SketchAgent: Generating Structured Diagrams from Hand-Drawn Sketches

Cheng Tan[1,2*], Qi Chen[3,4*], Jingxuan Wei[3,4†*], Gaowei Wu[3,4*], Zhangyang Gao[1,2], Siyuan Li[1,2], Bihui Yu[3,4], Ruifeng Guo[3,4], Stan Z. Li[1†]

[1]Westlake University
[2]Zhejiang University
[3]University of Chinese Academy of Sciences
[4]Shenyang Institute of Computing Technology, Chinese Academy of Sciences
tancheng@westlake.edu.cn, weijingxuan20@mails.ucas.edu.cn

Abstract

Hand-drawn sketches are a natural and efficient medium for capturing and conveying ideas. Despite significant advancements in controllable natural image generation, translating freehand sketches into structured, machine-readable diagrams remains a labor-intensive and predominantly manual task. The primary challenge stems from the inherent ambiguity of sketches, which lack the structural constraints and semantic precision required for automated diagram generation. To address this challenge, we introduce SketchAgent, a multi-agent system designed to automate the transformation of hand-drawn sketches into structured diagrams. SketchAgent integrates sketch recognition, symbolic reasoning, and iterative validation to produce semantically coherent and structurally accurate diagrams, significantly reducing the need for manual effort. To evaluate the effectiveness of our approach, we propose the Sketch2Diagram Benchmark, a comprehensive dataset and evaluation framework encompassing eight diverse diagram categories, such as flowcharts, directed graphs, and model architectures. The dataset comprises over 6,000 high-quality examples with token-level annotations, standardized preprocessing, and rigorous quality control. By streamlining the diagram generation process, SketchAgent holds great promise for applications in design, education, and engineering, while offering a significant step toward bridging the gap between intuitive sketching and machine-readable diagram generation.

1 Introduction

Hand-drawn sketches are a natural and powerful medium for rapidly conveying ideas, serving as a universal language in creative, technical, and educational workflows [Zhao and Lai, 2022; Zhao *et al.*, 2024; Tan *et al.*, 2024]. From rough brainstorming sessions to preliminary engineering designs, sketches offer an intuitive way to externalize concepts. However, translating these informal and ambiguous drawings into structured, machine-readable diagrams remains an open challenge. Unlike natural image generation tasks [Cao *et al.*, 2024; Huang *et al.*, 2024; Li *et al.*, 2019], which have seen remarkable progress in recent years through techniques such as controllable natural image generation, the sketch-to-diagram task demands more than just visual fidelity—it requires understanding and formalizing the underlying structural and semantic relationships inherent to diagrams.

We introduce a new task, sketch-to-diagram generation, which involves converting a hand-drawn sketch into a structured, machine-readable diagram. As shown in Figure 1, this task differs fundamentally from controllable natural image generation, as it focuses not on generating aesthetically pleasing visuals but on synthesizing a precise, semantically meaningful diagram that adheres to specific structural rules. This transformation requires solving several core challenges: **(1) handling the inherent ambiguity and variability in freehand sketches, (2) preserving the spatial and structural relationships between diagram components, and (3) producing an output that is both syntactically valid and semantically aligned with the user's intent**. These challenges make sketch-to-diagram generation a highly specialized and underexplored problem, distinct from existing works.

To address the lack of standardized resources for sketch-to-diagram research, we introduce the Sketch2Diagram Benchmark, a comprehensive dataset and evaluation framework designed to support the development and assessment of models for this task. The dataset spans eight diverse diagram categories, including flowcharts, directed graphs, and model architectures, and consists of over 6,000 high-quality examples. Each example includes a hand-drawn sketch paired with its corresponding structured diagram representation. The dataset is meticulously curated, featuring token-level annotations, standardized preprocessing, and rigorous quality control, ensuring its reliability for both training and evaluation purposes.

Building on this benchmark dataset, we propose SketchAgent, an end-to-end system for automating the transformation of hand-drawn sketches into structured diagrams. The system begins by converting an input sketch into a symbolic code representation, which abstracts its structural and spatial properties into a machine-readable format, bridging the gap between informal freehand drawings and precise computational diagrams. From there, SketchAgent performs iterative refinement to improve the accuracy, coherence, and validity of the

†Corresponding author.

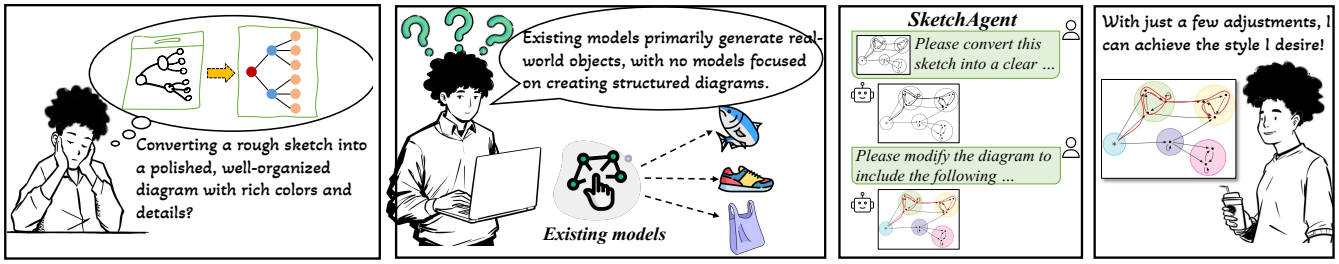

Figure 1: SketchAgent automates the transformation of hand-drawn sketches into structured diagrams.

code representation, ensuring the final diagram adheres to the user's intent while satisfying all structural constraints.

Our main contributions are as follows:

- We formally define the task of converting hand-drawn sketches into structured diagrams, distinguishing it from related tasks such as controllable image generation.
- We introduce a benchmark dataset of hand-drawn sketches and their corresponding structured diagram representations, offering a standardized resource for training and evaluation.
- We propose SketchAgent, a modular system that automates the sketch-to-diagram transformation process, integrating sketch recognition, symbolic reasoning, iterative refinement, and verification into a unified pipeline.

2 Related Work

2.1 Controllable Image Generation

Controllable image generation aims to synthesize images that adhere to specific constraints [Cao *et al.*, 2024; Huang *et al.*, 2024]. Existing methods can be categorized into three main approaches: GAN-based, diffusion-based, and multimodal fusion techniques. Early GAN-based methods, such as ControlGAN [Li *et al.*, 2019], introduced fine-grained text-conditioned image synthesis but suffered from instability and mode collapse. Diffusion models have since become the dominant paradigm, offering more stable and high-quality generation. Diffusion Self-Guidance [Epstein *et al.*, 2023] and MultiDiffusion [Bar-Tal *et al.*, 2023] enable explicit control over object positioning and spatial structure, while Control-GPT [Zhang *et al.*, 2023b] leverages GPT-4-generated sketches for improved spatial consistency. Other approaches integrate LLMs or additional modalities for enhanced control, such as MoMA [Song *et al.*, 2025], which fuses textual and visual embeddings, and MM-Diff [Wei *et al.*, 2024b], which refines personalization through CLIP-based representations. Furthermore, PALP [Arar *et al.*, 2024] enhances alignment with complex textual prompts by optimizing cross-modal score matching. Despite these advancements, existing approaches primarily focus on photorealistic image synthesis, making them insufficient for structured and logic-constrained generation tasks like diagrams [Cao *et al.*, 2024; Huang *et al.*, 2024]. While diffusion-based models offer control over spatial attributes [Epstein *et al.*, 2023; Bar-Tal *et al.*, 2023], they lack explicit structural reasoning capabilities required for diagram generation.

2.2 Controllable Code Generation

Controllable code generation aims to produce structured and executable code while adhering to specific constraints [Shin and Nam, 2021; Wei *et al.*, 2025]. Language model-based approaches leverage pre-trained models to improve code synthesis. Magicoder [Wei *et al.*, 2024a] enhances multi-language code generation through OSS-INSTRUCT, while VeriGen [Thakur *et al.*, 2024] tailors language models for Verilog synthesis by curating specialized training datasets. Structure-aware methods refine code generation by integrating abstract syntax trees and data flow graphs. StructCoder [Tipirneni *et al.*, 2024] introduces a structure-aware self-attention mechanism, and CoTexT [Phan *et al.*, 2021] applies multi-task learning to enhance text-to-code understanding. Planning-based techniques decompose complex tasks into stepwise solutions, as seen in Self-Planning Code Generation [Jiang *et al.*, 2024], while reinforcement learning-based approaches such as CodeRL [Le *et al.*, 2022] optimize model adaptation through reward-based fine-tuning. Execution-enhanced methods ensure generated code correctness by leveraging runtime validation. MBR-EXEC [Shi *et al.*, 2022] employs minimum Bayesian risk decoding based on execution, whereas CODET [Chen *et al.*, 2022] generates test cases to filter invalid code. Besides, real-world integration studies, such as In-IDE Code Generation [Xu *et al.*, 2022], evaluate the practical utility.

While these methods advance code generation in terms of syntax, semantics, and execution fidelity, they remain constrained to text-based inputs, lacking the capability to synthesize code from sketch-based conceptualizations. Furthermore, existing controllable code generation approaches do not inherently support structured diagram generation, limiting their applicability in domains requiring logical and hierarchical visual representations. [Ghosh *et al.*, 2018; Almazroi *et al.*, 2021] have focused on extracting structured representations from textual descriptions, but these methods do not generalize to sketch-driven workflows.

3 Method

The system consists of three modules: the Sketch-to-Code Agent, the Editing Code Agent, and the Check Agent, each responsible for specific tasks. Given a sketch S and a user-specified instruction set Q, SketchAgent generates an initial code representation, refines it based on additional instructions, and verifies the final output before rendering the structured diagram. The overall workflow is illustrated in Figure 2.

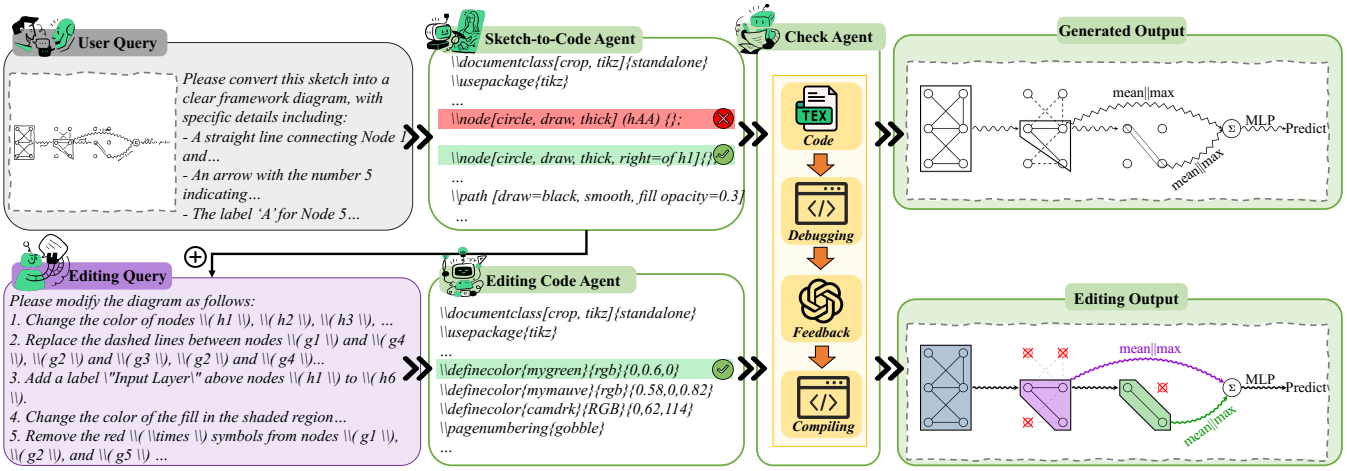

Figure 2: The SketchAgent pipeline, consisting of three main modules: Sketch-to-Code Agent, Editing Code Agent, and Check Agent.

3.1 Sketch-to-Code Agent

The Sketch-to-Code Agent maps a hand-drawn sketch S and an instruction set Q to an initial code representation C_k, capturing the structural semantics of the sketch. This process is formulated as:

$$C_k = \mathcal{F}_k(S, Q), \tag{1}$$

where $\mathcal{F}_k$ represents the transformation function. The output C_k is modeled as a sequence of tokens, where each token corresponds to a diagram component or an attribute.

To ensure the generated code aligns with the expected structure, we define the objective as minimizing the negative log-likelihood of the sequence:

$$\mathcal{L}_k = -\mathbb{E}_{C_k \sim P(C|S,Q)} \sum_{t=1}^{T} \log P(C_k^{(t)} \mid C_k^{(<t)}, S, Q), \tag{2}$$

where T is the sequence length, $C_k^{(t)}$ is the t-th token, and $P(\cdot)$ denotes its conditional probability given the preceding sequence. Optimizing this objective ensures C_k remains structurally valid and semantically aligned with S.

3.2 Editing Code Agent

The Editing Code Agent refines the initial code representation C_k based on an additional instruction set Q', generating an updated version C_e. This process is formalized as:

$$C_e = \mathcal{F}_e(C_k, Q'), \tag{3}$$

where $\mathcal{F}_e$ represents the refinement function. The objective is to minimize the discrepancy between C_e and the expected output, ensuring modifications align with the intended diagram structure. To achieve this, we minimize the negative log-likelihood of the sequence:

$$\mathcal{L}_e = -\sum_{t=1}^{T} \log P(C_e^{(t)} \mid C_e^{(<t)}, C_k, Q'), \tag{4}$$

where T is the sequence length, $C_e^{(t)}$ is the t-th token, and $P(\cdot)$ denotes its conditional probability given the preceding sequence and input conditions. Optimizing this objective ensures C_e effectively integrates the modifications specified in Q' while preserving the structural integrity of C_k.

3.3 Check Agent

The Check Agent verifies and refines the generated code C_e to produce the final executable representation C_f. This process is formalized as:

$$C_f = \mathcal{F}_f(C_e), \tag{5}$$

where $\mathcal{F}_f$ denotes the verification and debugging function. The goal is to ensure C_f is syntactically correct, executable, and aligned with the original sketch S and instructions Q.

To achieve this, the Check Agent begins by performing syntax validation and debugging to ensure that the generated code representation C_e is syntactically correct and compilable. If C_e fails to compile, it is sent back for regeneration by either the Sketch-to-Code Agent or the Editing Code Agent. Once compilation succeeds, we use GPT-4o to compare the compiled diagram D with the original sketch S and the user instructions Q. If the generated diagram aligns with the expected structure, the process is finalized; otherwise, the Check Agent triggers a fallback mechanism, prompting the responsible agent to regenerate C_e. This iterative validation process guarantees that only structurally coherent and semantically correct diagrams are finalized.

4 Sketch2Diagram Benchmark

The Sketch2Diagram Benchmark introduces a comprehensive dataset and evaluation framework for sketch-to-diagram tasks. It features eight diverse diagram categories, standardized with rigorous quality control and token-level statistics. Clear metrics evaluate sketch-to-code and code editing tasks, ensuring thorough assessment. The data collection and processing pipeline for the Sketch2Diagram Benchmark is meticulously designed to ensure a comprehensive and high-quality dataset. This process is divided into three key stages: *Data Collection*, *Data Processing*, and *Human Inspection*, as illustrated in Figure 3.

Data Collection. The first stage involves gathering open-source `.tex` files of logical diagrams from multiple repositories, including datikz-v2, GitHub, and Overleaf. These

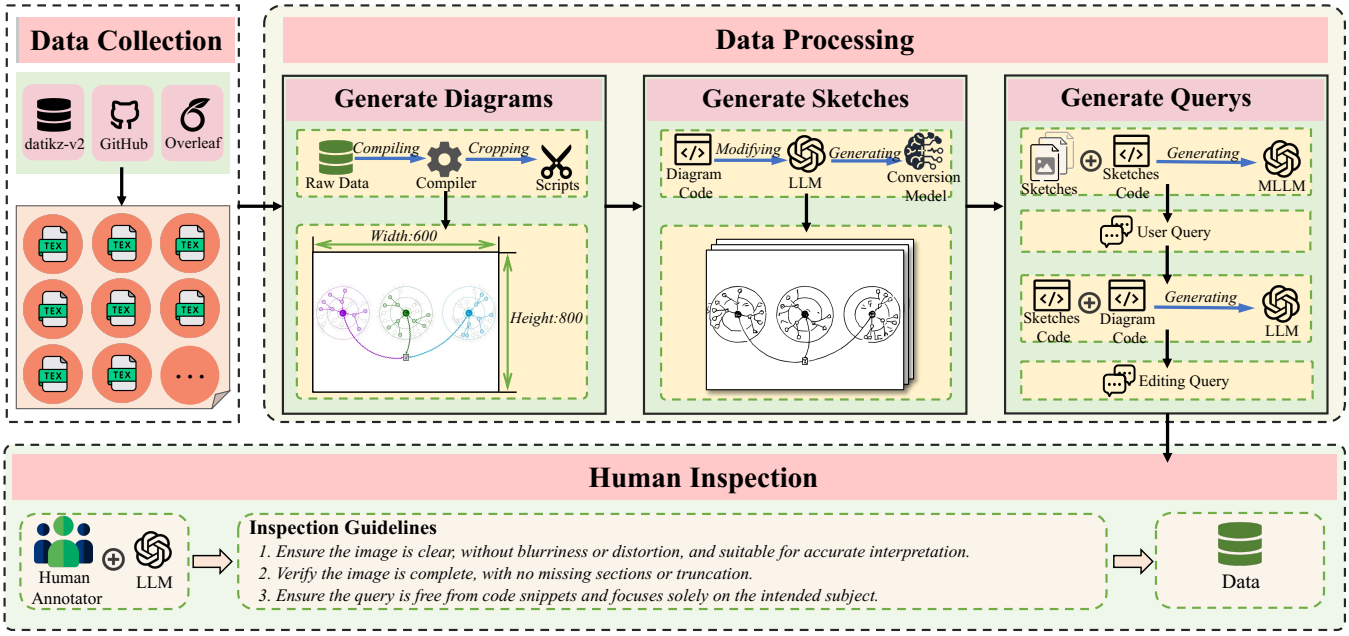

Figure 3: The data collection and processing pipeline for the Sketch2Diagram Benchmark.

sources provide a diverse range of diagrams across various domains, ensuring the dataset's broad applicability. The collected `.tex` files are then compiled into diagram images using standard LaTeX compilers. This step guarantees that the diagrams accurately reflect their logical structures as intended by their original creators.

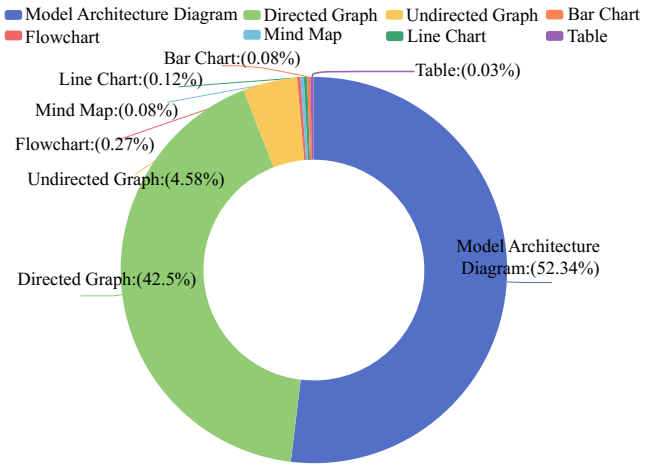

Figure 4: Category distribution in Sketch2Diagram.

Data Processing. We standardize the compiled diagrams to ensure uniformity and facilitate sketch-to-diagram tasks. Images are cropped to remove blank spaces and resized to 800×600 pixels. Colors and intricate details are stripped from diagrams to produce simplified sketch representations. GPT-4O generates corresponding sketch codes, ensuring consistency with the original diagrams. For model evaluation, we create two query types: (1) **User Queries**, which pair sketch images with codes to add supplementary details; and (2) **Editing Queries**, which capture differences between sketch and original diagram codes to guide refinement.

Human Inspection. The final stage involves a rigorous manual inspection process to ensure data quality. Human annotators adhere to three strict guidelines: First, images must be clear, free of blurriness or distortion, to ensure accurate interpretation. Second, images must be complete, with no missing sections or truncations. Third, queries are reviewed to exclude code snippets and focus on descriptive elements relevant to the task. These measures ensure the dataset's reliability and suitability for model evaluation.

4.1 Data Analysis

Diversity and Imbalance Challenges. Figures 5 and 4 illustrate the dataset's diversity and distribution. Figure 5 showcases examples from eight diagram categories, including undirected graphs, model architecture diagrams, and flowcharts, highlighting their real-world relevance. Figure 4 reveals an imbalance, with model architecture diagrams (52.34%) and directed graphs (42.5%) dominating, while categories like flowcharts and mind maps are underrepresented.

Token Length Statistics. Table 1 summarizes token length statistics for the Sketch2Diagram dataset, categorized by sketch-to-code (S2C) and code-editing (C2C) tasks. The dataset contains a total of 4824 training samples and 1206 test samples. Query lengths range from a minimum of 28 tokens for S2C tasks to a maximum of 170,894 tokens for C2C tasks. Answer lengths exhibit similar variability, with maximum values reaching 170,371 tokens. These diverse token lengths reflect the dataset's ability to evaluate models across tasks with varying complexities and input-output structures.

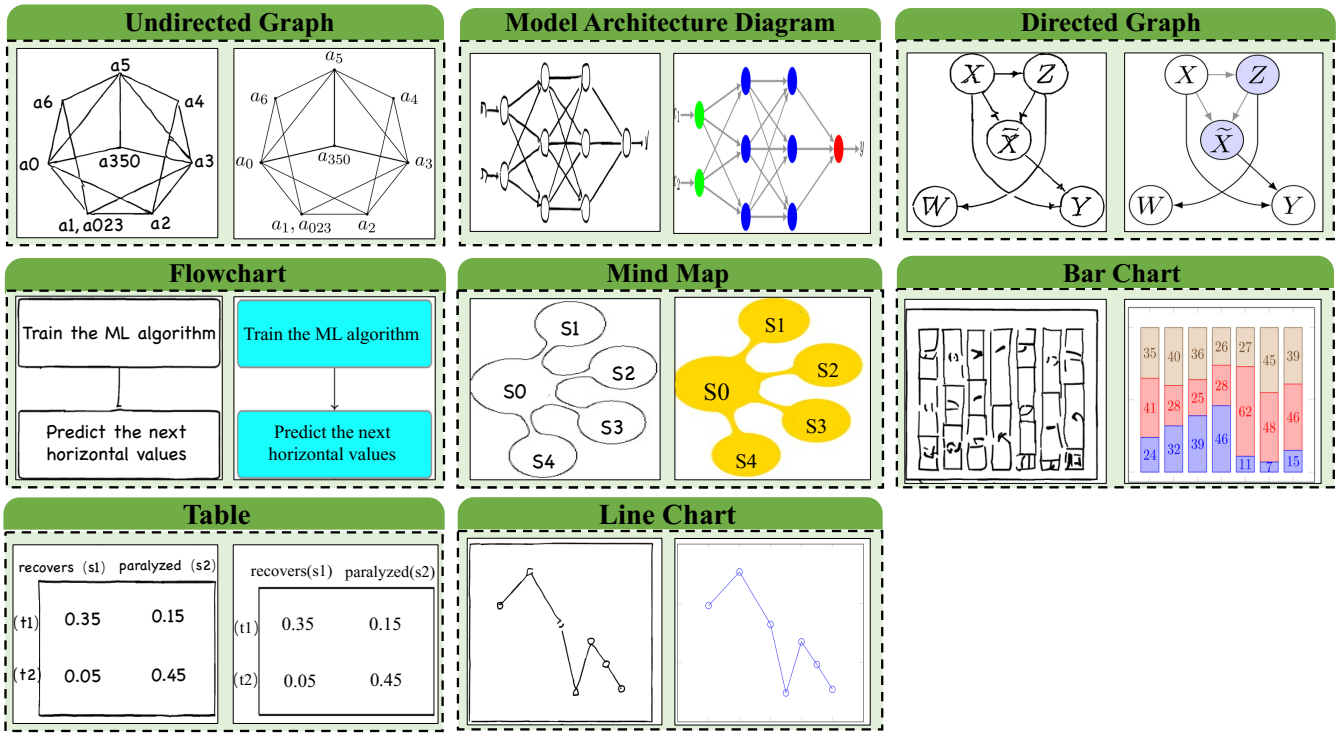

Figure 5: Examples of diagram types in the Sketch2Diagram.

Statistic	Train S2C	Train C2C	Test S2C	Test C2C
Total Samples				
Sample Count	4824	4824	1206	1206
Query Length (tokens)				
Minimum	28	165	39	208
Maximum	1118	170894	1115	28715
Average	243.9	1161.0	243.2	1074.3
Answer Length (tokens)				
Minimum	119	125	133	133
Maximum	170371	170371	28633	28633
Average	1010.0	1064.1	922.4	977.8

Table 1: Key statistics of the Sketch2Diagram dataset.

4.2 Evaluation Metrics

We employ a comprehensive set of evaluation metrics. For sketch-to-code, Pass@1 measures functional correctness, while ROUGE-L, BLEU, CodeBLEU (C-BLEU), chrF, Edit Distance (ED), BLEURT, and RUBY evaluate textual and semantic similarity. For code editing, we extend these metrics to include diagram fidelity measures such as FID, KID, CLIP-FID (C-FID), Inception Score (IS), LPIPS, and SSIM.

5 Experiment

Setup The Sketch-to-Code Agent is based on Qwen2-VL-7B [Wang *et al.*, 2024], while the Editing Code Agent utilizes Qwen2.5-Coder-7B [Hui *et al.*, 2024]. Both agents were fine-tuned over four epochs on a 4×80GB A100 GPU setup. The input token length for both agents is set to 4096 tokens.

Model For sketch generation, the Sketch-to-Code Agent is compared against several state-of-the-art models, including Yi-VL [Team, 2025c], Qwen2-VL [Wang *et al.*, 2024], Internlm-Xcomposer2.5 [Zhang *et al.*, 2023a], Llama-3.2-Vision [AI, 2025b], Phi-3.5-Vision [Abdin *et al.*, 2024], Llava-v1.6 [Liu and Others, 2025], Cogvlm2-Llama3 [Hong *et al.*, 2024], and DeepSeek-VL [Team, 2025f], with close-source models such as GPT-4o [Achiam *et al.*, 2023], GLM-4-plus [Team, 2025g], and Gemini-1.5-Pro [Team *et al.*, 2024] also implemented. For the sketch editing task, the Editing Code Agent is assessed alongside specialized code models, including Qwen2.5-Coder [Hui *et al.*, 2024], DeepSeek-Coder-Instruct [Guo *et al.*, 2024], Code-Llama [Roziere *et al.*, 2023], WizardCoder [Luo *et al.*, 2023], CodeGeeX4-All [Team, 2025d], Starcoder2 [Lozhkov *et al.*, 2024], Yi-Coder [Team, 2025b], Llama 3.1 [AI, 2025a], Baichuan2 [Yang *et al.*, 2023], Internlm2.5 [Cai *et al.*, 2024], Yi-1.5 [Team, 2025a], and Qwen2 [Team, 2024], and close-source models like GPT-4o [Achiam *et al.*, 2023], DeepSeekV2.5 [Team, 2025e], GLM-4-Plus [Team, 2025g], and Gemini-1.5-Pro [Team *et al.*, 2024].

5.1 Sketch generation

Main Results SketchAgent leverages compiler principles and integrates GPT-4o as a feedback mechanism to achieve state-of-the-art performance in translating logical structure diagrams into executable code. As demonstrated in Table 2, SketchAgent significantly outperforms both open-source and close-source models across key code generation metrics. Notably, it achieves a Pass@1 score of 82.34, substantially

Model	Size	Pass@1↑	ROUGE-L↑	C-BLEU↑	BLEU↑	ED↓	chrF↑	BLEURT↑	RUBY↑
(a) Main results									
Yi-VL-34B	34B	0.25	33.68	77.79	8.23	94.17	19.87	37.28	21.52
Qwen2-VL-7B-Instruct	7B	52.40	33.72	75.19	7.37	90.38	26.26	32.91	21.97
internlm-xcomposer2d5-7b	7B	0.08	30.32	78.65	3.61	94.50	19.98	37.18	18.57
Llama-3.2-11B-Vision-Instruct	11B	40.09	39.17	80.36	16.61	87.35	28.09	37.64	26.05
Phi-3.5-vision-instruct	4B	16.64	13.41	68.95	5.20	97.37	12.65	34.27	7.82
llava-v1.6-34b	34B	8.51	24.63	76.75	10.68	96.73	27.49	33.75	13.96
cogvlm2-llama3-chat-19B	19B	0.00	12.60	70.57	3.16	98.03	12.27	32.73	7.35
deepseek-vl-7b-chat	7B	0.33	26.94	80.74	12.06	95.68	22.47	37.18	15.21
GPT-4o	-	51.12	34.42	79.56	12.65	92.20	29.51	33.40	20.85
Gemini-1.5-pro	-	61.94	39.80	81.02	12.64	89.10	30.61	32.53	25.86
GLM-4V-Plus	-	66.09	40.53	81.30	12.73	88.16	30.01	33.00	26.70
SketchAgent	7B	**82.34**	**52.96**	**86.02**	**29.61**	**73.16**	**47.88**	**46.34**	**41.17**
(b) Ablation study									
w/o GPT-4o	7B	81.18	52.90	86.01	29.34	73.16	47.36	46.28	41.05
w/o Compiler	7B	80.85	52.83	86.01	29.33	73.23	47.78	46.25	41.02
w/o GPT-4o & Compiler	7B	78.52	52.39	85.98	25.88	73.36	47.82	46.38	40.78

Table 2: (a) Main results: Performance comparison on sketch generation with open-source and closed-source models on key code generation metrics. The best result is highlighted in bold. (b) Ablation study: Performance under different component configurations.

surpassing powerful large language models such as GPT-4o (51.12), Gemini-1.5-pro (61.94), and GLM-4V-Plus (66.09). In addition, SketchAgent excels in other critical metrics, including CodeBLEU (86.02), BLEU (29.61), and BLEURT (46.34), indicating its strong ability to generate syntactically and semantically accurate code. These evaluation results underscore SketchAgent's superior capability in producing high-quality, executable code from sketch representations.

Ablation Study We conduct an ablation study as detailed in Table 2. The results reveal the critical role played by both the compiler module and GPT-4o feedback mechanism. Removing the compiler results in a 1.49-point drop in Pass@1, indicating the compiler's significant contribution to precise code generation. Removing the GPT-4o feedback mechanism leads to a 1.16-point decline in Pass@1, demonstrating GPT-4o's role in validating outputs. More importantly, when both the compiler and GPT-4o are removed, the Pass@1 score experiences a substantial decrease of 3.82 points, falling to 78.52, highlighting their effect on SketchAgent.

Human Evaluation We employed three professional evaluators to assess the outputs. A score of 1 indicated the lowest quality, while 5 represented the highest quality. As shown in Figure 6, SketchAgent consistently achieved the highest quality based on objective assessment metrics. Furthermore, SketchAgent surpassed the accuracy of other models. Interestingly, the results revealed that SketchAgent performed better on the editing task compared to the generation task.

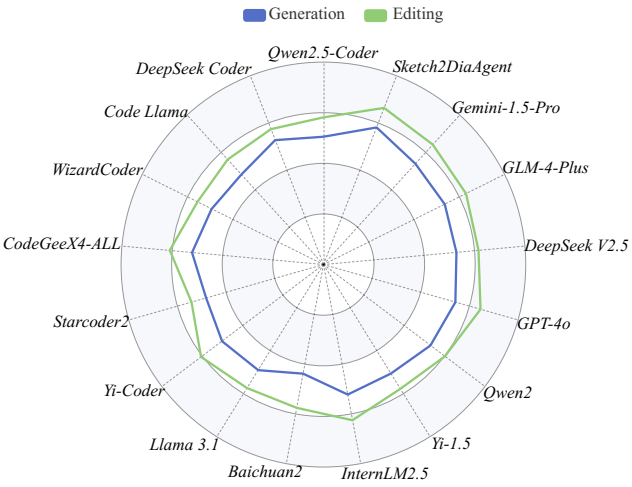

Figure 6: Human evaluation results for different models on diagram generation and Modify diagram generation tasks.

5.2 Sketch Editing

Main Results SketchAgent's Editing Code Agent demonstrates high performance across several key metrics, as shown in Table 3. It achieves the highest Pass@1 score of 93.12, surpassing other models such as Qwen2.5-Coder (62.19) and DeepSeek Coder (81.92). Moreover, SketchAgent excels in CodeBLEU and BLEU, with scores of 98.63 and 87.38, respectively, surpassing WizardCoder (96.78 and 76.39) and Code Llama (96.57 and 77.20). In terms of visual fidelity metrics, SketchAgent performs well with a FID of 130.1494, outperforming models such as DeepSeek Coder (222.42) and Qwen2.5-Coder (246.79), demonstrating its strong visual coherence and competitiveness.

Ablation Study The ablation study in Table 3 evaluates the impact of feedback module and compilation module on SketchAgent's performance. The full model achieves a Pass@1 score of 93.12, slightly lower than the 94.69 of the model without feedback, suggesting that the removal of feedback marginally improves performance. However, both models outperform the one without compilation, which scores 91.21, highlighting the importance of the compilation mod-

Model	Size	Pass@1↑	ROUGE↑	C-BLEU↑	BLEU↑	ED↓	chrF↑	BLEURT↑	RUBY↑	IS↑	FID↓	KID↓	C-FID↓	LPIPS↓	SSIM↑
(a) Main results															
Qwen2.5-Coder	7B	62.19	89.43	95.59	78.11	23.26	88.22	46.25	82.56	3.82	246.80	19.59	64.77	49.68	0.43
DeepSeek Coder	33B	81.92	88.53	95.37	75.38	22.31	87.67	46.28	81.66	3.95	222.42	1.90	29.16	54.07	0.51
Code Llama	34B	59.78	96.23	96.57	77.20	6.07	87.76	73.62	93.66	3.02	283.48	2.10	36.63	44.13	0.28
WizardCoder	15B	92.12	94.30	96.78	76.39	10.03	90.85	73.82	90.84	4.03	283.48	2.09	36.63	44.13	0.28
CodeGeeX4-ALL	9B	69.32	86.07	93.24	63.11	21.85	84.99	68.64	79.19	2.70	209.63	12.45	53.08	44.29	2.81
Starcoder2	15B	88.39	86.31	91.38	57.68	22.10	83.08	69.13	83.86	3.75	283.48	2.10	36.63	44.13	0.28
Yi-Coder	9B	66.67	34.17	39.34	8.14	30.87	33.83	49.12	31.68	3.35	228.94	2.09	30.11	53.44	0.44
Llama 3.1	8B	59.95	71.71	85.77	68.47	35.35	69.72	60.95	65.53	3.68	238.25	1.37	26.88	55.61	0.42
Baichuan2	13B	18.57	41.88	75.37	8.51	65.03	40.01	47.83	36.16	4.43	148.70	9.73	33.80	75.84	6.42
InternLM2.5	20B	80.43	90.09	94.81	74.78	16.03	88.64	70.86	84.59	3.09	220.07	1.00	29.11	48.84	0.51
Yi-1.5	34B	53.23	83.14	94.19	60.51	28.24	81.85	66.41	75.69	3.50	226.15	1.77	30.12	49.97	10.43
Qwen2	7B	58.96	78.93	93.01	60.22	37.30	76.56	61.60	69.73	3.22	237.49	1.34	27.85	50.50	0.42
GPT-4o	-	91.79	96.81	96.78	86.40	14.23	92.69	69.46	94.87	5.51	139.56	0.23	13.99	46.94	28.41
DeepSeek V2.5	-	83.67	94.30	96.77	80.54	17.02	90.85	71.01	90.84	4.57	203.27	0.98	33.18	42.16	0.43
GLM-4-Plus	-	87.06	95.88	96.78	86.06	10.02	91.02	73.27	88.81	**5.71**	243.45	1.32	23.44	47.38	0.61
Gemini-1.5-Pro	-	83.75	94.26	95.42	85.72	19.08	90.75	69.08	90.77	3.83	246.68	19.58	64.70	49.57	0.26
SketchAgent	7B	**93.12**	**97.68**	**98.63**	**87.38**	**5.92**	**96.38**	**74.33**	**96.26**	5.51	**130.15**	**0.22**	**12.24**	**42.00**	**30.13**
(b) Ablation study															
- w/o Feedback	7B	94.69	97.65	98.62	87.38	14.21	96.32	71.28	94.21	5.20	143.54	0.26	13.54	47.29	32.47
- w/o Compilation	7B	91.21	96.26	98.62	85.12	5.95	94.77	73.05	95.68	5.48	139.48	0.30	13.03	47.15	27.09
- w/o both	7B	88.39	95.39	96.77	83.56	10.05	92.68	70.21	90.11	4.92	154.23	0.51	14.25	50.71	26.33

Table 3: (a) Main results: Performance comparison on sketch editing with open-source and closed-source models on both code generation and fidelity metrics. The best result is highlighted in bold. (b) Ablation study: Performance under different component configurations.

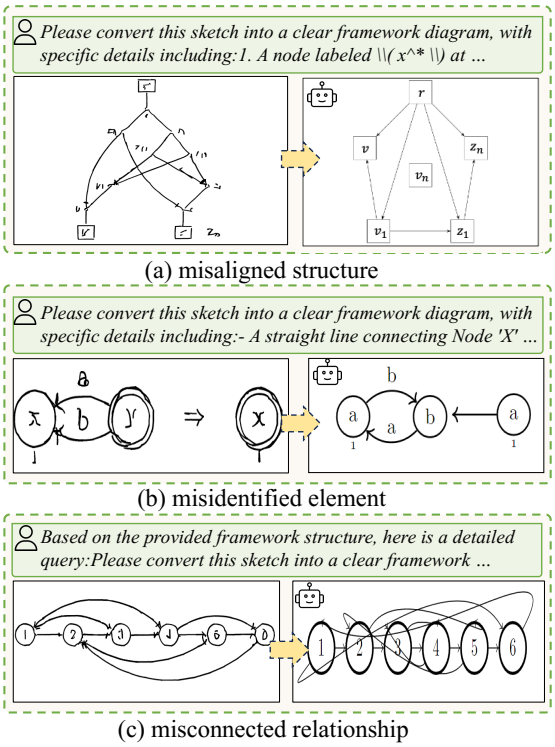

Figure 7: The error examples of SketchAgent.

ule. For ROUGE-L, the full model leads with 97.68, followed by the model without feedback at 97.65, while the model without compilation drops to 96.26, emphasizing the contribution of the compilation module. In FID, the full model outperforms both ablated versions with a score of 130.15, demonstrating the importance of both components in maintaining visual coherence. The model without both feedback and compilation shows higher KID and CLIP-FID scores but sacrifices performance in other key metrics, indicating the necessity of both components for optimal performance.

5.3 Error Analysis

As shown in Figure 7, SketchAgent encounters errors in three key areas: misaligned structures, misidentified elements, and misconnected relationships. Misaligned structures occur when the model fails to accurately capture the underlying structure of the objects in the sketch. This leads to incomplete or overly simplified outputs.

Misidentified elements refer to errors in recognizing individual components of a sketch. For instance, the model may distort characters, fail to reproduce their correct proportions, or misrepresent attributes such as font style or orientation. These errors stem from the variability and ambiguity in the sketches, where subtle variations in shape can confuse recognition. Misconnected relationships arise when the system misinterprets the spatial or relational connections between elements. For example, arrows may connect incorrect components, omit intermediate nodes, or bypass key elements, leading to diagrams with logical inconsistencies. Such errors undermine the semantic integrity of the generated diagram.

6 Conclusion

In this work, we present SketchAgent, a modular and end-to-end system for transforming hand-drawn sketches into structured diagrams. SketchAgent demonstrates its ability to produce accurate and semantically coherent diagrams with minimal human intervention. Moreover, we introduce the Sketch2Diagram Benchmark, a comprehensive dataset featuring over 6,000 high-quality examples spanning eight categories. Extensive experiments demonstrate that SketchAgent outperforms state-of-the-art models across key metrics, achieving superior accuracy and visual coherence.

Acknowledgements

This work was supported by National Science and Technology Major Project (No. 2022ZD0115101), National Natural Science Foundation of China Project (No. 624B2115, No. U21A20427), Project (No. WU2022A009) from the Center of Synthetic Biology and Integrated Bioengineering of Westlake University.

Contribution Statement

The first four authors are equal contribution.

References

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

[Achiam *et al.*, 2023] 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.

[AI, 2025a] Meta AI. Llama 3.1: A state-of-the-art large language model, 2025.

[AI, 2025b] Meta AI. Llama-3.2-11b-vision-instruct: A multimodal instruction-tuned model, 2025.

[Almazroi *et al.*, 2021] Abdulwahab Ali Almazroi, Laith Abualigah, Mohammed A Alqarni, Essam H Houssein, Ahmad Qasim Mohammad AlHamad, and Mohamed Abd Elaziz. Class diagram generation from text requirements: An application of natural language processing. *Deep Learning Approaches for Spoken and Natural Language Processing*, pages 55–79, 2021.

[Arar *et al.*, 2024] Moab Arar, Andrey Voynov, Amir Hertz, Omri Avrahami, Shlomi Fruchter, Yael Pritch, Daniel Cohen-Or, and Ariel Shamir. Palp: prompt aligned personalization of text-to-image models. In *SIGGRAPH*, pages 1–11, 2024.

[Bar-Tal *et al.*, 2023] Omer Bar-Tal, Lior Yariv, Yaron Lipman, and Tali Dekel. Multidiffusion: Fusing diffusion paths for controlled image generation. *ICLR*, 2023.

[Cai *et al.*, 2024] 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.

[Cao *et al.*, 2024] Pu Cao, Feng Zhou, Qing Song, and Lu Yang. Controllable generation with text-to-image diffusion models: A survey. *arXiv preprint arXiv:2403.04279*, 2024.

[Chen *et al.*, 2022] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397*, 2022.

[Epstein *et al.*, 2023] Dave Epstein, Allan Jabri, Ben Poole, Alexei Efros, and Aleksander Holynski. Diffusion self-guidance for controllable image generation. *NeurIPS*, 36:16222–16239, 2023.

[Ghosh *et al.*, 2018] Sutirtha Ghosh, Prasenjit Mukherjee, Baisakhi Chakraborty, and Rezaul Bashar. Automated generation of er diagram from a given text in natural language. In *iCMLDE*, pages 91–96. IEEE, 2018.

[Guo *et al.*, 2024] 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.

[Hong *et al.*, 2024] 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.

[Huang *et al.*, 2024] Shanshan Huang, Qingsong Li, Jun Liao, Shu Wang, Li Liu, and Lian Li. Controllable image synthesis methods, applications and challenges: a comprehensive survey. *Artificial Intelligence Review*, 57(12):336, 2024.

[Hui *et al.*, 2024] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[Jiang *et al.*, 2024] Xue Jiang, Yihong Dong, Lecheng Wang, Zheng Fang, Qiwei Shang, Ge Li, Zhi Jin, and Wenpin Jiao. Self-planning code generation with large language models. *TOSEM*, 33(7):1–30, 2024.

[Le *et al.*, 2022] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *NeurIPS*, 35:21314–21328, 2022.

[Li *et al.*, 2019] Bowen Li, Xiaojuan Qi, Thomas Lukasiewicz, and Philip Torr. Controllable text-to-image generation. *NeurIPS*, 32, 2019.

[Liu and Others, 2025] Haotian Liu and Others. Llava-v1.6-34b: Large multimodal language vision model, 2025.

[Lozhkov *et al.*, 2024] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier,

Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024.

[Luo *et al.*, 2023] 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. *arXiv preprint arXiv:2306.08568*, 2023.

[Phan *et al.*, 2021] Long Phan, Hieu Tran, Daniel Le, Hieu Nguyen, James Anibal, Alec Peltekian, and Yanfang Ye. Cotext: Multi-task learning with code-text transformer. *arXiv preprint arXiv:2105.08645*, 2021.

[Roziere *et al.*, 2023] 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.

[Shi *et al.*, 2022] Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I Wang. Natural language to code translation with execution. *arXiv preprint arXiv:2204.11454*, 2022.

[Shin and Nam, 2021] Jiho Shin and Jaechang Nam. A survey of automatic code generation from natural language. *Journal of Information Processing Systems*, 17(3):537–555, 2021.

[Song *et al.*, 2025] Kunpeng Song, Yizhe Zhu, Bingchen Liu, Qing Yan, Ahmed Elgammal, and Xiao Yang. Moma: Multimodal llm adapter for fast personalized image generation. In *ECCV*, pages 117–132. Springer, 2025.

[Tan *et al.*, 2024] 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.

[Team *et al.*, 2024] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

[Team, 2024] Qwen Team. Qwen2: A scalable and versatile language model, 2024.

[Team, 2025a] 01.AI Team. Yi-1.5: Open foundation models by 01.ai, 2025.

[Team, 2025b] 01.AI Team. Yi-coder: Open foundation models by 01.ai, 2025.

[Team, 2025c] 01.AI Team. Yi-vl-34b: A multimodal foundation model by 01.ai, 2025.

[Team, 2025d] CodeGeeX Team. Codegeex4: Open multilingual code generation model, 2025.

[Team, 2025e] DeepSeek Team. Deepseek v2.5: A scalable and efficient mixture-of-experts language model, 2025.

[Team, 2025f] DeepSeek AI Team. Deepseek-vl-7b-chat: A vision-language model for advanced multimodal understanding, 2025.

[Team, 2025g] Zhipu AI Team. Glm-4-plus: A multilingual foundation model for general purpose ai, 2025.

[Thakur *et al.*, 2024] Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. Verigen: A large language model for verilog code generation. *ACM Trans. on Design Automation of Electronic Systems*, 29(3):1–31, 2024.

[Tipirneni *et al.*, 2024] Sindhu Tipirneni, Ming Zhu, and Chandan K Reddy. Structcoder: Structure-aware transformer for code generation. *TKDD*, 18(3):1–20, 2024.

[Wang *et al.*, 2024] 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.

[Wei *et al.*, 2024a] Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with oss-instruct. In *ICML*, 2024.

[Wei *et al.*, 2024b] Zhichao Wei, Qingkun Su, Long Qin, and Weizhi Wang. Mm-diff: High-fidelity image personalization via multi-modal condition integration. *arXiv preprint arXiv:2403.15059*, 2024.

[Wei *et al.*, 2025] Jingxuan Wei, Cheng Tan, Qi Chen, Gaowei Wu, Siyuan Li, Zhangyang Gao, Linzhuang Sun, Bihui Yu, and Ruifeng Guo. From words to structured visuals: A benchmark and framework for text-to-diagram generation and editing. *CVPR*, 2025.

[Xu *et al.*, 2022] Frank F Xu, Bogdan Vasilescu, and Graham Neubig. In-ide code generation from natural language: Promise and challenges. *TOSEM*, 31(2):1–47, 2022.

[Yang *et al.*, 2023] 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.

[Zhang *et al.*, 2023a] Pan Zhang, Xiaoyi Dong, Bin Wang, Yuhang Cao, Chao Xu, Linke Ouyang, Zhiyuan Zhao, Haodong Duan, Songyang Zhang, Shuangrui Ding, et al. Internlm-xcomposer: A vision-language large model for advanced text-image comprehension and composition. *arXiv preprint arXiv:2309.15112*, 2023.

[Zhang *et al.*, 2023b] Tianjun Zhang, Yi Zhang, Vibhav Vineet, Neel Joshi, and Xin Wang. Controllable text-to-image generation with gpt-4. *arXiv preprint arXiv:2305.18583*, 2023.

[Zhao and Lai, 2022] Puning Zhao and Lifeng Lai. Analysis of knn density estimation. *TIT*, 68(12):7971–7995, 2022.

[Zhao *et al.*, 2024] Puning Zhao, Fei Yu, and Zhiguo Wan. A huber loss minimization approach to byzantine robust federated learning. In *AAAI*, 2024.