AMACE: Automatic Multi-Agent Chart Evolution for Iteratively Tailored Chart Generation

Hyuk Namgoong[1], **Jeesu Jung**[1], **Hyeonseok Kang**[1], **Yohan Lee**[2] and **Sangkeun Jung**[1*]
[1]Computer Science and Engineering, Chungnam National University, Republic of Korea
[2]Electronics and Telecommunications Research Institute, Republic of Korea
{hyuk199, jisu.jung5, dnfldjaak11,}@gmail.com,
carep@etri.re.kr and hugmanskj@gmail.com

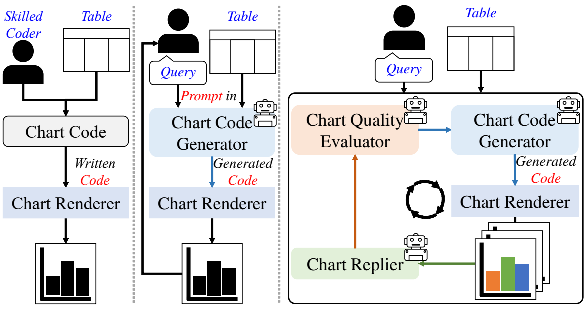
Abstract

Many statistical facts are conveyed through charts. While various methods have emerged for chart understanding, chart generation typically requires users to manually input code, intent, and other parameters to obtain the desired format on chart generation tools. Recently, the advent of image-generating Large Language Models has facilitated chart generation; however, even this process often requires users to provide numerous constraints for accurate results. In this paper, we propose a loop-based framework for automatically evolving charts in a multi-agent environment. Within this framework, three distinct agents—Chart Code Generator, Chart Replier, and Chart Quality Evaluator—collaborate for iterative, user-tailored chart generation using large language models. Our approach demonstrates an improvement of up to 29.97% in performance compared to first generation, while also reducing generation time by up to 86.9% compared to manual prompt-based methods, showcasing the effectiveness of this multi-agent collaboration in enhancing the quality and efficiency of chart generation.

1 Introduction

Charts are a key tool for visualizing data in both everyday life and scientific research. They are often used in reports and academic papers to simplify complex data and make it easier to understand. By converting difficult-to-read tables into visual formats, charts improve clarity and make information more persuasive.

Traditionally, creating charts has required manual coding with visualization libraries like matplotlib and AutoViz (Seshadri, 2020) (Figure 1a). This process is time-consuming and requires technical skills. Recently, large language models (LLMs), such as GPT-4 Vision (Yang et al., 2023), have enabled new automated methods that generate charts based on table data and user prompt (Figure 1b). These methods make chart creation more accessible by removing the need for *programming knowledge*.



(a) Programming Based Chart Approach

(b) Prompt Engineering Based Chart Approach

(c) Our approach: Multi-Agent Based Automatic Evolution Chart Approach

Figure 1: Comparison of our approach with typical chart generation methods. Manual tools require significant human intervention to convert tables into charts. Recent research leverages LLMs to generate charts from table data via code. Our approach utilizes multi-agents to iteratively refine and generate tailored charts automatically.

However, current single LLM approaches have limitations. These methods typically generate a single chart based solely on the user input, making the quality of the chart heavily dependent on precise *prompt-engineering*. Ambiguous instructions often result in ineffective charts, and these approaches require iterative user feedback, which demands continuous interaction and reduces efficiency.

To address these issues, we propose a new framework in which multiple LLM agents collaborate in a loop-like, evolutionary process to automatically generate and improve charts. Unlike traditional approaches, we take *zero-human-prompt-engineering*, utilizing iterative interactions among specialized agents to enable progressive chart quality improvements and gradual optimization over time.

Our proposed approach introduces three key LLM roles to achieve automated chart generation and refinement: 1) **Chart Code Generator** gen-

[*] Corresponding author

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 21472–21487
November 4-9, 2025 ©2025 Association for Computational Linguistics

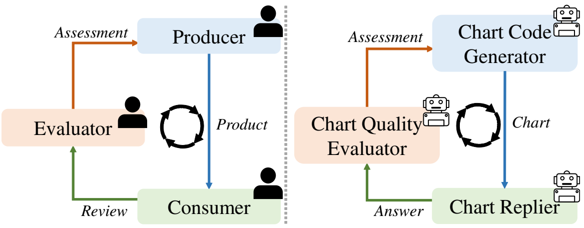
erates chart code, 2) **Chart Replier** has the LLM answer questions about the generated chart in relation to the user's objectives, and 3) **Chart Quality Evaluator** assesses whether the chart is suitable based on the table, user intent, and responses. This process is repeated at each step, allowing for gradual evolution.

To validate our framework, we first assessed whether each LLM agent effectively fulfilled its assigned role. We found that chart generation with the AMACE framework improved performance, with Chart Replier providing responses up to 29.97% better with additional steps compared to a single-step. Human evaluations, which rated chart preferences on a 1–5 scale, also showed consistent improvement across steps. Compared to manual chart generation using prompt engineering, AMACE achieved up to an 86.9% reduction in generation time. Additionally, by swapping models for each role, we demonstrated that better models tended to perform more effectively. The experiments illustrate how our framework progressively refines charts to better match user intent as the process advances.

2 Related Works

Recent frameworks enhance LLM performance by assigning roles to multiple models and leveraging their interactions. ART (Shridhar et al., 2024) and Self-Refine (Madaan et al., 2023) use feedback for iterative improvement. (Park et al., 2023; Xi et al., 2023), Eureka (Ma et al., 2024), and Guided-Code (Almorsi et al., 2025) apply multi-agent or loop-based strategies to simulate behavior or refine code. These works highlight the effectiveness of role-based collaboration in LLM systems.

LLMs are not limited to text-based input and output. With the advancement of Vision Transformers, multimodal LLMs that process both images and text to understand their interrelationships are gaining attention. Representative examples include BLIP2 (Li et al., 2023b), LLaVA (Liu et al., 2023b), and Gemini (Gemini, 2024). Frameworks that carry out interactions through the relationship between these models are being studied, such as (Wu et al., 2023), which extracts features from vision and generates images at each loop; (Li et al., 2023a), which collects similar examples to the input image through a loop structure and utilizes them for few-shot learning; and SILMM (Qu et al., 2025), which adopts a model-agnostic self-improvement loop where LLMs iteratively refine text-image alignment via preference optimization.



(a) Real-world Evaluation Framework Cycle (b) Multi-Agent Evaluation Framework Cycle

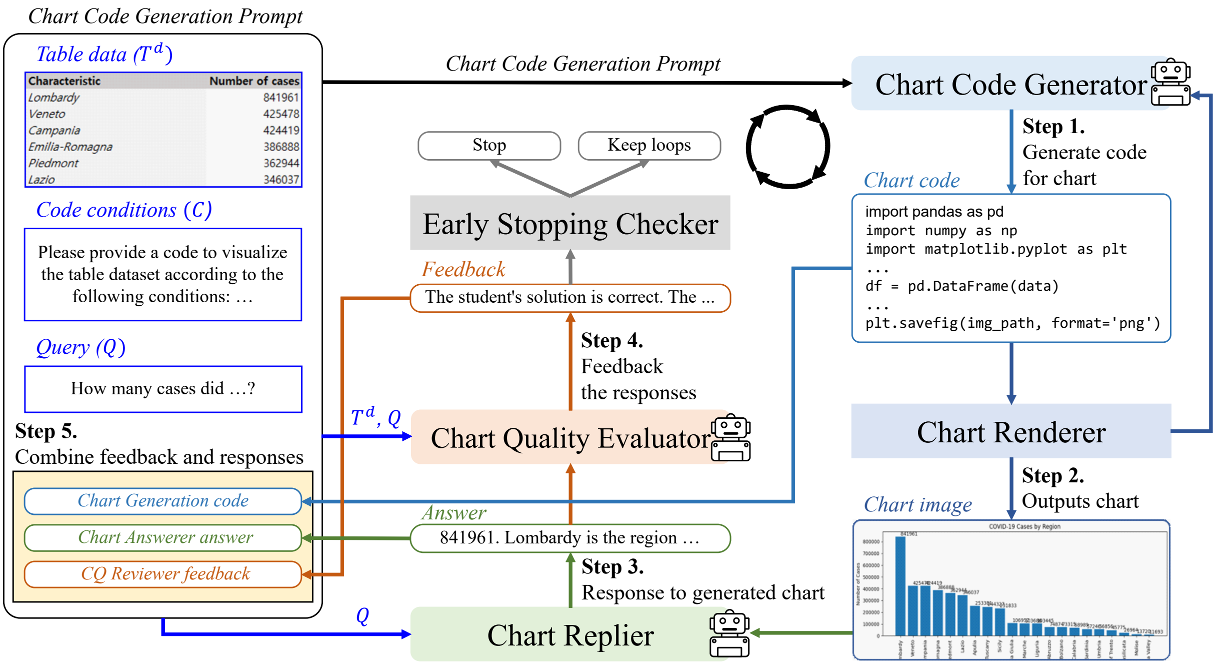
Figure 2: The real-world evaluation for product improvement and multi-agent evaluation framework. Our framework assigns the roles of the producer, consumer, and evaluator roles to specialized agents as the Chart Code Generator, Chart Replier, and Chart Quality Evaluator.

Chart generation using LLMs is closely tied to code generation, where textual and visual inputs guide the synthesis of chart-rendering code (Masry et al., 2023; Liu et al., 2023a). Text2Chart31 (Pesaran Zadeh et al., 2024) and Matplotbench (Yang et al., 2024) translate text and numeric data into executable chart code, while chart-specialized multimodal LLMs (Han et al., 2023; Masry et al., 2024) directly process chart images. ChartCitor (Goswami et al., 2025) uses multiple agents to extract tables from chart images and assign fine-grained answer citations. METAL (Li et al., 2025) introduces a multi-agent framework that refines existing charts, requiring a pre-existing chart as input and focusing on automatic improvement rather than generation from scratch.

Our study aims to evolve chart generation performance automatically by employing multiple LLMs in iterative multi-agent environments. In contrast to prior studies, our method adopts an iterative structure that refines charts to better align with user intent. We assign roles to different types of LLMs to construct a collaborative environment for generating charts that are tailored to specific user objectives.

3 Automatic Multi-Agent Chart Evolution(AMACE)

In this paper, we propose **Automatic Multi-Agent Chart Evolution (AMACE)**, a framework leveraging a cooperative multi-agent approach for chart generation. Traditional single-agent approaches often produce ambiguous outputs due to task complexity and cognitive load. Our framework addresses this by employing specialized LLM agents



Characteristic	Number of cases
Lombardy	841961
Veneto	425478
Campania	424419
Emilia-Romagna	386888
Piedmont	362944
Lazio	346037

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
...
df = pd.DataFrame(data)
...
plt.savefig(img_path, format='png')
```

Figure 3: Overall architecture of Automatic Multi-Agent Chart Evolution framework. In the initial step, the description is input into the Chart Generation module. The Chart Replier (CR) receives the query from the description, while the Chart Quality Evaluator (CQE) takes both the query and the table from the description as input. After generating the chart, the outputs from the CR and the CQE are combined in an appropriate format for the description, and this process is repeated at each step. Early Stopping Checker involves halting loops when deemed appropriate based on feedback from the CQE.

with distinct roles that collaborate interactively. This design ensures robust chart generation through evolutionary optimization. AMACE agent prompts are detailed in Appendix A.

3.1 Evaluation Framework Cycle

We identified real-world evaluation cycles, such as restaurants improving through delivery app reviews or research papers evolving through reviews and revisions, as shown in Figure 2a. Inspired by these processes, we designed a multi-agent evaluation framework cycle by assigning the roles of *producer*, *consumer*, and *evaluator* to specialized agents, as shown in Figure 2b. These agents collaborate to perform tasks efficiently and systematically.

Our chart generation framework leverages a multi-agent system as illustrated in Figure 3. The system orchestrates three distinct LLMs, including one with multimodal capabilities, each performing specialized tasks. The Chart Code Generator (CCG) generates code for table-to-chart transformation, the Chart Replier (CR) generates responses based on chart analysis, and the Chart Quality Evaluator (CQE) evaluates these responses. Through iterative refinement cycles, the system progressively optimizes chart generation. To enhance computational efficiency, we implement an Early Stopping Checker that terminates the iteration loops when the CQE determines that the generated chart meets the specified quality criteria.

3.2 Chart Code Generator (CCG)

The role of the CCG is to receive a series of text and conditions for code generation relevant to the input. in a text-to-code generation paradigm, where both input and output maintain textual format.

The input prompt structure, as illustrated in Figure 10, incorporates four key elements: role specification, table data, querys, and code conditions. The system assigns a Python programmer role to the LLM and provides text-formatted representations of the table data, query specifications, and code constraints. The query component encompasses visualization task objectives and, where applicable, example questions. The code conditions specify required libraries, table variable names, and output code structure specifications, which facilitate subsequent code execution post-processing.

The code regeneration process integrates feedback from multiple components, as depicted in Figure 11. This process incorporates outputs from the CR and CQE, alongside the previous CCG results. The system implements a conversational architecture where inputs comprise the previous CCG's input-output pairs, combined with CR and CQE feedback. This feedback loop structure enables iterative refinement, with new code generation instructions contextualized by prior process results.

3.3 Chart Renderer

The Chart Renderer module executes the code generated by the CCG to chart visualizations. It performs validation checks for requirement compliance and syntactic correctness, including table variable names and structural alignment. If the code fails, a call to the CCG for regenerate code.

Following successful validation, the system executes the code through a Python runtime environment. The resultant chart visualization is persisted to storage and transmitted to the CR. Additionally, the validated code is preserved to facilitate iterative refinement in subsequent processing stages.

3.4 Chart Replier (CR)

In the role of the CR, multimodal LLMs are utilized to process visual chart input alongside textual prompts and generate corresponding responses. This system architecture leverages the multimodal capabilities of LLMs to handle visual and textual input within a text-output framework.

As illustrated in Figure 12, the system processes chart visualizations and queries through a structured input. The query component includes task-specific objectives, with direct question input for question-answering tasks. The system generates responses that align with the specified task objectives through comprehensive chart analysis.

3.5 Chart Quality Evaluator (CQE)

The role of the CQE is to assess the quality of answers provided by the CR regarding the generated chart. It evaluates how effectively the chart conveys information and identifies any missing details. This evaluation is crucial in cases where ground truth is unavailable, as often occurs in real-world applications. Additionally, to account for potential missing information during the visualization process, the CQE takes a table input.

The input prompt structure, as shown in Figure 13, includes the role, table, query, and the response from the previous model. The system assigns the role of evaluating the correctness of responses to the LLM. The inputs consist of the table formatted in text, the query, and the response of the previous model. The updated query reflects the previous model's query in problem form. The output indicates whether the response is correct or incorrect, along with an explanation.

3.6 Iterative Multi-Agent Interaction

Our methodology adopts an iterative approach to chart generation:

1. The CCG generates charting code based on the table and query, while the Chart Renderer converts table data into charts using the generated code.
2. The CR processes questions relevant to the chart and provides answers.
3. The CQE evaluates based on the table, questions, and answers from the CR.
4. The feedbacks and answers from the CR and CQE are fed back to the CCG to generate new code, creating an iterative cycle.

During the iterative process, the chart evolves gradually through agent interactions. Each agent contributes responses, and feedback guides the evolution of the charting code, enabling continuous improvement in this collaborative environment.

3.7 Early Stopping Checker

To save time and resources in generating tailored charts, we introduce an Early Stopping mechanism, called the Early Stopping Checker. It autonomously halts loops once CQE feedback is confirmed as correct, preventing unnecessary steps and reducing both user waiting time and resource usage.

4 Experiments

This study explores the impact of the AMACE framework through multi-agent experiments with three main objectives:

- Quantify improvements in chart generation using the AMACE framework and validate them through human evaluation.
- Determine the influence of each LLM role on performance.
- Compare performance with and without Early Stopping to assess its potential as a replacement and highlight its time-saving benefits.

Dataset For the analysis of experiments, we employed the ChartQA (Masry et al., 2022) and Chart-to-Text (Kantharaj et al., 2022) datasets, which contain both table and chart data, making them suitable for multimodal evaluation.

AMACE-1

- **CCG**: GPT-3.5-turbo (Ye et al., 2023)

- **CR**: Gemini 1.0 Pro Vision (Gemini, 2024)
- **CQE**: GPT-4-turbo (OpenAI, 2024)

AMACE-1o

- **CCG**: GPT-4o mini (OpenAI, 2024)
- **CR**: Gemini 1.5 Flash (Team, 2024)
- **CQE**: GPT-4o (OpenAI, 2024)

In our experiments, AMACE-1o utilized *high-performing* LLMs, whereas AMACE-1 employed *earlier-generation* models, yielding inferior performance. Based on the ablation experiments, we designed the CR and CQE roles in AMACE using a more advanced model.

For the ablation experiments, we utilized GPT-4o mini and Gemini 1.0 Pro Vision as the baseline, while GPT-4o and Gemini 1.5 Flash served as the advanced models. Each role was tested by replacing its respective model. Dataset and all experiment settings are shown in Appendix B.

4.1 Performance Improvement Across Steps

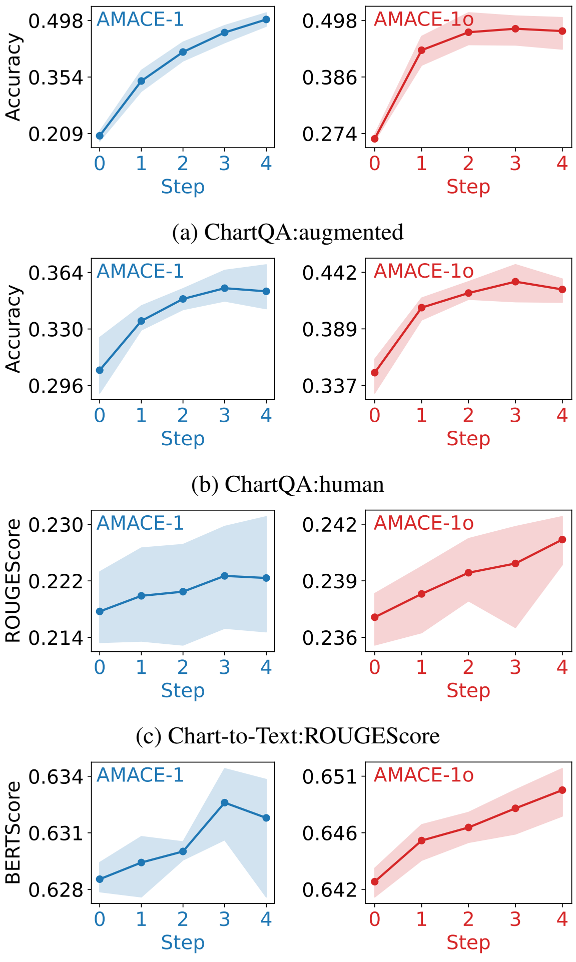
To assess whether the CR provides appropriate responses to the generated chart and user intent, we evaluate the extent of performance improvement across datasets as the steps progress.

The results in Figure 4a and 4b show performance improvements with each step in ChartQA. AMACE-1 saw a 29.79% accuracy increase for ChartQA:augmented, while AMACE-1o showed an 8.48% rise for ChartQA:human. ChartQA:augmented surpassed ChartQA:human due to LLM-generated questions.

Additionally, Figure 4c and 4d shows that the changes in ROUGE-L and BERTScore for Chart-to-Text across all steps are less pronounced compared to ChartQA. However, both AMACE-1 and AMACE-1o show an increase in these metrics. ROUGE-L and BERTScore exhibit wider bounds compared to Accuracy. These results indicate that AMACE performs well across all datasets. Numerical results are detailed in Table 4.

4.2 Human Evaluation of Chart Quality

To evaluate the quality of generated charts across iterative steps, we conducted a human evaluation in which five annotators rated charts generated from 150 data samples. For each sample, the AMACE framework generated five versions of a chart across five steps. Participants were asked to rate each chart on a scale from 1 (least suitable) to 5 (most suitable) based on how well the chart matched the intended query and conveyed the relevant information.



(a) ChartQA:augmented

(b) ChartQA:human

(c) Chart-to-Text:ROUGEScore

(d) Chart-to-Text:BERTScore

Figure 4: Performance graphs over five steps for each dataset. In Chart-to-Text, ROUGEScore represents the ROUGE-L F1 score. The *x*-axis shows step, with blue for AMACE-1 and red for AMACE-1o. Bounds denote the minimum and maximum values.

As shown in Figure 5, human ratings consistently improve as the chart generation progresses from Step 1 to Step 5, indicating that our framework gradually refines both the content and presentation of the charts. This trend is consistent across all datasets and models, demonstrating the effectiveness of our step-wise refinement strategy.

Further qualitative improvements are illustrated in Figure 6, which shows example charts generated by AMACE-1o at each step. As the steps progress, the charts include more informative elements, such as numeric labels on lines or modified bar layouts for better visual clarity. This highlights improvements in both informativeness and visual design.

Figure 14 in Appendix further shows how the underlying chart code evolves across steps. In particular, we observe that initial code generated by the CCG is updated in subsequent steps to reflect

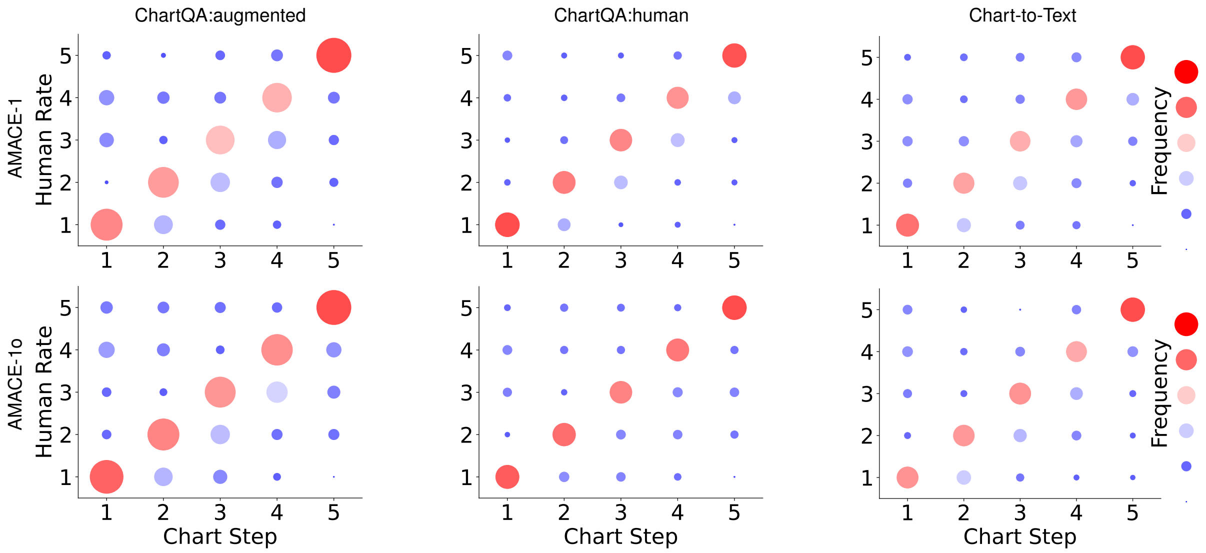


Figure 5: Human evaluation results for charts generated by AMACE-1 and AMACE-1o over five iterative step. Each chart was rated on a 5-point scale (1 = least preferred, 5 = most preferred) based on visual quality and relevance to the intent. The evaluations were conducted by five human judges across 150 chart instances from three datasets. Circle size and color represent the frequency of ratings.

feedback from the CQE, resulting in refined visual output. Additional chart types and refinement details can also be found in Appendix D.

4.3 Ablation Analysis of LLM Role Effects

To evaluate the impact of each role on performance, experiments were conducted by replacing the baseline model combinations with higher-performing models for each role. The performance was validated using ground truth from each dataset.

The results in Figure 7 show that while utilizing better models does not always guarantee improvement, the overall trend indicates a performance boost. In ChartQA, replacing the CQE yielded the best performance compared to other model replacements. In Chart-to-Text, replacing the CR further improved performance. These findings highlight the importance of generating sentences effectively and understanding context for accurate evaluation. Based on these insights, we developed the AMACE settings. Numerical results are provided in Table 5.

4.4 Impact of Early Stopping

We evaluate the impact of CQE-based Early Stopping on iterative chart generation, comparing it with generation without this mechanism.

Figure 8 shows the percentage of Early Stopping at each step for each dataset. In the first step, up to 63.9% of the processes stop, indicating that more than half of the datasets stop at the first step.

Figure 9 compares the final performance with and without Early Stopping. Overall, Early Stopping achieves comparable or higher performance, with AMACE-1 showing a notable 17.3% improvement in ChartQA. These results confirm the effectiveness of Early Stopping in enhancing performance. Numerical results are shown in Table 4.

Model	Dataset	w/o Early Stopping Checker	w/ Early Stopping Checker
AMACE -1	ChartQA:augmented	36.52s	24.27s (↓ 33.5%)
	ChartQA:human	77.47s	45.73s (↓ 41.0%)
	Chart-to-Text	80.04s	39.49s (↓ 50.7%)
AMACE -1o	ChartQA:augmented	31.09s	22.45s (↓ 27.8%)
	ChartQA:human	36.10s	23.83s (↓ 34.0%)
	Chart-to-Text	42.91s	15.74s (↓ 63.3%)

Table 1: Comparison of time (in seconds) taken per sample with and without the Early Stopping Checker. Each experiment was conducted on 100 samples per dataset. The values in parentheses indicate the percentage of time saved using Early Stopping.

Table 1 demonstrates the time efficiency gained by applying Early Stopping to terminate the iterative generation process. The results show that Early Stopping improves both the efficiency and overall effectiveness of chart generation. Notably, the chart-to-text setting with AMACE-1o achieved a 63.31% reduction in processing time, indicating substantial gains across datasets. In addition, Table 7 shows that Early Stopping reduced generation cost by up to 59.6%, along with improvements in token efficiency. For full results, see Appendix E.

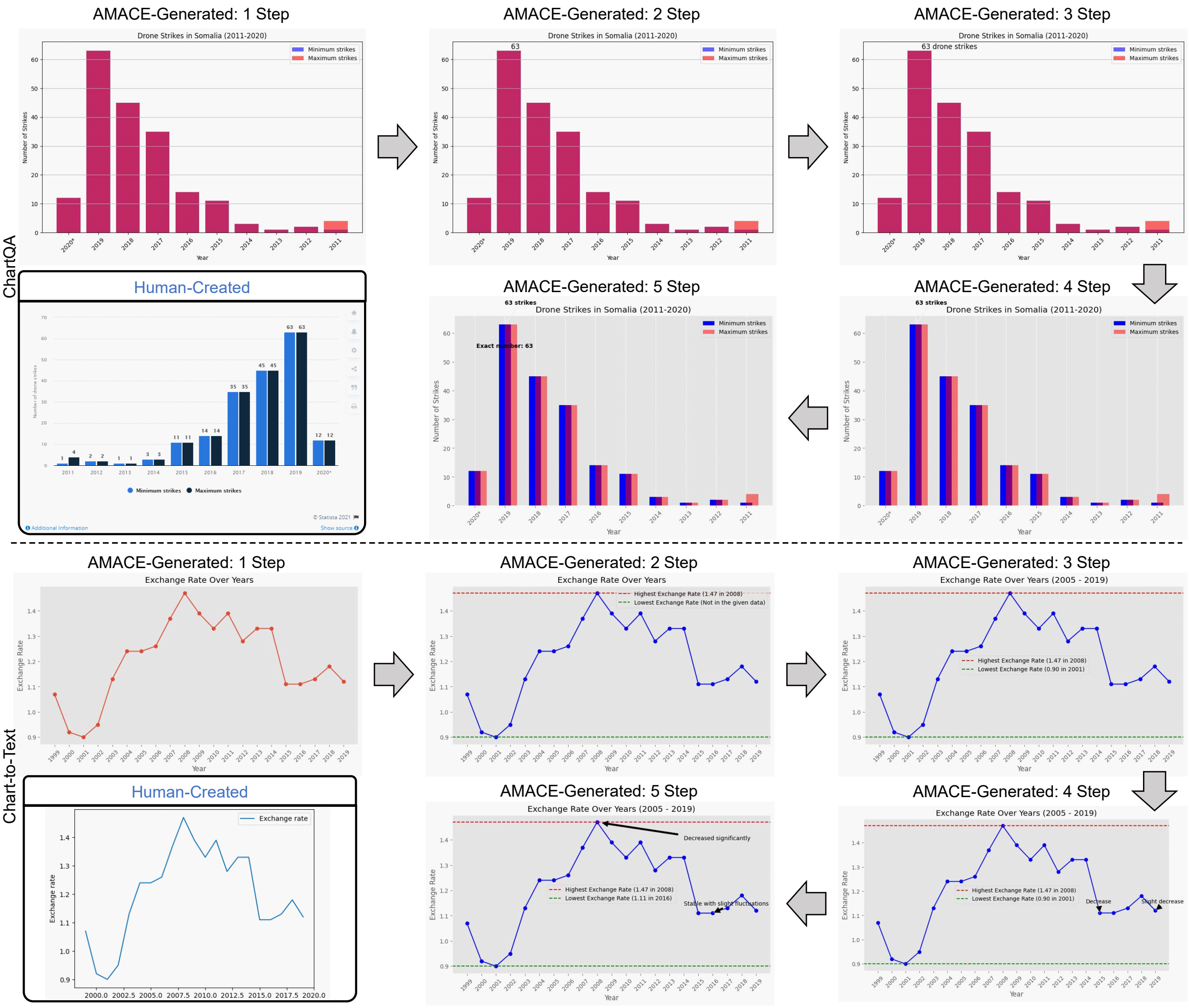


Figure 6: Step-by-step chart images generated by AMACE-1o across five refinement steps, shown alongside the original human-created charts for reference. Each row represents a dataset, and illustrates how AMACE-1o progressively improves the generated charts in terms of informativeness, visual clarity, and alignment with the underlying data.

4.5 Comparison with Prompt Engineering Based Chart Approach

We compare AMACE with a *Prompt Engineering Based Chart Approach*, illustrated in Figure 1b, where human participants manually revise prompts to iteratively generate charts (denoted as *Human Process*). The human process was evaluated on 50 samples, and the time required to generate each chart was measured. In this setting, humans stopped the process once they were satisfied with the generated chart, similar in principle to Early Stopping in AMACE. Participant details in Appendix B.

Table 2 compares the time taken for AMACE with Early Stopping Checker with *Human Process*. For *Human Process* charts, the process is stopped when the user is satisfied, and the next data point is processed. This is compared with the AMACE approach that uses Early Stopping Checker.

Dataset	Model	Time (sec)
ChartQA: augmented	Human Process	111.43s
	AMACE-1	24.27s (↓ 78.2%)
	AMACE-1o	22.45s (↓ 79.9%)
ChartQA: human	Human Process	144.66s
	AMACE-1	45.73s (↓ 68.4%)
	AMACE-1o	23.83s (↓ 83.6%)
Chart-to-Text	Human Process	120.24s
	AMACE-1	39.49s (↓ 67.2%)
	AMACE-1o	15.74s (↓ 86.9%)

Table 2: Compares the time per sample between AMACE with Early Stopping and the Prompt Engineering-based *Human Process*, based on 100 samples per dataset. Times are in seconds; Parentheses show time saved over the *Human Process*.

While the *Human Process* allows for fine-grained control through manual review and prompt

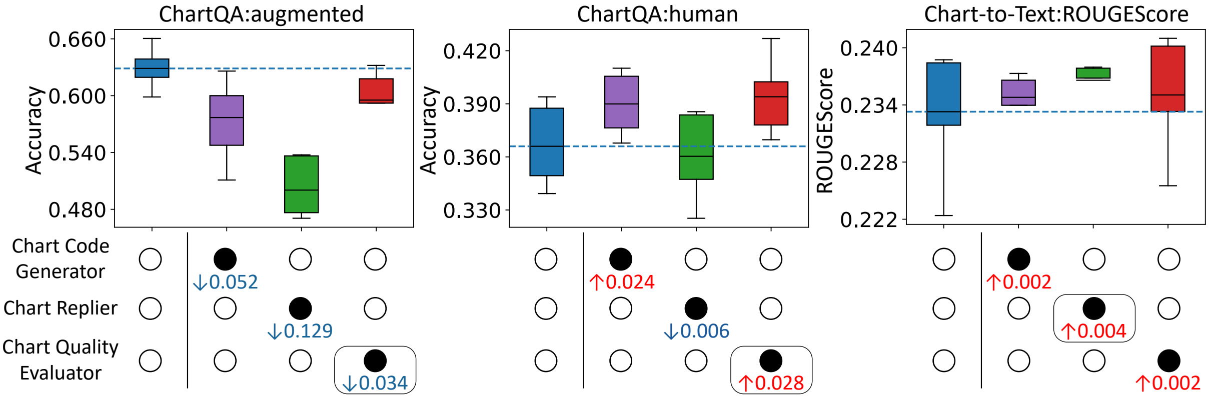


Figure 7: Influence of each role on performance, tested with 200 samples per dataset. ○ denotes the baseline model, while ● represents the advanced model for each role. The blue line indicates baseline performance, the number below ● shows the difference from the baseline, and the black line within the boxplot represents the mean performance. The rounded square around ● indicates the best-performing model, highlighting the most critical role.

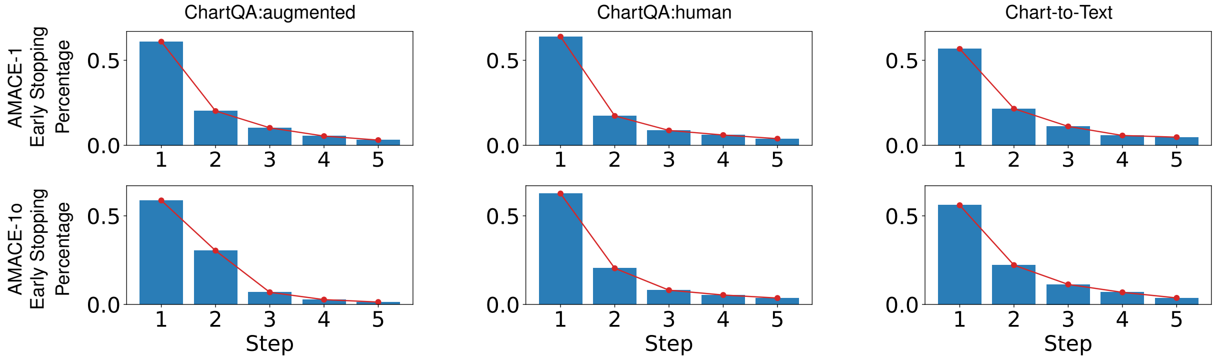
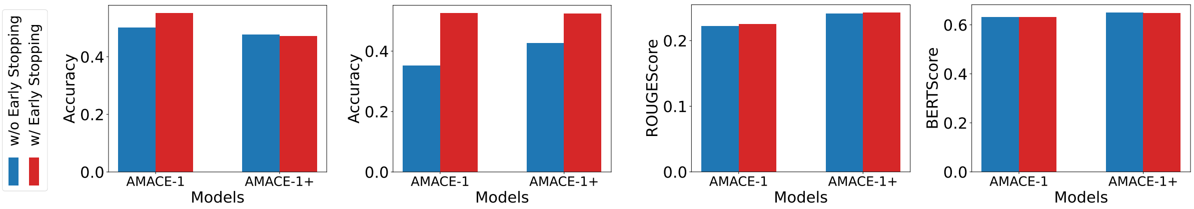


Figure 8: The Early Stopping percentage at each step. The datasets are listed at the top, and the models are indicated on the left. The vertical axis of the graph represents the Early Stopping percentage, ranging from 0 to 0.67.



(a) ChartQA:augmented (b) ChartQA:human (c) Chart-to-Text: ROUGEScore & BERTScore

Figure 9: This figure illustrates the performance with and without Early Stopping. For Early Stopping, the final performance is shown, while without Early Stopping, the performance after 5 steps is presented.

adjustments, it requires domain expertise and significantly more time. As shown in Table 2, AMACE-1o achieved up to 86.9% time reduction compared to the Human Process on the Chart-to-Text dataset. This overhead arises from the need for individualized judgment and iterative modifications for each data point when humans are involved.

Figure 15 shows that both approaches produce charts of comparable quality. Although manual refinement may yield slight improvements in specific visual elements, the results show that AMACE automates chart generation effectively while offering substantial time savings.

5 Conclusion

We introduced AMACE, a multi-agent LLM framework for automated chart generation that leverages a loop-based refinement process to align outputs with user intent. Through both quantitative metrics and human evaluations, we demonstrated that the generated charts progressively improve in quality and better reflect user goals. Compared to manual prompt engineering, AMACE achieves significant time savings while maintaining comparable output quality. We further analyzed the contributions of each LLM agent, confirming the effectiveness of role specialization and the impact of loop termination conditions on performance and cost efficiency.

As future work, we plan to enhance chart generation and leverage model distillation to transfer these capabilities into lightweight models for practical deployment. We also aim to replace the CQE module with a fine-tuned, chart-specific multimodal model to improve evaluation accuracy, while exploring efficient inference techniques and hierarchical agent architectures to ensure scalability and robust coordination.

Limitations

AMACE requires substantial computational resources due to its multi-agent design involving multiple LLMs, which may hinder scalability. This could be addressed by incorporating smaller, task-specific models specialized for chart generation.

In addition, our evaluation focuses on user-intent-driven chart generation, excluding datasets like Matplotbench and Text2Chart31, which prioritize chart structure over creation intent. Consequently, our experiments are limited to a narrow range of chart types.

Although our study centers on bar and line charts, this reflects the fact that these chart types dominate real-world usage, and LLMs naturally capture such tendencies during training. Importantly, this limitation can be alleviated by explicitly specifying alternative chart types or requesting diverse chart varieties through user instructions.

Acknowledgements

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT)(No. RS-2025-0055621731482092640101), Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (2019-0-00004, Development of semi-supervised learning language intelligence technology and Korean tutoring service for foreigners) and research fund of Chungnam National University.

References

Amr Almorsi, Mohanned Ahmed, and Walid Gomaa. 2025. Guided code generation with llms: A multi-agent framework for complex code tasks. *Preprint*, arXiv:2501.06625.

Gemini. 2024. Gemini: A family of highly capable multimodal models. *Preprint*, arXiv:2312.11805.

Kanika Goswami, Puneet Mathur, Ryan Rossi, and Franck Dernoncourt. 2025. Chartcitor: Multi-agent framework for fine-grained chart visual attribution. *Preprint*, arXiv:2502.00989.

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

Shankar Kantharaj, Rixie Tiffany Leong, Xiang Lin, Ahmed Masry, Megh Thakkar, Enamul Hoque, and Shafiq Joty. 2022. Chart-to-text: A large-scale benchmark for chart summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4005–4023, Dublin, Ireland. Association for Computational Linguistics.

Bingxuan Li, Yiwei Wang, Jiuxiang Gu, Kai-Wei Chang, and Nanyun Peng. 2025. Metal: A multi-agent framework for chart generation with test-time scaling. *Preprint*, arXiv:2502.17651.

Bo Li, Yuanhan Zhang, Liangyu Chen, Jinghao Wang, Fanyi Pu, Jingkang Yang, Chunyuan Li, and Ziwei Liu. 2023a. Mimic-it: Multi-modal in-context instruction tuning. *Preprint*, arXiv:2306.05425.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023b. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *Preprint*, arXiv:2301.12597.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Fangyu Liu, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Yasemin Altun, Nigel Collier, and Julian Eisenschlos. 2023a. MatCha: Enhancing visual language pretraining with math reasoning and chart derendering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12756–12770, Toronto, Canada. Association for Computational Linguistics.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023b. Visual instruction tuning. *Preprint*, arXiv:2304.08485.

AI @ Meta Llama Team. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Eureka: Human-level reward design via coding large language models. *Preprint*, arXiv:2310.12931.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *Preprint*, arXiv:2303.17651.

Ahmed Masry, Xuan Long Do, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. 2022. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2263–2279, Dublin, Ireland. Association for Computational Linguistics.

Ahmed Masry, Parsa Kavehzadeh, Xuan Long Do, Enamul Hoque, and Shafiq Joty. 2023. UniChart: A universal vision-language pretrained model for chart comprehension and reasoning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14662–14684, Singapore. Association for Computational Linguistics.

Ahmed Masry, Megh Thakkar, Aayush Bajaj, Aaryaman Kartha, Enamul Hoque, and Shafiq Joty. 2024. Chartgemma: Visual instruction-tuning for chart reasoning in the wild. *Preprint*, arXiv:2407.04172.

OpenAI. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *In the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*, UIST '23, New York, NY, USA. Association for Computing Machinery.

Fatemeh Pesaran Zadeh, Juyeon Kim, Jin-Hwa Kim, and Gunhee Kim. 2024. Text2Chart31: Instruction tuning for chart generation with automatic feedback. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11459–11480, Miami, Florida, USA. Association for Computational Linguistics.

Leigang Qu, Haochuan Li, Wenjie Wang, Xiang Liu, Juncheng Li, Liqiang Nie, and Tat-Seng Chua. 2025. Silmm: Self-improving large multimodal models for compositional text-to-image generation. *Preprint*, arXiv:2412.05818.

Ram Seshadri. 2020. Github - autoviml/autoviz: Automatically visualize any dataset, any size with a single line of code. `https://github.com/AutoViML/AutoViz`. Source code repository.

Kumar Shridhar, Koustuv Sinha, Andrew Cohen, Tianlu Wang, Ping Yu, Ramakanth Pasunuru, Mrinmaya Sachan, Jason Weston, and Asli Celikyilmaz. 2024. The ART of LLM refinement: Ask, refine, and trust. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5872–5883, Mexico City, Mexico. Association for Computational Linguistics.

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

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023. Visual chatgpt: Talking, drawing and editing with visual foundation models. *Preprint*, arXiv:2303.04671.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, and 10 others. 2023. The rise and potential of large language model based agents: A survey. *Preprint*, arXiv:2309.07864.

Zhengyuan Yang, Linjie Li, Kevin Lin, Jianfeng Wang, Chung-Ching Lin, Zicheng Liu, and Lijuan Wang. 2023. The dawn of lmms: Preliminary explorations with gpt-4v(ision). *Preprint*, arXiv:2309.17421.

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

Junjie Ye, Xuanting Chen, Nuo Xu, Can Zu, Zekai Shao, Shichun Liu, Yuhan Cui, Zeyang Zhou, Chao Gong, Yang Shen, Jie Zhou, Siming Chen, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. A comprehensive capability analysis of gpt-3 and gpt-3.5 series models. *Preprint*, arXiv:2303.10420.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with BERT. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

A Prompt Configurations for AMACE

This appendix presents the detailed prompt settings used in each module of the AMACE framework. The AMACE consists of three agents: the CCG, CR, and CQE. The prompt examples below illustrate how each agent is designed to interact with inputs and generate corresponding outputs in the overall chart generation process.

A.1 Chart Code Generator (CCG) Prompting

The CCG is responsible for generating chart code from textual data. Figure 10 illustrates the initial prompt where CCG receives text-formatted table data and returns chart code using a language model.

Chart Code Generator Prompt

You are python programmer.

Table:< Table data >

Please provide a code to visualize the table dataset according to the following conditions:
1. Create a Python code with pandas, numpy and matplotlib.
2. Visualized table will utilize for < Query >.
3. Name the variable for the table as df.
4. Provide only code. End with 'plt.savefig(img_path, format='png')'

Code:

Figure 10: Chart Code Generator(CCG) prompt template: CCG receives text-formatted table data as input, processed by a language model.

Chart Code Regenerate Prompt

You are python programmer.

Table:< Table data >

Please provide a code to visualize the table dataset according to the following conditions:
1. Create a Python code with pandas, numpy and matplotlib.
2. Visualized table will utilize for < Query >.
3. Name the variable for the table as df.
4. Provide only code. End with 'plt.savefig(img_path, format='png')'

Code:

< Previous code >

The resulting in
< Chart Replier answer > and
< Chart Quality Evaluator feedback >.
Recreate a Python code.

Figure 11: Chart Code Generator(CCG) prompt template for regenerating the code. In the prompt, both the answers from the Chart Replier and the Chart Quality Evaluator, as well as the output from the previous CCG step, are included as inputs.

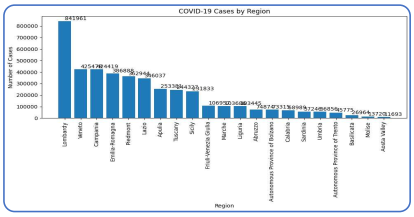
Figure 11 shows the regenerated code prompt, which includes previous CCG outputs along with responses from the CR and CQE modules to improve the result through iteration.

A.2 Chart Replier (CR) Prompting

The CR interprets the generated chart and provides textual responses. As shown in Figure 12, CR processes multimodal inputs—specifically, the generated chart image along with a corresponding question—allowing the language model to provide a human-like answer.

Chart Replier Prompt

< Chart image >



Please answer < Query > the chart image.

Figure 12: Chart Replier(CR) prompt template: CR takes inputs in a multimodal format, incorporating both text and images simultaneously.

Chart Quality Evaluator Prompt

Determine if the student's solution is correct or not .

Problem Statement:< Table data >

Please answer < Query > the table.

Student's Solution:
< Chart Replier answer >

Figure 13: Chart Quality Evaluator(CQE) prompt template: CQE evaluates the Chart Replier as if it were a student, conducting assessments accordingly.

A.3 Chart Quality Evaluator (CQE) Prompting

The CQE evaluates the quality of CR outputs. As depicted in Figure 13, CQE is prompted to behave like a grader, assessing whether the CR's response is accurate and informative based on the question and image input.

B Experiments Setting

Dataset Table 3 contains information about the datasets used in the experiments, including the types and quantities of data, as well as the dataset creation process. *ChartQA:human* represents questions and answers created by humans based on charts, while *ChartQA:augmented* refers to AI-generated questions and answers.

Performance Setting ChartQA involves answering questions based on charts, allowing us to evaluate the accuracy of multimodal responses by determining if there are match answers. Meanwhile, Chart-to-Text is evaluated using widely adopted metrics

Dataset	# of Table type	# of Table data	# of Query
ChartQA:augmented	2	987	1,250
ChartQA:human	2	625	1,250
Chart-to-Text	2	1,222	1,222

Table 3: Information about the test datasets used in the experiments. There are two types of tables: multi-column and single-column. In the description task, Chart-to-Text, the number of queries matches the number of table data entries. In the case of ChartQA, there may be more than one query per table data entry, resulting in an unequal count. ChartQA:human refers to queries written by humans, while ChartQA:augmented indicates augmented queries generated by the LLM.

such as *ROUGE-L (Lin, 2004)* and *BERTScore (Zhang et al., 2020)*, which assess the performance of multimodal Language Models in generating descriptions for charts. ROUGEScore refers to the F1 score of ROUGE-L. The performance of the CQE in distinguishing between correct and incorrect feedback is termed as *Feedback Accuracy*.

Loop setting In our experiment, we evaluate each component of the system. Primarily, we assess the performance of the Chart Generation module, followed by evaluations of the CR and the CQE. We also validate the Early Stopping Checker, which serves as the condition for terminating the loop. To account for potential errors in LLM output, each agent is allowed up to three retries, and the loop is set to stop after five steps.

Experiments Setting We utilized the AMACE-1 and AMACE-1o models as foundation models via API, while the other open LLMs were run on Nvidia V100 GPUs. Performance for AMACE-1 and AMACE-1o was obtained through five experimental runs, whereas the open LLM experiments were conducted only once due to resource and time constraints in the local environment.

Human Evaluation Five participants, all with university degrees in computer science, were recruited solely to evaluate chart quality and generation time.

C Experiments Performance

In the appendix, the tables provide numerical performance results for various experiments.

Table 4 provides the data for the experiments conducted on CR and CQE, as shown in Figure 4, as well as the performance with and without Early Stopping, depicted in Figure 9. Table 5 provides the performance results from the ablation study, where the Chart Replier model was replaced based on AMACE-1o, as shown in Figure 7.

AMACE Open Experiments We conducted experiments with open LLMs using the following configuration: AMACE Open employed Llama-3.1-8B (Llama Team, 2024) as both the CCG and CQE, while ChartLlama (Han et al., 2023) served as the CR. Table 4 presents the results of three experiments conducted on the same datasets as the original AMACE. These results, which show lower performance compared to the other AMACE, were not included in the main paper.

Original Performance In table 4, the original performance refers to the performance on charts created by humans, which are included in the original dataset. These charts were manually generated, and the dataset questions and answers were based on them. For example, when asked about the value of a specific-colored bar, AMACE sometimes failed to generate an appropriate chart due to a lack of prior knowledge. As a result, the original performance was higher than others. Considering this, the performance gap between *Human* and non-*Human* results is evaluated as small.

D Chart Generation Examples

Figure 14 presents example of the charts generated at each step using the Chart Generation module for the data "the car sales comparison for various manufacturers between July 2014 and January 2017." Additionally, we present an example of the outputs from LLM agents during step 1 and Chart Code Generator output on step 2.

Figure 15 shows that AMACE and *Human Process* generate nearly identical charts. In *Human Process*, *Step 2* introduces legends and numerical values, while *Step 3 and 4* adjust font styles and text positions. In contrast, AMACE-1o modifies numerical values in *Step 2* and adjusts titles and overall layout in *Step 3*. These refinements reflect the detailed adjustments made by humans.

Table 6 presents the type classification of charts generated by AMACE and the original charts. Chart types include bar, line, scatter, box, pie, and radar, classified using a multimodal LLM and verified through manual review. AMACE generated a distribution similar to the original charts, predominantly producing bar and line charts while also exploring other styles.

Model	Dataset	Performance	Step 1	Step 2	Step 3	Step 4	Step 5	Original
AMACE-1	ChartQA: augmented	Accuracy	0.2032	0.3435	0.4177	0.4676	0.5011	0.6335
		Accuracy w/ Early Stopping	0.3098	0.4346	0.4967	0.5314	0.5513	
		Feedback Accuracy	0.3151	0.4396	0.5115	0.5422	0.5875	-
	ChartQA: human	Accuracy	0.3054	0.3348	0.3479	0.3544	0.3525	0.5559
		Accuracy w/ Early Stopping	0.3611	0.4463	0.4862	0.5106	0.5255	
		Feedback Accuracy	0.3757	0.4215	0.4295	0.4485	0.4400	-
	Chart-to-Text	ROUGE-L	0.2174	0.2198	0.2204	0.2227	0.2224	0.3081
		ROUGE-L w/ Early Stopping	0.1295	0.1778	0.2022	0.2146	0.2249	
		BERTScore	0.6282	0.6291	0.6295	0.6297	0.6314	0.7099
		BERTScore w/ Early Stopping	0.3580	0.4942	0.5645	0.6007	0.6308	
		Feedback Accuracy	0.3994	0.4156	0.4267	0.4203	0.4449	-
AMACE-1o	ChartQA: augmented	Accuracy	0.2632	0.4392	0.4749	0.4819	0.4772	0.5883
		Accuracy w/ Early Stopping	0.2711	0.4218	0.4529	0.4650	0.4714	
		Feedback Accuracy	0.4835	0.7158	0.7640	0.7879	0.8052	-
	ChartQA: human	Accuracy	0.3486	0.4092	0.4228	0.4334	0.4261	0.6127
		Accuracy w/ Early Stopping	0.3460	0.4509	0.4867	0.5105	0.5238	
		Feedback Accuracy	0.4023	0.4780	0.4798	0.5002	0.5025	-
	Chart-to-Text	ROUGE-L	0.2366	0.2379	0.2391	0.2396	0.2409	0.2965
		ROUGE-L w/ Early Stopping	0.1379	0.1910	0.2181	0.2339	0.2426	
		BERTScore	0.6425	0.6457	0.6467	0.6482	0.6496	0.7087
		BERTScore w/ Early Stopping	0.3638	0.5070	0.5801	0.6240	0.6476	
		Feedback Accuracy	0.4464	0.4767	0.4940	0.5184	0.5271	-
AMACE Open	ChartQA: augmented	Accuracy	0.0242	0.0261	0.0276	0.0272	0.0288	0.0243
		Accuracy w/ Early Stopping	0.0340	0.0298	0.0362	0.0085	0.0196	
		Feedback Accuracy	0.5142	0.5179	0.5172	0.5182	0.5061	-
	ChartQA: human	Accuracy	0.0509	0.0579	0.0597	0.0649	0.0619	0.0528
		Accuracy w/ Early Stopping	0.0556	0.0505	0.0500	0.0828	0.0654	
		Feedback Accuracy	0.4458	0.4385	0.4364	0.4285	0.4632	-
	Chart-to-Text	ROUGE-L	0.0647	0.0650	0.0647	0.0633	0.0645	0.0641
		ROUGE-L w/ Early Stopping	0.0555	0.0546	0.0547	0.0565	0.0542	
		BERTScore	0.3751	0.3784	0.3783	0.3729	0.3786	0.3765
		BERTScore w/ Early Stopping	0.3579	0.3573	0.3664	0.3667	0.3393	
		Feedback Accuracy	0.4657	0.4882	0.4579	0.4779	0.4462	-

Table 4: The detailed performance metrics of the Chart Replier and Chart Quality Evaluator experiments, as illustrated in Figure 4 and performance comparison with and without Early Stopping, depicted in Figure 9. The *Original* performance was evaluated based on chart images included in the dataset created by humans, with three iterations conducted.

Chart Code Generator	Chart Replier	Chart Quality Evaluator	ChartQA:augmented	ChartQA:human	ROUGEScore	BERTScore
GPT-4o mini	Gemini 1.0 Pro	GPT-4o mini	0.629	0.366	0.233	0.640
GPT-4o	Gemini 1.0 Pro	GPT-4o mini	0.577	0.390	0.235	0.642
GPT-4o mini	Gemini 1.5 Flash	GPT-4o mini	0.500	0.360	0.237	0.649
GPT-4o mini	Gemini 1.0 Pro	GPT-4o	0.595	0.394	0.235	0.642

Table 5: The detailed performance metrics of the three roles on performance, as illustrated in Figure 7. Performance results are shown in Figure 7. In ChartQA, performance represents the accuracy of each dataset. In Chart-to-Text, ROUGEScore refers to the ROUGE-L F1 score. Experiments were conducted on 200 data samples for each dataset.

E Cost and Token Efficiency Analysis of Early Stopping

We compare the cost and token efficiency of applying the Early Stopping Checker in the AMACE frameworks across three datasets: ChartQA (augmented), ChartQA (human), and Chart-to-Text. It should be noted that the data is based on GPT models due to the unavailability of token-level cost data from Gemini.

As shown in Table 7, applying the Early Stopping Checker significantly reduces the monetary cost per data sample across all settings. For instance, in AMACE-1, the cost drops from 0.0483 $ to 0.0195 $ on the Chart-to-Text dataset, achieving a reduction of up to 59.6%. Similar reductions are observed in both human-authored and augmented ChartQA datasets. Even the lighter AMACE-1o model demonstrates consistent savings, with cost

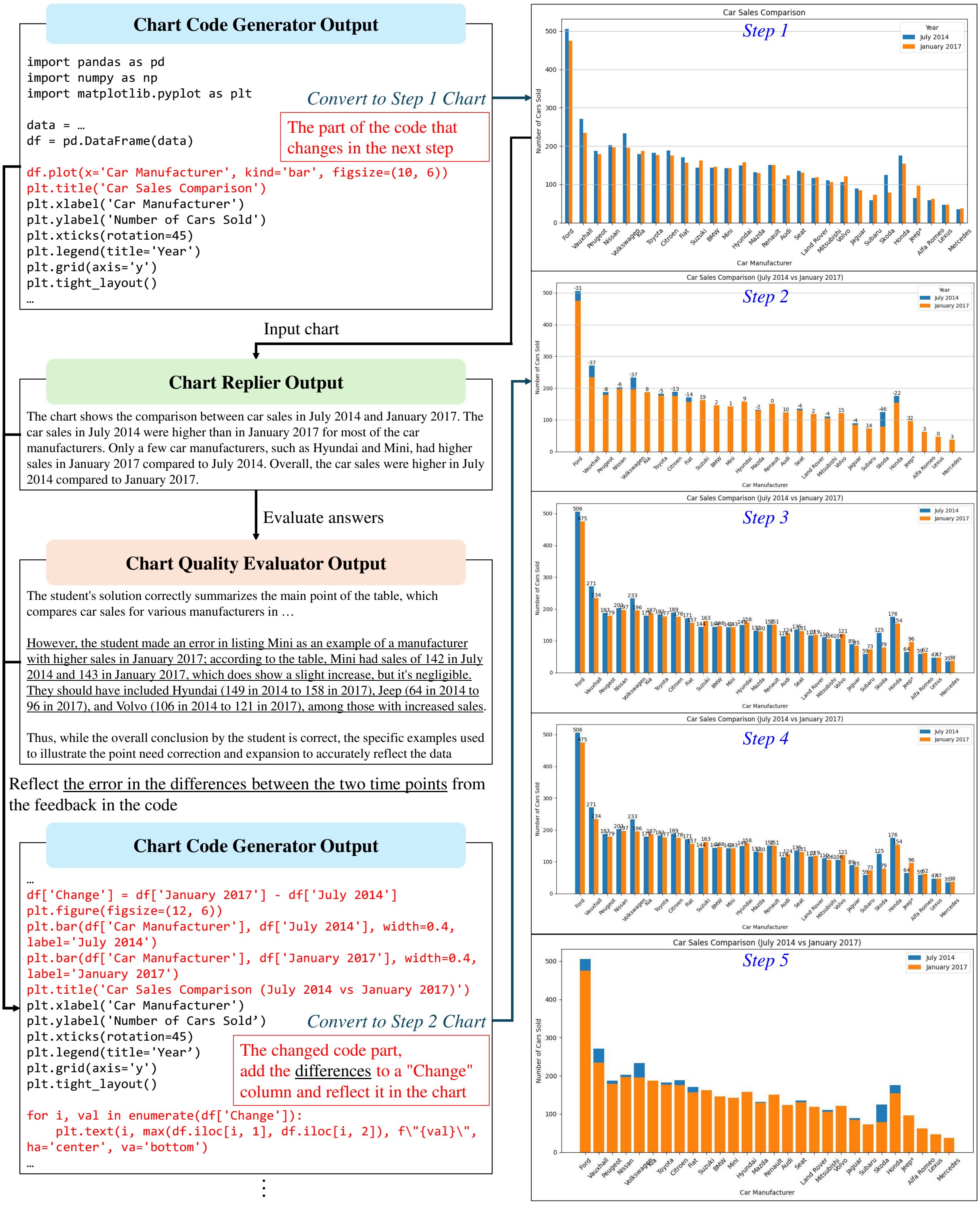


Chart Code Generator Output

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

data = …
df = pd.DataFrame(data)

df.plot(x='Car Manufacturer', kind='bar', figsize=(10, 6))
plt.title('Car Sales Comparison')
plt.xlabel('Car Manufacturer')
plt.ylabel('Number of Cars Sold')
plt.xticks(rotation=45)
plt.legend(title='Year')
plt.grid(axis='y')
plt.tight_layout()
…
```

Convert to Step 1 Chart

The part of the code that changes in the next step

Input chart

Chart Replier Output

The chart shows the comparison between car sales in July 2014 and January 2017. The car sales in July 2014 were higher than in January 2017 for most of the car manufacturers. Only a few car manufacturers, such as Hyundai and Mini, had higher sales in January 2017 compared to July 2014. Overall, the car sales were higher in July 2014 compared to January 2017.

Evaluate answers

Chart Quality Evaluator Output

The student's solution correctly summarizes the main point of the table, which compares car sales for various manufacturers in …

However, the student made an error in listing Mini as an example of a manufacturer with higher sales in January 2017; according to the table, Mini had sales of 142 in July 2014 and 143 in January 2017, which does show a slight increase, but it's negligible. They should have included Hyundai (149 in 2014 to 158 in 2017), Jeep (64 in 2014 to 96 in 2017), and Volvo (106 in 2014 to 121 in 2017), among those with increased sales.

Thus, while the overall conclusion by the student is correct, the specific examples used to illustrate the point need correction and expansion to accurately reflect the data

Reflect the error in the differences between the two time points from the feedback in the code

Chart Code Generator Output

```
…
df['Change'] = df['January 2017'] - df['July 2014']
plt.figure(figsize=(12, 6))
plt.bar(df['Car Manufacturer'], df['July 2014'], width=0.4,
label='July 2014')
plt.bar(df['Car Manufacturer'], df['January 2017'], width=0.4,
label='January 2017')
plt.title('Car Sales Comparison (July 2014 vs January 2017)')
plt.xlabel('Car Manufacturer')
plt.ylabel('Number of Cars Sold’)
plt.xticks(rotation=45)
plt.legend(title='Year’)
plt.grid(axis='y')
plt.tight_layout()

for i, val in enumerate(df['Change']):
    plt.text(i, max(df.iloc[i, 1], df.iloc[i, 2]), f\"{val}\",
ha='center', va='bottom')
…
```

Convert to Step 2 Chart

The changed code part, add the differences to a "Change" column and reflect it in the chart

⋮

Figure 14: Charts generated by the Chart Generation module and the process from Step 1 to Step 2. Each step visualizes the July 2014 and January 2017 Car Sales Comparison, displaying sales per manufacturer. The red text indicates the modified code from the previous step, while the underlined text highlights the identified feedback. The red text in the Chart Code Generator Output of Step 1 reflects the feedback from the Chart Replier and Chart Quality Evaluator, which identified the error in the differences between the two time points, resulting in the updated red text in the Chart Code Generator Output of Step 2.

reductions up to 54.8%.

Table 8 presents a marked decrease in the number of input tokens per data point when Early Stopping is enabled. Notably, AMACE-1 on the Chart-to-Text dataset shows a reduction from 52.33 to 15.77 tokens (69.9% decrease). Across the board,

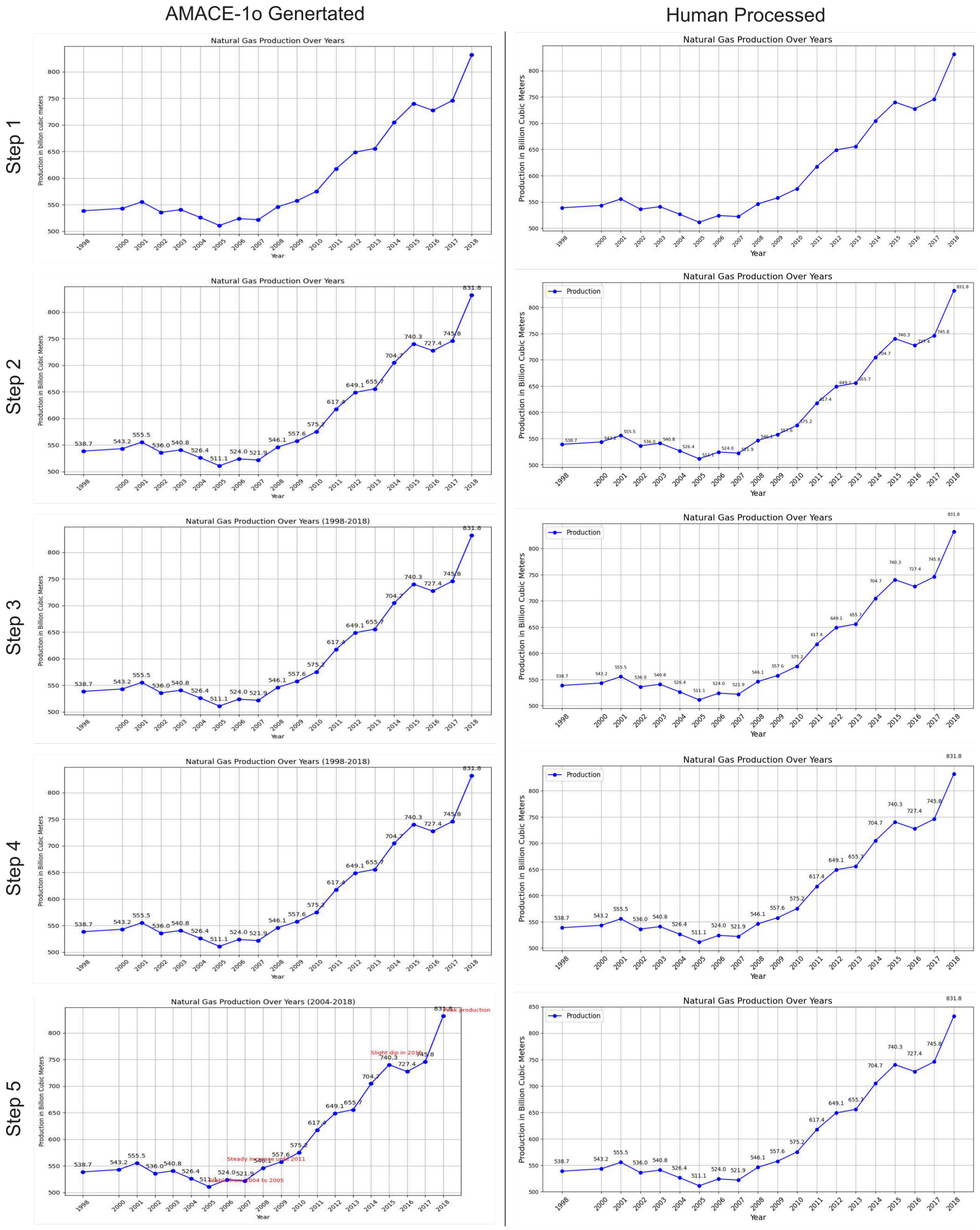


Figure 15: Step-by-step charts generated by AMACE-1o (left) and human-generated charts using prompt engineering, denoted as *Human Process* (right) using prompt engineering. Each chart is generated based on the same data.

the Early Stopping mechanism reduces unnecessary prompt inflation, improving processing efficiency.

Table 9 further highlights that the use of the Early Stopping Checker also reduces the output length, contributing to lower latency and cost. For example, the output token count in AMACE-1 on Chart-to-Text decreases by 62.3%, from 25.53 to 9.63 tokens. These improvements suggest that the Early Stopping Checker encourages more focused and concise responses without negatively affecting performance.

Dataset	Models	Bar	Line	Scatter	Box	Pie	Radar
Chart-to-Text	Original	44.70	55.30	0	0	0	0
	AMACE-1	42.80	56.38	0.29	0	0.51	0.02
	AMACE-1o	47.58	52.25	0	0	0.17	0
ChartQA	Original	80.73	14.09	0	0	5.18	0
ChartQA: augmented	AMACE-1	71.13	28.33	0.13	0	0.41	0
	AMACE-1o	63.00	36.73	0.02	0	0.24	0
ChartQA: human	AMACE-1	75.40	21.66	0.59	0	2.30	0.05
	AMACE-1o	77.73	20.08	0.51	0	1.68	0

Table 6: The type classification of charts generated by AMACE models compared to original charts in the dataset. Chart types are categorized as bar, line, scatter, box, pie, and radar.

Model	Dataset	w/o Early Stopping Checker	w/ Early Stopping Checker
AMACE -1	ChartQA:augmented	0.0238 $	0.0117 $ (↓ 51.0%)
	ChartQA:human	0.0391 $	0.0206 $ (↓ 47.3%)
	Chart-to-Text	0.0483 $	0.0195 $ (↓ 59.6%)
AMACE -1o	ChartQA:augmented	0.0047 $	0.0021 $ (↓ 54.8%)
	ChartQA:human	0.0053 $	0.0039 $ (↓ 26.5%)
	Chart-to-Text	0.0094 $	0.0069 $ (↓ 26.2%)

Table 7: Cost per data sample with and without Early Stopping Checker.

Model	Dataset	w/o Early Stopping Checker	w/ Early Stopping Checker
AMACE -1	ChartQA:augmented	35.45	15.27 (↓ 56.9%)
	ChartQA:human	41.89	18.84 (↓ 55.0%)
	Chart-to-Text	52.33	15.77 (↓ 69.9%)
AMACE -1o	ChartQA:augmented	35.00	16.54 (↓ 52.7%)
	ChartQA:human	34.22	23.60 (↓ 31.0%)
	Chart-to-Text	47.36	32.60 (↓ 31.2%)

Table 8: Input token usage per data sample.

Model	Dataset	w/o Early Stopping Checker	w/ Early Stopping Checker
AMACE -1	ChartQA:augmented	16.11	7.88 (↓ 51.1%)
	ChartQA:human	21.60	11.26 (↓ 47.9%)
	Chart-to-Text	25.53	9.63 (↓ 62.3%)
AMACE -1o	ChartQA:augmented	17.33	9.03 (↓ 47.9%)
	ChartQA:human	19.52	14.16 (↓ 27.5%)
	Chart-to-Text	20.92	14.79 (↓ 29.3%)

Table 9: Output token usage per data sample.

These results collectively demonstrate that incorporating Early Stopping into the inference process offers substantial efficiency gains in terms of cost and token usage, which are critical for scalable and economical deployment of LLM-based systems.