SKETCH2DIAGRAM: GENERATING VECTOR DIAGRAMS FROM HAND-DRAWN SKETCHES

Anonymous authors
Paper under double-blind review

ABSTRACT

We address the challenge of automatically generating high-quality vector diagrams from hand-drawn sketches. Vector diagrams are essential for communicating complex ideas across various fields, offering flexibility and scalability. While recent research has progressed in generating diagrams from text descriptions, converting hand-drawn sketches into vector diagrams remains largely unexplored, primarily due to the lack of suitable datasets. To address this, we introduce SKETI*k*Z, a dataset containing 3,231 pairs of hand-drawn sketches, reference diagrams, and corresponding Ti*k*Z codes. Our evaluations highlight current limitations of state-of-the-art vision and language models (VLMs), establishing SKETI*k*Z as a key benchmark for future research in sketch-to-diagram conversion. Along with SKETI*k*Z, we present IMGTI*k*Z, an image-to-Ti*k*Z model that integrates a 6.7B parameter code-specialized open-source large language model (LLM) with a pre-trained vision encoder. Despite its modest size, IMGTI*k*Z demonstrates performance comparable to more extensive models such as GPT-4o. The model's success is largely driven by using our two data augmentation techniques and a multi-candidate inference strategy. These findings provide promising avenues for future research in sketch-to-diagram conversion and may have broader implications for image-to-code generation tasks. SKETI*k*Z is publicly available.[1]

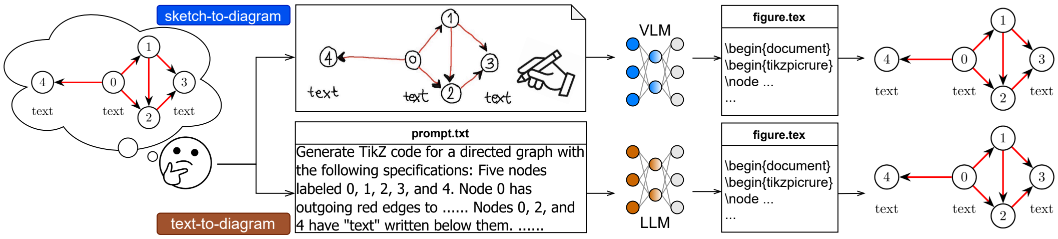
1 INTRODUCTION

Diagrams are powerful visual tools widely used in academia and various professional fields to convey complex ideas. They are essential for clear communication and knowledge sharing by effectively simplifying complex information. Vector graphics, in particular, are commonly used to create these high-quality diagrams due to their scalability and precision. Their scalability and flexibility make them particularly suitable for professional and academic contexts. These properties allow seamless resizing and modification without losing quality, enabling efficient adaptation to various presentation formats and requirements. While established tools and languages such as Ti*k*Z and Graphviz are popular for creating high-quality vector graphics, they often require significant manual effort and specialized expertise. Recent advances in large language models (LLMs), such as GPT-4o, have triggered a growing interest in automating the generation of vector graphic diagrams from textual descriptions (Belouadi et al., 2023; Zala et al., 2023; Zou et al., 2024). This emerging research area has significant potential to streamline the diagram creation process and make high-quality visualizations more accessible. Despite the significant advancements in text-to-code generation, generating diagrams from sketches remains largely unexplored. Sketch-based input often provides a more intuitive and user-friendly way to express visual ideas (Figure 1). This approach leverages the inherent human ability to quickly and effectively communicate complex visual information through simple drawings. A primary reason for the limited research in this area is the lack of publicly available datasets that pair hand-drawn sketches with their corresponding codes. Such datasets are essential for training and evaluating models that translate sketch-based input into structured diagram code.

To address this gap, we introduce SKETI*k*Z, a new dataset designed for benchmarking *sketch-to-diagram generation*. SKETI*k*Z comprises 3,231 pairs of hand-drawn sketches and corresponding Ti*k*Z codes. The sketches were created using several tools commonly employed in real-world scenarios: paper, whiteboards, and tablets. This diverse collection provides a valuable resource for advancing research in automated diagram generation from sketches. SKETI*k*Z aims to facilitate the development of models capable of generating high-quality diagrams from hand-drawn inputs for real-world applications. We also developed IMGTI*k*Z, a Vision-Language Model (VLM) specifically

---

[1] The dataset link is provided in the reproducibility statement section.



Generate TikZ code for a directed graph with the following specifications: Five nodes labeled 0, 1, 2, 3, and 4. Node 0 has outgoing red edges to ...... Nodes 0, 2, and 4 have "text" written below them. ......

```
\begin{document}
\begin{tikzpicrure}
\node ...
...
```

Figure 1: Overview of *sketch-to-diagram*. We consider scenarios where users hand-drawn diagrams that they want to create. Sketch-to-diagram models (e.g., VLM) take these sketches I_s and pre-defined instructions X, and generate code Y_g for producing vector graphics. Y_g is subsequently rendered into generated image I_g. The process of *text-to-diagram* is also provided for comparison.

designed for this task. Our model combines three components: an open-source LLM specialized in code generation, a vision encoder, and an adapter. This combination aims to create a model capable of efficiently converting sketches into Ti*k*Z code. We confirmed the effectiveness of two strategies: expanding our dataset using two data augmentation methods and employing an inference strategy that generates multiple candidates and selects the best one. As a result, IMGTI*k*Z achieved performance comparable to GPT-4o in subjective evaluations despite having a relatively small model size of 6.7B parameters. However, both IMGTI*k*Z and the latest state-of-the-art models still struggle to accurately generate code that captures all elements and layouts of sketches, indicating the potential for further advances. We aim for our dataset and findings to drive future research and development in this field. Our contributions are summarized as follows:

- We introduce SKETI*k*Z: A new dataset containing 3,231 pairs of hand-drawn sketches, reference diagrams, and corresponding Ti*k*Z codes, addressing the lack of real-world data for sketch-to-vector diagram conversion and serving as a benchmark for future research.
- We develop IMGTI*k*Z: A image-to-Ti*k*Z model that combines a 6.7B parameter code-specialized LLM with a pre-trained vision encoder, achieving accuracy comparable to larger models despite its modest size.
- We empirically demonstrate the effectiveness of two types of data augmentation and a multi-candidate inference strategy.

2 RELATED WORK

Vision and language models Constructing VLMs that understand images and generate text has become increasingly feasible with advancements in LLMs. A particularly effective approach involves integrating vision encoders, such as CLIP (Radford et al., 2021), and LLMs using adapter modules. This method has demonstrated promising results (Liu et al., 2023; Dai et al., 2023; Ye et al., 2023; Zhu et al., 2023; Li et al., 2024; Wang et al., 2024), efficiently creating VLMs that leverage the extensive knowledge base of pre-trained models. In this study, on the same line as these approaches, we build a VLM to generate Ti*k*Z code from images.

Image to code generation While VLMs typically focus on generating natural language outputs, such as answers to questions or image descriptions, research that produces code for rendering images, such as HTML, LaTeX, or SVG, has proven to be a valuable application. For instance, models have been developed to generate LaTeX code from screenshots of mathematical formulas or handwritten images (Deng et al., 2016; Gervais et al., 2024), HTML code from web page screenshots (Soselia et al., 2023; Si et al., 2024; Laurençon et al., 2024; Gui et al., 2024), and SVG code from icon images (Rodriguez et al., 2023). While generating LaTeX and TikZ code are similar in terms of code output, our research tackles significantly more complex problems than previous formula-to-LaTeX conversion studies. It involves much longer output sequences (739 tokens on average compared to 65 tokens in prior work) and requires an understanding of two-dimensional layouts. We introduce three key advances to handle this increased complexity: code-specialized VLM, two data augmentation strategies, and multi-candidate generation.

Diagram understanding Understanding diagrams has been an important and long-standing research topic, including question answering (Kembhavi et al., 2016; Lu et al., 2023; Wang et al.),

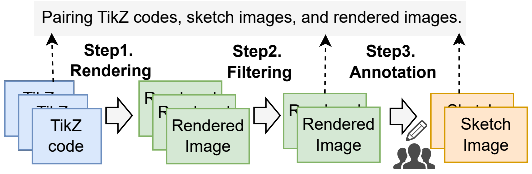


Figure 2: Dataset construction process.

Figure 3: Sketch Tool Usage Statistics.

Tool	Number	Proportion
Paper	2,545	78.8%
Whiteboard	346	10.7%
Tablet	340	10.5%
All	3,231	100%

caption generation (Hsu et al., 2021; Singh et al., 2023; Huang et al., 2023), and gene rating descriptions (Hu et al., 2023; Bhushan & Lee, 2022; Bhushan et al., 2024). Recent research proposed benchmark datasets to assess not only the understanding of diagram images, but also the direct comprehension of vector graphics code (Zou et al., 2024; Qiu et al., 2024). This expanding research area reflects the growing interest in understanding vector graphics diagrams.

Diagram generation Ellis et al. (2017) proposed a model generating Ti*k*Z code for primitive geometric sketches, limiting their scope to three basic shapes (circles, rectangles, and lines) without text. Our work differs significantly as we address real-world diagrams with unrestricted shapes and text elements. Furthermore, our dataset reflects realistic environments by including sketch images from various sources such as paper, whiteboards, and tablets. Regarding the model structure, while they used a two-stage approach for code identification and generation, we developed an end-to-end model to handle diverse, unrestricted shapes. Recently, there has been growing interest in generating real-world diagrams from text inputs (Belouadi et al., 2023; Zala et al., 2023). For example, Belouadi et al. (2023) proposed a method for generating Ti*k*Z code to render diagram images from caption text. Generating diagrams through code synthesis provides better controllability and editability than pixel-based image generation methods, while enabling LLM integration. Belouadi et al. (2023) also highlights the challenge of image-to-diagram generation, which remains limited due to the scarcity of paired image-code data. Concurrent work by (Belouadi et al., 2024) addresses the task of generating diagrams from images, which is closely related to our task. However, their evaluation of sketch-based generation is limited to a small dataset, which lacks corresponding Ti*k*Z code and thus cannot be used for image-to-code training. Our dataset provides the largest and most diverse sketch-to-diagram dataset with Ti*k*Z code, captured under real-world conditions. The dataset serves as a valuable benchmark for evaluating model robustness on real sketches since hand-drawn data cannot be collected in large quantities from the web. Beyond sketch-to-Ti*k*Z applications, it can enable the development of more general models through multitask learning with other image-to-code datasets. We also contribute novel data augmentation methods and multi-candidate generation strategies, providing new insights for future research directions in this field. Another line of research addresses the generation of CAD sketches (referring to parameters and constraints within CAD systems rather than hand-drawn images) (Para et al., 2021; Seff et al., 2021; Ritchie et al., 2023). These approaches are specifically designed for CAD generation, which differs from our focus.

3 DATASET AND TASK

3.1 TASK DEFINITION

We introduce a *sketch-to-diagram* task (Figure 1), where the input consists of a sketch image of a diagram I_s and a language instruction X, and the output is a sequence of Ti*k*Z code Y_g. Then generated Ti*k*Z code Y_g are compiled to render the diagram image I_g.

3.2 DATASET CONSTRUCTION

We constructed our dataset in three steps: rendering, filtering, and sketch annotation (Figure 2).

Step1: Rendering diagrams from TikZ code We first rendered diagrams from Ti*k*Z code in the DaTi*k*Z (Belouadi et al., 2023) by using pdflatex. We then paired the rendered reference diagrams I_r with the corresponding Ti*k*Z code Y_r. We refer to the rendered diagrams as the reference images.

Step2: Diagram classification and filtering Diagrams can be classified into various categories, as demonstrated by ACL-Fig (Karishma et al., 2023) with its 19-category dataset. For our sketch-to-diagram task, we focused on diagrams composed of geometric shapes and arrows, excluding those primarily based on numerical data. We specifically targeted diagrams categorized as Tree, Graph,

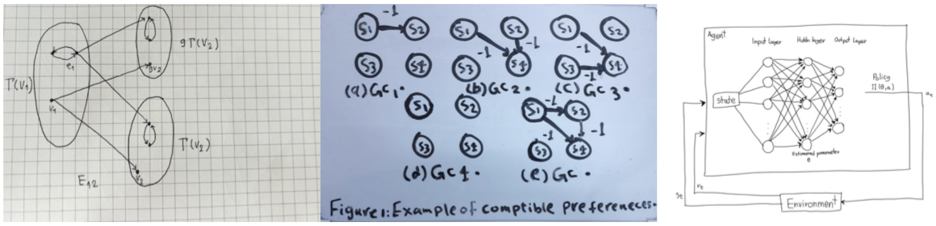


Figure 4: Examples of sketch images. Left: paper, Center: whiteboard, Right: tablet.

Table 1: Datasets used for training ImgTi*k*Z.

No	Name	Input	Output	Size	Stage1	Stage2
1	arXiv figure	Figure or table image	OCR text	1.2M	✓	
2	arXiv figure	Figure or table image	Caption text	1.1M	✓	
3	LLaVA-Pretrain[2]	Multi-domain image	Caption text	558K	✓	
4	SkeTi*k*Z	Diagram sketch image	Ti*k*Z source code	2.5K		✓
5	RenderTikZ	Diagram image	Ti*k*Z source code	155K		✓
6	AugTikZ	Diagram image	Ti*k*Z source code	556K		✓
7	ImgAugTikZ	Noised Diagram image	Ti*k*Z source code	714K		✓
8	DaTi*k*Z-v2[3]	Diagram image	Ti*k*Z source code	46K		✓

Architecture Diagram, Neural Networks, and Venn Diagram according to ACL-Fig labels. We chose these categories because sketch-to-diagram generation is particularly effective for visually oriented diagrams. These diagrams often involve complex combinations of shapes and interconnections, making manual creation time-consuming and precise linguistic instructions challenging. Using an image classification model trained on the ACL-fig dataset (details in Appendix E), we extracted and sampled 4,000 diagram images from our targeted categories for annotation. We present the detailed breakdown of categories in Table 9 and Figure 9 in Appendix E.

Step3: Sketch data collection 28 annotators created sketch images I_s based on filtered reference images I_r. Annotators used black pens primarily, with red, blue, and green for colored elements, excluding complex diagrams and ignoring color filling. They chose from paper, whiteboard, or tablet tools. Table 3 shows the distribution of sketches by tool, with the paper being the most common. Figure 4 illustrates examples from each tool. The dataset includes diverse sketches mimicking real-world scenarios, with paper and whiteboard sketches showing varied lighting and backgrounds. We aligned sketches I_s with corresponding Ti*k*Z codes Y_r and reference images I_r, creating a dataset of 2,585 training, 323 validation, and 323 test samples. More examples are shown in Appendix F

4 ImgTi*k*Z: Vision-Language Model for Image-to-Ti*k*Z Generation

4.1 Model Structure

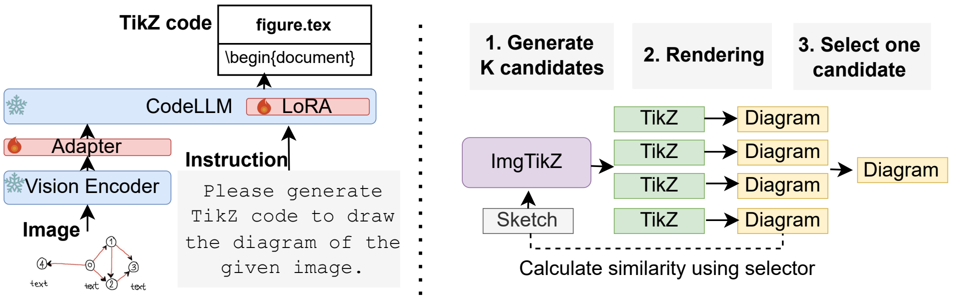
We developed ImgTi*k*Z, a VLM specifically designed for this task using the model architecture of LLaVA 1.5 (Liu et al., 2023). The model architecture comprises three key components: a code-specialized LLM, a vision encoder, and an adapter, illustrated in Figure 5 (a). The model inputs a diagram image and generates a corresponding TikZ code. While the original LLaVA1.5 employs a language model for natural language generation, we replaced this component with a code-specific LLM, a model specialized for code generation tasks. Specifically, we used the instruction-tuned DeepSeek coder (Guo et al., 2024) of 6.7B size as the code-specific LLM and SigLIP model Zhai et al. (2023) for our vision encoder. We employed the same architecture as LLaVA 1.5 for the adapter module - a simple two-layer MLP. We trained our model in two stages: first updating only the adapter parameters, then training both adapter and LoRA (Hu et al., 2021) parameters. The language and vision model parameters remained frozen throughout training. For more detailed information about the model hyperparameters, please refer to supplementary material B and Table 7.

4.2 Training Data

---

[2] `https://huggingface.co/datasets/liuhaotian/LLaVA-Pretrain`

[3] `https://huggingface.co/datasets/nllg/datikz-v2`



Please generate TikZ code to draw the diagram of the given image.

(a) ImgTikZ model structure

(b) Multi-candidate generation

Figure 5: ImgTi*k*Z model structure (a) and multi-candidate generation process for inference (b).

Datasets used in stage 1 training In stage 1 training, we incorporated arXiv figure data (No. 1 and 2 in Table 1) in addition to LLaVA-pretrain data (No. 3). This arXiv figure dataset was created by extracting figures, tables, and captions from arXiv paper PDFs in arXiv bulk dataset using PDFFigure2.0.[4] We also used Google Cloud Vision API[5] to extract text from these images. The arXiv data served two purposes: (1) generating OCR text from images to improve text recognition and (2) generating captions from diagram images to enhance diagram image understanding.

Datasets used in stage 2 training In the second stage of training, we focused on enhancing the model's ability to generate Ti*k*Z code. Given the limited size of the SkeTi*k*Z dataset alone, we supplemented our training data by creating paired rendered diagram images and Ti*k*Z code collected from arXiv source file in bulk data, which is referred to as RenderTi*k*Z (No. 5). We implemented two data augmentation techniques to increase diagram and image variations further. The first method involves generating Ti*k*Z code using GPT-3.5 to increase the variety of diagrams, referred to as AugTi*k*Z (No. 6). The second is an image augmentation technique designed to account for various types of noise commonly found in sketches, such as background images, lighting conditions, and rotation, which is referred to as ImgAugTi*k*Z (No. 7). In addition to these, we also utilized existing pairs of Ti*k*Z code and images (No. 8). The subsequent paragraphs will explain these two augmentation methods in more detail.

Data augmentation for increasing diagram variations While we collected approximately 916K original TikZ codes from arXiv sources, many failed to compile during RenderTi*k*Z creation. We used GPT-3.5 to fix these compilation errors with a prompt such as 'Please modify the code to make it compilable.' To increase diagram variety, we instructed GPT-3.5 to modify the original diagram into a different diagram, producing altered versions of the original diagrams. These augmentation techniques resulted in 556K AugTi*k*Z data samples. Previous data augmentation for VLMs used other VLMs to generate instruction-response pairs from images, which was costly due to image processing. Instead, we generate data efficiently by modifying only Ti*k*Z code using text-based LLMs. This approach could be applied to various image-to-code tasks. More details are in Appendix G.1.

Data augmentation for increasing image variations Hand-drawn sketch diagrams inherently contain more image noise compared to rendered images. This noise can appear as background interference or lighting variations when capturing sketches from paper or whiteboards. Furthermore, handwritten text and lines often exhibit significant distortions, and diagrams are frequently stored with angular rotations. To address these issues, we applied multiple image augmentation techniques to RenderTi*k*Z and AugTi*k*Z datasets, such as synthesizing notebook backgrounds, adding Gaussian noise, varying brightness and contrast, and introducing distortion. Figure 6 illustrates an example of the augmented

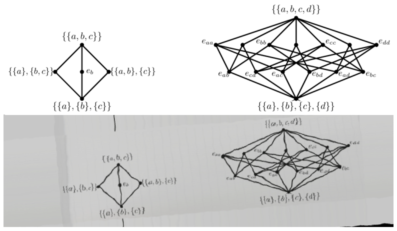


Figure 6: Example of ImgAugTi*k*Z data. Top: original image, bottom: augmented image.

---

[4] `https://github.com/allenai/pdffigures2`

[5] `https://cloud.google.com/vision/docs?hl=en`

image. This augmentation approach is general-purpose and can be applied to various sketch-to-diagram tasks. More details are in Appendix G.2.

4.3 INFERENCE

We implemented two inference methods: iterative generation and multi-candidate generation. In the paper, we refer to them as IMGTI*k*Z-IG and IMGTI*k*Z-MCG, respectively.

Iterative genetaion Iterative generation produces one candidate per test sample, regenerating upon compilation failure until success. We set a maximum number of generation attempts M to limit this process. This method is straightforward and can be considered a baseline approach.

Multi-candidate generation Multi-candidate generation creates K candidates simultaneously, selecting the best one (Figure. 5 (b)) using a selector model. In our study, we generate multiple Ti*k*Z codes and render them as images. The selector selects the best candidate by maximizing the similarity between the input sketch image I_s and the generated diagram image I_g. As general vision encoders cannot accurately measure diagram similarity, we propose D-SigLIP (Diagram-Specialized SigLIP) as our selector. D-SigLIP adds a trainable linear layer to a pre-trained SigLIP model, and we fine-tune only this layer through contrastive learning with noise-augmented diagram pairs from RenderTi*k*Z and AugTi*k*Z (Chen et al., 2020). More details are in Appendix C. To calculate the similarity score, we utilized the vector obtained from the CLS token of the D-SigLIP and computed the cosine similarity, as in CLIPScore (Radford et al., 2021). Our task requires generating lengthy code sequences (averaging 739 tokens), making producing error-free code in a single-generation attempt challenging. Furthermore, since the model training is based on next-token prediction loss for code sequences, metrics related to image quality are not explicitly considered during code generation. The multi-candidate generation and selection strategy allows us to evaluate these metrics after code generation, which could not be considered during the training phase. While similar approaches have been proposed for text inference and coding tasks (Brown et al., 2024), our work is the first to use image similarity for candidate selection in image-to-diagram conversion.

5 EVALUATION METRICS

5.1 AUTOMATIC EVALUATION

We used four aspects of automatic evaluation: compilation success rate, image similarity, code similarity, and character similarity.

Compilation success rate The compilation success rate (CSR) represents the percentage of generated Ti*k*Z codes that successfully compile into images. In this study, we employ two CSR metrics. The first is the averaged CSR, which calculates the ratio of successful compilations N_{success} to the total number of generation attempts N_{gen}, expressed as $CSR_{\text{avg}} = \frac{N_{\text{success}}}{N_{\text{gen}}}$. This metric indicates how often a model succeeds in compilation on average. The second is the cumulative CSR, which represents the number of test samples that compiled successfully $N_{\text{test\_success}}$ out of the total number of test samples N_{test}, expressed as $CSR_{\text{cum}} = \frac{N_{\text{test\_success}}}{N_{\text{test}}}$. This metric shows the proportion of test samples that were correctly compiled. Detailed examples are provided in Appendix J.

Image similarity We used cosine similarity between image embeddings to measure the similarity between the generated image I_g and the reference diagram image I_r. We used our D-SigLIP (see Sec. 4.3) for calculating image embeddings. This approach can be considered as a modification of widely-used CLIPScore, where the CLIP model is replaced with D-SigLIP. We also calculated CLIPScore using the original CLIP model; however, CLIPScore showed a lower correlation with human evaluations compared to the similarity calculated using D-SigLIP. If the compilation failed, we set the similarity score to 0.

Code similarity We used cosine similarity in the embedding space between Y_g and Y_r. We generated the code embeddings using OpenAI's text embedding model.[6]

Character similarity The character similarity calculates the similarity between the text in the generated image I_g and the text in the reference image I_r using Rouge-1 score (Lin, 2004). We used the OCR included in the Google Cloud Vision API to extract text. This metric indicates how well the model can read and generate text from the sketch.

---

[6] We used `text-embedding-3-small` version.

5.2 SUBJECTIVE EVALUATION

We conducted a subjective evaluation focusing on two key aspects: alignment and quality following established practices in previous studies(Otani et al., 2023; Ku et al., 2023). In our study, alignment measures the similarity between the generated and reference images, while quality assesses the coherence and appropriate arrangement of elements within the generated diagram. We employed a five-point scale for both metrics to ensure a nuanced evaluation.

Alignment Annotators assessed alignment by visually comparing the generated diagram image I_g to the reference diagram image I_r. The sketch diagram image I_s was also provided for evaluation. A score of 1 indicated that the diagram's elements were completely misaligned, while a score of 5 signified that they were almost perfectly aligned. To illustrate a score of 1, a randomly selected rendered diagram image from the training dataset was displayed.

Quality Annotators assessed the quality of the generated diagram images independently of the reference images, focusing on the structural clarity and arrangement of elements within the layout. A score of 1 was assigned to diagrams with poorly arranged, overlapping elements that were nearly unreadable. Conversely, a score of 5 was given to well-structured diagrams with logically arranged shapes and text that closely resembled human-created diagrams. The scale reflects the overall layout quality, ranging from incomprehensible to highly coherent visual representations.

Annotation We comprehensively evaluated each model's outputs across the entire test set using Amazon Mechanical Turk. The manual assessment involved 40 annotators, each evaluated by five distinct annotators, ensuring coverage of all six models. Diagrams that failed to compile were automatically assigned the minimum score of 1 for alignment and quality metrics. We computed the final score for each system and instance by averaging the three median evaluation scores, excluding potential outliers. A detailed description is provided in Appendix H.

6 EXPERIMENTAL SETUP

Models for Comparison We evaluated several state-of-the-art models in our study.[7] GPT-4o, OpenAI's most efficient multimodal model. We also included GPT-4o mini, their top small model. From Anthropic, we employed Claude 3.5 Sonnet, the latest in their multimodal large language model series. Lastly, we assessed LLaVA-Next, a popular open-source model.

Training parameters for IMGTI*k*Z We set the LoRA tuning parameters for training to r=128 and α=256. The stage 1 training was conducted with a batch size of 256 for 6,000 steps. Stage 2 training used a batch size of 128 for 1 epoch. We used 8 A100 GPUs for training IMGTI*k*Z, and 1 H100 GPU for inference. More details are in Appendix B.

Inference We applied iterative generation as the baseline for the four comparison models (see Sec.6), while for IMGTI*k*Z, we implemented both iterative and multi-candidate generation. The maximum number of attempts M for iterative sampling was set to 5, and the number of candidates K for multi-candidate generation was set to 20. More details are in Appendix A.

7 RESULTS

7.1 MAIN RESULTS

Can models generate compilable Ti*k*Z code for diagrams? Table 2 presents the averaged CSR results (CSR_avg). IMGTI*k*Z significantly outperformed other models in averaged CSR. Other models showed relatively low CSR_avg values (approximately 0.35-0.54), indicating insufficient adaptation to TikZ data. Since averaged CSR directly impacts user convenience, achieving higher scores is crucial. Figure 7 illustrates the progression of cumulative CSR across iterative generation attempts. IMGTI*k*Z achieved nearly 100% success after five attempts for the test data, while other methods leveled off at 0.8-0.9. This result indicates that 10-20% of samples remain uncompilable even after five attempts with these models.

---

[7] We used the `gpt-4o-2024-05-13` version for GPT-4o, the `gpt-4o-mini-2024-07-18` version for GPT-4o mini, the `claude-3-5-sonnet-20240620` version for Claude 3.5, and the `llama3-llava-next-8b` version, which is trained on the 8B Llama 3 model, for LLaVA-Next.

Table 2: The results of the automatic (0-1) and subjective (1-5) evaluations. The best results are highlighted in bold.

	Automatic				Subjective	
Model	**ImageSim**	**CodeSim**	**CharSim**	**CSR_avg**	**Alignment**	**Quality**
Closed models						
GPT-4o	0.695	0.821	0.611	0.479	3.00	3.20
GPT-4o-mini	0.595	0.814	0.514	0.376	2.39	2.71
Claude 3.5 Sonnet	0.753	0.813	**0.671**	0.544	**3.32**	**3.54**
Open source models						
LLaVA-Next	0.315	0.727	0.206	0.350	1.43	1.93
ImgTi*k*Z-IG (*ours*)	0.734	0.815	0.503	0.767	2.78	2.92
ImgTi*k*Z-MCG (*ours*)	**0.821**	**0.822**	0.594	**0.799**	3.13	3.30

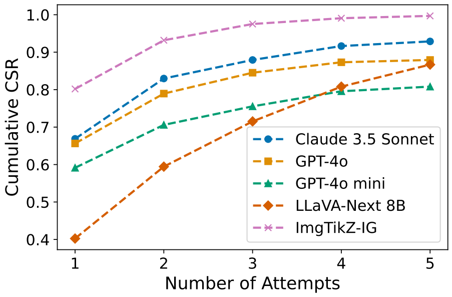


Figure 7: Progression of cumulative compilation success rate with varying number of attempts in an iterative generation.

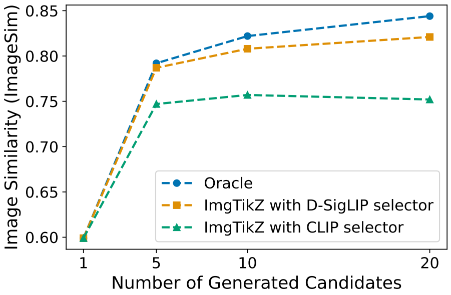


Figure 8: Progression of image similarity with varying number of candidates in multi-candidate generation

Can models generate diagram images close to the reference images? ImageSim and Alignment in Table 2 present the similarity between generated and reference images. Claude 3.5 achieved the highest performance in Alignment, with ImgTi*k*Z-MCG following as the second-best. Conversely, for ImageSim, ImgTi*k*Z-MCG performed best, while Claude 3.5 attained the second-highest score. LLaVA-Next, with a comparable model size to ImgTi*k*Z but without Ti*k*Z data training, performed poorly and rarely generated correct output. ImgTi*k*Z-MCG achieved comparable performance to GPT-4o on the Alignment score despite being smaller, highlighting the effectiveness of our adaptation and multi-candidate generation inference process. Overall, even the best-performing Claude 3.5 model achieves an average Alignment score of only about 3.3. This indicates that the generated diagrams, on average, only match about 50-60% of the correct diagrams based on the subjective assessment criteria. This suggests that the task remains challenging even for state-of-the-art models.

Can models generate Ti*k*Z code close to the reference code? Table 2 indicates that ImgTi*k*Z-MCG achieved the highest similarity scores for code similarity. However, code similarity scores are generally high with minimal inter-model differences. This indicates that high code similarity does not necessarily guarantee quality image generation. This discrepancy highlights a critical insight for model training: generating code that closely resembles the ground truth is insufficient. Similar to conventional VLMs, ImgTi*k*Z training relies on loss based on the next-word prediction of code. However, our findings suggest the need to incorporate image similarity metrics in training or inference phrases. This result aligns with the significant performance improvements of ImgTi*k*Z-MCG.

Can models accurately render text in sketch images? The CharSim in Table 2 provides insight into each model's ability to recognize characters in sketch images and render them accurately in Ti*k*Z diagram. Claude 3.5 achieved the highest CharSim score, followed by GPT-4o. While ImgTi*k*Z achieved comparable performance to GPT-4o in Alignment, it significantly underperforms in CharSim. This suggests that ImgTi*k*Z has enhanced diagram shape recognition but struggles with detailed character recognition. This limitation may reflect the resolution constraints of the SigLIP vision encoder. However, the substantial improvement in CharSim with multi-candidate generation indicates the need to strengthen character recognition during training.

Table 3: Evaluation of the effectiveness of SKETI*k*Z as training data.

Model	ImageSim	CharSim	CSR_avg
IMGTI*k*Z-IG	0.734	0.502	0.767
w/ SKETI*k*Z only	0.513	0.358	0.533
LLaVA-Next 8B	0.315	0.205	0.350

Table 4: Effectiveness of two data augmentation: (a) ImgAugTi*k*Z and (b) AugTi*k*Z.

Model	ImageSim	CharSim	CSR_avg
IMGTI*k*Z-IG	0.734	0.502	0.767
w/o (a)	0.668	0.457	0.635
w/o (a) and (b)	0.601	0.439	0.541

Can models generate high-quality diagrams? Table 2 presents quality scores from subjective evaluations. Claude 3.5 achieved the highest average score of 3.54 out of 5, followed by IMGTI*k*Z-MCG. Even the best-performing Claude model produces approximately 38% of samples with quality scores below 3 (indicating significant overlap of shapes and text), demonstrating that current VLMs still struggle with correct diagram layout rendering. This limitation in spatial reasoning is a common challenge among current VLMs. Our task and dataset can be considered one of the benchmark datasets for evaluating VLMs' spatial reasoning capabilities.

How does the number of candidates in multi-candidate generation affect performance? Figure 8 illustrates the image similarity trends for ImgTi*k*Z-MCG as the number of candidates K in multi-candidate generation varies. The oracle represents the highest achievable performance by selecting the best candidate based on image similarity to the reference diagram I_r. Results show significant performance improvement when increasing candidates from 1 to 5. Both oracle and IMGTI*k*Z demonstrate enhanced image similarity with more candidates. However, when replacing the selection model from D-SigLIP to CLIP, performance doesn't increase beyond 5 candidates. This indicates the importance of selection model quality in multi-candidate generation.

Do subjective evaluations correlate with automated evaluations? We analyzed correlations between subjective alignment ratings and automatic evaluation metrics. Pearson's correlation coefficients were calculated between human-rated alignment and image similarity (0.759), code similarity (0.365), and character similarity (0.592). Image similarity metric showed a high correlation with subjective evaluation, while code similarity demonstrated a low correlation. Character similarity exhibited moderate correlation, highlighting the importance of textual information in diagram evaluation. Image similarity metrics often fail to capture this local textual similarity.

Are the subjective evaluations consistent? To assess inter-annotator agreement in subjective evaluations, we employed Krippendorff's α (Krippendorff, 1980), a measure commonly used in related research (Otani et al., 2023; Ku et al., 2023). The analysis showed Krippendorff's α of 0.761 for alignment and 0.662 for quality, indicating substantial to moderate agreement among annotators in their subjective assessments.

7.2 DETAILED ANALYSIS

How effective is SKETI*k*Z alone as training data? We evaluated the effectiveness of our SKETI*k*Z dataset, comprising only 2.5k hand-drawn sketch samples, as training data. We evaluated the performance of a model trained solely on SKETI*k*Z in step 2. Results are presented in Table 3. While the SKETI*k*Z-only model underperforms compared to the full-data model, it significantly outperforms LLaVA-Next, indicating meaningful adaptation even with this limited dataset.

Is data augmentation effective? To assess the impact of our two data augmentation methods, we trained models excluding ImgAugTi*k*Z and both ImgAugTi*k*Z and AugTi*k*Z. Results are presented in Table 4. The observed significant decrease in image similarity, character similarity, and CSR_avg when excluding these datasets confirms the effectiveness of both augmentation methods.

To what extent does image augmentation improve sketch recognition? While the ablation study in Table 4 confirmed improved performance through image augmentation, we further investigated its impact on sketch recognition. Specifically, we compared the performance gap between using rendered reference images I_r and sketch images I_s as input. The closer the performance of sketch input approaches that of rendered image input, the more robust the model's understanding of sketch noise can be considered. Results are shown in Table 5. Without ImgAugTi*k*Z, image similarity decreased by approximately 12.5% and character similarity by 22.7%. In contrast, ImgTi*k*Z limited these reductions to 6.97% for image similarity and 17.0% for character similarity.

Table 5: Performance gap between rendered and sketch image inputs: comparison IMGTIkZ-IG and IMGTIkZ-IG without ImgAugTikZ data.

	Model	
Metric	**IMGTIkZ-IG**	**w/o ImgAugTikZ**
ImageSim		
Rendered Image	**0.789**	0.763
Sketch Image	**0.734**	0.668
Performance Gap	**-6.97%**	-12.5%
CharSim		
Rendered Image	**0.605**	0.591
Sketch Image	**0.503**	0.457
Performance Gap	**-16.9%**	-22.7%

Table 6: Performance gap between rendered and sketch image inputs across different sketching tools. Evaluation conducted using the IMGTIkZ-IG.

	Tool		
Metric	**Paper**	**Whiteboard**	**Tablet**
ImageSim			
Rendered Image	0.793	0.796	0.754
Sketch Image	0.735	0.716	0.740
Performance Gap	**-7.31%**	**-10.1%**	**-1.90%**
CharSim			
Rendered Image	0.587	0.627	0.581
Sketch Image	0.502	0.451	0.570
Performance Gap	**-14.5%**	**-28.1%**	**-1.89%**

However, ImgTikZ still doesn't match rendered image input performance, suggesting potential for further improvement through more noise-robust model construction.

Does image augmentation improve performance for non-sketch images? Comparing ImageSim and CharSim results for Rendered Images in Table 5 reveals that ImgTikZ outperforms the model without it. Image augmentation enhanced both ImageSim (0.763→0.789) and CharSim (0.591→0.605) scores, showing improved recognition even for clean, computer-rendered images.

Does image recognition difficulty vary across sketch tools? Table 6 presents the performance gap in image and character similarity when using rendered images versus sketches as inputs across different sketching tools. Results show that tablet sketches maintain image and character similarity close to rendered images. However, sketches from paper and whiteboard tools show significant performance degradation. These exhibit a 7-10% drop in image similarity and a 14-28% decline in character similarity. This performance drop suggests that paper and whiteboard sketches are more challenging for the model to process, likely due to their increased noise variety compared to tablet sketches. Whiteboard sketches showed the most significant decline in performance. While our image augmentation techniques have relatively minimized the gap with rendered image input, further performance improvements will require developing methods more robust to real-world noise.

8 CONCLUSION

We introduced SKETIkZ, a benchmark dataset with 3,231 pairs of hand-drawn sketches and their corresponding TikZ codes for generating diagrams. Our experiments demonstrate that current VLMs face considerable challenges in this task, highlighting the value of SKETIkZ as a benchmark for future research. We also developed IMGTIkZ, an image-to-TikZ model. Despite being smaller, this model performed as well as GPT-4o in subjective evaluations. This success came from using two data augmentation techniques and generating multiple candidates during inference. SKETIkZ is publicly available, and we expect these data resources and insights to drive the development of more advanced and efficient methods for automating vector graphics creation from hand-drawn sketches.

9 LIMITATION

Currently, SKETIkZ is restricted to generating diagrams using TikZ. However, the methodology could be extended to other formats such as SVG, HTML, Python, and JavaScript for diagram generation from code. Exploring these additional formats could enhance the dataset's generality and applicability. Transforming sketches into well-formed diagrams involves information completion, which can potentially lead to hallucination. An important direction for future work is developing an interactive system that allows users to modify generated diagrams through instructions. Furthermore, while our multi-candidate generation strategy considers code correctness and image quality metrics after code generation, incorporating these metrics directly into the training phase could potentially lead to better generation results, representing a promising direction for future work.

ETHICS STATEMENT

Were annotators for sketch creation told what the dataset would be used for, and did they consent? Yes. BAOBAB Inc. was fully responsible for managing the annotators. BAOBAB Inc. provides task descriptions, training, and agreements for each project with the annotators `https://baobab-trees.com/en/service`.

Data License SKETI*k*Z is derived from a publicly available subset of DaTi*k*Z (Belouadi et al., 2023), which permits copying and redistributing content under a Creative Commons Attribution License,[8] the GNU Free Documentation License,[9] or the MIT License.[10]

Potential ethical considerations we believe that there are minimal ethical considerations within the scope of this current research. However, as more accurate automatic diagram generation becomes feasible in the future, several considerations may arise. These potential issues include the misuse of highly accurate auto-generated diagrams to spread misinformation, the risk of AI models perpetuating or amplifying biases from their training data, and the possibility of advanced systems inadvertently reproducing copyrighted diagram designs, thereby raising intellectual property and copyright infringement issues; all of these challenges necessitate the establishment of appropriate guidelines to address them effectively.

REPRODUCIBILITY STATEMENT

Dataset Distribution All sketch image data in SKETI*k*Z is available at `https://storage.googleapis.com/sketikz/sketch_images.tar.gz`. The TikZ code and other information for train, val, and test can be downloaded from the following link `https://storage.googleapis.com/sketikz/sketikz_data.json`. The metadata of SKETI*k*Z is also available at this link `https://storage.googleapis.com/sketikz/sketikz_meta_data.json`. This metadata provides information about the fields in the data.

Details of models, hyperparameters, and manual evaluation Appendices B, C, and E provide detailed information about the models developed in this study. Appendix A describes the specifics of our inference process. Appendix H presents details regarding the subjective evaluation. Additionally, Appendices D and G presents details of the data creation process.

REFERENCES

Jonas Belouadi, Anne Lauscher, and Steffen Eger. AutomaTikZ: Text-guided synthesis of scientific vector graphics with TikZ. *arXiv [cs.CL]*, 2023. URL `http://arxiv.org/abs/2310.00367`.

Jonas Belouadi, Simone Paolo Ponzetto, and Steffen Eger. DeTikZify: Synthesizing graphics programs for scientific figures and sketches with TikZ. *arXiv [cs.CL]*, 2024. URL `http://arxiv.org/abs/2405.15306`.

Shreyanshu Bhushan and Minho Lee. Block diagram-to-text: Understanding block diagram images by generating natural language descriptors. In Yulan He, Heng Ji, Sujian Li, Yang Liu, and Chua-Hui Chang (eds.), *Findings of the Association for Computational Linguistics: AACL-IJCNLP 2022*, pp. 153–168. Association for Computational Linguistics, 2022. URL `https://aclanthology.org/2022.findings-aacl.15`.

Shreyanshu Bhushan, Eun-Soo Jung, and Minho Lee. Unveiling the power of integration: Block diagram summarization through local-global fusion. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics ACL 2024*, pp. 13837–13856. Association for Computational Linguistics, 2024. URL `https://aclanthology.org/2024.findings-acl.822`.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv [cs.LG]*, 2024. URL `http://arxiv.org/abs/2407.21787`.

---

[8] `https://creativecommons.org/licenses`

[9] `https://www.gnu.org/licenses/fdl-1.3.en.html`

[10] `https://opensource.org/license/mit`

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv [cs.LG]*, 2020. URL `http://arxiv.org/abs/2002.05709`.

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. InstructBLIP: Towards general-purpose vision-language models with instruction tuning. *arXiv [cs.CV]*, 2023. URL `http://arxiv.org/abs/2305.06500`.

Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M Rush. Image-to-markup generation with coarse-to-fine attention. *arXiv [cs.CV]*, 2016. URL `http://arxiv.org/abs/1609.04938`.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Joshua B Tenenbaum. Learning to infer graphics programs from hand-drawn images. *arXiv [cs.AI]*, 2017. URL `http://arxiv.org/abs/1707.09627`.

Philippe Gervais, Asya Fadeeva, and Andrii Maksai. MathWriting: A dataset for handwritten mathematical expression recognition. *arXiv [cs.CV]*, 2024. URL `http://arxiv.org/abs/2404.10690`.

Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Yi Su, Shaoling Dong, Xing Zhou, and Wenbin Jiang. VISION2UI: A real-world dataset with layout for code generation from UI designs. *arXiv [cs.CV]*, 2024. URL `http://arxiv.org/abs/2404.06369`.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, Y K Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. DeepSeek-coder: When the large language model meets programming – the rise of code intelligence. *arXiv [cs.SE]*, 2024. URL `http://arxiv.org/abs/2401.14196`.

Ting-Yao Hsu, C Lee Giles, and Ting-Hao Huang. SciCap: Generating captions for scientific figures. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-Tau Yih (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 3258–3264. Association for Computational Linguistics, 2021. URL `https://aclanthology.org/2021.findings-emnlp.277`.

Anwen Hu, Yaya Shi, Haiyang Xu, Jiabo Ye, Qinghao Ye, Ming Yan, Chenliang Li, Qi Qian, Ji Zhang, and Fei Huang. mPLUG-PaperOwl: Scientific diagram analysis with the multimodal large language model. *arXiv [cs.MM]*, 2023. URL `http://arxiv.org/abs/2311.18248`.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *arXiv [cs.CL]*, 2021. URL `http://arxiv.org/abs/2106.09685`.

Chieh-Yang Huang, Ting-Yao Hsu, Ryan Rossi, Ani Nenkova, Sungchul Kim, Gromit Yeuk-Yin Chan, Eunyee Koh, Clyde Lee Giles, and Ting-Hao 'kenneth Huang. Summaries as captions: Generating figure captions for scientific documents with automated text summarization. *arXiv [cs.CL]*, 2023. URL `http://arxiv.org/abs/2302.12324`.

Zeba Karishma, Shaurya Rohatgi, Kavya Shrinivas Puranik, Jian Wu, and C Lee Giles. ACL-fig: A dataset for scientific figure classification. *arXiv [cs.AI]*, 2023. URL `http://arxiv.org/abs/2301.12293`.

Aniruddha Kembhavi, Mike Salvato, Eric Kolve, Minjoon Seo, Hannaneh Hajishirzi, and Ali Farhadi. A diagram is worth a dozen images. In *Computer Vision – ECCV 2016*, pp. 235–251. Springer International Publishing, 2016. URL `http://link.springer.com/10.1007/978-3-319-46493-0_15`.

Klaus Krippendorff. *Content analysis: An introduction to its methodology*. SAGE Publications, 1980. URL `https://books.google.at/books?id=CyW7WBRzOqIC`.

Max Ku, Tianle Li, Kai Zhang, Yujie Lu, Xingyu Fu, Wenwen Zhuang, and Wenhu Chen. ImagenHub: Standardizing the evaluation of conditional image generation models. *arXiv [cs.CV]*, 2023. URL `http://arxiv.org/abs/2310.01596`.

Hugo Laurençon, Léo Tronchon, and Victor Sanh. Unlocking the conversion of web screenshots into HTML code with the WebSight dataset. *arXiv [cs.HC]*, 2024. URL `http://arxiv.org/abs/2403.09029`.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Yanwei Li, Ziwei Liu, and Chunyuan Li. LLaVA-OneVision: Easy visual task transfer. *arXiv [cs.CV]*, 2024. URL `http://arxiv.org/abs/2408.03326`.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81. Association for Computational Linguistics, 2004. URL `https://aclanthology.org/W04-1013`.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. *arXiv [cs.CV]*, 2023. URL `http://arxiv.org/abs/2310.03744`.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. MathVista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv [cs.CV]*, 2023. URL `http://arxiv.org/abs/2310.02255`.

Mayu Otani, Riku Togashi, Yu Sawai, Ryosuke Ishigami, Yuta Nakashima, Esa Rahtu, J Heikkila, and Shin'ichi Satoh. Toward verifiable and reproducible human evaluation for text-to-image generation. *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 14277–14286, 2023. URL `http://openaccess.thecvf.com/content/CVPR2023/html/Otani_Toward_Verifiable_and_Reproducible_Human_Evaluation_for_Text-to-Image_Generation_CVPR_2023_paper.html`.

Wamiq Reyaz Para, Shariq Farooq Bhat, Paul Guerrero, Tom Kelly, Niloy Mitra, Leonidas Guibas, and Peter Wonka. SketchGen: Generating constrained CAD sketches. *arXiv [cs.LG]*, 2021. URL `http://arxiv.org/abs/2106.02711`.

Zeju Qiu, Weiyang Liu, Haiwen Feng, Zhen Liu, Tim Z Xiao, Katherine M Collins, Joshua B Tenenbaum, Adrian Weller, Michael J Black, and Bernhard Schölkopf. Can large language models understand symbolic graphics programs? *arXiv [cs.LG]*, 2024. URL `http://arxiv.org/abs/2408.08313`.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pp. 8748–8763. PMLR, 2021. URL `https://proceedings.mlr.press/v139/radford21a.html`.

Daniel Ritchie, Paul Guerrero, R Kenny Jones, Niloy J Mitra, Adriana Schulz, Karl D D Willis, and Jiajun Wu. Neurosymbolic models for computer graphics. *arXiv [cs.GR]*, 2023. URL `http://arxiv.org/abs/2304.10320`.

Juan A Rodriguez, Shubham Agarwal, Issam H Laradji, Pau Rodriguez, David Vazquez, Christopher Pal, and Marco Pedersoli. StarVector: Generating scalable vector graphics code from images. *arXiv [cs.CV]*, 2023. URL `http://arxiv.org/abs/2312.11556`.

Ari Seff, Wenda Zhou, Nick Richardson, and Ryan P Adams. Vitruvion: A generative model of parametric CAD sketches. *arXiv [cs.LG]*, 2021. URL `http://arxiv.org/abs/2109.14124`.

Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2Code: How far are we from automating front-end engineering? *arXiv [cs.CL]*, 2024. URL `http://arxiv.org/abs/2403.03163`.

Ashish Singh, Prateek Agarwal, Zixuan Huang, Arpita Singh, Tong Yu, Sungchul Kim, Victor Bursztyn, Nikos Vlassis, and Ryan A Rossi. FigCaps-HF: A figure-to-caption generative framework and benchmark with human feedback. *arXiv [cs.CL]*, 2023. URL `http://arxiv.org/abs/2307.10867`.

Davit Soselia, Khalid Saifullah, and Tianyi Zhou. Learning UI-to-code reverse generator using visual critic without rendering. *arXiv [cs.CV]*, 2023. URL `http://arxiv.org/abs/2305.14637`.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. Qwen2-VL: Enhancing vision-language model's perception of the world at any resolution. *arXiv [cs.CV]*, 2024. URL `http://arxiv.org/abs/2409.12191`.

Shaowei Wang, Lingling Zhang, Longji Zhu, Tao Qin, Kim-Hui Yap, Xinyu Zhang, and Jun Liu. CoG-DQA: Chain-of-guiding learning with large language models for diagram question answering. `https://openaccess.thecvf.com/content/CVPR2024/papers/Wang_CoG-DQA_Chain-of-Guiding_Learning_with_Large_Language_Models_for_Diagram_Question_CVPR_2024_paper.pdf`.

Qinghao Ye, Haiyang Xu, Guohai Xu, Jiabo Ye, Ming Yan, Yiyang Zhou, Junyang Wang, Anwen Hu, Pengcheng Shi, Yaya Shi, Chenliang Li, Yuanhong Xu, Hehong Chen, Junfeng Tian, Qian Qi, Ji Zhang, and Fei Huang. mPLUG-owl: Modularization empowers large language models with multimodality. *arXiv [cs.CL]*, 2023. URL `http://arxiv.org/abs/2304.14178`.

Abhay Zala, Han Lin, Jaemin Cho, and Mohit Bansal. DiagrammerGPT: Generating open-domain, open-platform diagrams via LLM planning. *arXiv [cs.CV]*, 2023. URL `http://arxiv.org/abs/2310.12128`.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. *arXiv [cs.CV]*, 2023. URL `http://arxiv.org/abs/2303.15343`.

Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. MiniGPT-4: Enhancing vision-language understanding with advanced large language models. *arXiv [cs.CV]*, 2023. URL `http://arxiv.org/abs/2304.10592`.

Bocheng Zou, Mu Cai, Jianrui Zhang, and Yong Jae Lee. VGBench: Evaluating large language models on vector graphics understanding and generation. *arXiv [cs.CV]*, 2024. URL `http://arxiv.org/abs/2407.10972`.

A DETAILS OF INFERENCE

Inference Procedure We used pdflatex from TeX Live 2023[11] to compile generated Ti*k*Z code into a diagram image. We first cropped the rendered image using pdfcrop and then converted it to a PNG file to calculate image similarity.

Hyperparameters for closed models We used the API's default parameters for the closed models GPT-4o, GPT-4o mini, and Claude. The max_token parameter was set to 2,048 for all models.

Hyperparameters for LLaVA1.6 and IMGTI*k*Z We set the maximum number of newly generated tokens to 2,048 and generated the code through sampling. The sampling temperature was set to 0.6, a value determined through evaluation using the validation set.

B HYPERPARAMETERS FOR TRAINING IMGTI*k*Z

We conducted the training using the official code of LLaVA.[14] Table 7 details the hyperparameters used for stage 2 training of IMGTI*k*Z. For stage 2 training, we used a total batch size of 128. The stage 1 training employed similar hyperparameters, with a few exceptions: we set the batch size to 32

---

[11] `https://tug.org/texlive/`

[14] https://github.com/haotian-liu/LLaVA

Table 7: Configuration for the ImgTi*k*Z model training.

Option	Value
model_name (LLM)	`deepseek-ai/deepseek-coder-6.7b-instruct`[12]
model_name (Vision encoder)	`google/siglip-so400m-patch14-384`[13]
lora_r	128
lora_alpha	256
mm_projector_lr	2e-5
mm_projector_type	mlp2x_gelu
group_by_modality_length	True
bf16	True
num_train_epochs	1
batch_size	16
gradient_accumulation_steps	8
weight_decay	0
warmup_ratio	0.03
lr_scheduler_type	cosine
model_max_length	4096
gradient_checkpointing	True

with gradient accumulation over 4 steps, resulting in a total batch size of 128, and we increased the max_length to 2048. These parameters were derived from the original implementation of LLaVA1.5. The training process consisted of 6000 steps for stage 1 and a full epoch for stage 2. We conducted the training using 8 A100 GPUs. The total training time was approximately 24 hours for stage 1 and 60 hours for stage 2.

C D-SigLIP: An SigLIP Model Adapted for diagram

We trained D-SigLIP using a contrastive learning framework using the Hugging Face's code.[15] We used the `google/siglip-so400m-patch14-384` version of SigLIP as the vision encoder. During training, we applied augmentation twice to each image, aiming to maximize the similarity between augmented versions of the same image within the batch. Image augmentation was performed on-the-fly using imgaug.[16] The noise pipeline applied through imgaug is detailed below.

Listing 1: Image Augmentation Pipeline for D-SigLIP Training

```
pipeline = iaa.Sequential([
    iaa.Affine(scale={"x": (0.7, 1.0), "y": (0.7, 1.0)}, cval=255),
    iaa.Affine(rotate=(-5, 5), cval=255),
    iaa.Affine(translate_percent={"x": (-0.1, 0.1), "y": (-0.1, 0.1)},
        cval=255),
    iaa.Sometimes(0.2, iaa.ChangeColorTemperature((1100, 3000))),
    iaa.Sometimes(0.3, iaa.AdditiveGaussianNoise(scale=(10, 20))),
    iaa.Sometimes(0.3, iaa.MultiplyAndAddToBrightness(mul=(0.8, 1.2), add
        =(-5, 5))),
    iaa.Sometimes(0.3, iaa.GammaContrast((0.8, 1.2))),
    iaa.Sometimes(0.3,
        iaa.BlendAlphaSimplexNoise(
        iaa.Multiply((1.5, 2.5), per_channel=True),
        upscale_method=’cubic’,
        iterations=(1, 2)
    )),
    iaa.Sometimes(0.1, iaa.LinearContrast((0.8, 1.2))),
    iaa.ElasticTransformation(alpha=(15.0, 40.0), sigma=(5.0, 10.0)),
])
```

---

[15] `https://github.com/huggingface/transformers/tree/main/examples/pytorch/contrastive-image-text`

[16] `https://imgaug.readthedocs.io/en/latest/`

The training was conducted using four H100 80G GPUs. We set the batch size to 1024, the learning rate to 5e-5, and the warmup steps to 0, with training carried out for 200 steps.

D DATASET COLLECTION PROCESS

First, we compiled the TikZ code from DaTi*k*Z (Belouadi et al., 2023) to render the diagram images. Then, we developed a diagram classification model (See Section E) using the ACL-fig (Karishma et al., 2023) data, which was subsequently employed to classify the rendered diagrams from the DaTi*k*Z dataset.
We then extracted diagrams with the predicted labels Tree, Graph, Architecture diagram, Neural networks, and Venn diagram and sampled 4,000 instances from them.
BAOBAB Inc. coordinated multiple annotators to create the corresponding sketches for sampled instances. We excluded diagrams that were too complex to be sketched, diagrams of bar charts and line graphs that require numerical data, overly simplistic diagrams comprising only straight lines or dots, diagrams with illegible text, diagrams containing non-English text, and incomplete diagrams that were unnaturally truncated from the tasks during this process. The annotators selected one of the following tools to create the sketches: paper, whiteboard, or tablet. When using paper or whiteboard, they captured photos of the hand-drawn images with a smartphone camera. They used the drawing tool’s save function for tablets to save the images. All images were then converted to PNG format. As a result of these processes, we ultimately created 3,231 instances.

E DIAGRAM IMAGE CLASSIFICATION MODEL FOR DATA CONSTRUCTION

We developed a model to classify diagram images into categories by fine-tuning a pre-trained vision transformer on the ACL-fig dataset.[17] For the pre-trained VIT, we used Google’s `vit-large-patch16-224-in21k`.[18] The training was conducted using Hugging Face’s tools.[19] The parameters used for the training are listed in Table 8. We trained the model using a NVIDIA A100 GPU. The model achieved a classification accuracy of 0.886 on the evaluation dataset.

Table 8: Configuration for the image classification model.

Option	**Value**
model_name	`google/vit-large-patch16-224-in21k`
learning_rate	2e-5
num_train_epochs	5
batch_size	8
warmup_ratio	0
weight_decay	0

Table 9 presents the breakdown of estimated image labels within the sampled data. Furthermore, Figure 9 illustrates example diagrams for each estimated label category. While these are estimated labels and may potentially include diagrams that do not strictly conform to any specific category or contain estimation errors, we confirmed that there are diverse types of diagrams in our dataset.

F SKETCH IMAGE EXAMPLES

Figure 10 shows a subset of the collected sketch images.

G DETAILS OF THE DATA AUGMENTATION

G.1 AUGTI*K*Z: THE AUGMENTATION FOR INCREASING DIAGRAM VARIATION

From the arXiv source files,[20] we initially obtained 916,123 TikZ code samples. However, only 155K of these were successfully compiled. We utilized these compilable codes as RenderTi*k*Z.

---

[17] `https://huggingface.co/datasets/citeseerx/ACL-fig`

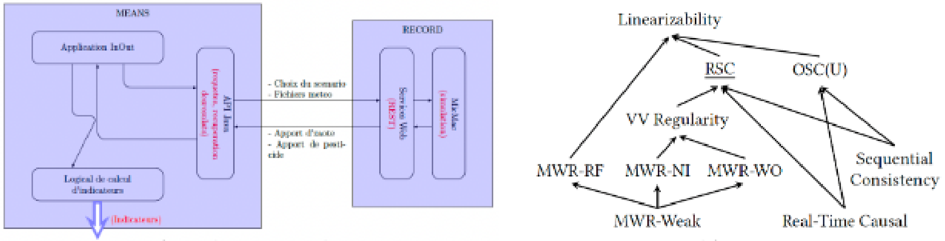
[18] `https://huggingface.co/google/vit-large-patch16-224-in21k`

[19] `https://github.com/huggingface/transformers/blob/main/examples/pytorch/image-classification/run_image_classification.p`

[20] `https://info.arxiv.org/help/bulk_data_s3.html`

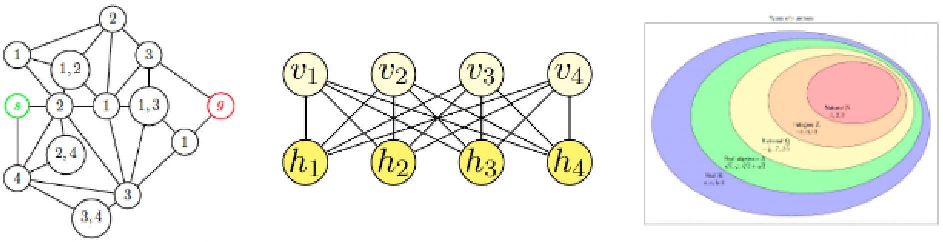
Table 9: Proportion of estimated image labels in the sampled data.

Category	**Number**	**Proportion**
Tree	1,799	45.0 %
Graph	1,046	26.2 %
Architecture diagram	646	16.2 %
Neural networks	459	11.5 %
Venn diagram	50	1.1 %
All	4,000	100%



a) Architecture diagram

b) Tree



c) Graph

d) Neural networks

e) Venn diagram

Figure 9: Examples of estimated image labels and their diagrams.

While the remaining codes failed to compile, we recognized their potential to significantly increase diagram variations if effectively utilized. To achieve this, we employed two types of augmentation prompts. The first prompt focused on code revision and was applied to the initially failed compilations. The second prompt, aimed at code modification, was applied to the entire dataset. The specific instructions provided were as follows. We used the `gpt-3.5-turbo-0125` version of GPT-3.5 to create the augmentation data.

Prompts for data augmentation

- Please modify the given LaTeX source code to make it compilable, including only the required preamble statements. If any external files are referenced, please modify the code to avoid referencing external files and include the content directly. The output should consist solely of the code itself, without any supplementary text.
- Please generate Ti*k*Z source code that modifies parts of the following code to create a different diagram. Ensure the code is compilable and includes only the required preamble statements. If any external files are referenced, please modify the code to avoid referencing external files and include the content directly. The output should consist solely of the code itself, without any supplementary text.

We included only the code that successfully compiled and rendered images correctly in our dataset AugTi*k*Z. Furthermore, we excluded images that were rendered at extreme scales (either too large or too small) from the training dataset.

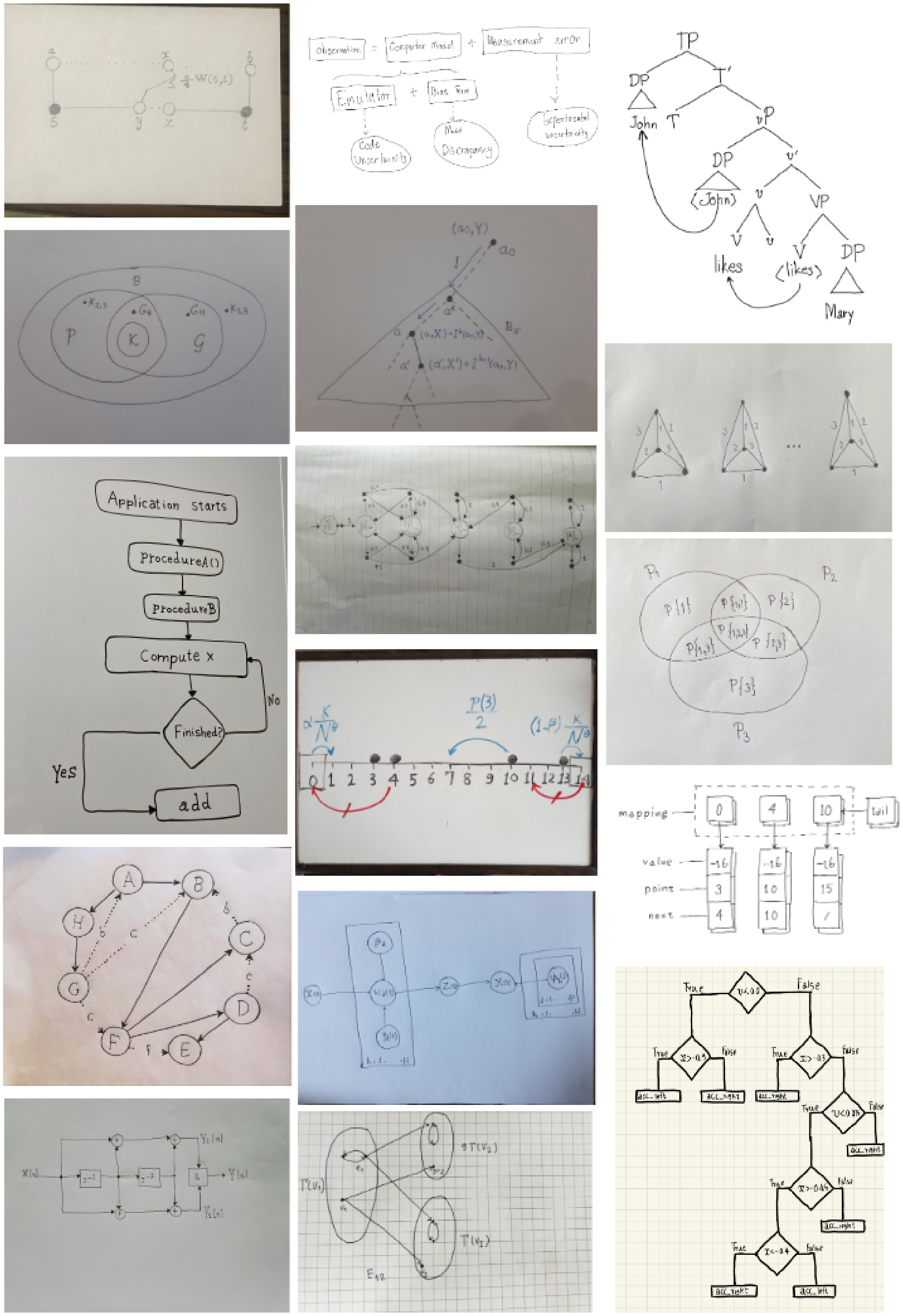


Figure 10: Examples of collected sketch images.

G.2 IMGAUGTI*k*Z: THE AUGMENTATION FOR INCREASING IMAGE VARIATION

To simulate the noise typically present in sketches, we applied several augmentation techniques to both RenderTi*k*Z and AugTi*k*Z. These included compositing with notebook background images, augmentation using imgaug, and white balance augmentation.[21] For notebook backgrounds, we created eight unique images independently of the sketch annotation process. The imgaug library was used to generate variations in rotation, distortion, Gaussian noise, brightness, and contrast. The specific augmentation pipeline created with imgaug is detailed below.

Listing 2: Image Augmentation Pipeline for Image Augmentation

```
pipeline = iaa.Sequential([
    iaa.Pad(percent=0.3, pad_mode="median"),
    iaa.Sometimes(0.3, iaa.AdditiveGaussianNoise(scale=(10, 20))),
    iaa.Sometimes(0.3, iaa.MultiplyAndAddToBrightness(mul=(0.8, 1.2), add
        =(-5, 5))),
    iaa.Sometimes(0.3, iaa.GammaContrast((0.8, 1.2))),
    iaa.Sometimes(0.3,
        iaa.BlendAlphaSimplexNoise(
        iaa.Multiply((1.5, 2.5), per_channel=True),
        upscale_method='cubic',
        iterations=(1, 2)
    )),
    iaa.Sometimes(0.1, iaa.LinearContrast((0.8, 1.2))),
    iaa.Affine(rotate=(-5, 5)),
    iaa.ElasticTransformation(alpha=(15.0, 30.0), sigma=(5.0, 10.0)),
    iaa.CropToFixedSize(width=int(width*0.8), height=int(height*0.8))
])
```

H SUBJECTIVE EVALUATION

For each test sample, annotators evaluated the alignment and quality of the six systems' outputs, GPT-4o, GPT-4o mini, Claude 3.5 Sonnet, LLaVA-Next, IMGTI*k*Z-IG, IMGTI*k*Z-MCG. We compensated annotators at a rate of $1.5 per test sample.
We provided annotators with the following instructions for conducting their evaluations:

Instructions

For each image A-F, please assign a score from 1 to 5 based on the following two aspects. You may also use 0.5 increments, such as 1.5 or 3.5.

- **Alignment:** The extent to which the generated diagram image matches the layout and content of the hand-drawn image.
- **Quality:** The overall completeness of the generated diagram image, regardless of the presence or absence of the hand-drawn and reference image.

The specific evaluation criteria for alignment that we instructed the annotators to follow are as follows:

---

[21] https://github.com/mahmoudnafifi/WB_color_augmenter

Evaluation Criteria for Alignment

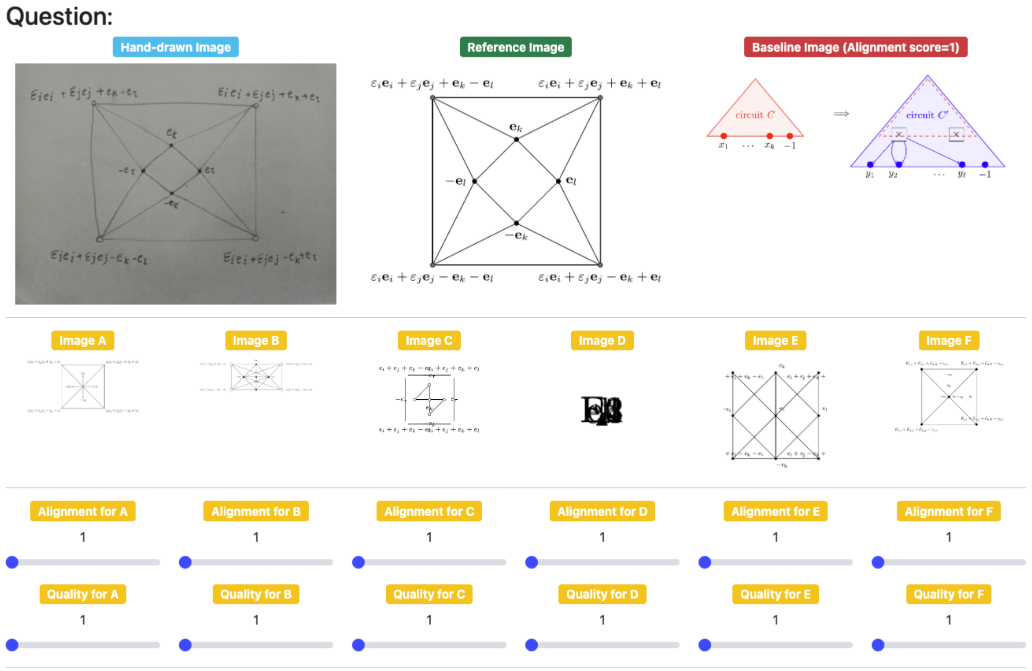
- **1:** The elements of the diagram in the generated image and the hand-drawn image do not match at all.
- **2:** The elements of the diagram in the generated image and the hand-drawn image match approximately 25%.
- **3:** The elements of the diagram in the generated image and the hand-drawn image match approximately 50%.
- **4:** The elements of the diagram in the generated image and the hand-drawn image match approximately 75%.
- **5:** The elements of the diagram in the generated image and the hand-drawn image match almost perfectly.

The specific evaluation criteria for quality that we instructed the annotators to follow are as follows:

Evaluation Criteria for Quality

- **1:** Almost complete overlap of text or shapes, making the diagram unreadable.
- **2:** Significant overlap of text or shapes, and the arrangement of elements is unnatural.
- **3:** Significant overlap of text or shapes, making some elements unreadable, or some elements are arranged unnaturally.
- **4:** Some overlap of text or shapes, but the arrangement of elements is neat.
- **5:** No overlap of text or shapes, and the arrangement of elements is as neat as a human-created diagram.

Figure 9 presents a partial screenshot of the annotation system interface. The complete template file for the annotation system, which includes all instructions, can be accessed this link `https://storage.googleapis.com/sketikz/template_202409_example.html`.


Question:
Hand-drawn Image
Reference Image
Baseline Image (Alignment score=1)
$\varepsilon_i\mathbf{e}_i+\varepsilon_j\mathbf{e}_j+\mathbf{e}_k-\mathbf{e}_l$
$\varepsilon_i\mathbf{e}_i+\varepsilon_j\mathbf{e}_j+\mathbf{e}_k+\mathbf{e}_l$
$\mathbf{e}_k$
$-\mathbf{e}_l$
$\mathbf{e}_l$
$-\mathbf{e}_k$
$\varepsilon_i\mathbf{e}_i+\varepsilon_j\mathbf{e}_j-\mathbf{e}_k-\mathbf{e}_l$
$\varepsilon_i\mathbf{e}_i+\varepsilon_j\mathbf{e}_j-\mathbf{e}_k+\mathbf{e}_l$
circuit C
circuit C'
Image A
Image B
Image C
Image D
Image E
Image F

Alignment for A 1
Alignment for B 1
Alignment for C 1
Alignment for D 1
Alignment for E 1
Alignment for F 1
Quality for A 1
Quality for B 1
Quality for C 1
Quality for D 1
Quality for E 1
Quality for F 1

Figure 11: Screenshot of the annotation interface: In the HTML, each image can be clicked to enlarge, allowing annotators to view the details of the diagrams.

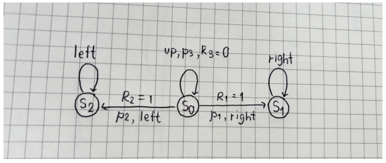
I GENERATED DIAGRAM EXAMPLES WITH EVALUATION SCORES

Tables 10 and 11 show some examples of generated diagrams. ImgTikZ-MCG generally selects better candidates compared to ImgTikZ-IG.

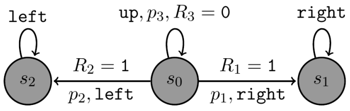
Table 10: Examples of generated diagrams and their metric scores.⊠ indicates a compile error and, therefore, has no score.

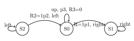
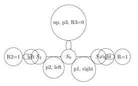
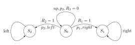
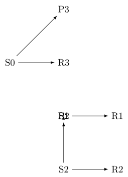
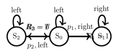
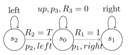
(a)

sketch diagram



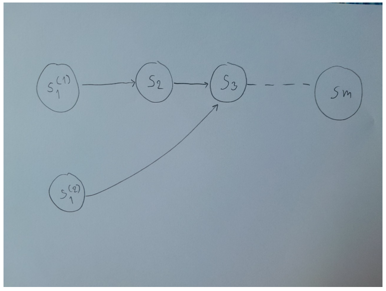
reference diagram



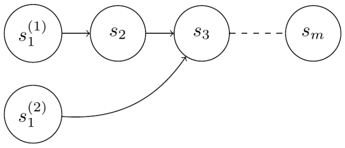
Models	GPT-4o	GPT-4o mini	Claude 3.5	LLaVA	ImgTikZ-IG	ImgTikZ-MCG
Diagram						
Alignment	3.67	2.83	3.83	1.67	4.00	4.67
Quality	3.67	2.67	4.17	2.17	3.67	5.00
ImageSim	0.86	0.69	0.82	0.38	0.81	0.91

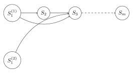
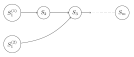
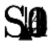
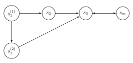
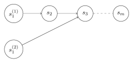
(b)

sketch diagram



reference diagram



Models	GPT-4o	GPT-4o mini	Claude 3.5	LLaVA	ImgTikZ-IG	ImgTikZ-MCG
Diagram		⊠				
Alignment	3.83	N/A	4.50	1.00	4.17	4.67
Quality	4.00	N/A	4.83	1.00	4.83	4.83
ImageSim	0.79	N/A	0.92	0.05	0.87	0.92

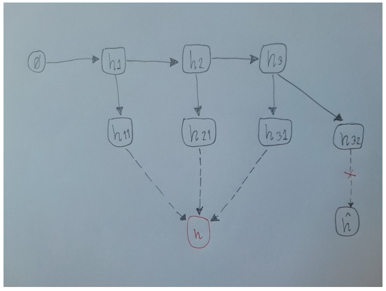
J DETAILED EXPLANATION OF COMPILATION SUCCESS RATE (CSR)

To better illustrate the difference between CSR_{avg} and CSR_{cum}, we provide examples below.
CSR_{avg} represents the success rate across all generation attempts. For example, if a model attempts N generation for each of the 100 test samples and succeeds in compilation K times, then

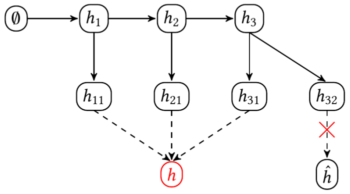
$$CSR_{\text{avg}} = \frac{N_{\text{success}}}{N_{\text{gen}}} = \frac{K}{(100 \times N)}. \tag{1}$$

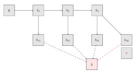
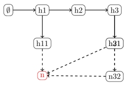
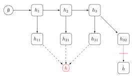
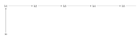
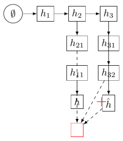
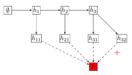
Table 11: Examples of generated diagrams and their metric scores.☒ indicates a compile error and, therefore, has no score.

(c) sketch diagram

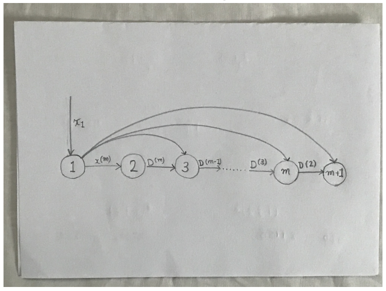


reference diagram

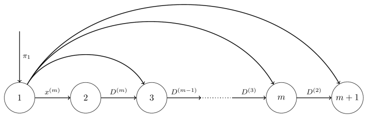


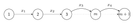
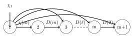
Models	GPT-4o	GPT-4o mini	Claude 3.5	LLaVA	ImgTikZ-IG	ImgTikZ-MCG
Diagram						
Alignment	3.83	3.17	4.67	2.33	3.00	3.83
Quality	4.17	3.50	4.83	3.83	3.33	4.17
ImageSim	0.70	0.76	0.88	0.27	0.77	0.81

(d) sketch diagram



reference diagram



Models	GPT-4o	GPT-4o mini	Claude 3.5	LLaVA	ImgTikZ-IG	ImgTikZ-MCG
Diagram	☒	☒	☒	☒		
Alignment	N/A	N/A	N/A	N/A	2.83	4.33
Quality	N/A	N/A	N/A	N/A	4.33	3.50
ImageSim	N/A	N/A	N/A	N/A	0.75	0.86

To illustrate, if we make 10 generation attempts for each of the 100 test samples (totaling 1,000 generations) and achieve successful compilation in 400 cases, then

$$CSR_{\text{avg}} = \frac{400}{1000} = 0.4. \tag{2}$$

CSR_{cum}, which is exclusively used for iterative generation, measures the cumulative proportion of test samples achieving successful compilation across multiple attempts. Consider the following sequential process for 100 test samples:

- first generation: 50 of the 100 samples compile successfully
- Second generation: 20 of the remaining 50 (100 - 50) samples compile successfully
- Third generation: 10 of the remaining 30 (50 - 20) samples compile successfully

In this scenario,

$$CSR_{\text{cum}} = \frac{N_{\text{test\_success}}}{N_{\text{test}}} = \frac{50 + 20 + 10}{100} = 0.8. \quad (3)$$

This metric specifically quantifies the proportion of test samples that eventually achieve successful compilation, independent of the total generation attempts.

The motivation for utilizing these two distinct evaluation metrics arises from their complementary analytical perspectives: CSR_{avg} represents the average compilation success rate, enabling fair model comparison. CSR_{cum} measures the proportion of successfully compiled test samples across multiple attempts, analogous to a recall metric.