes in self-correction, we investigate the regressed samples in the second turn. We discover three major failure modes: (a) **diagnosis error** - the thinking trace presents incorrect chart discrepancies or revision items; (b) **coding error** - the thinking trace is correct, but the code fails to fix the issue; (c) **regression** - the identified issues are fixed, but the existing correct functionalities (e.g., text, color) are broken. We count the failure mode frequencies of MM-ReCoder and the baselines in Table A6. Compared to Qwen3-VL-8B, MM-ReCoder fails in diagnosis less frequently. Qwen3-VL-235B-A22B has fewer from diagnosis errors but more regression errors.



First Turn

<think> Looking at the pie chart, I need to:

1. Extract the approximate values based on percentages shown
2. Create a pie chart with appropriate colors
3. Add proper labels and percentages
4. Position the legend appropriately
5. Use appropriate formatting for percentages

</think>

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

### Data to plot
labels = ['David', 'John', 'Marry', 'Peter']
sizes = [11, 29, 20, 40] ### Approximate percentages based on visual estimation
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']

### Create figure and axis
fig, ax = plt.subplots(figsize=(5, 5))

### Create pie chart
wedges, texts, autotexts = ax.pie(sizes, labels=labels, colors=colors,
autotext='%1.1f%%', startangle=90, textprops={'weight': 'bold'},
pctdistance=0.85)

### Add title if needed (optional, but improves appearance)
### ax.set_title('', fontsize=12, fontweight='bold')

### Display percentages on wedges
autotexts[0].set_text('11.0%')
autotexts[1].set_text('29.0%')
autotexts[2].set_text('20.0%')
autotexts[3].set_text('40.0%')

### Add legend in top-left corner with proper positioning
ax.legend(labels, loc='upper left', bbox_to_anchor=(-0.1, 1.1), fontsize=11,
framealpha=True, edgecolor='black')

### Display chart
plt.tight_layout()
plt.show()
```

Second Turn

<think> The labels are appearing incorrectly and overlapping improperly. Also, the title/spacing needs adjustment better. The legend needs proper alignment without labels interfering. Also, the percentages need proper placement without labels interfering. Also, the x-axis labels need proper alignment without being displayed incorrectly.

I need to:

1. Remove proper labels from wedges and only show percentages accurately
2. Properly position the legend in top-left corner without causing issues
3. Remove the title properly or adjust better
4. Make sure proper spacing without labels on wedges interfering

</think>

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

### Data to plot
labels = ['David', 'John', 'Marry', 'Peter']
sizes = [11, 29, 20, 40] ### Approximate percentages based on visual estimation
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']

### Create figure and axis
fig, ax = plt.subplots(figsize=(5, 5))

### Create pie chart
wedges, texts, autotexts = ax.pie(sizes, labels=None, colors=colors,
autotext='%1.1f%%', startangle=90, textprops={'weight': 'bold'})

### Add percentages
autotexts[0].set_text('11.0%')
autotexts[1].set_text('29.0%')
autotexts[2].set_text('20.0%')
autotexts[3].set_text('40.0%')

### Add custom legend in top-left corner with proper positioning
ax.legend(labels, loc='upper left', bbox_to_anchor=(0, 1.1), fontsize=11,
framealpha=True, edgecolor='black')

### Display chart
plt.tight_layout()
plt.show()
```

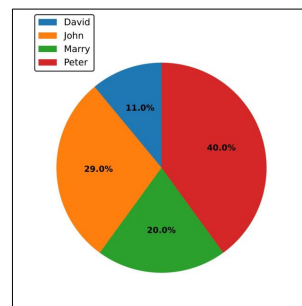
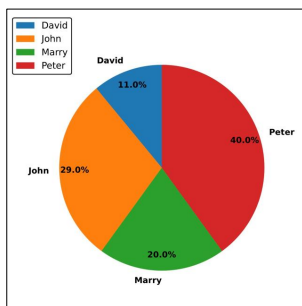
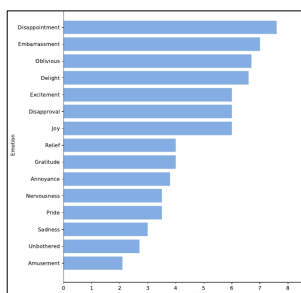


Figure A2. Qualitative result of MM-ReCoder for self-correction. The model removes the text labels from the pie chart in the second turn.

Input



First Turn

<think> Looking at the chart, I need to create a horizontal bar plot with appropriate spacing and proper alignment.

Steps to reproduce:

1. Import matplotlib.pyplot
2. Define the emotion labels (y-axis ticks)
3. Define corresponding frequency values (approximate based on visual estimation)
4. Create the horizontal bar plot
5. Set appropriate axis ticks and labels
6. Display the plot properly

</think>

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

#### Data for the chart (estimated frequencies based on visual appearance)
emotions = [
    'Amusement', 'Unbothered', 'Sadness', 'Pride', 'Nervousness',
    'Annoyance', 'Gratitude', 'Relief', 'Joy', 'Disapproval',
    'Excitement', 'Delight', 'Oblivious', 'Embarrassment', 'Disappointment'
]
frequencies = [
    2.2, 2.6, 3.0, 3.4, 3.5, 3.8, 4.0, 4.0, 6.0, 6.1, 6.1, 6.6, 6.7, 7.0, 7.5
]

#### Create the figure and axis
fig, ax = plt.subplots(figsize=(8, 8))

#### Create horizontal bar chart
ax.barh(range(len(emotions)), frequencies, height=0.6, color='#6495ED') ####
Using a similar blue color

#### Set labels and title
ax.set_xlabel('Frequency (%)', fontsize=12, fontweight='bold')
ax.set_ylabel('Emotion!', fontsize=12, fontweight='bold')
ax.set_title('', fontsize=16)

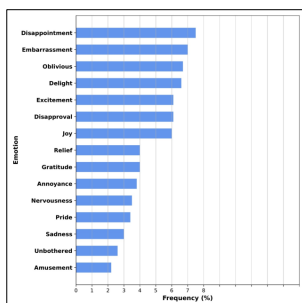
#### Set ticks and tickLabels
ax.set_yticks(range(len(emotions)))
ax.set_yticklabels(emotions, rotation=0, ha='right', fontsize=11,
fontweight='bold')

#### Set x-axis ticks appropriately
ax.set_xticks([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
ax.set_xticklabels(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14',
'', '', ''], rotation=0, ha='center')

#### Display grid lines more accurately
ax.grid(axis='x', which='major', linestyle='-', linewidth=0.5)

#### Adjust layout
plt.tight_layout()

#### Display plot
plt.show()
```



Second Turn

<think> The x-axis ticks appear wrong and there are grid lines that shouldn't be there. Also, the spacing needs adjustment better without causing issues. The y-axis labels need proper alignment without being skewed incorrectly.

Also, the frequencies x-axis ticks need proper spacing without being skewed incorrectly. The grid lines need proper control without being visible as lines that interfere with proper appearance.

Also, the spacing needs adjustment better without causing issues properly.

</think>

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

#### Data for the chart (estimated frequencies based on visual appearance)
emotions = [
    'Amusement', 'Unbothered', 'Sadness', 'Pride', 'Nervousness',
    'Annoyance', 'Gratitude', 'Relief', 'Joy', 'Disapproval',
    'Excitement', 'Delight', 'Oblivious', 'Embarrassment', 'Disappointment'
]
frequencies = [
    2.2, 2.6, 3.0, 3.4, 3.5, 3.8, 4.0, 4.0, 6.0, 6.1, 6.1, 6.6, 6.7, 7.0, 7.5
]

#### Create the figure and axis
fig, ax = plt.subplots(figsize=(8, 8))

#### Create horizontal bar chart
ax.barh(range(len(emotions)), frequencies, height=0.6, color='#6495ED') ####
Using a similar blue color

#### Set labels and title
ax.set_xlabel('Frequency (%)', fontsize=11, fontweight='bold')
ax.set_ylabel('Emotion!', fontsize=12, fontweight='bold')
ax.set_title('', fontsize=14)

#### Set ticks and tickLabels
ax.set_yticks(range(len(emotions)))
ax.set_yticklabels(emotions, rotation=0, ha='right', fontsize=11,
fontweight='bold')

#### Set x-axis ticks appropriately
ax.set_xticks([0, 1, 2, 3, 4, 5, 6, 7, 8])
ax.set_xticklabels(['0', '1', '2', '3', '4', '5', '6', '7', '8'], rotation=0,
ha='center')

#### Remove y-axis ticks properly without grid issues
#### Set appropriate ticks without grid lines interfering properly

#### Display plot properly without grid lines interfering badly
plt.tight_layout()

#### Display plot
plt.show()
```

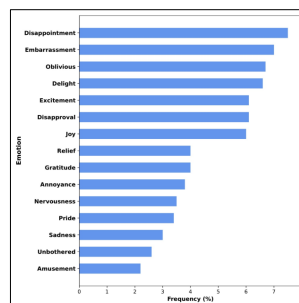
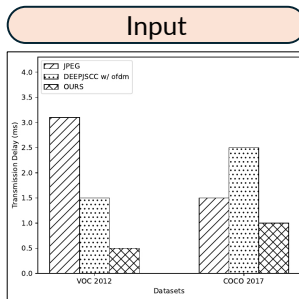


Figure A3. Qualitative result of MM-ReCoder for self-correction. The model rescales the x-axis range and removes the grid lines in the second turn.



First Turn

<think> Looking at the chart, I need to create proper bar plots with appropriate spacing and proper legend formatting. Since I don't have actual data, I need to estimate reasonable values based on the visual appearance.

Steps to reproduce:

1. Import matplotlib.pyplot
2. Define appropriate bar positions and widths
3. Use appropriate hatch patterns for bars
4. Create proper legend with correct formatting
5. Set appropriate axis ticks and labels

</think>

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

#### Data for the chart (estimated values based on visual appearance)
labels = ['VOC 2012', 'COCO 2017']
jpeg_values = [3.1, 1.5]
deepjssc_values = [1.5, 2.5]
ours_values = [0.5, 1.0]

#### Create figure and axis
fig, ax = plt.subplots(figsize=(6, 5))

#### Set width for bars
width = 0.25

#### Create bars
rect1 = ax.bar(np.arange(2) + 0*width, jpeg_values, width, label='JPEG',
hatch='/', edgecolor='black')
rect2 = ax.bar(np.arange(2) + 1*width, deepjssc_values, width,
label='DEEPS3CC w/ ofdm', hatch='.', edgecolor='black', linestyle=':')
rect3 = ax.bar(np.arange(2) + 2*width, ours_values, width, label='OURS',
hatch='x', edgecolor='black')

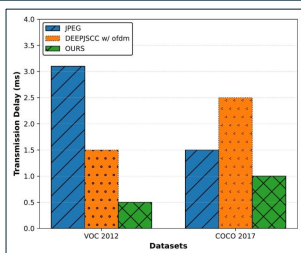
#### Add labels and title
ax.set_title('')
ax.set_xlabel('Datasets', fontsize=11, fontweight='bold')
ax.set_ylabel('Transmission Delay (ms)', fontsize=11, fontweight='bold')
ax.set_xticks(np.arange(2) + width)
ax.set_xticklabels(labels, rotation=0, ha='center')

#### Add legend in top-left corner with proper positioning
ax.legend(loc='upper left', bbox_to_anchor=(0, 1), fontsize=10,
framealpha=True, edgecolor='black')

#### Set y-axis ticks
ax.set_yticks([0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0])

#### Display grid for better appearance
ax.grid(axis='y', linestyle=':', linewidth=0.5)

#### Show plot
plt.tight_layout()
plt.show()
```



Second Turn

<think> The rendered chart has incorrect legend content and colors, and the bars appear with wrong styling. Also, the proper spacing and alignment need adjustment better.

Also, the legend needs proper alignment without conflicting with bars incorrectly. The hatching needs adjustment more accurately without colors.

Also, the y-axis ticks need proper control without grid lines interfering incorrectly.

Also, the proper spacing needs adjustment better without overlap issues.

</think>

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

#### Data for the chart (estimated values based on visual appearance)
labels = ['VOC 2012', 'COCO 2017']
jpeg_values = [3.1, 1.5]
deepjssc_values = [1.5, 2.5]
ours_values = [0.5, 1.0]

#### Create figure and axis
fig, ax = plt.subplots(figsize=(6, 5))

#### Set width for bars
width = 0.25

#### Create bars
rect1 = ax.bar(np.arange(2) + 0*width, jpeg_values, width, label='JPEG',
hatch='/', edgecolor='black', color='FFFFFF')
rect2 = ax.bar(np.arange(2) + 1*width, deepjssc_values, width,
label='DEEPS3CC w/ ofdm', hatch='.', edgecolor='black', linestyle=':',
color='FFFFFF')
rect3 = ax.bar(np.arange(2) + 2*width, ours_values, width, label='OURS',
hatch='x', edgecolor='black', color='FFFFFF')

#### Add labels and title
ax.set_title('')
ax.set_xlabel('Datasets', fontsize=11, fontweight='bold')
ax.set_ylabel('Transmission Delay (ms)', fontsize=11, fontweight='bold')
ax.set_xticks(np.arange(2) + width)
ax.set_xticklabels(labels, rotation=0, ha='center')

#### Add proper legend in top-left corner with correct formatting
ax.legend(loc='upper left', bbox_to_anchor=(0, 1), fontsize=10,
framealpha=True, edgecolor='black')

#### Set y-axis ticks
ax.set_yticks([0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0])

#### Display proper legend without conflicting issues
#### Remove grid or proper control without issues

#### Show plot
plt.tight_layout()
plt.show()
```

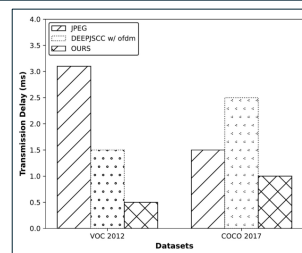
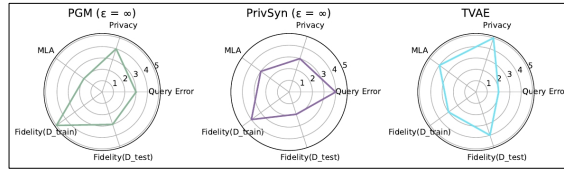


Figure A4. Qualitative result of MM-ReCoder for self-correction. The model removes the hatch color in the second turn.

Input



First Turn

<think> Looking at the radar charts, I need to create proper polar plots with appropriate data points. Since I don't have actual data, I need to estimate reasonable values based on the visual appearance. Each chart has 5 categories with values ranging from ~1 to ~5. I'll create appropriate angles for each category and plot lines with appropriate colors.

```

import matplotlib.pyplot as plt
import numpy as np

### Create figure with appropriate size
fig, axes = plt.subplots(1, 3, figsize=(15, 5), subplot_kw=dict(polar=True))

### Data for PGM
labels = ['Query Error', 'Privacy', 'MLA', 'Fidelity(D_train)', 'Fidelity(D_test)']
values_pgm = [3.5, 4.5, 2.5, 5.0, 4.8]

### Data for PrivSyn
labels2 = ['Query Error', 'Privacy', 'MLA', 'Fidelity(D_train)', 'Fidelity(D_test)']
values_privsyn = [4.2, 3.0, 2.8, 4.5, 3.8]

### Data for TVAE
labels3 = ['Query Error', 'Privacy', 'MLA', 'Fidelity(D_train)', 'Fidelity(D_test)']
values_tvae = [2.8, 5.0, 3.7, 4.2, 4.0]

### First subplot - PGM
ax1 = axes[0]
angles = np.linspace(0, 2 * np.pi, len(labels), endpoint=False).tolist()
angles += angles[:1]

### Plot data
values_pgm_extended = values_pgm + values_pgm[:1]
ax1.plot(angles, values_pgm_extended, color='#8FBC8F', linewidth=1.5)
ax1.fill(angles, values_pgm_extended, color='#8FBC8F', alpha=0.25)

ax1.set_thetagrids(np.degrees(angles[:-1]), labels, fontsize=11, fontweight='bold')
ax1.set_title('PGM (ε = ∞)', size=14, fontweight='bold', pad=50)
ax1.set_ylim(0, 5)
ax1.set_yticks([1, 2, 3, 4, 5])
ax1.set_yticklabels(['1', '2', '3', '4', '5'], fontsize=11)

### Second subplot - PrivSyn
ax2 = axes[1]
values_privsyn_extended = values_privsyn + values_privsyn[:1]
ax2.plot(angles, values_privsyn_extended, color='#483D8B', linewidth=1.5)
ax2.fill(angles, values_privsyn_extended, color='#483D8B', alpha=0.25)

ax2.set_thetagrids(np.degrees(angles[:-1]), labels2, fontsize=11, fontweight='bold')
ax2.set_title('PrivSyn (ε = ∞)', size=12, fontweight='bold')
ax2.set_ylim(0, 5)
ax2.set_yticks([1, 2, 3, 4, 5])
ax2.set_yticklabels(['1', '2', '3', '4', '5'], fontsize=11)

### Third subplot - TVAE
ax3 = axes[2]
values_tvae_extended = values_tvae + values_tvae[:1]
ax3.plot(angles, values_tvae_extended, color='#00FFFF', linewidth=1.5)
ax3.fill(angles, values_tvae_extended, color='#00FFFF', alpha=0.25)

ax3.set_thetagrids(np.degrees(angles[:-1]), labels3, fontsize=11, fontweight='bold')
ax3.set_title('TVAE', size=12, fontweight='bold')
ax3.set_ylim(0, 5)
ax3.set_yticks([1, 2, 3, 4, 5])
ax3.set_yticklabels(['1', '2', '3', '4', '5'], fontsize=11)

### Adjust layout
plt.tight_layout()
plt.show()

```

Second Turn

<think> The charts appear too filled and the lines need proper control without filling being dominant. Also, the titles need proper alignment without being too bold or inconsistent. Also, the filled areas need proper control without being too prominent or causing issues with transparency. I need to remove the filled areas properly or adjust them better. Also, the boldness of titles needs adjustment more accurately without causing issues. Also, the radial grid lines need proper control without being too prominent but accurate.

```

import matplotlib.pyplot as plt
import numpy as np

### Create figure with appropriate size
fig, axes = plt.subplots(1, 3, figsize=(15, 5), subplot_kw=dict(polar=True))

### Data for PGM
labels = ['Query Error', 'Privacy', 'MLA', 'Fidelity(D_train)', 'Fidelity(D_test)']
values_pgm = [3.5, 4.5, 2.5, 5.0, 4.8]

### Data for PrivSyn
labels2 = ['Query Error', 'Privacy', 'MLA', 'Fidelity(D_train)', 'Fidelity(D_test)']
values_privsyn = [4.2, 3.0, 2.8, 4.5, 3.8]

### Data for TVAE
labels3 = ['Query Error', 'Privacy', 'MLA', 'Fidelity(D_train)', 'Fidelity(D_test)']
values_tvae = [2.8, 5.0, 3.7, 4.2, 4.0]

### First subplot - PGM
ax1 = axes[0]
angles = np.linspace(0, 2 * np.pi, len(labels), endpoint=False).tolist()
angles += angles[:1]

### Plot data
values_pgm_extended = values_pgm + values_pgm[:1]
ax1.plot(angles, values_pgm_extended, color='#8FBC8F', linewidth=1.5)
ax1.set_thetagrids(np.degrees(angles[:-1]), labels, fontsize=11, fontweight='bold')
ax1.set_title('PGM (ε = ∞)', size=12, fontweight='bold', pad=50)
ax1.set_ylim(0, 5)
ax1.set_yticks([1, 2, 3, 4, 5])
ax1.set_yticklabels(['1', '2', '3', '4', '5'], fontsize=11)

### Second subplot - PrivSyn
ax2 = axes[1]
values_privsyn_extended = values_privsyn + values_privsyn[:1]
ax2.plot(angles, values_privsyn_extended, color='#483D8B', linewidth=1.5)
ax2.set_thetagrids(np.degrees(angles[:-1]), labels2, fontsize=11, fontweight='bold')
ax2.set_title('PrivSyn (ε = ∞)', size=11.5, fontweight='bold')
ax2.set_ylim(0, 5)
ax2.set_yticks([1, 2, 3, 4, 5])
ax2.set_yticklabels(['1', '2', '3', '4', '5'], fontsize=10)

### Third subplot - TVAE
ax3 = axes[2]
values_tvae_extended = values_tvae + values_tvae[:1]
ax3.plot(angles, values_tvae_extended, color='#00FFFF', linewidth=1.5)
ax3.set_thetagrids(np.degrees(angles[:-1]), labels3, fontsize=11, fontweight='bold')
ax3.set_title('TVAE', size=11.5, fontweight='bold')
ax3.set_ylim(0, 5)
ax3.set_yticks([1, 2, 3, 4, 5])
ax3.set_yticklabels(['1', '2', '3', '4', '5'], fontsize=10)

### Remove unnecessary fills property
### Adjust layout
plt.tight_layout()
plt.show()

```

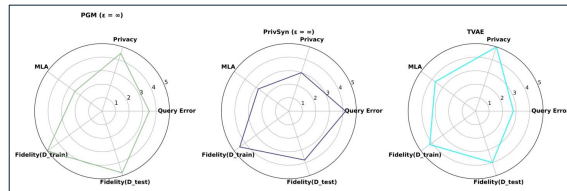
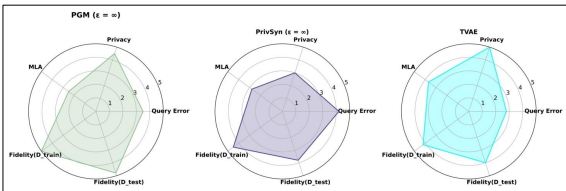
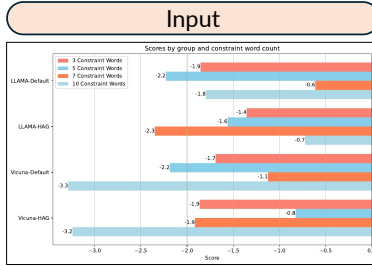


Figure A5. Qualitative result of MM-ReCoder for self-correction. The model removes the colors filled in the radar charts in the second turn.



First Turn

<think> Looking at the chart, I need to create a horizontal bar plot with proper spacing and annotations.
 Steps to reproduce:
 1. Import necessary libraries (matplotlib.pyplot)
 2. Define the categories and corresponding scores
 3. Create appropriate bar widths and positions
 4. Add proper labeling, titles, and ticks
 5. Display annotations on bars
 6. Position bars appropriately with proper spacing
 </think>

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

### Data extraction from the chart (approximate values based on visual inspection)
labels = ['LLAMA-Default', 'LLAMA-HAG', 'Vicuna-Default', 'Vicuna-HAG']
groups = ['3 Constraint Words', '5 Constraint Words', '7 Constraint Words', '10 Constraint Words']

### Scores for each group and constraint word count
scores_3 = [-1.9, -1.4, -1.7, -1.9]
scores_5 = [-2.2, -1.6, -1.7, -1.9]
scores_7 = [-0.6, -0.7, -1.1, -0.8]
scores_10 = [-2.2, -1.8, -2.2, -3.2]

### Create figure and axis
fig, ax = plt.subplots(figsize=(10, 6))

### Set y-axis positions
y_pos = np.arange(4)

### Create horizontal bars
ax.barh(y_pos, scores_3, height=0.2, color='#FAB972', label='3 Constraint Words')
ax.barh(y_pos + 0.15, scores_5, height=0.18, color='#87CEEB', label='5 Constraint Words')
ax.barh(y_pos + 0.3, scores_7, height=0.18, color='#FF7F50', label='7 Constraint Words')
ax.barh(y_pos + 0.45, scores_10, height=0.18, color='#ADD8E6', label='10 Constraint Words')

### Add labels and title
ax.set_title('Scores by group and constraint word count', fontsize=22, fontweight='bold')
ax.set_xlabel('Score', fontsize=12, fontweight='bold')
ax.set_ylabel('', fontsize=12)

### Set ticks and grid
ax.set_xticks([-3.0, -2.5, -2.0, -1.5, -1.0, -0.5, 0.0])
ax.set_yticks(y_pos + 0.25)
ax.set_yticklabels(labels, fontsize=11, weight='bold')

### Add legend in top-left corner
ax.legend(loc='upper left', bbox_to_anchor=(0, 1), fontsize=11, framealpha=True, edgecolor='black')

### Add annotations
for i, (score3, score5, score7, score10) in enumerate(zip(scores_3, scores_5, scores_7, scores_10)):
    ax.text(score3, i, f'{score3:.1f}', va='center', ha='right', fontsize=11, weight='bold')
    ax.text(score5, i + 0.15, f'{score5:.1f}', va='center', ha='right', fontsize=11)
    ax.text(score7, i + 0.3, f'{score7:.1f}', va='center', ha='right', fontsize=11)
    ax.text(score10, i + 0.45, f'{score10:.1f}', va='center', ha='right', fontsize=11)

### Display grid lines
ax.xaxis.grid(True, linestyle='-', linewidth=0.5)
ax.yaxis.grid(False)

### Show plot
plt.tight_layout()
plt.show()
```

Second Turn

<think> The chart is inverted and the y-axis labels are reversed. Also, the legend needs proper alignment without being cut off incorrectly. The groups need proper alignment without being reversed incorrectly. Also, the y-axis labels need proper orientation without being reversed incorrectly. I need to fix the proper alignment and orientation. Also, the legend needs all items properly displayed with correct spacing.
 I need to properly control the orientation and spacing correctly without inverting the axes incorrectly.
 </think>

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

### Data extraction from the chart (approximate values based on visual inspection)
labels = ['LLAMA-Default', 'LLAMA-HAG', 'Vicuna-Default', 'Vicuna-HAG']
groups = ['3 Constraint Words', '5 Constraint Words', '7 Constraint Words', '10 Constraint Words']

### Scores for each group and constraint word count
scores_3 = [-1.9, -1.4, -1.7, -1.9]
scores_5 = [-2.2, -1.6, -1.7, -1.9]
scores_7 = [-0.6, -0.7, -1.1, -0.8]
scores_10 = [-2.2, -1.8, -2.2, -3.2]

### Create figure and axis
fig, ax = plt.subplots(figsize=(10, 6))

### Set y-axis positions
y_pos = np.arange(4)

### Create horizontal bars
ax.barh(y_pos, scores_3, height=0.18, color='#FAB972', label='3 Constraint Words')
ax.barh(y_pos + 0.15, scores_5, height=0.18, color='#87CEEB', label='5 Constraint Words')
ax.barh(y_pos + 0.3, scores_7, height=0.18, color='#FF7F50', label='7 Constraint Words')
ax.barh(y_pos + 0.45, scores_10, height=0.18, color='#ADD8E6', label='10 Constraint Words')

### Add labels and title
ax.set_title('Scores by group and constraint word count', fontsize=22, fontweight='bold')
ax.set_xlabel('Score', fontsize=12, fontweight='bold')
ax.set_ylabel('', fontsize=12)

### Set ticks and grid
ax.set_xticks([-3.0, -2.5, -2.0, -1.5, -1.0, -0.5, 0.0])
ax.set_yticks(y_pos + 0.25)
ax.set_yticklabels(labels, fontsize=11, weight='bold')

### Add legend in top-left corner with proper positioning
ax.legend(loc='upper left', bbox_to_anchor=(0, 1), fontsize=11, framealpha=True, edgecolor='black')

### Add annotations
for i, (score3, score5, score7, score10) in enumerate(zip(scores_3, scores_5, scores_7, scores_10)):
    ax.text(score3, i, f'{score3:.1f}', va='center', ha='right', fontsize=11, weight='bold')
    ax.text(score5, i + 0.15, f'{score5:.1f}', va='center', ha='right', fontsize=11)
    ax.text(score7, i + 0.3, f'{score7:.1f}', va='center', ha='right', fontsize=11)
    ax.text(score10, i + 0.45, f'{score10:.1f}', va='center', ha='right', fontsize=11)

### Invert y-axis properly
ax.invert_yaxis()

### Display grid lines
ax.xaxis.grid(True, linestyle='-', linewidth=0.5)
ax.yaxis.grid(False)

### Show plot
plt.tight_layout()
plt.show()
```

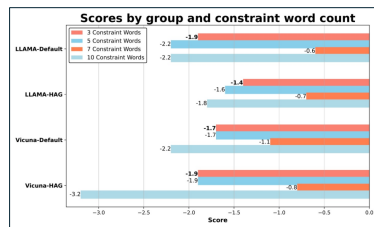
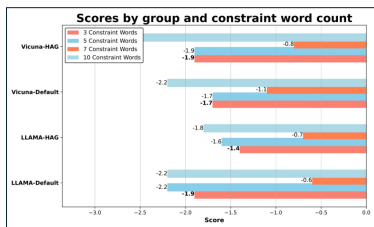
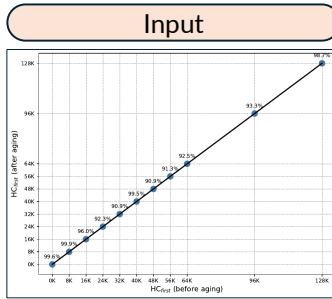


Figure A6. Qualitative result of MM-ReCoder for self-correction. The model corrects the category orders in the second turn.



First Turn

<think> Looking at the plot, I need to:

1. Create appropriate x and y data points that match the trend
2. Plot them with proper spacing
3. Add proper axis ticks and labels
4. Display percentages on points accurately
5. Use appropriate colors and styles
6. Add grid lines properly

</think>

```

import matplotlib.pyplot as plt
import numpy as np

#### Create figure with appropriate size
fig, ax = plt.subplots(figsize=(8, 6))

#### Data points based on visual estimation
x_values = [0, 8, 16, 24, 32, 40, 48, 56, 64, 96, 128]
y_values = [0, 8, 16, 24, 32, 40, 48, 56, 64, 96, 128]
percentages = [99.6, 99.9, 96.0, 92.3, 90.9, 99.5, 90.9, 91.3, 92.5, 93.3, 98.7]

#### Plot the points and connecting line
ax.scatter(x_values, y_values, color='#1f77b4', s=100, marker='o')
ax.plot(x_values, y_values, color='black', linewidth=1.5)

#### Add annotations for percentages
for i, txt in enumerate(percentages):
    ax.annotate(f'{txt:.1f}%', (x_values[i], y_values[i]), textcoords="offset
points", xytext=(0,10), ha='center', fontsize=11, fontweight='bold')

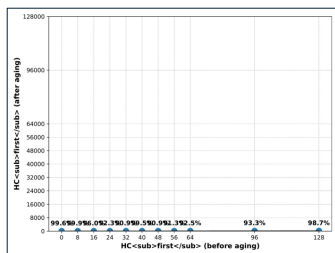
#### Set axis ticks and labels
ax.set_xticks([0, 8, 16, 24, 32, 40, 48, 56, 64, 96, 128])
ax.set_yticks([0, 8000, 16000, 24000, 32000, 40000, 48000, 56000, 64000,
96000, 128000])
ax.set_xlabel('HCsub-first<sub></sub> (before aging)', fontsize=12,
fontweight='bold')
ax.set_ylabel('HCsub-first<sub></sub> (after aging)', fontsize=12,
fontweight='bold')

#### Set grid with appropriate style
ax.grid(True, linestyle='--', linewidth=0.5)

#### Set ticks and labels properly
ax.set_title('', fontsize=14)

#### Display plot
plt.tight_layout()
plt.show()

```



Second Turn

<think> The issue is with proper axis formatting and proper spacing. Also, the y-axis ticks need proper alignment without being skewed incorrectly. The x-axis labels need proper alignment without being too skewed. Also, the percentages need proper positioning without being cut off incorrectly. The y-axis needs proper scientific notation formatting without being wrong. Also, the title/spacing needs adjustment better. Also, the proper formatting needs adjustment better without being skewed wrong.

</think>

```

import matplotlib.pyplot as plt
import numpy as np

#### Create figure with appropriate size
fig, ax = plt.subplots(figsize=(8, 6))

#### Data points based on visual estimation
x_values = [0, 8, 16, 24, 32, 40, 48, 56, 64, 96, 128]
y_values = [0, 8000, 16000, 24000, 32000, 40000, 48000, 56000, 64000, 96000,
128000]
percentages = [99.6, 99.9, 96.0, 92.3, 90.9, 99.5, 90.9, 91.3, 92.5, 93.3,
98.7]

#### Plot the points and connecting line
ax.scatter(x_values, y_values, color='#1f77b4', s=100, marker='o')
ax.plot(x_values, y_values, color='black', linewidth=1.5)

#### Add annotations for percentages
for i, txt in enumerate(percentages):
    ax.annotate(f'{txt:.1f}%', (x_values[i], y_values[i]), textcoords="offset
points", xytext=(0,10), ha='center', fontsize=11, fontweight='bold')

#### Set axis ticks and labels properly
ax.set_xticks([0, 8, 16, 24, 32, 40, 48, 56, 64, 96, 128])
ax.set_yticks([0, 8000, 16000, 24000, 32000, 40000, 48000, 56000, 64000,
96000, 128000])
ax.set_xlabel('HCsub-first<sub></sub> (before aging)', fontsize=12,
fontweight='bold')
ax.set_ylabel('HCsub-first<sub></sub> (after aging)', fontsize=12,
fontweight='bold')

#### Set proper ticks without scientific notation issues
ax.set_title('', fontsize=14)

#### Display proper grid with appropriate style
ax.grid(True, linestyle='--', linewidth=0.5)

#### Add proper tick labels without being skewed wrong
#### Remove y-axis scientific notation properly

#### Display plot
plt.tight_layout()
plt.show()

```

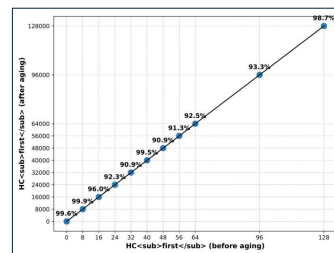
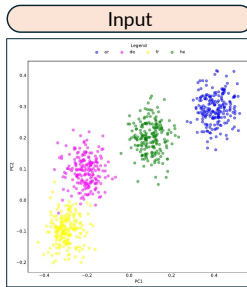


Figure A7. Qualitative result of MM-ReCoder for self-correction. The model corrects the y-axis values of the data points in the second turn.



First Turn

<think> Looking at the plot, I need to create scatter plots with appropriate colors and proper spacing. Since I don't have actual data, I'll need to generate synthetic data that matches the distribution shown in the plot. The data appears to be clustered in different regions with distinct colors. I'll create:

- Properly spaced points for each category
- Appropriate colors matching the original (blue, pink, yellow, green)
- Proper axis ticks and labels
- Legend positioned correctly

</think>

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

#### Set figure size
fig, ax = plt.subplots(figsize=(8, 8))

#### Generate synthetic data (approximating the distribution shown)
np.random.seed(42)

#### Create sample data for each category
#### ar = blue/purple
ar_x = np.random.normal((0.45, 0.35), (0.05, 0.05), 200)
ar_y = np.random.normal((0.35, 0.05), 200)
ax.scatter(ar_data[:, 0], ar_data[:, 1], color='k0000FF', marker='o', s=30, label='ar')

#### de = pink/magenta
de_x = np.random.normal((-0.25, 0.15), (0.05, 0.05), 200)
de_y = np.random.normal((0.15, 0.05), 200)
ax.scatter(de_data[:, 0], de_data[:, 1], color='FF00FF', marker='o', s=30, label='de')

#### fr = yellow
fr_x = np.random.normal((-0.3, -0.15), (0.05, 0.05), 200)
fr_y = np.random.normal((-0.15, 0.05), 200)
ax.scatter(fr_data[:, 0], fr_data[:, 1], color='FFFF00', marker='o', s=30, label='fr')

#### he = green
he_x = np.random.normal((0.15, 0.25), (0.05, 0.05), 200)
he_y = np.random.normal((0.25, 0.05), 200)
ax.scatter(he_data[:, 0], he_data[:, 1], color='008000', marker='o', s=30, label='he')

#### Add labels and title
ax.set_title('Legend', fontsize=16, fontweight='bold')
ax.set_xlabel('PC1', fontsize=12, fontweight='bold')
ax.set_ylabel('PC2', fontsize=12, fontweight='bold')

#### Add legend in top center position
ax.legend(title='Legend', loc='upper center', bbox_to_anchor=(0.5, 1.15),
         fontsize=11, title_fontsize=12, framealpha=True, edgecolor='black')

#### Set appropriate ticks
ax.set_xticks([-0.4, -0.2, 0.0, 0.2, 0.4])
ax.set_yticks([-0.2, -0.1, 0.0, 0.1, 0.2, 0.3, 0.4])

#### Display plot
plt.tight_layout()
plt.show()
```

Second Turn

<think> I need to fix the issue with generating proper 2D data. Instead of trying to create a 2D array directly, I should generate proper x and y values separately. Also, the legend needs proper alignment without causing errors. I need to create proper scatter plots without trying to combine them incorrectly. I'll generate proper 1D arrays and plot them appropriately.

</think>

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

#### Set figure size
fig, ax = plt.subplots(figsize=(8, 8))

#### Generate synthetic data (approximating the distribution shown)
np.random.seed(42)

#### Create sample data for each category
#### ar = blue/purple
ar_x = np.random.normal(0.45, 0.07, 200)
ar_y = np.random.normal(0.35, 0.05, 200)
ax.scatter(ar_x, ar_y, color='k0000FF', marker='o', s=30, label='ar')

#### de = pink/magenta
de_x = np.random.normal(-0.25, 0.07, 200)
de_y = np.random.normal(0.15, 0.05, 200)
ax.scatter(de_x, de_y, color='FF00FF', marker='o', s=30, label='de')

#### fr = yellow
fr_x = np.random.normal(-0.3, 0.07, 200)
fr_y = np.random.normal(-0.15, 0.05, 200)
ax.scatter(fr_x, fr_y, color='FFFF00', marker='o', s=30, label='fr')

#### he = green
he_x = np.random.normal(0.15, 0.07, 200)
he_y = np.random.normal(0.25, 0.05, 200)
ax.scatter(he_x, he_y, color='008000', marker='o', s=30, label='he')

#### Add labels and title
ax.set_title('Legend', fontsize=16, fontweight='bold')
ax.set_xlabel('PC1', fontsize=12, fontweight='bold')
ax.set_ylabel('PC2', fontsize=12, fontweight='bold')

#### Add legend in top center position with proper positioning
ax.legend(title='Legend', loc='upper center', bbox_to_anchor=(0.5, 1.15),
         fontsize=11, title_fontsize=12, framealpha=True, edgecolor='black')

#### Set appropriate ticks
ax.set_xticks([-0.4, -0.2, 0.0, 0.2, 0.4])
ax.set_yticks([-0.2, -0.1, 0.0, 0.1, 0.2, 0.3, 0.4])

#### Display plot
plt.tight_layout()
plt.show()
```

Third Turn

<think> The legend needs proper alignment without being skewed incorrectly, and it should appear in the top center without causing confusion. Also, the plot needs proper spacing without the title being cut incorrectly. Also, the positioning needs adjustment better without causing overlap issues properly. The proper positioning needs adjustment without creating an inconsistent legend. I need to create proper spacing and have the legend in the correct position without causing conflicts. Also, the y-axis ticks need proper alignment without being skewed wrong. Also, the proper spacing needs adjustment better.

</think>

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

#### Set figure size
fig, ax = plt.subplots(figsize=(8, 8))

#### Generate synthetic data (approximating the distribution shown)
np.random.seed(42)

#### Create sample data for each category
#### ar = blue/purple
ar_x = np.random.normal(0.45, 0.07, 200)
ar_y = np.random.normal(0.35, 0.05, 200)
ax.scatter(ar_x, ar_y, color='k0000FF', marker='o', s=30, label='ar')

#### de = pink/magenta
de_x = np.random.normal(-0.25, 0.07, 200)
de_y = np.random.normal(0.15, 0.05, 200)
ax.scatter(de_x, de_y, color='FF00FF', marker='o', s=30, label='de')

#### fr = yellow
fr_x = np.random.normal(-0.3, 0.07, 200)
fr_y = np.random.normal(-0.15, 0.05, 200)
ax.scatter(fr_x, fr_y, color='FFFF00', marker='o', s=30, label='fr')

#### he = green
he_x = np.random.normal(0.15, 0.07, 200)
he_y = np.random.normal(0.25, 0.05, 200)
ax.scatter(he_x, he_y, color='008000', marker='o', s=30, label='he')

#### Add labels and title
ax.set_title('', fontsize=16, fontweight='bold')
ax.set_xlabel('PC1', fontsize=12, fontweight='bold')
ax.set_ylabel('PC2', fontsize=12, fontweight='bold')

#### Add proper legend in top center without title conflict
ax.legend(title='Legend', loc='upper center', bbox_to_anchor=(0.5, 1.15),
         fontsize=11, title_fontsize=12, framealpha=True, edgecolor='black')

#### Set appropriate ticks
ax.set_xticks([-0.4, -0.2, 0.0, 0.2, 0.4])
ax.set_yticks([-0.2, -0.1, 0.0, 0.1, 0.2, 0.3, 0.4])

#### Display plot
plt.tight_layout()
plt.show()
```

Traceback (most recent call last):
 File "<string>", line 50, in code
 ar_data = np.random.normal((0.45, 0.35), (0.05, 0.05), 200)
 File "numpy/random/mtrand.pyx", line 1557, in numpy.random.mtrand.RandomState.normal
 File "_common.pyx", line 600, in numpy.random._common.cont
 File "_common.pyx", line 517, in numpy.random._common.cont_broadcast_2
 File "_init_.cython-30.pxd", line 780, in numpy.PyArray_MultiIterNew3
 ValueError: shape mismatch: objects cannot be broadcast to a single shape. Mismatch is between arg 0 with shape (200,) and arg 1 with shape (2,).

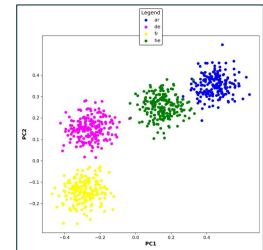
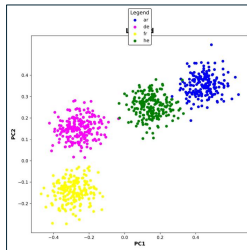


Figure A8. Qualitative result of MM-ReCoder for self-correction. In the second turn, the model addresses the runtime error. In the third turn, the model removes the chart title overlapped with the legend.

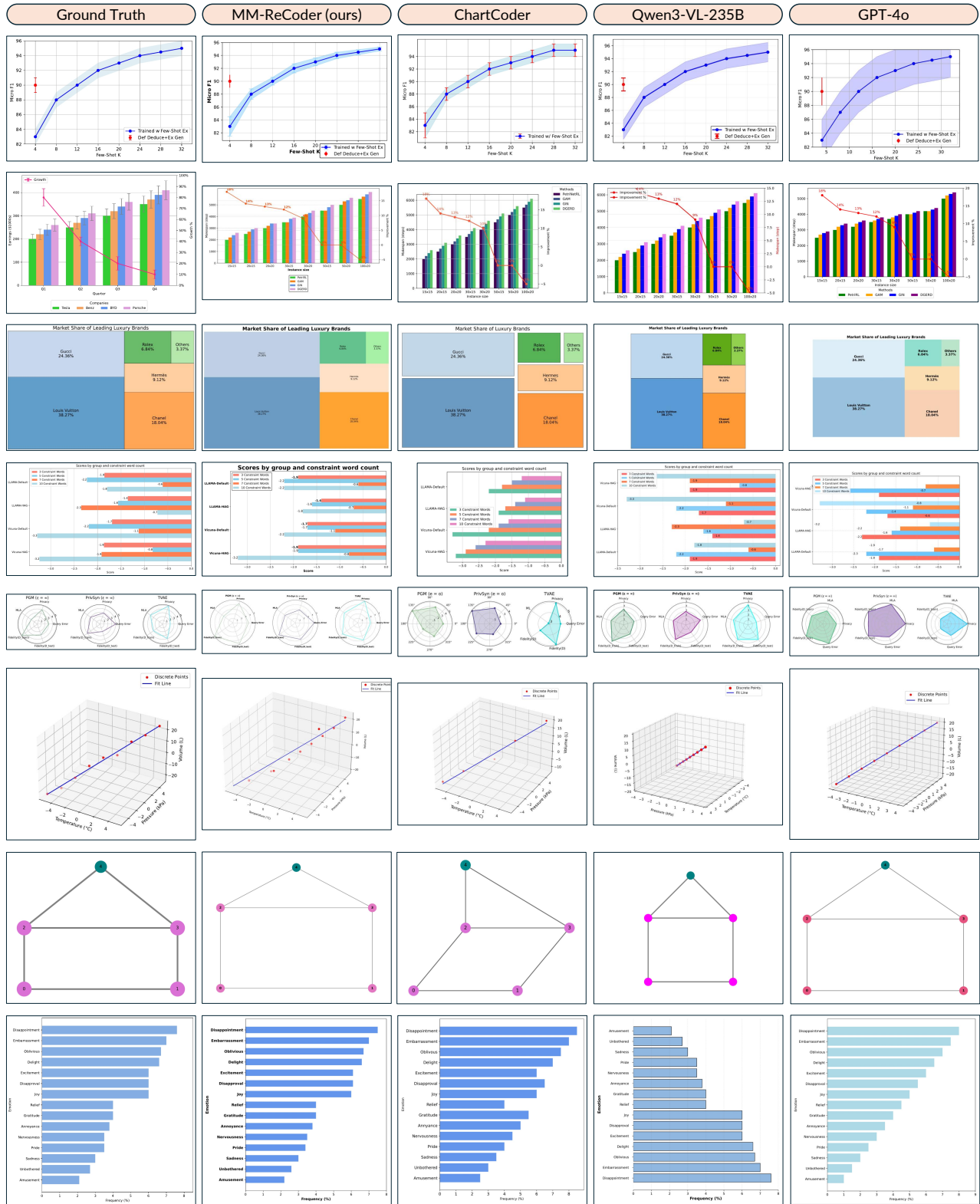


Figure A9. Qualitative results of MM-ReCoder and baselines.

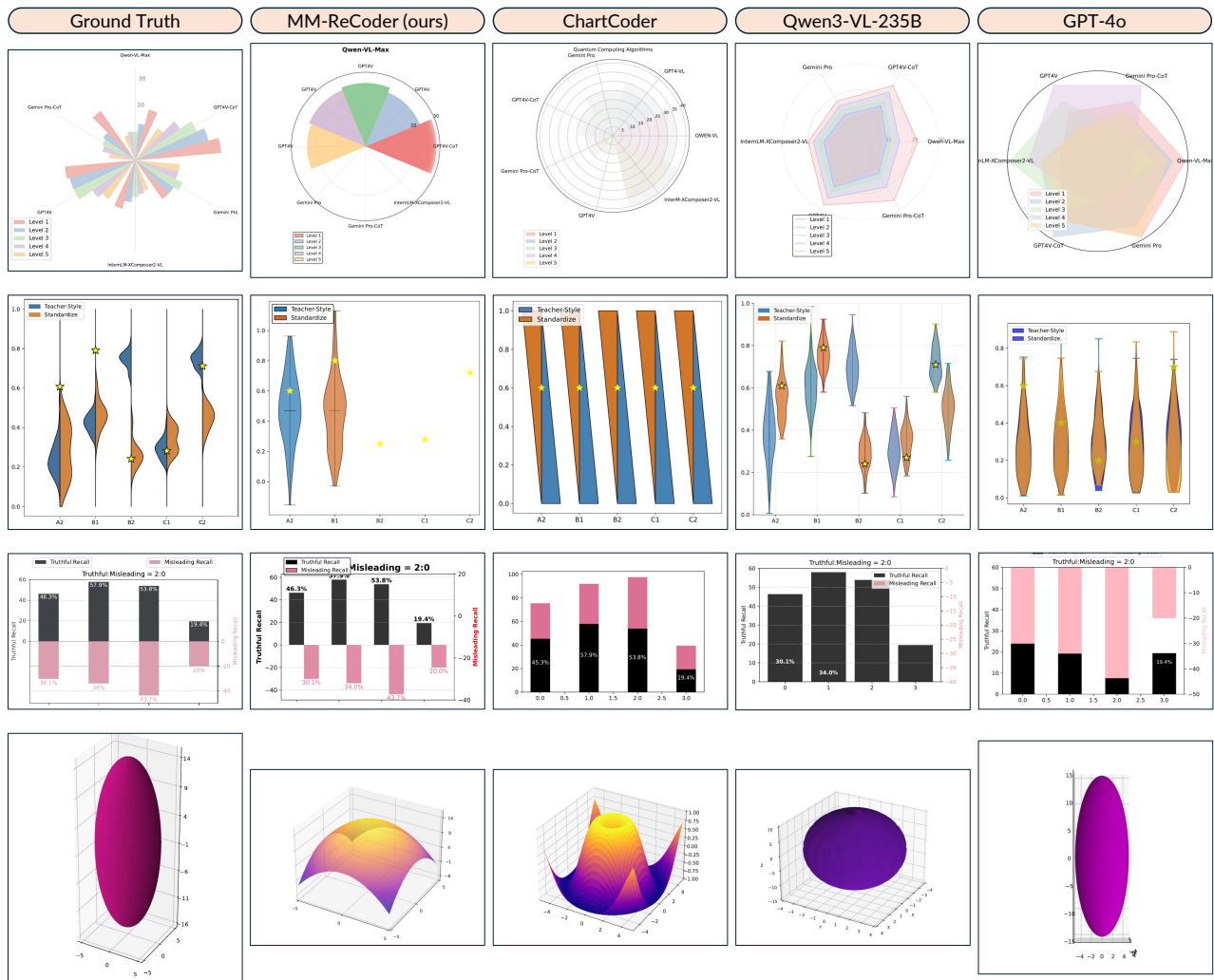


Figure A10. Failure cases of MM-ReCoder and baselines. The models tend to fail when the chart type is rare.