SciSketch: An Open-source Framework for Automated Schematic Diagram Generation in Scientific Papers

Zihang Wang [Y*] **Yilun Zhao** [Y*] **Kaiyan Zhang** [Y]
Chen Zhao [N] **Manasi Patwardhan** [T] **Arman Cohan** [Y]
[Y] Yale University [T] TCS Research [N] New York University

https://github.com/yale-nlp/SciSketch

Abstract

High-quality schematic diagrams, which provide a conceptual overview of the research, play a crucial role in summarizing and clarifying a study's core ideas. However, creating these diagrams is time-consuming for authors and remains challenging for current AI systems, as it requires both a deep understanding of the paper's content and a strong sense of visual design. To address this, we introduce SciSketch, an open-source framework that supports two automated workflows for schematic diagram generation using foundation models, shown in Figure 1. 1) In the graphic-code-based workflow, SciSketch follows a two-stage pipeline: it first produces a layout plan expressed in a graphical code language with a self-refinement and self-verification mechanism. It then integrates empirical images and symbolic icons to create a visually coherent, informative diagram. 2) In the image-based workflow, SciSketch directly synthesizes the diagram image through image generation with a self-refinement mechanism. Through both automatic and human evaluations, we show that SciSketch outperforms several state-of-the-art foundation models, including GPT-4o, Claude-3.7-Sonnet and Gemini-2.5-Pro, in generating schematic diagrams for scientific papers. We make SciSketch fully open-sourced, providing researchers with an accessible, extensible tool for high-quality schematic diagram generation in scientific fields.

1 Introduction

Schematic diagrams have long been recognized as a critical component in scientific and engineering research (Larkin and Simon, 1987). High-quality schematic diagrams can effectively convey complex information in scientific paper through structured visual representations, enabling readers to grasp key concepts more efficiently and accurately.

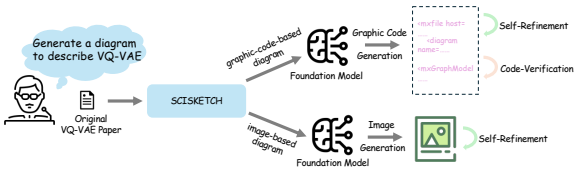

Figure 1: Overview of SciSketch framework

While recent works (Belouadi et al., 2024; Zala et al., 2024; Wei et al., 2025) have explored the generation of diagram from short textual descriptions (typically under 50 words), they fall short of addressing the more challenging task of generating schematic diagrams directly from longer contexts—specifically, full-length scientific papers.

This task is both challenging and important for several reasons. First, scientific papers contain dense, domain-specific language and complex multi-section structures, which require models to understand not just isolated sentences but also the broader context and interdependencies across sections. Second, identifying what constitutes schematic-worthy content involves deep semantic understanding and abstraction, as the relevant information is often scattered and implicitly stated. Third, the generated diagrams must not only be visually coherent but also semantically faithful to the source material, which imposes strict constraints on layout, labeling, and relational structure.

To address these challenges and support researcher's productivity, we introduce SciSketch, an open-source framework powered by foundation models for sketching schematic diagrams in scientific papers. It aims to reduce the manual effort involved in creating high-quality, informative visuals for scientific research. As illustrated in Figure 2 and Figure 3, SciSketch supports two workflows: In the graphic-code-based workflow, it operates in two key phases: (1) Layout planning, which comprises a text analysis module, a layout refinement module, and a graphic code verification module to

*Equal Contributions. Correspondence: Yilun Zhao (yilun.zhao@yale.edu).

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 403–417
November 4-9, 2025

produce a coherent layout plan (Feng et al., 2023; Lian et al., 2023; Cho et al., 2023). (2) Diagram generation, where an element discriminator module collaborates with an empirical figure retrieval module and an SVG generation module to enrich the layout plan with visual elements, ultimately producing the final schematic diagram that enhances both visual clarity and semantic richness. In the image-based workflow, the framework uses a text analysis module and an image refinement module to synthesize a coherent diagram image directly.

To evaluate the effectiveness of diagram generation, we introduce an automated assessment method that leverages multimodal foundation models. Our validation shows that the model's evaluations closely align with human judgments. Through extensive experiments and analysis, we demonstrate that the SciSketch framework surpasses state-of-the-art models, including GPT-4o, Claude-3.7-Sonnet, and Gemini-2.5-Pro, in both the quality and usability of the generated diagrams.

2 SciSketch Framework

Given a scientific paper, SciSketch is designed to automatically generate schematic diagrams based on user requests in two workflows.

2.1 Graphic-code-based Workflow

In the graphic-code-based workflow: it coordinates multiple modules in a workflow consisting of two main phases: **Layout Planning** and **Diagram Generation**, which we describe in detail below.

2.1.1 Layout Planning

At a high level, the layout planning phase begins by interpreting the user's intent and the provided scientific paper, producing a textual description of the target diagram. This description is then translated into structured graphic code, iteratively refined for clarity and accuracy, and finally verified to ensure syntactic correctness and adherence to the target specification. We detail the design as follows:

Text Analysis. Unlike typical text-to-image generation tasks, where an image is generated based on a short description, scientific schematic diagrams generation requires reasoning over long-form inputs, such as entire scientific papers that may span dozens of pages. To accurately capture the user's intent and extract relevant content, we employ a foundation model (specifically GPT-4o[1] for all SciSketch modules involving foundation models) to analyze the full text. Based on the user's request, the model generates a structured textual description of the target diagram layout, detailing key components and their relationships. This intermediate representation serves as a high-level blueprint, guiding the subsequent diagram generation process rather than generating directly from the raw paper.

Layout Planning. Schematic diagrams are typically represented using vector graphics due to their scalability and clarity. We adopt an XML-based representation, specifically the XML schema used by draw.io, which offers both programmatic control and ease of manual refinement. Given the textual description produced in the previous stage, we prompt a foundation model to generate the corresponding XML code that structurally encodes the diagram as the initial diagram layout.

Layout Refinement. Foundation models can enhance their outputs through iterative self-evaluation (Madaan et al., 2023). To improve the precision and quality of the initial layout, we prompt the foundation model to critically assess its own output. This self-evaluation checks for missing elements, ambiguous or inconsistent connections, and opportunities for aesthetic improvement—while maintaining semantic fidelity. Based on its own feedback, the foundation model is prompted to revise the layout as needed. If the feedback indicates no changes are necessary, the current layout is accepted as final. This feedback-refinement loop continues for up to four iterations, ensuring a high-quality and semantically sound diagram layout.

Graphic Code Verification. The refined diagram layout plan in graphic code language may occasionally contain syntactic errors or hallucinated structures, resulting in invalid or uncompilable code. For instance, draw.io's XML schema requires each `<mxCell>` element (except the root) to have a parent attribute, and special characters must be properly escaped. To mitigate such issues, we employ an iterative verification process in which we attempt a foundation model to find potential errors. If any error is reported, we prompt the foundation model to produce the corrected diagram code. This loop continues until no error is reported.

[1]We use `gpt-4o-2024-11-20` for SciSketch due to its superior performance.

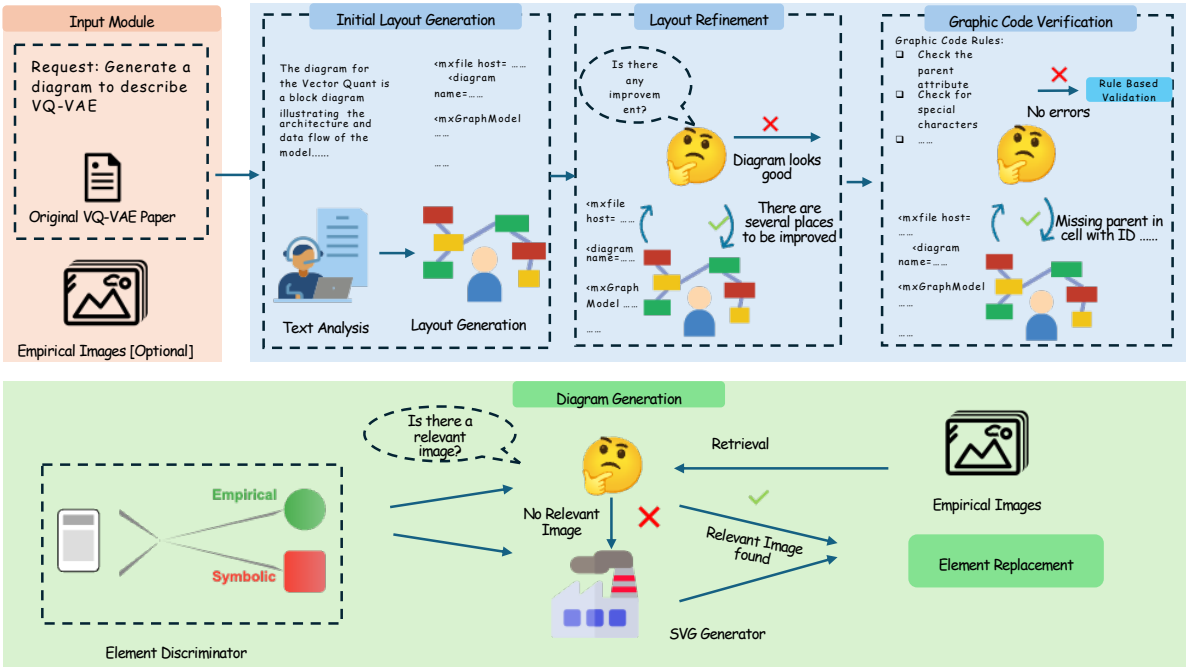

Figure 2: Overview of the graphic-code-based workflow of SciSketch two-stage framework. The first stage, *layout generation*, produces an initial diagram layout plan, which is iteratively refined by a layout refinement module to enhance structural coherence and completeness. The refined layout is then validated by a code verification module. In the second stage, *diagram generation*, an element discriminator identifies components suitable for replacement and routes them either to an empirical image retrieval module or an SVG generator—based on their type—to construct the final diagram.

2.1.2 Diagram Generation

Scientific schematic diagrams typically consist of two categories of visual elements: (1) *empirical figures*, which present data-driven content such as experimental results or input samples; and (2) *symbolic representations*, which illustrate abstract concepts like databases, models, or neural networks. Generating such diagrams requires the incorporation of domain-relevant visuals to enhance clarity and expressiveness. In the diagram generation phase, the discriminator module systematically analyzes the diagram layout to determine whether each component should be represented as a symbolic icon or an empirical figure. Identified components are then handled by either a symbolic generator or an empirical figure retrieval module respectively.

Element Discriminator. The element discriminator takes the verified diagram layout as input and determines a replacement plan by prompting a foundation model. The output is a structured data representation containing four fields for each element: `id`, `value`, `type`, and `description`. The `id` serves as a unique identifier to locate the corresponding element within the diagram. The `type` field is critical, as it dictates the subsequent processing step: if the type is empirical, the element is forwarded to the empirical image retrieval module; if symbolic, it is passed to the symbolic generator.

Symbolic Generator. Scalable Vector Graphics (SVGs) are widely used for symbolic icons due to their high precision, consistent visual quality across varying resolutions, compact file size and ease of editing. The symbolic generator module takes the element's `value` and `description` as input and instruct GPT-4o to produce a simple yet informative SVG icon to replace the corresponding element in diagram.

Empirical Figure Retrieval. For elements identified as empirical, the empirical figure retrieval module attempts to match them with relevant figures provided by the user. Authors may optionally supply a set of candidate figures as input to the system. The module compares each element's `value` with the filenames or associated metadata of the provided figures. If a suitable match is found, the image is selected as the replacement. If no relevant figure is identified, the element is passed to

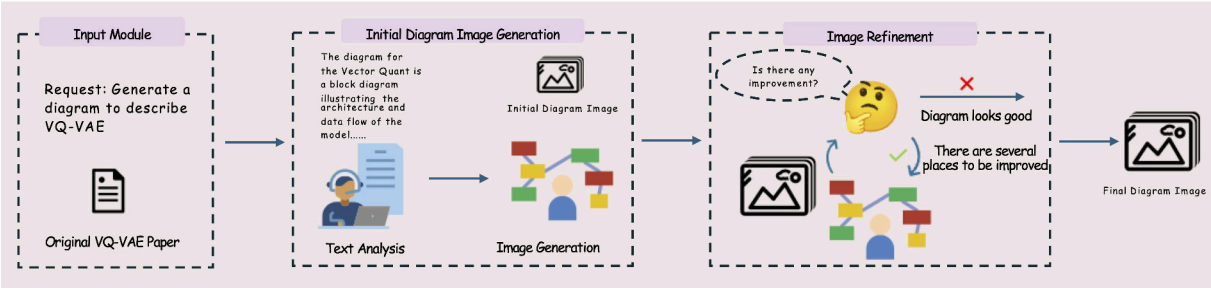

Figure 3: Overview of the image-based workflow of SCISKETCH two-stage framework. An initial diagram image is synthesized using an image generation API. This image is then iteratively refined under the guidance of an image refinement module to improve clarity and structure.

the symbolic form generator module to produce a fallback symbolic representation.

2.2 Image-based Workflow

In the image-based workflow: multiple modules are coordinated to synthesize the diagram image. Similar to the layout planning phase in the graphic-code-based workflow, the process begins with generating a textual description of the target diagram. However, rather than translating this description into graphical code, the image-based workflow leverages a foundation model to generate a diagram image directly from the description. The image is then refined iteratively for clarity and accuracy.

3 Experiment

3.1 Evaluation Data

To construct our evaluation set within a limited budget, we carefully curated 40 scientific papers from Hugging Face Daily Papers[2] and arXiv. The selection includes a balance mix of classic papers and recently published papers in computer science. Each selected paper contains a clear conceptual pipeline figure, which we use as the reference for evaluating diagram generation quality.

3.2 Evaluation Protocol

Evaluation Criteria. We evaluate the quality of the generated schematic diagram using three criteria: (1) Completeness — whether all key components described in the source paper are represented in the diagram, with no critical elements omitted; (2) Logical Consistency — whether the relationships and connections between components accurately reflect the logical structure and intended semantics of the paper; and (3) Aesthetic Quality — the visual clarity of the diagram, including layout organization, appropriate spacing, and the absence of overlapping or misaligned elements.

Automated Evaluation. Leveraging foundation models as judges has received increasing attention and adoption in recent work (Zheng et al., 2023). As demonstrated in the previous LLM-as-a-judge work, strong LLMs like GPT-4 show very high agreements with human experts. We employ GPT-4o to assess the quality of generated diagrams by comparing them to their corresponding reference diagrams from the original papers. For each evaluation criterion—Completeness, Logical Consistency, and Aesthetic Quality—the model is prompted to first provide a rationale for its assessment, followed by a score on a 1–5 scale, where 1 denotes the lowest quality and 5 denotes the highest.

Human Evaluation. We employed four evaluators with Master's degree in Computer Science to assess the quality of the generated diagrams. We randomly sampled 20 papers from our evaluation set. We ask the evaluator to assign a score from 1 to 5 to each generated diagram with regarding to the three criteria, Completeness, Logical Consistency and Aesthetic Quality. We document the definition of each metric in detail in Appendix C. We provide the ground truth diagram from the original paper as the reference.

3.3 Baseline Systems

Due to the limited prior work on generating diagrams directly from full scientific papers, we compare our framework against state-of-the-art foundation models capable of handling document-level inputs, *i.e.,* GPT-4o, Claude-3.7-Sonnet, and Gemini-2.5-Pro. To ensure a fair comparison and avoid information leakage, we manually remove the original diagram from each paper before inputting it into the baseline models.

[2]https://huggingface.co/papers

System	Model-based Evaluation				Human Evaluation			
	Comp.	Logical Cons.	Aesthetic Quality	Average	Comp.	Logical Cons.	Aesthetic Quality	**Average**
GPT-4o	4.1	4.7	3.5	4.1	3.8	3.9	2.1	3.3
Gemini-2.5-Pro	4.5	4.7	2.4	3.9	4.1	4.0	2.5	3.5
Claude-3.7-Sonnet	4.2	4.2	4.4	4.3	4.2	4.3	3.6	4.0
SciSketch (drawio)	4.4	4.6	4.2	4.4	4.3	4.1	3.7	4.0
SciSketch (image)	4.4	4.0	4.6	4.3	4.3	3.8	4.3	4.1

Table 1: Evaluation of diagram quality across models by both model-based and human evaluation. Metrics include completeness, logical consistency, and aesthetic quality.

3.4 Experimental Results and Analysis

Main Findings. Table 1 presents the evaluation scores from both the multimodal model and human evaluators. Both sources indicate that the SciSketch framework achieves high scores. All methods—including the baselines and SciSketch—perform comparably well in terms of completeness and logical consistency. The most notable discrepancy lies in aesthetic quality, where human evaluators showed a clear preference for diagrams that incorporate visual elements.

In addition to quantitative evaluation, we also conducted a qualitative analysis of the generated diagrams. Selected examples from both the baseline models and our proposed framework are shown in Appendix A. With the latest model update on March 25th, GPT-4o is now capable of generating images. In most cases, its outputs consist of generic rectangular boxes with minimal structure, rendering them largely unusable in scientific contexts.

Gemini 2.5 Pro often produces diagrams in Markdown-inspired formats such as Mermaid. While it is capable of capturing logical structure, its outputs tend to be overly verbose and visually cluttered, making them less suitable for scientific papers. Claude 3.7 Sonnet attempts to use SVG representations and is generally more concise than Gemini. However, it lacks visual elements and is difficult to edit or post-process manually. Moreover, due to the syntax constraints of the graphical languages, the Mermaid code generated by Gemini often fails to render successfully on the first attempt and typically requires manual correction.

In contrast, our proposed framework generates diagrams that are both concise and semantically faithful to the original paper. It captures the key components while avoiding unnecessary elements that may distract the reader. Furthermore, by integrating symbolic icons and empirical images, the diagrams are better suited for inclusion in scientific papers. Additionally, in the graphic-code based workflow, the output is editable XML. Users can conveniently refine the diagrams post-generation to meet specific formatting or presentation needs.

Setting	Iteration Numbers	Executable Rate
w/o Verification	0	0.882
w Verification	0.118	1.0

Table 2: Average iteration counts and execution rates, illustrating the effectiveness of the verification module.

Effectiveness of the framework. All experiments are initially conducted using GPT-4o. To assess the robustness and generalizability of our framework, we replace GPT-4o with the open-source language model DeepSeek. We randomly sampled three papers for this comparison, and the results are presented in Appendix D. Despite with the open-sourced language model, the framework can still produce high-quality diagrams: the generated outputs effectively capture the core concepts of the papers and maintain logical coherence in the arrangement of components.

Ablation Study. Our framework uses three key modules to produce the final diagram layout: the text analysis module, the refinement module, and the code verification module. To evaluate the contribution of the verification module, we conduct an ablation experiment by generating diagrams without the verification module. As shown in Table 2, without verification, the code executable rate drops to 0.883. When the verification module is enabled, with an average of 0.118 correction iterations, the framework achieves a 100% code execution success rate. We also assess the impact of the refinement module. On average, each diagram generation involves 2.12 iterations of refinement. Each iteration introduces improvements such as adding missing components, adjusting layouts, or enhanc-

ing visual clarity through colors and highlights. Figure 10 shows an example. While the first iteration already produces a solid initial layout, the second refinement iteration groups multiple outputs into a single cluster, enhancing both clarity and readability.

Case Study. To further illustrate the strengths and limitations of SCISKETCH in both workflows, we conduct a detailed analysis of both successful and unsuccessful cases, as shown in Figure 11 and Figure 12.

- **Graphic-code-based Workflow:** The top example corresponds to the VQ-VAE paper and demonstrates a high-quality generation. The left side shows the original diagram from the paper. As seen in the generated version, the system successfully captures the core components—such as the encoder, decoder, and embedding table—and integrates both input/output data and symbolic icons, resulting in a visually appealing and informative diagram. Despite these strengths, the SCISKETCH framework still has some limitations. In the bottom example which corresponds to the CoCA paper, although the diagram correctly represents the two generative paths and includes an amplification process, the complexity of the logic makes it challenging for the system to produce a clear and structured layout.
- **Image-based Workflow:** The top example corresponds to the FABLES paper and demonstrates a high-quality generation. In addition to the same strengths of the graphic-code-based workflow, the image-based-workflow can generate a more coherent diagram with elements are placed harmoniously and the icons are consistent due to the direct generation property. However, this approach also has certain limitations. In the bottom example which corresponds to the TEMPLE-MQA paper, although the diagram successfully conveys the core idea, some words are misspelled in the image. Furthermore, due to the generative nature, occasional hallucinated elements are introduced.

These issues highlight open challenges and potential directions for improving the framework in future work.

4 Related Work

Recent advances in text-to-image generation have focused predominantly on producing photorealistic images (Ramesh et al., 2021; Saharia et al., 2022; Zhang et al., 2023; Dai et al., 2023; Chang et al., 2023). While these models achieve impressive visual fidelity, they offer limited utility for generating structured, symbolic representations such as scientific diagrams. In contrast, emerging work on vector graphic generation in SVG format (Frans et al., 2022; Jain et al., 2023; Wu et al., 2023, 2024) shows potential for creating simple illustrations. However, these approaches fall short in handling the complex reasoning and spatial planning required for scientific schematics.

More directly relevant efforts include AutomaTikZ (Belouadi et al., 2024), which fine-tunes foundation models to generate TikZ code for vector graphics, and DiagrammerGPT (Zala et al., 2024), which adopts a two-stage pipeline involving layout planning followed by diagram synthesis using diffusion models. Similarly, DiagramAgent (Wei et al., 2025) coordinates multiple specialized agents to plan, code, and edit diagrams, showcasing the promise of large language models in structured content generation. Despite these innovations, such systems are typically limited to brief prompts and struggle with the dense, interrelated content characteristic of full-length scientific papers. Moreover, they offer limited support for hybrid diagrams that integrate empirical data visualizations with symbolic or conceptual elements—core features of scientific schematics.

To our knowledge, SCISKETCH is the first open-source framework designed specifically for schematic diagram generation for scientific research, making it accessible and extensible for the broader research community.

5 Conclusion

This work tackles the challenge of generating schematic diagrams from full-length scientific papers. We introduce the first open-source framework specifically designed for this task, enabling the creation of scientific diagrams directly from full-text content. Our system outperforms SOTA models in both quantitative metrics and human evaluations. By automating diagram generation, our approach has the potential to significantly boost researchers' productivity across academia and industry, streamlining the creation of high-quality visualizations. Future work could focus on semantic abstraction of generated SVG icons and enhancing layout robustness for complex diagram structures.

Acknowledgments

This project is supported by Tata Sons Private Limited, Tata Consultancy Services Limited, and Titan.

References

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

Huiwen Chang, Han Zhang, Jarred Barber, Aaron Maschinot, Jose Lezama, Lu Jiang, Ming-Hsuan Yang, Kevin Patrick Murphy, William T. Freeman, Michael Rubinstein, Yuanzhen Li, and Dilip Krishnan. 2023. Muse: Text-to-image generation via masked generative transformers. In Proceedings of the 40th International Conference on Machine Learning, volume 202 of Proceedings of Machine Learning Research, pages 4055–4075. PMLR.

Jaemin Cho, Abhay Zala, and Mohit Bansal. 2023. Visual programming for step-by-step text-to-image generation and evaluation. In Advances in Neural Information Processing Systems, volume 36, pages 6048–6069. Curran Associates, Inc.

Xiaoliang Dai, Ji Hou, Chih-Yao Ma, Sam Tsai, Jialiang Wang, Rui Wang, Peizhao Zhang, Simon Vandenhende, Xiaofang Wang, Abhimanyu Dubey, Matthew Yu, Abhishek Kadian, Filip Radenovic, Dhruv Mahajan, Kunpeng Li, Yue Zhao, Vladan Petrovic, Mitesh Kumar Singh, Simran Motwani, Yi Wen, Yiwen Song, Roshan Sumbaly, Vignesh Ramanathan, Zijian He, Peter Vajda, and Devi Parikh. 2023. Emu: Enhancing image generation models using photogenic needles in a haystack. Preprint, arXiv:2309.15807.

Weixi Feng, Wanrong Zhu, Tsu-jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. 2023. Layoutgpt: Compositional visual planning and generation with large language models. Advances in Neural Information Processing Systems, 36:18225–18250.

Kevin Frans, Lisa Soros, and Olaf Witkowski. 2022. Clipdraw: Exploring text-to-drawing synthesis through language-image encoders. Advances in Neural Information Processing Systems, 35:5207–5218.

Ajay Jain, Amber Xie, and Pieter Abbeel. 2023. Vectorfusion: Text-to-svg by abstracting pixel-based diffusion models. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 1911–1920.

Jill H Larkin and Herbert A Simon. 1987. Why a diagram is (sometimes) worth ten thousand words. Cognitive science, 11(1):65–100.

Long Lian, Boyi Li, Adam Yala, and Trevor Darrell. 2023. Llm-grounded diffusion: Enhancing prompt understanding of text-to-image diffusion models with large language models. arXiv preprint arXiv:2305.13655.

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. In Advances in Neural Information Processing Systems, volume 36, pages 46534–46594. Curran Associates, Inc.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. In Proceedings of the 38th International Conference on Machine Learning, volume 139 of Proceedings of Machine Learning Research, pages 8821–8831. PMLR.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. 2022. Photorealistic text-to-image diffusion models with deep language understanding. In Advances in Neural Information Processing Systems, volume 35, pages 36479–36494. Curran Associates, Inc.

Jingxuan Wei, Cheng Tan, Qi Chen, Gaowei Wu, Siyuan Li, Zhangyang Gao, Linzhuang Sun, Bihui Yu, and Ruifeng Guo. 2025. From words to structured visuals: A benchmark and framework for text-to-diagram generation and editing. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.

Ronghuan Wu, Wanchao Su, and Jing Liao. 2024. Chat2svg: Vector graphics generation with large language models and image diffusion models. Preprint, arXiv:2411.16602.

Ronghuan Wu, Wanchao Su, Kede Ma, and Jing Liao. 2023. Iconshop: Text-guided vector icon synthesis with autoregressive transformers. ACM Transactions on Graphics (TOG), 42(6):1–14.

Abhay Zala, Han Lin, Jaemin Cho, and Mohit Bansal. 2024. Diagrammergpt: Generating open-domain, open-platform diagrams via llm planning. In COLM.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. 2023. Adding conditional control to text-to-image diffusion models. In Proceedings of the IEEE/CVF international conference on computer vision, pages 3836–3847.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang,

Joseph E Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In Advances in Neural Information Processing Systems, volume 36, pages 46595–46623. Curran Associates, Inc.

A Select Examples

Example diagrams generation results from state-of-the-art foundation models (GPT-4o, Gemini-2.5-Pro, Claude-3.7-Sonnet) and SciSketch are shown in Figure 4, Figure 5, Figure 6, Figure 7, Figure 8, Figure 9. Compared to the baselines, our method produces diagrams that better align with the core concepts and logical structure of the original paper, while also exhibiting higher aesthetic quality. Moreover, our framework captures essential components without including verbose or irrelevant elements, resulting in concise and clear diagrams that effectively convey the intended information.

B Prompts

B.1 Layout Planning Phase in Graphic-Code-Based Workflow

Text Analysis

```
You are an expert in describing a scientific diagram
given the caption and the content of the paper.
You should give a detailed description of the
diagram which will be used to generate the diagram.
Output the description within the <description>
</description> tag.
You should follow the following steps:
1. Read the paper comprehensively and extract all
the information that relates to the caption.
2. Based on the caption and the paper, decide the
type of the diagram.
3. Based on the caption, paper content and the type
of the diagram, identify the components and layout
of the diagram.
4. Generate a detailed description of the diagram.
The description should comepletely come from the
paper. And it should match the caption.
5. Check the description with the caption and the
paper, make sure the description is correct and
complete. If not, revise the description until it
is correct and complete.
6. Wrap the description within the <description>
</description> tag.
Paper Content: {paper_content}
Caption: {caption}
```

Layout Generation

```
You are an expert in generating a scientific diagram
based on the description of the diagram. You should
generate an executable code of {language} to fully
represent the description. After generation, you
need to verify to make sure it could be executed
without any error.
You could follow the following steps:
1. Analyze the description and identify the type,
components and layout of the diagram.
2. Generate the executable {language} code for all
the components and layout.
3. Make sure the code covers all the components of
the description.
4. Verify the code to make sure it could be
executed without any error.
5. Return your result in a code block wrapped with
“‘.
Think step by step and make sure the diagram is
clear and matches the description and the caption.
Description: {description}
Code:
```

Layout Refinement

```
You are an expert in planning and designing a
scientific diagram with the {language}. Given
the description, the caption, and the code of
the diagram, you need to check if there is any
improvement to make the diagram more precise and
appealing to the reader. You should follow the
following steps:
1. Analyze the description and the caption of the
diagram to understand the components and layout of
the diagram.
2. Analyze the code and consider how the scientific
diagram can be enhanced to make it match the
description and caption.
3. Consider how to arrange the components and
layout of the diagram to make it more appealing to
the reader.
4. Try to avoid overlap of components as much as
possible.
5. If the code is good enough, you should not make
any changes to the code.
You will first need to decide if any improvement is
needed. You should wrap your decision inside the
<decision></decision> tag.
You should only input yes or no inside the
<decision></decision> tag.
If the decision is yes, you should then generate the
improved code in a code block wrapped with “‘.
Think step by step and make sure the code is
self-contained and executable.
Description: {description}
Caption: {caption}
Diagram: {diagram}
```

Graphic Code Verification

```
You are an expert of {language}. You know the rules
and regulations of {language}. You will be given
the {language} code of the diagram. You should
verify the code to make sure it could be executed
without any error. If you identify any error, you
should fix the error and output the fixed code.
You will first decide if there is any error in the
code. You should wrap your decision inside the
<decision></decision> tag. You should only input
yes or no inside the <decision></decision> tag.
If there are errors in the code, you should then
generate the fixed code in a code block wrapped with
“‘.
Think step by step and make sure the code is
self-contained and executable.
Here are some rules to follow:
1. The child-parent relationship should be correct
-- no child should be the parent of itself or the
parent of its parent.
2. For each mxGeometry object, there should be an
as="geometry" attribute at the end.
3. Each mxCell should have a parent.
Diagram: {diagram}
```

B.2 Diagram Generation Phase in Graphic-Code-Based Workflow

Element Discriminator

```
You are a scientific researcher with professional
design skills. Given a drawio diagram which shows a
```

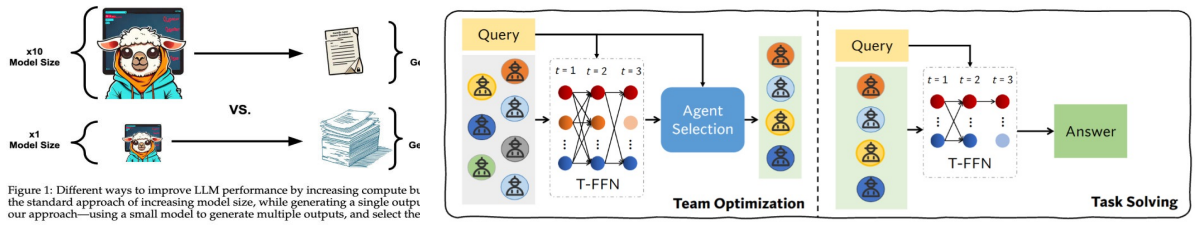

Figure 4: Example diagram from the original paper. The left is the Budget Relocation paper, the right is the DyLAN paper

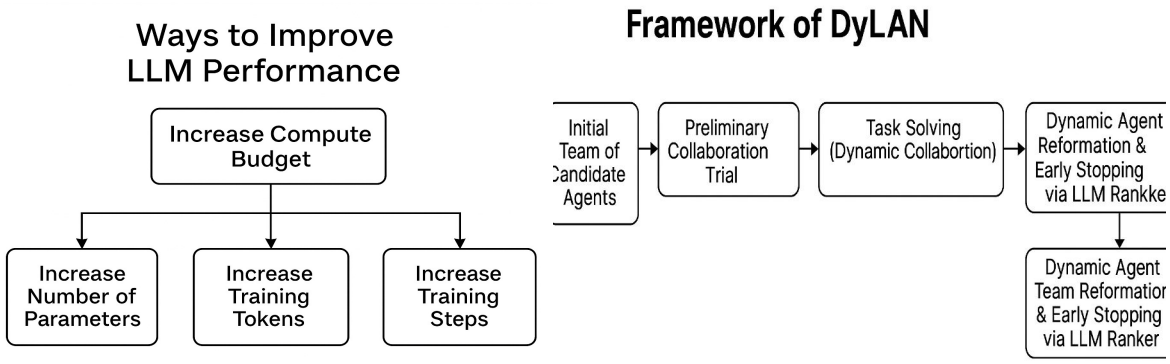

Figure 5: Example diagram generated by GPT-4o

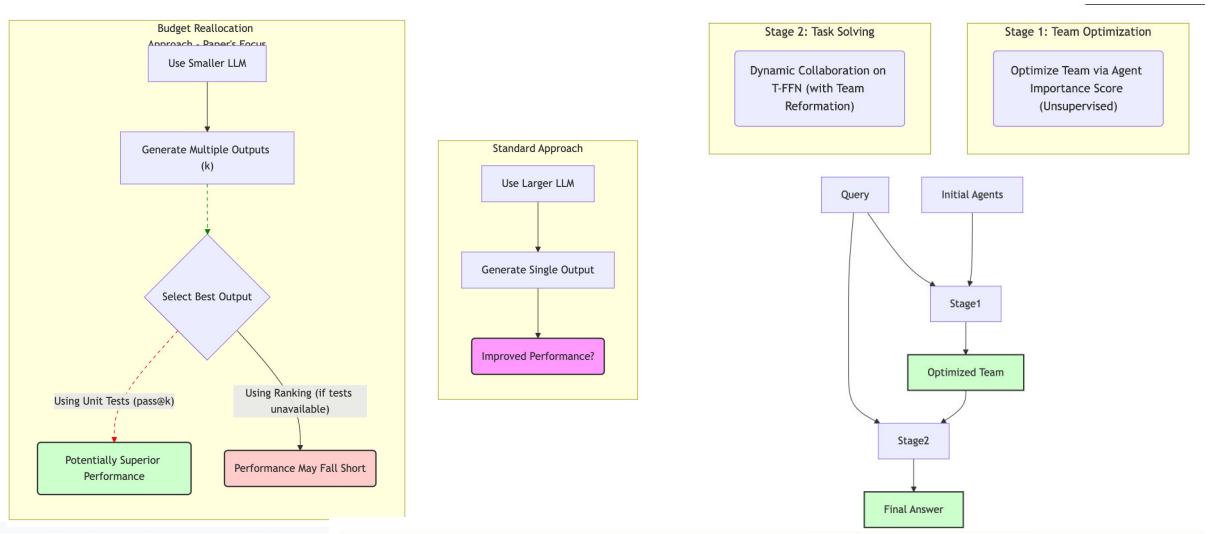

Figure 6: Example diagram generated by Gemini-2.5-Pro

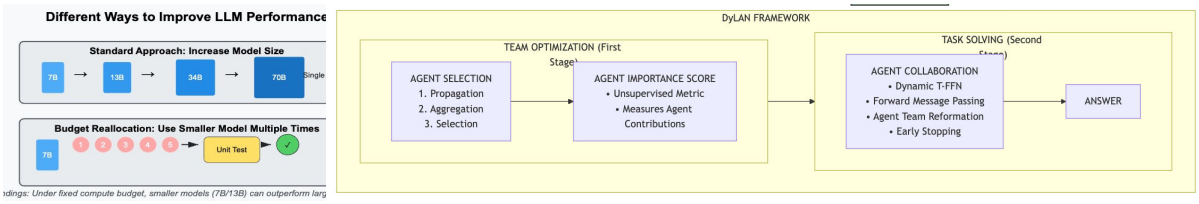

Figure 7: Example diagram generated by Claude-3.7-Sonnet

diagram of a scientific paper, your job is to select some of the components to be replaced by images to make it more appealing. The images should come from either the author's input, or could be replaced by a vivid svg icon.
You should follow the following steps:
1. Comprehend the drawio diagram thoroughly.
2. Identify which component should be replaced with

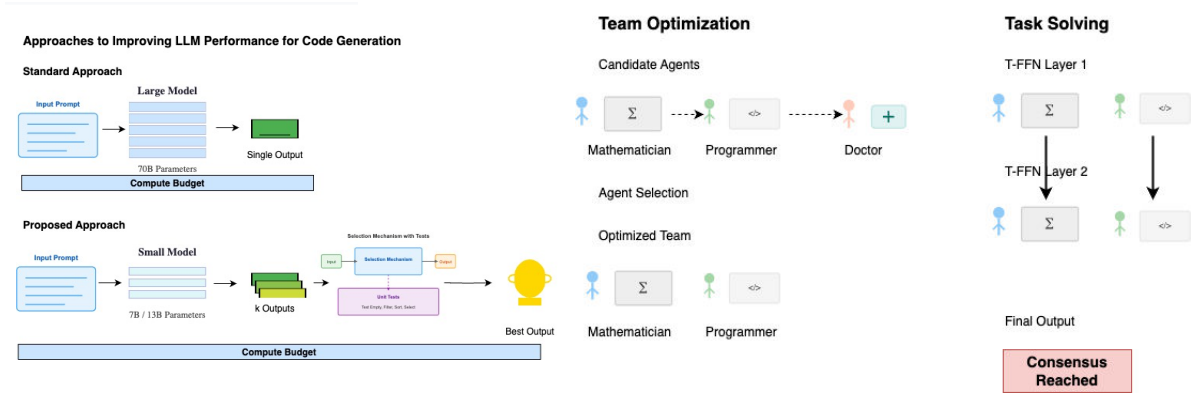

Figure 8: Example diagram generated by SCISKETCH graphic-code-based workflow

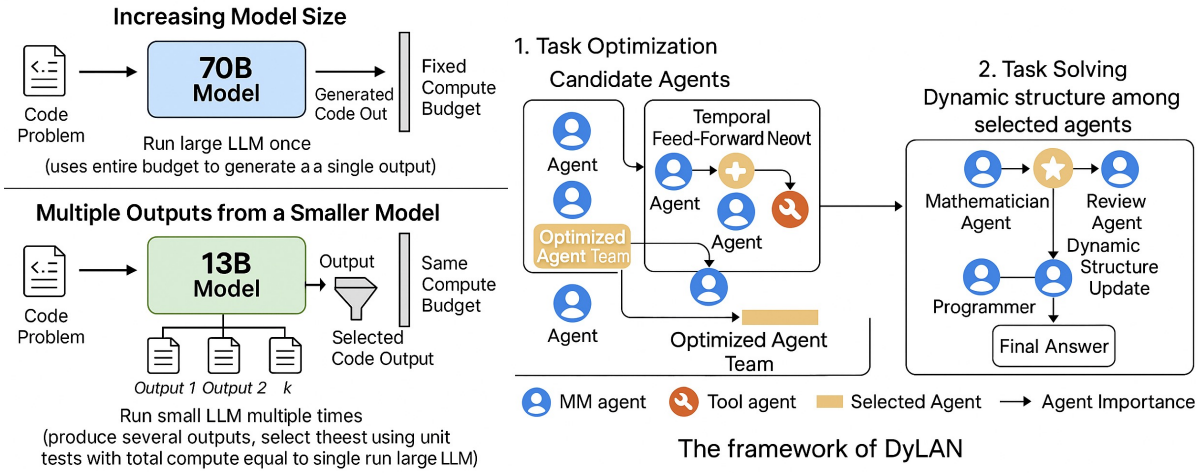

Figure 9: Example diagram generated by SCISKETCH image-based workflow

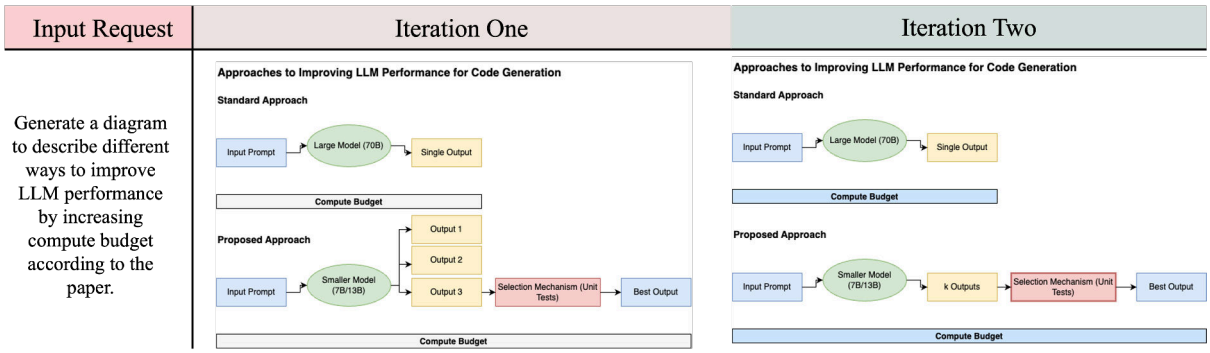

Figure 10: An example illustrating how the iterative refinement process improves diagram quality. In the second iteration, the color of the "Compute Budget" bars is adjusted for better visibility and thematic alignment, and multiple outputs are grouped into a single cluster to enhance clarity.

```
an image to make the diagram look better.
3. If the component is a concrete example or data
from the paper, then it should be provided by the
author.
4. If the component is an abstract component which
could be replaced by a svg icon, then it should be
provided by a svg generator.
5. You need to output the components result which
could be replaced by an image in a valid json string
as a list of dictionaries.
The dictionary should strictly follow the rules:
1. source: the replaced image source either be
"author" or "svg" according to the rules.
2. id: the id of the mxCell which should be
replaced.
3. title: the value of the mxCell.
4. description: a detailed description of the
component which will help generate the image later
```

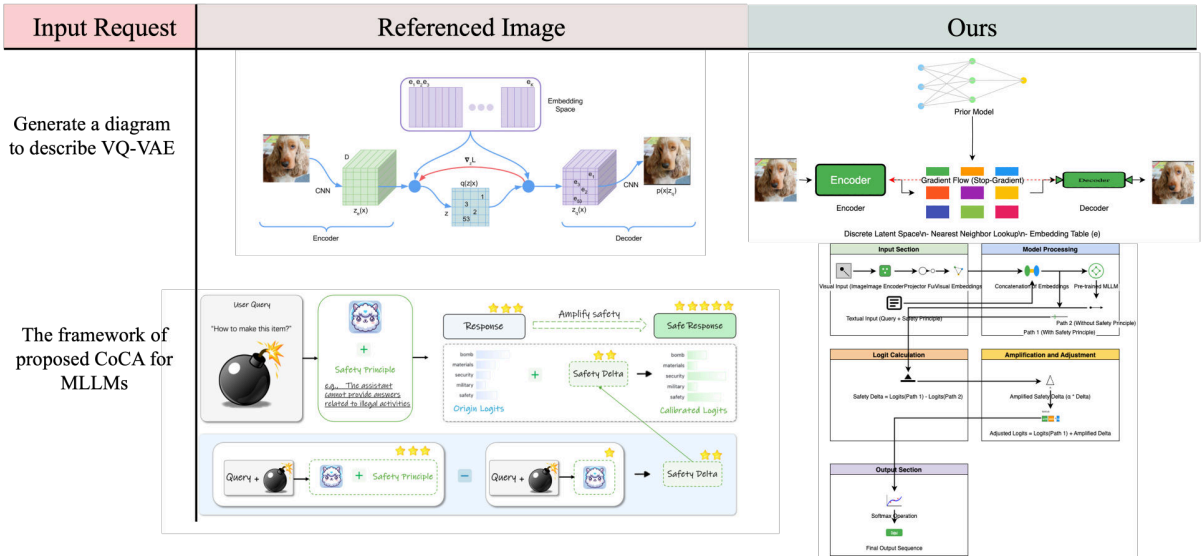

Figure 11: Examples of diagrams generated by SciSketch framework graphic-code-based workflow. The top example illustrates a high-quality generation with accurate structure and visual clarity. The bottom example demonstrates a case with structural or semantic errors.

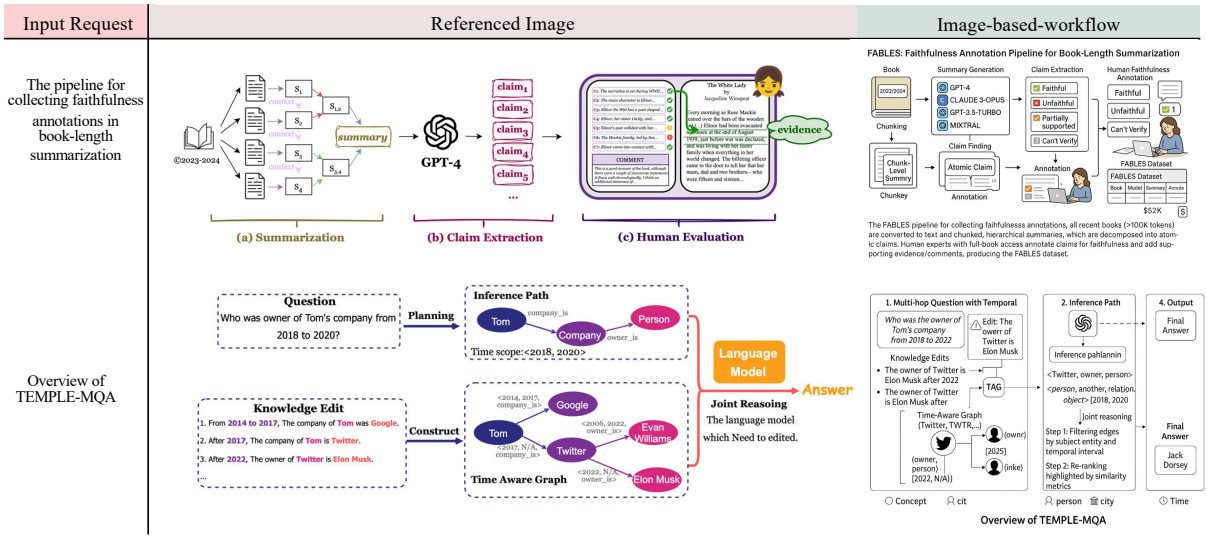

Figure 12: Examples of diagrams generated by SciSketch framework image-based workflow. The top example illustrates a high-quality generation with accurate structure and visual clarity. The bottom example demonstrates a case with typos and hallucinations.

```
on.
Think step by step and put the final result in a
valid json string wrapped by triple "`.
Diagram: {diagram}
```

SVG Generation

```
You are an excellent svg designer. Your job is to
generate an svg given the title and description of
the svg. Make it as simple as possible, otherwise
it will mess up with other components in a diagram.
Output the svg and wrap it within triple "`.
title: {title}
description: {description}
```

Empirical Figure Retrieval

```
You are an expert in matching for an image in a
given candidate_list with image names based on the
title and description. You will need to give the
index if the image is found, otherwise output -1.
The index is zero based. Only output the number of
the index.
title: {title}
description: {description}
candidate_list: {candidate_list}
index:
```

B.3 Diagram Generation in Image-Based Workflow

Diagram Image Refinement

```
You are an expert in assessing the quality of an
image given the description. You need to find out
if there are any issues in the image, and if so, how
```

```
to fix them.  Output the decision and wrap it inside
the <decision></decision> tag.  You should only
output yes or no inside the <decision></decision>
tag.  If nothing needs to be fixed, you should
output no.
Output your assessment and suggestions inside the
<suggestion></suggestion> tag.
Description:  {description}
```

C Evaluation

C.1 Human Evaluation Criteria

We ask the evaluator to give a score from 1 to 5 with regarding to the three fine-grained aspects, Completeness, Logical Consistency and Aesthetic Quality. The definition and instruction of scoring is shown in Table 3

D Experiments

D.1 Experiments with DeepSeek

To evaluate the adaptability of our framework to open-source foundation models, We replace all the foundation model in the framework with DeepSeek. And we generate the diagrams for three sampled papers. The result is shown in Figure 13

Aspect	Definition	Instructions
Completeness	Evaluate if the diagram includes all essential components and elements described in the source paper without omitting critical information.	Score 1 means the diagram is missing most of the main components or fails to capture essential parts of the source paper. Score 5 means the diagram includes all relevant components, fully representing the source material without any omissions.
Logical Consistency	Evaluate if the relationships and connections between diagram components accurately reflect the logical structure and intended semantics of the original paper.	Score 1 means the diagram has misleading, incorrect, or missing connections between components, failing to capture the logical flow. Score 5 means the diagram's relationships and connections are accurate and coherent, perfectly mirroring the logic and intended meaning of the source paper.
Aesthetic Quality	Evaluate the visual clarity and professional presentation of the diagram including layout organization, spacing, and element alignment.	Score 1 means the diagram is cluttered, visually confusing, or contains significant overlap or misalignment. Score 5 means the diagram is visually clear, well-organized, with good spacing, and all elements are neatly aligned.

Table 3: Evaluation Criteria Descriptions

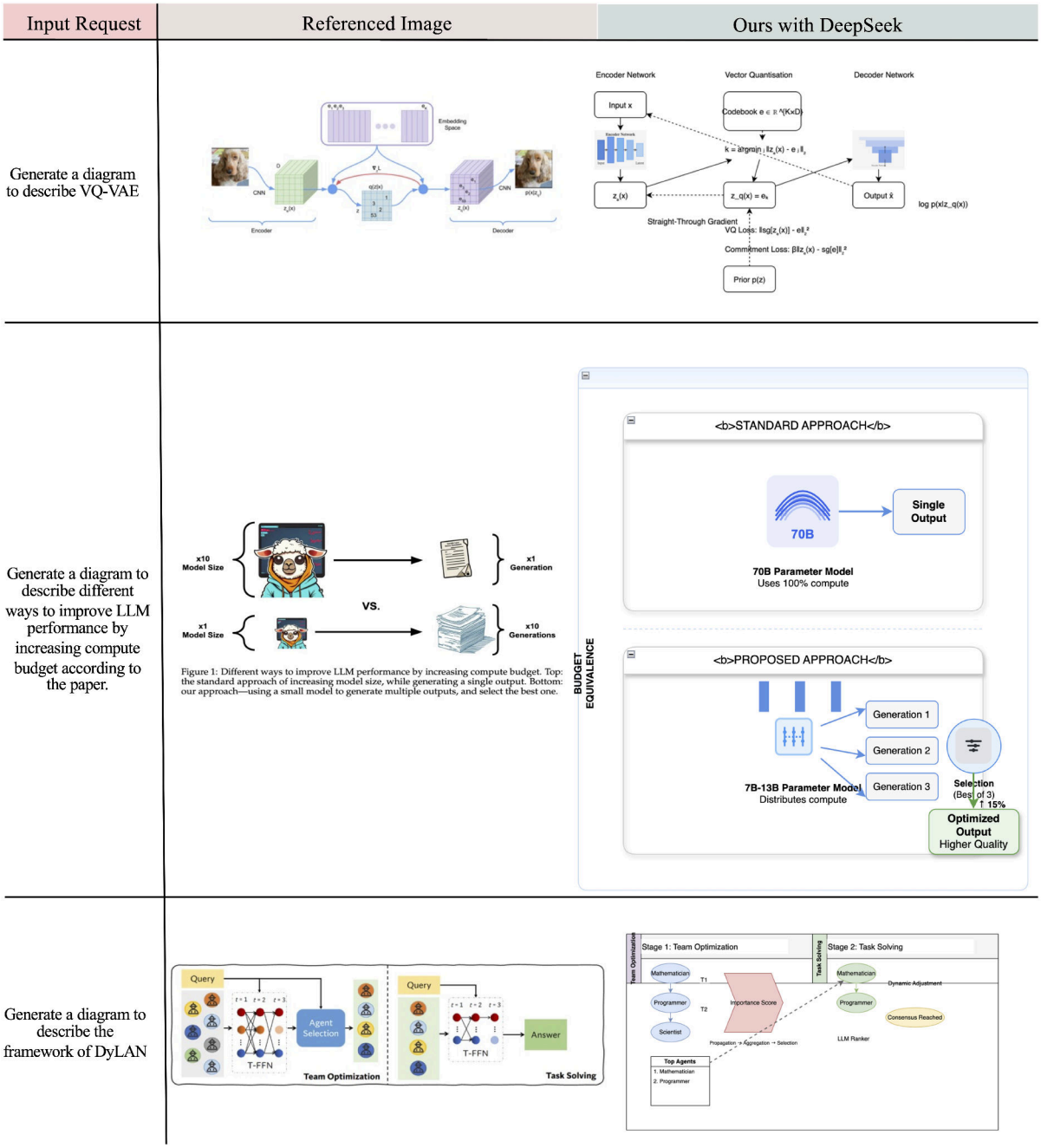

Figure 13: Examples of diagrams generated by the SCISKETCH framework using an open-source language model. The left example shows the ground-truth diagram from the original paper. The right example presents the diagram generated by SCISKETCH using DeepSeek. Despite the use of an open-source model, the generated diagram successfully captures the core concepts and presents the components in a logically coherent manner.