This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

From Words to Structured Visuals: A Benchmark and Framework for Text-to-Diagram Generation and Editing

Jingxuan Wei[1,3*], Cheng Tan[2*†], Qi Chen[1,3*], Gaowei Wu[1,3*], Siyuan Li[2], Zhangyang Gao[2], Linzhuang Sun[1,3], Bihui Yu[1,3†], Ruifeng Guo[1,3]
[1]Shenyang institute of computing technology, Chinese academy of sciences
[2] Zhejiang University [3]University of Chinese Academy of Sciences
weijingxuan20@mails.ucas.edu.cn, tancheng@westlake.edu.cn

Abstract

We introduce the task of text-to-diagram generation, which focuses on creating structured visual representations directly from textual descriptions. Existing approaches in text-to-image and text-to-code generation lack the logical organization and flexibility needed to produce accurate, editable diagrams, often resulting in outputs that are either unstructured or difficult to modify. To address this gap, we introduce DiagramGenBenchmark, a comprehensive evaluation framework encompassing eight distinct diagram categories, including flowcharts, model architecture diagrams, and mind maps. Additionally, we present DiagramAgent, an innovative framework with four core modules—Plan Agent, Code Agent, Check Agent, and Diagram-to-Code Agent—designed to facilitate both the generation and refinement of complex diagrams. Our extensive experiments, which combine objective metrics with human evaluations, demonstrate that DiagramAgent significantly outperforms existing baseline models in terms of accuracy, structural coherence, and modifiability. This work not only establishes a foundational benchmark for the text-to-diagram generation task but also introduces a powerful toolset to advance research and applications in this emerging area.

1. Introduction

"Drawing is not what one sees but what one can make others see." — Edgar Degas.

The task of transforming textual descriptions into structured diagrams, referred to as *text-to-diagram generation*, is crucial for domains that rely on clear, logical representations of complex information [8, 15, 27, 37, 40]. Unlike natural scene images, structured diagrams—such as flowcharts, mind maps, and model architecture diagrams—require precise relationships and logical coherence to effectively communicate intricate ideas. This study focuses on developing robust methods for automatic diagram generation and editing, aiming to facilitate efficient, accurate diagram creation from textual input. Such advancements hold great potential to enhance productivity in domains like education, scientific research, and industry, where structured visualizations are essential for effective communication and analysis.

While recent advancements in text-to-image generation have produced impressive results in synthesizing photorealistic and aesthetically pleasing images [8, 53], these methods are largely optimized for naturalistic visuals rather than structured, logical content. Models such as GANs [43, 49] and diffusion models [28, 33] have excelled at generating images with rich visual detail, but they inherently lack the ability to enforce the precise relationships and hierarchical organization required for structured diagrams. Alternative approaches have sought to first convert text into code representations [4, 18, 41], which are then compiled into charts or simple plots. These methods leverage advancements in large language models to interpret textual descriptions as code that can produce visual output. However, while effective for generating basic visualizations such as bar charts and line graphs, this approach has limitations when applied to more complex diagrams. Specifically, they lack the flexibility and structural logic required for diagrams that demand intricate organization and interactivity to edits—a necessity for broader scientific and industrial applications.

As illustrated in Figure 1, existing methods fall short when faced with tasks that require both logical precision and post-generation modifiability. Text-to-image models often produce outputs that are visually appealing but logically incoherent [22, 53], failing to capture the structured relationships that intricate diagrams demand. This gap in logical structure limits the practical utility of these models in scenarios where accurate and adaptable visual represen-

*Equal contribution
†Corresponding author

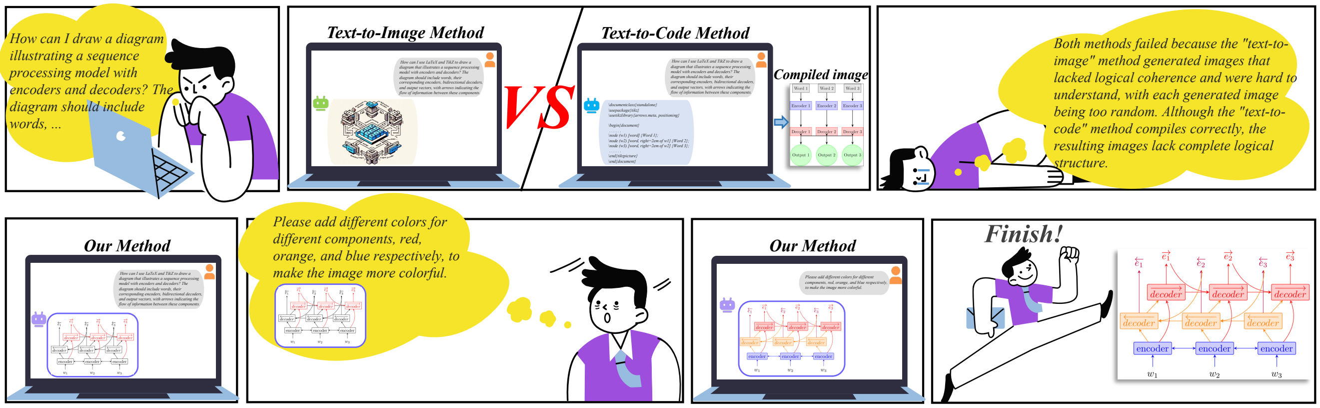

Figure 1. Challenges in Existing Text-to-Image and Text-to-Code Methods for Diagram Generation

tations are essential. On the other hand, current text-to-code approaches, while suitable for simpler visualizations, struggle to represent complex diagram features such as hierarchical relationships, intricate color coding, and layered structures [4, 18, 35]. Furthermore, there is a significant lack of comprehensive datasets tailored for generating and modifying diverse types of structured diagrams. Existing datasets tend to focus on either image realism or simplistic data visualizations, neither of which fully address the demands of text-to-diagram generation.

To address these challenges, we introduce **DiagramGenBenchmark**, a rigorously curated dataset for evaluating structured diagram generation and editing. This benchmark includes eight distinct types of diagrams, such as flowcharts, bar charts, and model architectures, providing a solid foundation for advancing and assessing text-to-diagram tasks. Alongside this benchmark dataset, we propose **DiagramAgent**, a framework for text-to-diagram generation and editing. DiagramAgent integrates multiple agents, including a *Plan Agent* for interpreting textual commands, a *Code Agent* for generating diagram code, a *Check Agent* for logical verification, and a *Diagram-to-Code Agent*. This framework not only facilitates robust diagram generation but also enables editing and adaptation, accommodating complex structural requirements.

Our extensive experiments on DiagramAgent, conducted using the DiagramGenBenchmark dataset, demonstrate its superior performance across various diagram types and tasks. Notably, DiagramAgent outperforms existing works in terms of accuracy and overall quality. For example, in the diagram generation task, DiagramAgent achieved top scores with Pass@1 reaching 58.15, ROUGE-L scoring 51.97, and CodeBLEU at 86.83, highlighting its efficacy in generating accurate and logically coherent diagram code. We further validated DiagramAgent's versatility by testing it on both open-source models, such as Qwen2 [45], and closed-source models, including GPT-4o [3], demonstrating its adaptability across different model architectures. To substantiate these results, we conducted human evaluations that aligned closely with the objective metrics, confirming DiagramAgent's accuracy and usability in diagram generation tasks. We summarize our contributions as follows:

- We introduce **DiagramGenBenchmark**, a benchmark dataset covering eight diverse types of diagrams, specifically tailored for diagram generation and editing.
- We present **DiagramAgent**, a text-to-diagram framework capable of both generating and editing diagrams, providing flexibility and accuracy in complex logical structures.
- We conduct comprehensive experiments demonstrating DiagramAgent's superior performance in terms of accuracy and logical coherence, establishing DiagramAgent as a reliable baseline for future research.

2. Related Work

2.1. Text-to-image generation

Text-to-image generation involves creating images based on natural language descriptions, ranging from photorealistic scenes to semantically rich and abstract visuals [8, 38, 53]. Existing methods can be broadly categorized into three main approaches: Generative Adversarial Networks (GANs), diffusion models, and autoregressive approaches. GANs represent some of the earliest efforts in this field, with models such as StackGAN [49] and AttnGAN [43] improving image quality through progressive generation strategies and attention mechanisms. Despite their early successes, GANs are often hampered by issues like instability in adversarial training and mode collapse, making them less suitable for generating images of complex scenes. Diffusion models have emerged as the dominant approach, offering significant improvements in image quality by iteratively refining images through a denoising process. Notable examples include Imagen [33], which generates high-resolution images with remarkable detail. PreciseControl [28] enable fine-grained control over specific attributes, while OpenBias [10] incorporates mechanisms for bias detection to ensure fairness and diversity in generated outputs. Transformer-based autoregressive models have also gained prominence due to their strong cross-modal alignment capabilities. For instance, DALL-E [30] employs attention mechanisms to closely map textual descriptions to visual elements. The STAR model [26] integrates multi-scale autoregressive generation with enhanced Rotary Position Embeddings (RoPE) to improve consistency between text and images in intricate scenarios. Similarly, Fluid [12] leverages multi-level attention to fine-tune the alignment of object layouts and sizes, ensuring a precise correspondence between textual inputs and visual outputs. While these methods have made significant strides in creating natural scene imagery, they fall short when it comes to tasks that require the generation of diagrams with logical structures and explicit relationships, such as flowcharts, architecture diagrams, or other technical illustrations.

2.2. Text-to-code generation

Text-to-code generation involves the automatic conversion of natural language descriptions into executable code. Notable examples include Qwen2.5-Coder [17], DeepSeek-Coder-Instruct [14], Code-Llama [32], and WizardCoder [25], which excel at generating sophisticated code snippets that integrate architectural logic. In contrast, generalist text-to-code models, although not explicitly designed for code generation, have demonstrated remarkable adaptability due to their extensive pre-trainings. Open-sourced models such as Llama3 [11], Baichuan2 [44], InternLM2.5 [7], and Qwen2 [45] exhibit strong capabilities across multiple programming languages, making them well-suited for a broad range of coding challenges. Moreover, several closed-sourced models, including GPT-4o [3], Gemini [31], DeepSeek-2.5 [23], and GLM-4-Plus [13], offer exceptional accuracy and performance in complex code generation. Text-to-code models are also paving the way for innovations that extend beyond conventional programming tasks. The methodologies employed in text-to-code generation have inspired research into adjacent areas, such as automatic diagram generation in this work.

3. Preliminaries

We outline the foundational components of text-to-diagram system, detailing the processes involved in generating, coding, and editing diagrams based on user instructions.

① **Diagram Generation** Given an instruction x_{ins}, our system generates code c_{diag}, which can be compiled into a diagram D_{gen}. Unlike standard code generation—which focuses on executable logic—our approach emphasizes both the logical structure and the aesthetic design.

Let $f_{code}(x_{ins})$ denote the function that translates a textual instruction x_{ins} into diagram-specific code c_{diag}. Additionally, let $f_{img}(c_{diag})$ represent the function that compiles the diagram-specific code c_{diag} into the generated diagram D_{gen}. The process can be formally expressed as follows:

$$D_{gen} = f_{img}(f_{code}(x_{ins})), \tag{1}$$

where D_{gen} is the generated diagram based on the structured and logical elements specified in the user instruction.

② **Diagram Coding** Given a diagram D_{gen}, our system generates structured code c'_{diag} that accurately captures the logical framework and design elements of the diagram. This process ensures that each diagram not only conveys the intended information but also adheres to the necessary structural integrity. The process can be formulated as:

$$c'_{diag} = f^{-1}_{\text{code}}(D_{gen}). \tag{2}$$

③ **Diagram Editing** The process of modifying an existing diagram involves three main steps: *diagram coding*, *code modification*, and *code-to-diagram* recompilation. Initially, the original diagram D_{ori} is translated back into its underlying code c_{ori}. This code is subsequently updated according to the user's editing instruction x_{edit}, resulting in the modified code c_{mod}. Finally, this modified code c_{mod} is compiled into a new diagram D_{mod}. The editing process can be mathematically represented as:

$$D_{mod} = f_{img}(f_{mod}(f^{-1}_{\text{code}}(D_{ori}), x_{edit})), \tag{3}$$

where $f^{-1}_{\text{code}}(D_{ori})$ indicates the function that extracts the code c_{ori} from the original diagram D_{ori}. The function $f_{mod}(c_{ori}, x_{edit})$ reflects the process of updating the code based on the instruction x_{edit}, ultimately producing the modified diagram D_{mod} after recompilation via f_{img}.

4. Method

We summarize DiagramAgent, which comprises four core agents: Plan Agent, Code Agent, Check Agent, and Diagram-to-Code Agent in Figure 2, highlighting the interactions between the agents and the processes involved in diagram generation, coding, and editing.

4.1. Plan Agent

The Plan Agent (④) [17, 29] is the central component responsible for interpreting user instructions and determining the necessary steps. The agent analyzes the instruction for completeness. If the instruction is incomplete (e.g., missing node or label information), the Plan Agent performs *Query Expansion* using a Large Language Model (LLM), generating a more comprehensive query x_{comp}, ensuring necessary elements are included. The process can be expressed as:

$$x_{comp} = f_{expand}(x_{ins}) \tag{4}$$

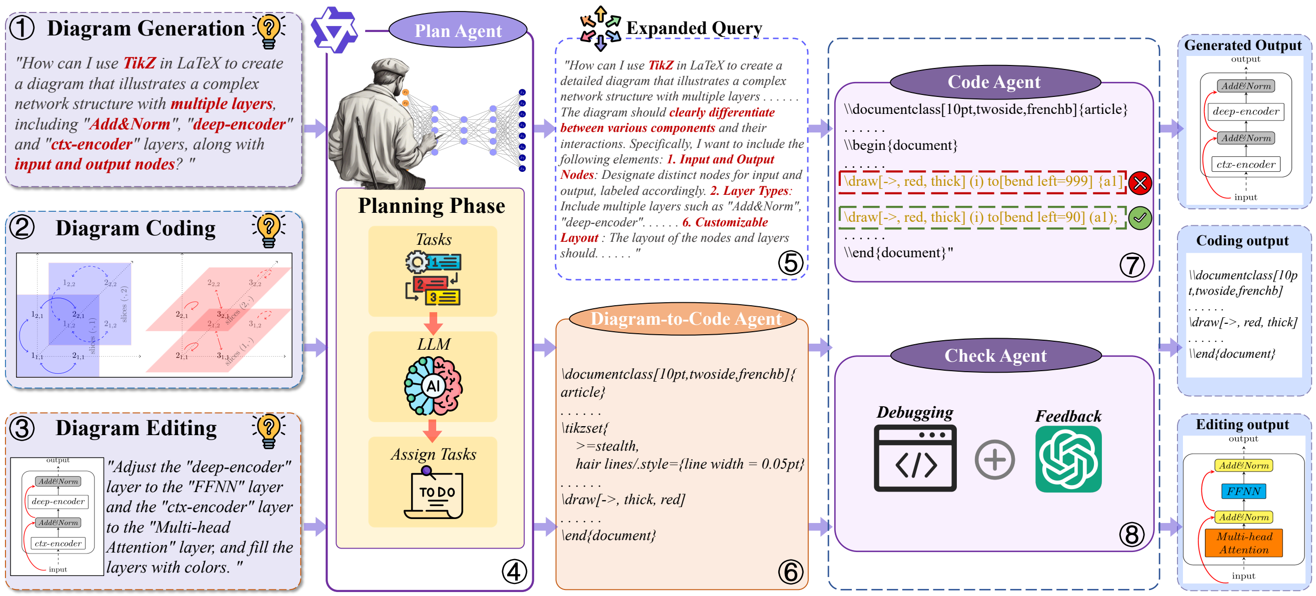

Figure 2. Workflow of DiagramAgent. The DiagramAgent handles diagram generation, coding, and editing tasks, processing the user query (①-③) through the Plan Agent (④), which communicates with the Code Agent (⑦) for diagram generation or with the Diagram-to-Code Agent (⑥) for diagram coding or editing. Code is verified and feedback provided by the Check Agent (⑧).

Next, the complete query x_{comp} is passed to the Code Agent, which generates the corresponding code c_{diag}.

For diagram coding, the Plan Agent handles the initial diagram D_{ori} by invoking the *Diagram-to-Code Agent* $f^{-1}_{code}(D_{ori})$ (⑥), which translates the diagram into its code form. For diagram editing, the Plan Agent receives the editing query x_{edit} along with the diagram D_{ori}. Based on the user's request for changes, the Plan Agent again calls the *Diagram-to-Code Agent* $f^{-1}_{code}(D_{ori})$ (⑥) to convert the provided diagram into code. Then, the Plan Agent sends both the code and the editing query x_{edit} to the *Code Agent* (⑦).

4.2. Code Agent

The Code Agent (⑦) [17, 51] is responsible for transforming the processed user instructions from the Plan Agent (④) into executable code. For diagram generation, the Code Agent takes the complete query x_{comp} and generates diagram-specific code c_{diag} as follows:

$$c_{diag} = f_{code}(x_{comp}) \tag{5}$$

For diagram editing, the Code Agent takes the original code c_{ori} and applies the modifications according to the query x_{edit} to generate modified code c_{mod}. The code is passed to the Check Agent (⑧) for debugging and refinement, ensuring that the code is both syntactically and logically correct.

We optimize the Code Agent by training it to generate code c that closely matches a reference code c_{ref}. This can be formalized as minimizing the following objective:

$$\mathcal{L} = \min_{\theta} \mathbb{E}_{(x,c_{ref})} \left[\mathcal{L}_{\text{code}}(f_{code_\theta}(x), c_{ref}) \right], \tag{6}$$

where θ represents the parameters of the model, and $\mathcal{L}_{\text{code}}$ is the loss function, which measures the discrepancy between the generated code and the reference code.

4.3. Diagram-to-Code Agent

The Diagram-to-Code Agent (⑥) is responsible for converting a given diagram D_{ori} into its corresponding code c'_{diag}, facilitating the diagram coding task. The Diagram-to-Code Agent captures the diagram's logical structure and visual elements, translating them into structured code. This transformation process is formally expressed as:

$$c'_{diag} = f_{\text{diagram-to-code}}(D_{ori}) \tag{7}$$

where c'_{diag} denotes the generated code based on the elements of the diagram D_{ori}.

The Diagram-to-Code Agent then sends the generated c'_{diag} to the Check Agent (⑤) for validation. If necessary, feedback is provided to refine c'_{diag}. The objective is to generate code c'_{diag} that closely matches a reference code c_{ref}, ensuring that the code preserves the diagram's logical and visual integrity. This optimization is formalized as minimizing the following loss function:

$$\mathcal{L} = \min_{\theta} \mathbb{E}_{(D_{ori},c_{ref})} \left[\mathcal{L}_{\text{code}}(f_{\text{diagram-to-code}_\theta}(D_{ori}), c_{ref}) \right]. \tag{8}$$

where $\mathcal{L}_{\text{code}}$ measures the discrepancy between the generated code and the reference code.

4.4. Check Agent

The Check Agent (⑧) [17, 36] is responsible for verifying the correctness and completeness of the code generated by the Code Agent (⑦) and the Diagram-to-Code Agent(⑥). Its role involves two critical stages: *debugging* and *verification*. First, the Check Agent compiles the generated code. If errors are detected, they are returned to the Code Agent or Diagram-to-Code Agent for correction and regeneration. Once the code compiles successfully, the Check Agent uses GPT-4o [3] to verify the completeness of the code, ensuring that all necessary elements are included. The process can be summarized as follows:

$$f_{\text{check}}(c) = f_{\text{debug}}(c_{diag}, c_{mod}, c'_{diag}) + f_{\text{verify}}(c_{diag}, c_{mod}, c'_{diag}) \quad (9)$$

where $f_{\text{debug}}(c)$ returns errors if present, and $f_{\text{verify}}(c)$ evaluates the completeness of the diagram.

5. DiagramGenBenchmark

We introduce *DiagramGenBenchmark*, which focuses on transforming textual descriptions into structured diagram representations. It includes various types of diagrams such as model architecture diagrams, flowcharts, line charts, directed and undirected graphs, tables, bar charts, and mind maps. The data is sourced from HuggingFace's VGQA [54], datikz [6], and datikz-v2 [6] datasets, as well as open-source repositories on GitHub and Overleaf, which are licensed under CC BY 4.0 or MIT. These repositories predominantly feature diagram code written in LaTeX or DOT languages. Due to the limited space, we provide the benchmark dataset details in the Appendix A.

5.1. Data Statistics

Figure 3 provides example queries and diagrams, inlcuding model architecture, flowcharts, line charts, directed/undirected graphs, tables, bar charts, and mind maps. As detailed in Table 1, the diagram generation/coding task contains 6713 training samples and 270 testing samples, while the diagram editing task includes 1400 training samples and 200 testing samples. The query lengths and code lengths vary significantly, reflecting the complexity of the tasks.

5.2. Evaluation Metrics

- **Code Generation Metrics**: For the diagram generation and editing tasks, we evaluate the generated code using:
  - *Pass@1* (↑): Measures accuracy of generated code.
  - *ROUGE-L* (↑): Measures similarity with reference code based on longest common subsequence.
  - *CodeBLEU* (↑): Assess semantic similarity with reference code.
  - *Edit Distance* (↓): Computes the edit operations required to match the reference.
  - *chrF* (↑), *RUBY* (↑): Evaluate text similarity and understanding.
- **Diagram Quality Metrics**: For assessing the quality of generated diagrams, we use:
  - *CLIP-FID* (↓), *LPIPS* (↓): Cross-modal similarity.
  - *PSNR* (↑), *MS-SSIM* (↑): Structural and clarity metrics.

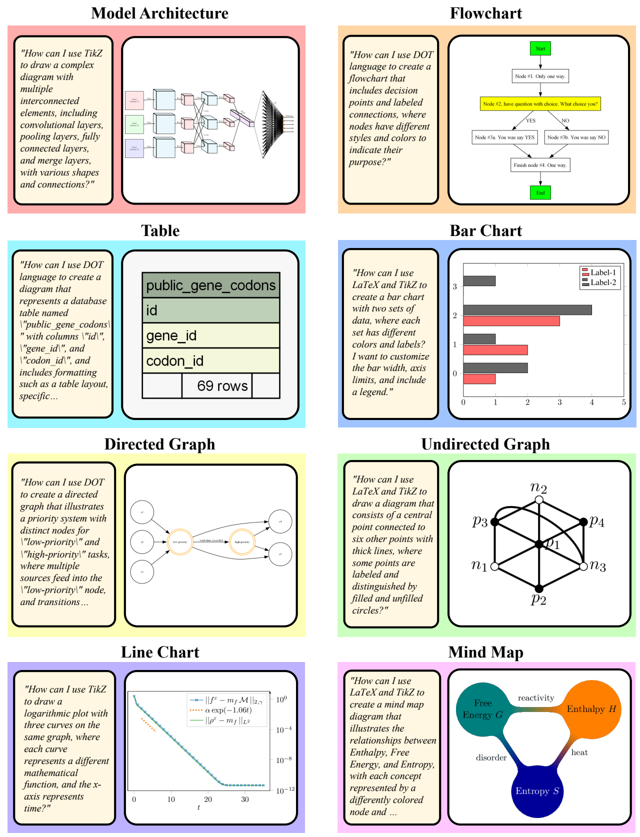

Figure 3. Example queries and diagrams.

Table 1. Key statistics of DiagramGenBenchmark.

Statistic	Generation / Coding	Editing
Total Samples	6983	1600
Training Samples	6713	1400
Testing Samples	270	200
Diagram Type		
Model Architecture	3000 (42.96%)	706 (44.13%)
Flowchart	1500 (21.48%)	320 (20%)
Line Chart	1142 (16.35%)	227 (14.19%)
Directed Graph	787 (11.27%)	217 (13.56%)
Table	114 (1.63%)	19 (1.19%)
Bar Chart	227 (3.25%)	77 (4.81%)
Mind Map	140 (2.00%)	16 (1%)
Undirected Graph	73 (1.05%)	18 (1.12%)
Query Length (tokens)		
Minimum	18	40
Maximum	561	499
Average	247	133
Complete Query Length (tokens)		
Minimum	404	115
Maximum	4015	3419
Average	1900	98
Code Length (tokens)		
Minimum	60	116
Maximum	75612	4727
Average	1701	1009.6

- **Human Evaluation**: Three evaluators with graduate degrees assess the test set based on criteria rated from 1 to

5, with higher scores indicating better quality.

6. Experiment

Setup DiagramAgent performs diagram generation and editing through four core agents: Plan Agent, Code Agent, Check Agent, and Diagram-to-Code Agent. The Plan Agent utilizes the Qwen-72B [17] to effectively distinguish and interpret the user's query tasks. The Code Agent is based on Qwen2.5-Coder-7B [17], fine-tuned for four epochs with a maximum input length of 8192 tokens. Diagram-to-Code Agent uses Qwen2-VL-7B [39] in its default configuration. Experiments are conducted on an 8×80G A100 GPUs.

Model Comparison We evaluate DiagramAgent's Code Agent against 16 mainstream models. The specialized code models include Qwen2.5-Coder [17], DeepSeek-Coder-Instruct [14], Code-Llama [32], WizardCoder [25], Codegeex4-all [52], Starcoder2 [21], and Yi-Coder [1]. Open-source models feature Llama-3.1 [11], Baichuan2 [44], Internlm2_5 [7], Yi-1.5 [48], and Qwen2 [17], while close-source models include GPT-4o [3], DeepSeek V2.5 [23], GLM-4-plus [13], and Gemini [31]. Similarly, we compare the Diagram-to-Code Agent's performance on the diagram coding against open-source models include Yi-VL [47], Qwen2-VL [39], Internlm-Xcomposer2.5 [50], Llama-3.2-Vision [11], Phi-3.5-vision [2], Llava-v1.6 [19], Cogvlm2-llama3 [16], and Deepseek-vl [24]. Close-source models include GPT-4o [3], GLM-4-plus [13], and Gemini-1.5-pro [31].

6.1. Diagram Generation

Main Results DiagramAgent's Code Agent demonstrates outstanding performance on the diagram generation task, achieving top scores across both code accuracy and image fidelity metrics, as shown in Table 2. In terms of code quality, DiagramAgent achieves leading results with metrics such as Pass@1 (58.15), ROUGE-L (51.97), and CodeBLEU (86.83), among others, highlighting its capability to generate accurate and robust code representations. These results demonstrate the effectiveness of DiagramAgent in generating structured, accurate, and high-quality diagrams. For image quality, it also excels with PSNR (6.38), LPIPS (45.95), and others, confirming its ability to maintain high visual fidelity in generated diagrams.

Ablation Study The ablation study in Table 3 demonstrates the critical role of the Check Agent in enhancing DiagramAgent's performance. Removing the verification module (GPT-4o) leads to the chrF decrease of -0.68 and the LPIPS increase of +0.71. Excluding the debugging component (compiler) results in a significant chrF drop of -1.75 and an LPIPS increase of +2.18, impacting both code quality and image fidelity. Together, the debugging and verification achieve the highest scores across metrics, underscoring their combined value in producing precise diagrams. Overall, this ablation study demonstrates that both debugging and verification are essential in diagram generation.

6.2. Diagram Coding

Main Results DiagramAgent's model, configured with compiler-based debugging followed by GPT-4o verification, achieves the highest performance across several metrics in the diagram coding task, as illustrated in Table 4. Key metrics include Pass@1 (68.89), ROUGE-L (48.99), codeBLEU (84.64), demonstrating DiagramAgent's effectiveness in generating high-quality code from images. Compared to both open-source models like Qwen2-VL-7B-Instruct and closed-source models such as GPT-4o, DiagramAgent consistently excels, highlighting its robustness in tasks requiring precise visual-to-code translation.

Ablation Study The ablation study in Table 5 demonstrates the effectiveness and necessity of the **Check Agent**'s debugging (compiler) and verification (GPT-4o) components within DiagramAgent. Removing the verification module results in a notable Pass@1 reduction of -6.30 and a ROUGE-L reduction of -0.28, underscoring the importance of verification for accurate code generation. Excluding the compiler module reduces Pass@1 by -15.56 and ROUGE-L by -0.53. The full model configuration, which includes both debugging and verification, achieves optimal results across all metrics, affirming the essential role of these components in enhancing diagram coding performance.

Due to space limitations, the full results presented in Table 4 and Table 5 are provided in Appendix E.

6.3. Diagram Editing

Main results DiagramAgent's Code Agent achieves superior results on the diagram editing task, demonstrating exceptional performance in both code accuracy and image quality metrics, as shown in Table 6. In terms of code generation, DiagramAgent leads with top scores in Pass@1 (98.00), ROUGE-L (98.41), and CodeBLEU (99.93), underscoring its capability for precise and reliable code outputs. For image quality, DiagramAgent also excels, achieving outstanding results in CLIP-FID (1.08), LPIPS (40.64), and MS-SSIM (97.00), although its PSNR (13.18) is slightly lower than WizardCoder, potentially due to DiagramAgent's focus on overall image fidelity rather than absolute sharpness. DiagramAgent consistently outperforms baseline models across comprehensive metrics, validating its strong adaptability and reliability.

Ablation Study As shown in Table 7, removing the verification module results in a Pass@1 decrease of -1.00 and a chrF reduction of -0.08, highlighting its role in ensuring code accuracy. Excluding the debugging module no-

Table 2. Main results for diagram generation (Code Agent). The best result in each metric is bolded.

Model	Size	Pass@1↑	ROUGE-L↑	CodeBLEU↑	Edit Dist.↓	chrF↑	RUBY↑	CLIP-FID↓	LPIPS↓	PSNR↑	MS-SSIM↑
Qwen2.5-Coder [17]	7B	32.22	41.94	82.58	83.45	38.14	28.09	18.90	60.54	3.72	13.04
DeepSeek-Coder [14]	33B	55.56	44.26	83.29	81.85	42.01	30.55	15.49	60.99	6.02	19.80
Code-Llama [32]	34B	8.89	22.92	76.78	95.89	28.77	13.60	30.12	59.80	0.89	2.32
WizardCoder [25],	15B	28.89	29.93	78.96	91.30	31.38	19.73	27.38	55.96	3.36	11.66
Codegeex4-all [52]	9B	49.63	42.14	82.94	86.31	41.36	28.69	13.86	61.08	5.48	17.37
Starcoder2 [21]	15B	27.41	26.49	78.56	90.67	25.98	17.74	31.63	56.54	3.11	10.53
Yi-Coder [1]	9B	37.04	41.38	82.46	83.91	39.20	28.00	22.40	57.10	3.91	14.11
Llama-3.1 [11]	8B	33.58	37.04	80.45	88.24	36.80	24.79	17.91	58.80	3.78	11.94
Baichuan2 [44]	13B	16.30	33.28	79.94	87.96	31.83	21.51	23.43	61.49	1.81	4.94
Internlm2_5 [7]	20B	34.44	39.45	81.79	87.00	38.44	26.21	24.56	56.81	3.91	13.39
Yi-1.5 [48]	34B	35.19	42.56	82.91	85.52	42.03	28.43	20.03	58.04	3.82	12.83
Qwen2 [17]	7B	41.48	41.74	82.49	84.86	39.72	27.93	15.57	58.89	4.60	15.48
GPT-4o [3]	-	49.81	44.59	82.83	85.17	43.83	30.08	13.26	63.07	5.56	18.21
DeepSeek V2.5 [23]	-	54.44	43.00	82.83	85.67	43.63	28.75	13.32	62.32	5.56	16.98
GLM-4-plus [13]	-	42.96	46.42	83.91	82.40	44.51	32.13	14.70	63.38	4.47	13.89
Gemini [31]	-	43.23	44.86	82.37	84.44	43.75	30.46	21.69	54.93	3.16	18.70
DiagramAgent	7B	**58.15**	**51.97**	**86.83**	**74.62**	**53.49**	**39.71**	**11.16**	**45.95**	**6.38**	**24.78**

Table 3. Ablation study for diagram generation task (Code Agent). Each result shows the performance of DiagramAgent Code Agent under various component configurations, with the decrease in each metric from the full model indicated in parentheses.

Diagram generation	Size	Pass@1↑	ROUGE-L↑	CodeBLEU↑	Edit Dist.↓	chrF↑	RUBY↑	CLIP-FID↓	LPIPS↓	PSNR↑	MS-SSIM↑
DiagramAgent	7B	**58.15**	**51.97**	**86.83**	**74.62**	**53.49**	**39.71**	**11.16**	**45.95**	**6.38**	**24.78**
– w/o GPT-4o	7B	57.78 (-0.37)	51.95 (-0.02)	86.35 (-0.48)	74.71 (+0.09)	52.81 (-0.68)	39.33 (-0.38)	11.23 (+0.07)	46.66 (+0.71)	6.33 (-0.05)	20.80 (-3.98)
– w/o Compiler	7B	57.41 (-0.74)	51.94 (-0.03)	85.78 (-1.05)	74.68 (+0.06)	51.74 (-1.75)	39.65 (-0.06)	11.29 (+0.13)	48.13 (+2.18)	5.93 (-0.45)	24.10 (-0.68)
– w/o GPT-4o & Compiler	7B	57.41 (-0.74)	51.87 (-0.10)	85.74 (-1.09)	74.79 (+0.17)	51.69 (-1.80)	38.77 (-0.94)	11.30 (+0.14)	48.17 (+2.22)	5.85 (-0.53)	20.37 (-4.41)

Table 4. Main results for diagram coding task (Diagram-to-Code Agent). The best result in each metric is bolded.

Model	Size	Pass@1↑	ROUGE-L↑	codeBLEU↑
Yi-VL [47]	34B	2.22	20.01	70.57
Qwen2-VL [39]	7B	28.89	31.74	80.04
Internlm-xcomposer2.5 [50]	7B	3.33	28.47	77.35
Llama-3.2-Vision [11]	11B	27.78	21.94	75.37
Phi-3.5-vision [2]	4B	24.07	27.53	76.56
Llava-v1.6 [19]	34B	8.89	26.68	76.53
Cogvlm2-llama3 [16]	19B	3.70	14.42	70.72
Deepseek-vl [24]	7B	50.74	25.18	76.48
GPT-4o [3]	-	64.07	39.95	81.78
GLM-4-plus [13]	-	51.48	35.92	80.16
Gemini-1.5-pro [31]	-	17.78	38.66	80.75
DiagramAgent	7B	**68.89**	**48.99**	**84.64**

Table 5. Ablation study for diagram coding task (Diagram-to-Code Agent). Each result shows the performance of DiagramAgent under various component configurations, with the decrease in each metric from the full model indicated in parentheses.

Diagram coding	Size	Pass@1↑	ROUGE-L↑	codeBLEU↑
DiagramAgent	7B	**68.89**	**48.99**	**84.64**
– w/o GPT-4o	7B	62.59 (-6.30)	48.71 (-0.28)	84.57 (-0.07)
– w/o Compiler	7B	53.33 (-15.56)	48.46 (-0.53)	84.52 (-0.12)
– w/o GPT-4o & Compiler	7B	52.59 (-16.30)	47.81 (-1.18)	84.21 (-0.43)

tably impacts CodeBLEU by -0.74 and increases LPIPS by +1.53, affecting both code quality and visual fidelity. Interestingly, while removing the debugging module led to a decrease in overall fidelity, PSNR showed a slight improvement, possibly due to the model's focus shifting towards simpler representations, which, though less precise, may result in visually sharper images. Removing both components leads to larger performance declines, including a Pass@1 reduction of -2.50 and an MS-SSIM decrease of -3.07, confirming the necessity of debugging and verification for maintaining both high accuracy and fidelity.

6.4. Human Evaluation

To assess DiagramAgent's performance, three raters with master's degrees evaluated two core tasks—the diagram generation task and the diagram editing task—using a 1-5 scale. A score of 1 represents minimal satisfaction in terms of code accuracy and diagram fidelity, while a score of 5 indicates the highest satisfaction. The evaluation criteria focused on the similarity between the generated diagram and the reference diagram. Figure 4 presents the scores across models. DiagramAgent achieves state-of-the-art scores in both tasks, demonstrating superior accuracy compared to other models. Notably, the results show that DiagramAgent performs better on the diagram editing task than on the diagram generation task. This difference suggests that diagram editing task, which involves modifying an existing codebase, is comparatively easier due to the established

Table 6. Main results for diagram editing (Code Agent). The best result in each metric is bolded.

Model	Size	Pass@1↑	ROUGE-L↑	CodeBLEU↑	Edit Dist.↓	chrF↑	RUBY↑	CLIP-FID↓	LPIPS↓	PSNR↑	MS-SSIM↑
Qwen2.5-Coder-7B [17]	7B	71.50	91.86	97.42	13.26	89.91	86.99	4.79	46.45	11.16	66.76
DeepSeek-Coder-Instruct [14]	33B	90.50	96.64	98.48	5.80	95.73	94.68	2.63	46.25	15.84	86.42
Code-Llama [32]	34B	87.00	52.51	92.55	50.96	65.83	40.22	4.95	44.62	24.10	82.42
WizardCoder [25]	15B	87.50	74.59	95.20	28.91	84.23	63.71	4.92	44.18	**24.24**	82.88
Codegeex4-all [52]	9B	90.00	96.73	98.71	5.39	95.99	95.69	1.93	43.43	11.47	92.35
Starcoder2 [21]	15B	41.00	21.34	90.28	80.79	34.04	14.36	9.44	44.76	11.50	37.93
Yi-Coder [1]	9B	81.50	96.03	98.08	7.00	95.43	93.41	2.68	45.28	13.05	78.59
Llama-3.1-8B-Instruct [11]	8B	24.00	50.85	89.20	55.86	57.03	44.52	14.06	48.59	4.09	21.37
Baichuan2-13B-Chat [44]	13B	39.50	82.07	92.51	30.60	82.33	75.16	10.04	44.80	6.50	37.06
Internlm2_5-20b-chat [7]	20B	57.00	84.31	95.57	21.13	87.98	77.90	6.58	43.85	12.14	55.96
Yi-1.5-34B-chat [48]	34B	90.50	96.64	98.38	6.85	95.78	94.52	2.13	45.86	16.06	85.70
Qwen2-7B-Instruct [17]	7B	81.50	91.51	96.40	17.87	91.34	87.63	3.72	44.87	15.59	76.67
GPT-4o [3]	-	92.42	96.22	97.73	7.31	95.49	94.50	1.89	43.53	14.23	88.43
DeepSeek V2.5 [23]	-	95.00	96.77	98.83	5.04	96.10	94.96	1.63	43.16	12.81	91.97
GLM-4-plus [13]	-	92.00	97.05	98.63	6.06	96.04	95.12	1.54	45.79	13.89	88.31
Gemini [31]	-	72.00	95.09	95.34	7.00	93.32	93.45	2.08	47.57	12.50	85.59
DiagramAgent Code Agent	7B	**98.00**	**98.41**	**99.93**	**3.58**	**97.96**	**97.05**	**1.08**	**40.64**	13.18	**97.00**

Table 7. Ablation study for diagram editing task (Code Agent). Each result shows the performance of DiagramAgent Code Agent under various component configurations, with the decrease in each metric from the full model indicated in parentheses.

Diagram editing	Size	Pass@1↑	ROUGE-L↑	CodeBLEU↑	Edit Dist.↓	chrF↑	RUBY↑	CLIP-FID↓	LPIPS↓	PSNR↑	MS-SSIM↑
DiagramAgent	7B	**98.00**	**98.41**	**99.93**	**3.58**	**97.96**	**97.05**	**1.08**	**40.64**	13.18	**97.00**
– w/o GPT-4o	7B	97.00 (-1.00)	98.19 (-0.22)	99.29 (-0.64)	3.63 (+0.05)	97.88 (-0.08)	96.96 (-0.09)	1.13 (+0.05)	41.95 (+1.31)	13.33 (+0.15)	96.02 (-0.98)
– w/o Compiler	7B	95.50 (-2.50)	98.12 (-0.29)	99.19 (-0.74)	3.84 (+0.26)	97.45 (-0.51)	96.91 (-0.14)	1.26 (+0.18)	42.17 (+1.53)	**16.72** (+3.54)	96.22 (-0.78)
– w/o GPT-4o & Compiler	7B	95.50 (-2.50)	97.92 (-0.49)	99.11 (-0.82)	3.96 (+0.38)	97.30 (-0.66)	96.73 (-0.32)	1.36 (+0.28)	42.21 (+1.57)	15.60 (+2.42)	93.93 (-3.07)

structure, whereas the diagram generation task requires generating a diagram from scratch, posing a higher challenge given the dependency on instruction quality.

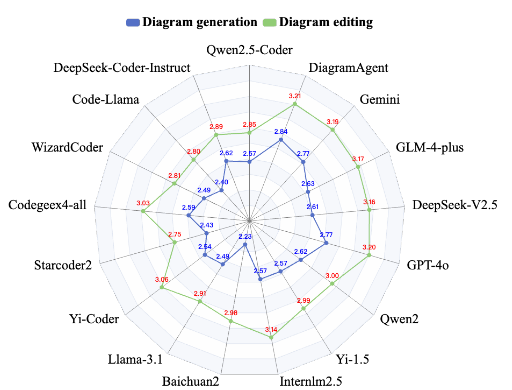

Figure 4. Human evaluation results for different models on diagram generation and Modify diagram generation tasks.

Consequently, achieving accurate diagrams in a single generation step remains a significant challenge, often requiring multiple iterations to refine the output. This observation highlights the importance of DiagramAgent's capability for iterative diagram editing, enabling the generation of precise and refined diagrams. We also provide detailed error analysis in Appendix F.

7. Conclusion and Limitations

In this work, we present DiagramAgent, a framework for Text-to-Diagram generation, alongside DiagramGenBenchmark, a comprehensive dataset that encompasses eight distinct types of structured diagrams. Our extensive experimental evaluations, which include both objective metrics and human assessments, demonstrate that DiagramAgent significantly outperforms existing open-source and close-source models in diagram generation, coding, and editing tasks. To the best of our knowledge, this is the first framework specifically designed not only for generating structured diagrams from textual descriptions but also for editing them. This work sets a new benchmark for text-to-diagram tasks, contributing significantly to the fields of structured diagram generation and editing.

However, there are still areas for further refinement. DiagramAgent's handling of more complex structures, particularly with regard to intricate spatial relationships, remains an area for future development. Additionally, while diagram editing has reached a high level of accuracy, challenges persist in diagram generation, where achieving consistent and precise coding remains critical for producing reliable and accurate modifications.

Acknowledgement

This work was supported by National Science and Technology Major Project (No. 2022ZD0115101), National Natural Science Foundation of China Project (No. 624B2115, No. U21A20427), Shenyang Science and Technology Program (No. 23-407-3-29), Project (No. WU2022A009) from the Center of Synthetic Biology and Integrated Bioengineering of Westlake University, Project (No. WU2023C019) from the Westlake University Industries of the Future Research Funding.

References

[1] 01.AI. Meet yi-coder: A small but mighty llm for code, 2024. 6, 7, 8

[2] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024. 6, 7, 4

[3] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 2, 3, 5, 6, 7, 8, 1, 4

[4] Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu, and Charles Sutton. A survey of machine learning for big code and naturalness. *ACM Computing Surveys*, 51(4):1–37, 2018. 1, 2

[5] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. 3

[6] Jonas Belouadi, Anne Lauscher, and Steffen Eger. Automatikz: Text-guided synthesis of scientific vector graphics with tikz. *ICLR*, 2024. 5, 1

[7] Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, et al. Internlm2 technical report. *arXiv preprint arXiv:2403.17297*, 2024. 3, 6, 7, 8

[8] Hanqun Cao, Cheng Tan, Zhangyang Gao, Yilun Xu, Guangyong Chen, Pheng-Ann Heng, and Stan Z Li. A survey on generative diffusion models. *IEEE Transactions on Knowledge and Data Engineering*, 2024. 1, 2

[9] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. 3

[10] Moreno D'Incà, Elia Peruzzo, Massimiliano Mancini, Dejia Xu, Vidit Goel, Xingqian Xu, Zhangyang Wang, Humphrey Shi, and Nicu Sebe. Openbias: Open-set bias detection in text-to-image generative models. In *CVPR*, pages 12225–12235, 2024. 3

[11] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 3, 6, 7, 8, 4

[12] Lijie Fan, Tianhong Li, Siyang Qin, Yuanzhen Li, Chen Sun, Michael Rubinstein, Deqing Sun, Kaiming He, and Yonglong Tian. Fluid: Scaling autoregressive text-to-image generative models with continuous tokens. *arXiv preprint arXiv:2410.13863*, 2024. 3

[13] Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024. 3, 6, 7, 8, 4

[14] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024. 3, 6, 7, 8

[15] Ruifeng Guo, Jingxuan Wei, Linzhuang Sun, and et.al. A survey on advancements in image-text multimodal models: From general techniques to biomedical implementations. *Computers in Biology and Medicine*, page 108709, 2024. 1

[16] Wenyi Hong, Weihan Wang, Ming Ding, Wenmeng Yu, Qingsong Lv, Yan Wang, Yean Cheng, Shiyu Huang, Junhui Ji, Zhao Xue, et al. Cogvlm2: Visual language models for image and video understanding. *arXiv preprint arXiv:2408.16500*, 2024. 6, 7, 4

[17] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024. 3, 4, 6, 7, 8

[18] Satyam Kumar, Dayima Musharaf, Seerat Musharaf, and Anil Kumar Sagar. A comprehensive review of the latest advancements in large generative ai models. In *ICACIS*, pages 90–103. Springer, 2023. 1, 2

[19] Feng Li, Renrui Zhang, Hao Zhang, Yuanhan Zhang, Bo Li, Wei Li, Zejun Ma, and Chunyuan Li. Llava-next-interleave: Tackling multi-image, video, and 3d in large multimodal models. *arXiv preprint arXiv:2407.07895*, 2024. 6, 7, 4

[20] Kaixin Li, Yuchen Tian, Qisheng Hu, Ziyang Luo, and Jing Ma. Mmcode: Evaluating multi-modal code large language models with visually rich programming problems. *arXiv preprint arXiv:2404.09486*, 2024. 3

[21] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023. 6, 7, 8

[22] Zhiqiu Lin, Deepak Pathak, Baiqi Li, Jiayao Li, Xide Xia, Graham Neubig, Pengchuan Zhang, and Deva Ramanan. Evaluating text-to-visual generation with image-to-text generation. In *ECCV*, pages 366–384. Springer, 2025. 1

[23] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and

efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024. 3, 6, 7, 8

[24] Haoyu Lu, Wen Liu, Bo Zhang, Bingxuan Wang, Kai Dong, Bo Liu, Jingxiang Sun, Tongzheng Ren, Zhuoshu Li, Yaofeng Sun, et al. Deepseek-vl: towards real-world vision-language understanding. *arXiv preprint arXiv:2403.05525*, 2024. 6, 7, 4

[25] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. In *ICLR*, 2024. 3, 6, 7, 8

[26] Xiaoxiao Ma, Mohan Zhou, Tao Liang, Yalong Bai, Tiejun Zhao, Huaian Chen, and Yi Jin. Star: Scale-wise text-to-image generation via auto-regressive representations. *arXiv preprint arXiv:2406.10797*, 2024. 3

[27] W Michael McCracken and Wendy C Newstetter. Text to diagram to symbol: Representational transformations in problem-solving. In *FIE*, pages F2G–13. IEEE, 2001. 1

[28] Rishubh Parihar, VS Sachidanand, Sabariswaran Mani, Tejan Karmali, and R Venkatesh Babu. Precisecontrol: Enhancing text-to-image diffusion models with fine-grained attribute control. In *ECCV*, pages 469–487. Springer, 2025. 1, 3

[29] Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Huajun Chen, et al. Autoact: Automatic agent learning from scratch for qa via self-planning. In *ICLR*, 2024. 3

[30] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *ICML*, pages 8821–8831. Pmlr, 2021. 3

[31] Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024. 3, 6, 7, 8, 4

[32] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023. 3, 6, 7, 8

[33] Chitwan Saharia, William Chan, et al. Photorealistic text-to-image diffusion models with deep language understanding. *NeurIPS*, 35:36479–36494, 2022. 1, 2

[34] Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: How far are we from automating front-end engineering? *arXiv preprint arXiv:2403.03163*, 2024. 3

[35] Cheng Tan, Siyuan Li, Zhangyang Gao, Wenfei Guan, Zedong Wang, Zicheng Liu, Lirong Wu, and Stan Z Li. Openstl: A comprehensive benchmark of spatio-temporal predictive learning. *Advances in Neural Information Processing Systems*, 36:69819–69831, 2023. 2

[36] Cheng Tan, Dongxin Lyu, Siyuan Li, Zhangyang Gao, Jingxuan Wei, Siqi Ma, Zicheng Liu, and Stan Z Li. Peer review as a multi-turn and long-context dialogue with role-based interactions. *arXiv preprint arXiv:2406.05688*, 2024. 4

[37] Cheng Tan, Jingxuan Wei, Zhangyang Gao, Linzhuang Sun, Siyuan Li, Ruifeng Guo, Bihui Yu, and Stan Z Li. Boosting the power of small multimodal reasoning models to match larger models with self-consistency training. In *European Conference on Computer Vision*, pages 305–322. Springer, 2024. 1

[38] Cheng Tan, Jingxuan Wei, Linzhuang Sun, Zhangyang Gao, Siyuan Li, Bihui Yu, Ruifeng Guo, and Stan Z Li. Retrieval meets reasoning: Even high-school textbook knowledge benefits multimodal reasoning. *arXiv preprint arXiv:2405.20834*, 2024. 2

[39] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024. 6, 7, 4

[40] Jingxuan Wei, Linzhuang Sun, Yichong Leng, Xu Tan, Bihui Yu, and Ruifeng Guo. Sentence-level or token-level? a comprehensive study on knowledge distillation. In *IJCAI-24*, pages 6531–6540, 2024. 1

[41] Jingxuan Wei, Linzhuang Sun, Yichong Leng, Xu Tan, Bihui Yu, and Ruifeng Guo. Sentence-level or token-level? a comprehensive study on knowledge distillation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pages 6531–6540, 2024. 1

[42] Chengyue Wu, Yixiao Ge, Qiushan Guo, Jiahao Wang, Zhixuan Liang, Zeyu Lu, Ying Shan, and Ping Luo. Plot2code: A comprehensive benchmark for evaluating multi-modal large language models in code generation from scientific plots. *arXiv preprint arXiv:2405.07990*, 2024. 3

[43] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. In *CVPR*, pages 1316–1324, 2018. 1, 2

[44] Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, et al. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*, 2023. 3, 6, 7, 8

[45] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024. 2, 3

[46] Zhiyu Yang, Zihan Zhou, Shuo Wang, Xin Cong, Xu Han, Yukun Yan, Zhenghao Liu, Zhixing Tan, Pengyuan Liu, Dong Yu, et al. Matplotagent: Method and evaluation for llm-based agentic scientific data visualization. *arXiv preprint arXiv:2402.11453*, 2024. 3

[47] Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, et al. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*, 2024. 6, 7, 4

[48] Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, et al. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*, 2024. 6, 7, 8

[49] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stack-

gan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *ICCV*, pages 5907–5915, 2017. 1, 2

[50] Pan Zhang, Xiaoyi Dong, Yuhang Zang, Yuhang Cao, Rui Qian, Lin Chen, Qipeng Guo, Haodong Duan, Bin Wang, Linke Ouyang, et al. Internlm-xcomposer-2.5: A versatile large vision language model supporting long-contextual input and output. *arXiv preprint arXiv:2407.03320*, 2024. 6, 7, 4

[51] Ziyin Zhang, Chaoyu Chen, Bingchang Liu, Cong Liao, Zi Gong, Hang Yu, Jianguo Li, and Rui Wang. Unifying the perspectives of NLP and software engineering: A survey on language models for code. *TMLR*, 2024. 4

[52] Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Lei Shen, Zihan Wang, Andi Wang, Yang Li, et al. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x. In *ACM SIGKDD*, pages 5673–5684, 2023. 6, 7, 8, 1

[53] Yutong Zhou and Nobutaka Shimada. Vision+ language applications: A survey. In *CVPR*, pages 826–842, 2023. 1, 2

[54] Bocheng Zou, Mu Cai, Jianrui Zhang, and Yong Jae Lee. Vgbench: Evaluating large language models on vector graphics understanding and generation. *arXiv preprint arXiv:2407.10972*, 2024. 5, 1