TechING: Towards Real World Technical Image Understanding via VLMs

Tafazzul Nadeem[*†] **Bhavik Shangari**[*‡] **Manish Rai**[‡]
Gagan Raj Gupta[‡] **Ashutosh Modi**[†]
[†]IIT Kanpur [‡]IIT Bhilai
{tafazzul23, ashutoshm}@cse.iitk.ac.in,
{bhaviks, manishr, gagan}@iitbhilai.ac.in

Abstract

Professionals working in technical domain typically hand-draw (on whiteboard, paper, etc.) technical diagrams (e.g., flowcharts, block diagrams, etc.) during discussions; however, if they want to edit these later, it needs to be drawn from scratch. Modern day VLMs have made tremendous progress in image understanding but they struggle when it comes to understanding technical diagrams. One way to overcome this problem is to fine-tune on real world hand-drawn images, but it is not practically possible to generate large number of such images. In this paper, we introduce a large synthetically generated corpus (reflective of real world images) for training VLMs and subsequently evaluate VLMs on a smaller corpus of hand-drawn images (with the help of humans). We introduce several new self-supervision tasks for training and perform extensive experiments with various baseline models and fine-tune Llama 3.2 11B-instruct model on synthetic images on these tasks to obtain `LLama-VL-TUG`, which significantly improves the **ROUGE-L** performance of Llama 3.2 11B-instruct by **2.14x** and achieves the best all-round performance across all baseline models. On real-world images, human evaluation reveals that we **achieve minimum compilation errors** across all baselines in 7 out of 8 diagram types and improve the average **F1 score** of Llama 3.2 11B-instruct by **6.97x**.

1 Introduction

Technical discussions typically involve ideation and translating those ideas into hand-drawn diagrams (e.g., flowcharts, class, packet diagrams) on a whiteboard or paper. Such diagrams are often digitized in the form of images in jpg/png formats for documentation purposes, with contents of the image not readily machine-understandable, making it challenging to edit these later via natural language queries. Consequently, we need techniques that enable computers to understand digitized images of technical diagrams, making it feasible to edit these via natural language queries.

Recently, Vision Language Models (VLMs) have shown impressive performance on various benchmarks (Yue et al., 2024; Liu et al., 2024), however, one area that remains under-explored is an assessment of the abilities of VLMs with regard to technical diagram understanding and generation. Our initial experiments (§5) reveal that many of the pre-trained VLMs have limited understanding of technical diagrams. Many commercial tools are being developed (e.g. `www.eraser.io`, `www.whimsical.com`, `www.notegpt.io`, `www.diagrammingai.com`, `www.visily.ai`) with the aim of helping users generate technical diagrams via prompts. These tools have limited performance (based on our initial study) and focus mainly on system diagrams or workflow-related images. There is a need for models that can understand technical images, as these can be beneficial for various applications such as diagram editing, accelerating design iterations, enhancing collaboration and versioning, automation of document workflows, requirements-driven modeling, and diagram creation.

One possible way to address this gap is to develop domain-specific VLMs that can understand multiple types of technical diagrams. However, this comes with several challenges: **1)** a lacuna of corpus of technical diagrams that are reflective of real-world settings, particularly hand-drawn technical diagrams. **2)** Technical diagrams are domain-specific and image types vary across domains by a large margin; therefore, the model should be able to adapt to and generalize the domain. For example, *class diagrams* are used in software documentation to represent the system design, and *packet diagrams* are used for standardizing the packet structures of various protocols in network specifications such as TCP/IP; both types of diagram are

[*]Equal contribution

Findings of the Association for Computational Linguistics: EACL 2026, pages 2720–2749
March 24-29, 2026

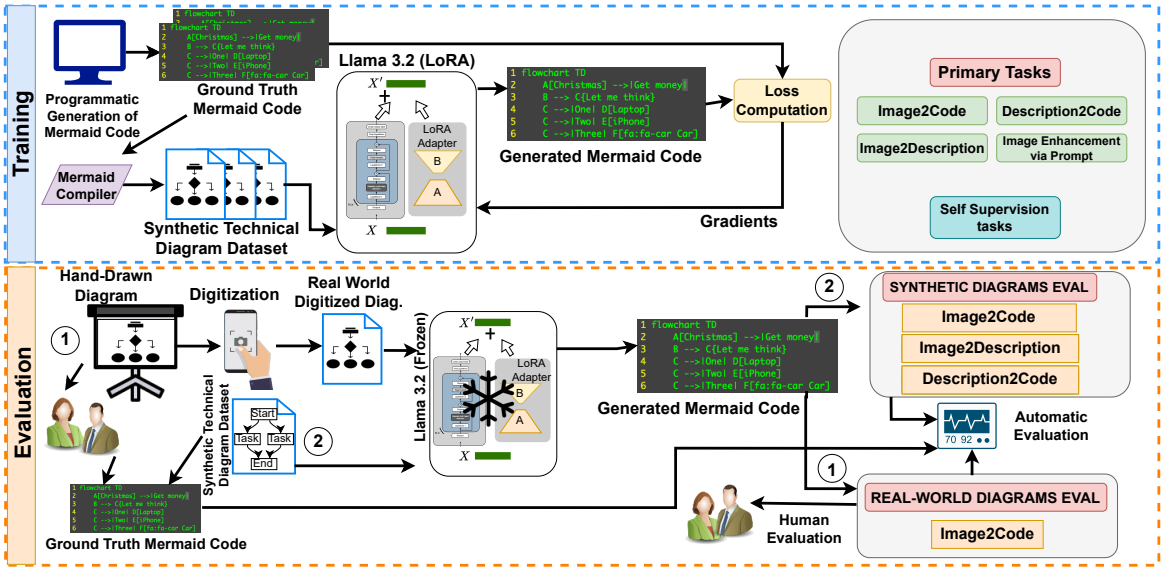

Figure 1: Figure illustrating the pipeline for technical diagram understanding

very different (see App. B). **3)** Technical diagrams pose representational challenges due to the tight coupling between geometric shapes (e.g., boxes, circles, arrows) and the text within or on these geometrical shapes (e.g., text on an arrow, text inside a box, etc.), requiring models to process text and geometrical shapes jointly. **4)** Hand-drawn technical diagrams can have large variations, same diagram can be made by different people in various ways. **5)** An effective way to build understanding in a VLM is by introducing an intermediate representation (in the form of computer code) that encodes the structure and semantics of drawings in a simplified, symbolic form. However, manually constructing a large corpus of hand-drawn images paired with their corresponding code representations is highly resource-intensive, *leading to a low-resource setting*. To address these challenges, we propose an alternate approach:

- We propose a large (115, 014 images) corpus (referred as **`TechING`**) of synthetically generated diverse technical diagrams, covering 8 different diagram types along with a manually created corpus of 545 hand-drawn technical diagrams of various types with the help of human participants. The corpus consists of triplets: synthetic technical diagram image, corresponding Mermaid[1] code, and corresponding image description. Collectively, the corpus spans a broad spectrum of commonly used diagram types, including flowcharts, block diagrams, state diagrams, graphs, C4 models, sequence diagrams, packet diagrams, and class diagrams. Fig. 1 shows the entire pipeline.
- To enable technical diagram understanding in models, we introduce various primary tasks (Image2code, Image2Description, Description2Code, and ImageEnhancement-via-Prompt) and self-supervision based tasks (CodeCompletion, Image-Code similarity).
- We experiment with a battery of pre-trained VLMs on these tasks via a zero-shot setup. Given the poor performance on zero-shot setting, we fine-tune Llama 3.2 11B-instruct (Meta, 2024) model on synthetic images on these tasks and obtain **`LLama-VL-TUG`**, which significantly improves the **ROUGE-L performance by 2.14x** and achieves **best all-round performance across all baseline models**. Further, we assess model generalization by manually evaluating several models on the Image2Code task with human participants. On real-world images, human evaluation reveals that we achieve the **minimum compilation errors** and average **F1 score** on correctly generating diagram structures by **6.97x**.
- We create Github (`https://github.com/Exploration-Lab/TechING`) for the project having the details of dataset (`https://huggingface.co/datasets/Exploration-Lab/TechING`) and models (`https://huggingface.co/Exploration-Lab/LLama-VL-TUG`) for the

[1] `https://mermaid.js.org/`

Corpus	Diagram types	Image Dataset Size (English)	Real World Images (Hand Drawn)	Real World Images (Scraped)	Structural Summary	Diagram Code
FC Database (Awal et al., 2011)	Flowchart	78	78	0	×	×
CBD (Bhushan and Lee, 2022)	Block	502	0	502	×	×
FlowchartQA (Tannert et al., 2023)	Flowchart	~1,000,000	0	166	×	×
BD-EnKo (Bhushan et al., 2024)	Flowchart, Graph, Journey, Sequence, State, C4	46,599	0	91	×	×
FlowVQA (Singh et al., 2024)	Flowchart	2,272	0	0	×	×
Ai et al. (2024)	Knowledge Graph, Route Map Flowchart, Mind Map, Gantt Chart	1,314	0	1,314	×	×
FlowLearn (Pan et al., 2024)	Flowchart	13,858	0	3,858	✓	✓
TechING (ours)	Block, C4, Class, Flowchart, Graph, Packet, Sequence, State	115, 014	545	0	✓	✓

Table 1: Comparison among different available corpora for technical diagram understanding

research community.

The core contribution of this work lies in the integration and large-scale construction of a corpus that enables effective vision–language learning for hand-drawn technical diagrams and subsequent evaluation of existing VLMs on tasks related to the corpus.

2 Related Works

Vision Language Models (VLMs) (e.g., PaliGemma-2 (Beyer et al., 2024), GPT-4o (OpenAI, 2024a), QwenVL (Bai et al., 2023), DeepSeek-VL (Lu et al., 2024), and BLIP-2 (Li et al., 2023)) have been developed in recent years to process and understand image and text modalities together. VLMs have shown remarkable performance on a variety of tasks such as visual question answering (dos Sants et al., 2023; Goyal et al., 2017; Hudson and Manning, 2019; Schwenk et al., 2022), image captioning (Chen et al., 2015; Young et al., 2014; Agrawal et al., 2019; Chen et al., 2024, 2015; Young et al., 2014; Agrawal et al., 2019), image-text retrieval (Zhang et al., 2024), text to image generation (Li et al., 2024a), text-guided image editing (Kembhavi et al., 2016), visual story telling (Zellers et al., 2019), and document understanding (Li et al., 2024b; Aggarwal et al., 2023). However, technical diagram understanding is under-explored, with few recent works such as diagram understanding and reasoning (Kembhavi et al., 2016), diagram summarization (Bhushan and Lee, 2022; Bhushan et al., 2024), and question answering (Masry et al. (2022); Methani et al. (2020); Kahou et al. (2018)). Accordingly, various corpora have been introduced. Table 1 summarizes various available corpora along with **TechING**. As can be seen, **TechING** covers more diagram types than existing works and includes a much larger set of hand-drawn real-world images. In contrast, most existing real-world diagram corpora are scraped from structured sources, such as documentation or websites, where the diagrams are already clean and well-formatted. This makes **TechING** more challenging and realistic, as hand-drawn diagrams often exhibit variations in layout, noise, handwriting, and structure, reflecting practical scenarios more closely. FlowLearn (Pan et al., 2024) solely focuses on flowchart diagrams and lacks coverage of other diagram types, limiting its generalization scope. The diagram types we cover go beyond graph structures (i.e., some of these types cannot be represented as a graph, e.g., Packet, Sequence, C4, and Class Diagrams). Due to space limitations, more details are provided in App. A.

3 Corpus Creation and Tasks

Due to the unavailability of large-scale real-world open-source hand-drawn images with their corresponding code, we programmatically generated a large number of synthetic images with different diagram types (Table 2 and see App. B for examples). These images are used to create the corpus **TechING**. In total, we constructed 115, 014 samples comprising of synthetic Image–Mermaid

Dataset	Difficulty	Block	C4	Class	Flowchart	Graph	Packet	Sequence	State	Total
D1	Easy	4402 (2.0)	4738 (1.5)	6000 (2.0)	1500 (4.5)	3695 (3.3)	1500 (3.0)	6000 (2.3)	5000 (3.4)	**32835**
	Medium	2000 (4.5)	3000 (4.5)	2000 (4.2)	3000 (6.0)	5001 (4.1)	2981 (5.0)	1500 (3.9)	2500 (10.6)	**21554**
	Hard	2500 (6.1)	2500 (4.5)	2500 (6.0)	3500 (5.9)	1729 (4.7)	3500 (8.2)	1786 (7.6)	2000 (8.3)	**20443**
D2	Combined	2731	0	7646	0	5038	6683	8786	8753	**39637**
D3	Combined	74	62	74	72	77	53	58	75	**545**

Table 2: **TechING** distribution across different tasks and diagram types for synthetic images and real-world hand-drawn images. The average number of primary components indicating the difficulty level is indicated in parentheses.

Code–Description triplets and the derived image enhancement corpus for eight different diagram types; each diagram type has three difficulty levels (easy, medium, and hard) based on the number of components and relationship between them. To test performance in the real-world setting, we manually created a real-world images corpus for evaluation, having 545 hand-drawn technical images (on white board and paper) along with corresponding Mermaid code.

3.1 TechING Creation

The corpus contains three parts as described below:

Image-Mermaid Code-Description Corpus (D1): We programmatically created synthetic images and their description via Mermaid programming language (also see App. C). As the first step, in order to populate content in various components of a diagram, for each diagram type, we constructed discipline-specific (e.g., engineering, business, health) lists of keywords using GPT-4o. The lists were thoroughly reviewed to prevent harmful/toxic keywords. For instance, a keyword such as "electrical" could be assigned as a block name in a block diagram. The synthetic corpus creation system works as follows: based on the difficulty level (for instance, Easy (2-3 Headers), Medium (3-8 Headers), or Hard (6-9 Headers) in Packet diagrams – Diagram of data packet used in communication where headers has information like IP Addresses, Flags etc.) and diagram type, the system first randomly selects a few components and edges that should be in the diagram, and subsequently selects a discipline and samples corresponding keywords. Similarly, according to the number of edges sampled, a pair of random blocks is connected with/without an edge label. Random sampling at each step was done using a uniform probability distribution. These were then populated into templatized Mermaid code, from which diagrams are rendered. We also manually created templates for generating descriptions corresponding to each diagram type. Mermaid code for each diagram in **D1** is parsed with regular-expression extractors to identify components, like blocks and edges, which were then inserted into the description templates to produce image–description pairs. The templates are simple python f-strings which are populated by the sampled keywords and edge information updated in a loop line by line. Owing to the diversity of diagram structures, for each diagram type, a dedicated Python script is developed to automatically generate the corpus (see App. D for corpus statistics). This pipeline mitigates the model's inherent bias toward specific semantic content while strengthening its ability to learn diagram structures, thereby enabling robust training without requiring domain expertise. Also, all automatically generated data is created using predefined templates, ensuring consistency and correctness by design. As a result, no additional data cleaning or selection is required after generation. Note that the difficulty level of diagrams (based on headers) was decided empirically based on initial experiments where we found that basic building blocks (very easy examples) of a diagram were required to make the model learn the structure of the diagrams. For instance, in the case of Block Diagram, two nodes connected with an edge can be defined as a basic building block. We have found out that if these diagrams are not present in the training set the model struggles to learn to produce correct Mermaid codes. Hence, these building blocks are assigned as Easy level and subsequently higher levels are defined relative to this level.

Why Mermaid? Mermaid programming language, used for representing technical images, was selected due to its human readability, being editable, wide coverage across diagram types, and widespread popularity. Additionally, Mermaid also provides its CLI tools (compiler) to generate images through the command line interface, hence it can be easily integrated with python scripts for automatic creation of large scale corpus. It is also possible to use multiple diagramming languages, such

as GraphViz[2], to increase corpus size and complexity; however, our primary focus is not on training VLMs to learn multiple programming languages, but rather on enabling the model to understand and reason over technical images. Moreover, introducing multiple languages could increase variability in syntax, which may actually degrade fine-tuning performance by adding unnecessary complexity.

Image-Enhancement Corpus (D2): Using **D1**, we created Image-Enhacement Corpus (**D2**) that consists of an incomplete image and a corresponding natural language prompt that directs a model to complete the image. For creating **D2**, we remove specific parts of technical images in **D1** by deleting triplets (two components connected by an edge in the Mermaid Code). The remaining code is then compiled again to get the incomplete image, which is coupled with an enhancement prompt. The prompt is generated automatically by verbalizing the removed triplet into textual description using Gemma3 (4B) model (Aishwarya Kamath, 2025) resulting in Image-Enhancement corpus with images and enhancement prompts. The full dataset construction pipelines are illustrated in App. C.

Real World Images Corpus (D3): To test a model's technical image understanding and generalization capabilities, we created a corpus of hand-drawn images (with the help of humans) reflective of real-world setting. We curated 545 images with corresponding Mermaid Code, covering all diagram types. To ensure the Mermaid code accurately reflects the original image, it was compiled using the Mermaid compiler, and the generated image was compared to the hand-drawn image. These diagrams were hand drawn by annotators on different mediums (whiteboard, paper, and electronic whiteboard) to provide diversity in the corpus, which were subsequently digitized using a mobile phone (see examples in App. B). The size of this corpus is relatively small compared to the synthetic corpus; this is primarily due to the significant difficulty in creating such a corpus. Each example requires not only manual drawing but also precise transcription into correct Mermaid code, which is highly time-consuming due to multiple iterations (compiling and comparing with the original image), prone to errors due to oversight, and hence difficult to scale. Even when using samples from existing hand-drawn corpora like books and research papers, generating the corresponding code remains a tedious and labor-intensive task. This corpus provides a realistic setting where images may be blurred or may have glare on them. The annotators were mainly graduate and under-graduate students (8 in total); they had knowledge of various diagram types, and they were not part of the project, hence minimizing potential biases. They were instructed to draw these diagram types based on how these diagrams look in real world scenarios. They performed the task on a pro-bono basis in order to contribute towards the advancement of technology.

[2] https://graphviz.org/

3.2 Technical Diagram Understanding Tasks

We selected many tasks inspired from real-life use cases to help the model learn alignment between image, code, and textual description. The tasks are categorized into primary (used for both training and evaluation) (App Fig. 8) and self-supervision-based (employed exclusively during training). See App. E for example figures of all tasks.

Image2Code (Primary): Accurately translating images into structured code is essential for enabling automated diagram understanding and downstream reasoning. To this end, we frame the Image2Code task, where the model is required to generate corresponding Mermaid code for a given image. Once well-formed code is produced, rendering the technical diagram becomes straightforward using the Mermaid compiler.

Description2Code (Primary): Enabling the translation of structural descriptions into executable code is crucial for rapid prototyping, documentation, and iterative editing of technical diagrams. We therefore frame the task of converting natural language descriptions – capturing topological information such as the number of components and their connections – into code, using code as an intermediate representation. This approach allows diagrams to be generated and refined efficiently through iterative prompts.

Image2Description (Primary): Generating Descriptions from technical diagram images is essential for efficient search, retrieval, and question-answering tasks. These descriptions can also serve as contextual input for building advanced research agents, eliminating the need for the original image in later stages. By providing structured representations, the model enables diagrams to be indexed for both keyword-based queries (e.g., "find all diagrams with retry loops") and vector-based retrieval to identify structurally similar diagrams.

Image Enhancement via Prompt (Primary): Cer-

tain applications like versioning require us to update our existing diagrams, but these diagrams are in the form of an image and not directly editable. We can train a model so that it can directly generate the code of the updated image when given an image and a natural language enhancement prompt as input. This updated code can then be rendered into the desired image using the Mermaid compiler.

Self Supervision Tasks: We introduce a set of self-supervision tasks (only for training) to learn fine-grained alignment between images, Mermaid code, and textual descriptions. In the **image enhancement via description task**, the model receives a description of the target image and identifies which components are present and which need to be added in the code. In **code enhancement via prompt**, the model updates given Mermaid code based on an enhancement prompt, while in **code enhancement via description**, it completes missing parts of the code from a description. The **positive/negative image-code pair Q&A task** involves predicting whether a given image and code pair match, while the **partial match image-code pair Q&A** task requires identifying partial matches between incomplete and complete image-code pairs to capture sub-part relationships.

4 Methodology and Experiments

We benchmark various VLMs/LLMs Llama3.2-11B-Vision-Instruct (Meta, 2024), MiniCPM-V-2-6(8B) (Yao et al., 2024), Qwen2.5-VL-7B-Instruct (Bai et al., 2025), Gemma3-12B-Instruction-Tuned (Team et al., 2025) and GPT-4o-mini (OpenAI, 2024b)) in zero-shot setting for the baseline (App. F). Zero-shot evaluation was deliberately chosen for the baselines to establish a fair and widely adopted point of comparison without requiring fine-tuning. Moreover, typically in a practical setting, VLMs/LLMs are used in a zero-shot setting. Our intent is to highlight the effectiveness of the proposed corpus and training pipeline, rather than to introduce new architectural or prompt-driven innovations. Given the poor performance of baselines (§5), we fine-tuned a pre-trained Llama-3.2-11B-Vision-Instruct (Meta, 2024) model using LoRA (Hu et al., 2021) on the synthetic part of the corpus using tasks described earlier. Subsequently, the trained model is tested on real-world data and evaluated by humans to gauge its practical applicability and generalizability. To further validate our corpus and fine-tuning strategy, we also fine-tuned Gemma3-12B-Instruction-Tuned and evaluated it on the Synthetic Corpus Evaluation set across the three primary tasks, following the same procedure as with our fine-tuned model, **`LLama-VL-TUG`**. The results (Fig. 2) are consistent with those of **`LLama-VL-TUG`**, supporting the effectiveness of our corpus and fine-tuning approach.

Fine-tuning: We fine-tuned Llama3.2-11B-Vision-Instruct using LoRA (image encoder as well as text decoder) on the combination of all tasks (both Primary and Self Supervision) using **D1** and **D2**. While there are several options available with regard to the choice of VLM, due to limited compute availability, we focused on fine-tuning just two VLMs. Out aim is to validate the effectiveness of the training pipeline and code-generation approach, and insights gained from these experiments can easily be generalized to other models. Our goal is to demonstrate the benefits of the methodology rather than compare fine-tuning performance across every model. During fine-tuning, a task is selected randomly (all tasks having equal probability), and accordingly, a data point is also sampled randomly with suitable input-output pair selected from the corpus, which then goes into the model, and gradient updates are made (see App. H for hyperparameter details). For instance, in Image2Code task, a sample from **D1** corpus with Image as input and Mermaid Code as target output is selected for the forward and backward pass, respectively. Eventually, the model is fine-tuned on a dynamic mixture of tasks sampled in a random order at each training step. This helps the model generalize across tasks, handle different input types, and transfer knowledge to various technical reasoning scenarios. We refer to the model as **`LLama-VL-TUG`** (LLama-Vision Language-Technical image Understanding and Generation). To improve the diversity of the data and bring it closer to a real-world setting, we also performed data augmentation on every training sample by performing various transformations while training, such as blur, noise, lighting changes, and distortions (App. G).

Evaluation: We evaluated all the models on synthetic as well as real-world diagrams (Fig. 1). The synthetic evaluation corpus was created by the same strategy as **D1**, i.e., for each diagram type, we programmatically generated 500 Image-Mermaid Code-Description triplets and selected the input output pair according to the task. For the real world evaluation, real world corpus **D3** was used. For the synthetic diagrams, we conducted evaluation of our fine-tuned model and the

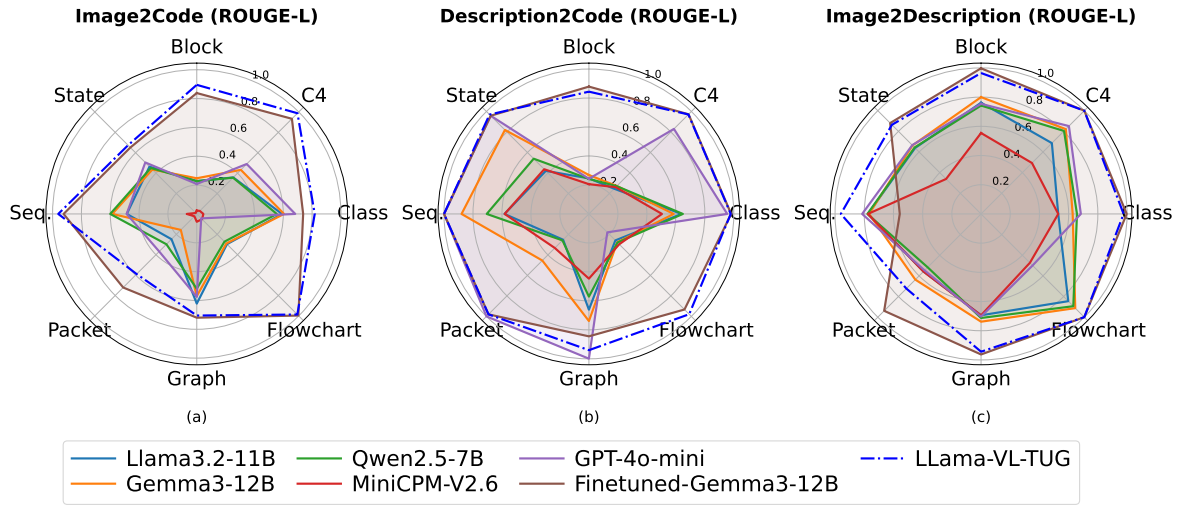

Figure 2: Results on Image2Code, Description2Code, and Image2Description tasks for the synthetic corpus (**D1**).

baselines on the three primary tasks, Image2Code, Description2Code, and Image2Description using standard metrics, BLEU (Papineni et al., 2002), SACREBLEU (Post, 2018), METEOR (Banerjee and Lavie, 2005), chrF (Popović, 2015), BLEURT (Sellam et al., 2020), and ROUGE-L (Lin, 2004) (deatils in App. I). Although order-agnostic evaluation tools such as *mtool* (Oepen et al., 2019) are available, they are limited to graph-based representations; therefore, we restrict our evaluation to the aforementioned automatic metrics for consistency across graph-based and non-graph based diagrams. The real-world images corpus (**D3**) was used to evaluate on Image2Code task with the same aforementioned metrics. All the experiments were conducted in a zero shot setting, and seeds were saved for reproducibility.

Human Evaluation: Automatic evaluation metrics such as BLEURT and ROUGE-L, while widely used in natural language tasks, are not well-suited for evaluating the Image2Code task. These metrics primarily capture surface-level textual similarity and struggle to account for structural correctness or semantic equivalence in code, where multiple variations may yield the same diagram (order-agnostic). Owing to these limitations, and to ensure reliable assessment, we conducted human evaluation on **D3**, focusing on issues such as incorrect diagram types, compilation errors (errors in the Mermaid code syntax that cause image generation to fail), spurious or missing structures (blocks, edges, attributes, etc.). Eight human annotators (participating on a pro bono basis) (different from those who created diagrams) were asked to compare the original image with the image regenerated from the code (for the hand-drawn image) generated by the model, and they were asked to evaluate on various metrics. For the Image2Code task, the evaluators first evaluate % of compilation errors of the generated Mermaid code. Among the successfully compiled images, the evaluators compute models' precision, recall, and F1 score for each structure (blocks, edges, attributes, etc.), taking the hand-drawn image as the ground truth (details in App. I).

Ablation Study: An ablation study was also performed where we fine-tuned the model only on primary tasks (without self-supervision tasks) and evaluated on the Image2Code task using **D3** to compare its performance against **`LLama-VL-TUG`**.

5 Results, Analysis and Discussion

Automatic Eval. Results and Analysis: Fig. 2 shows radar plots (with ROUGE-L scores per diagram type) for Image2Code, Description2Code, and Image2Description. In **Image2Code** task **`LLama-VL-TUG`** consistently outperforms all baselines, including GPT-4o-mini on all 6 metrics (results in App. Table 7). **`LLama-VL-TUG`** has near perfect ROUGE-L scores on C4, Flowchart and Sequence Diagrams (Fig. 2(a)). The effectiveness of finetuning on **`TechING`** is evident, as our model improves the average ROUGE-L scores (average over all diagram types) of Llama3.2-11B-Vision-Instruct by 2.29x and over GPT-4o-mini by 4.53x. A huge improvement (22.6x) is seen on the flowchart type image over GPT-4o-mini. In **Description2Code** task (Fig. 2(b)) highlights that **`LLama-VL-TUG`** has the best average ROUGE-L performance. Our model is competitive with GPT-4o-

Model	Diag	CEr*↓	CBl*↑	CEd*↑	CLE*↑/CAM*↑	Diag	CEr*↓	CBl*↑	CEd*↑	CLE*↑/CBi*↑
Llama3.2-11B		0.76	0.33	0.20	0.05		0.56	0.52	0.37	0.03
Llama-VL-TUG (ours)		**0.03** (25x)	0.80 (2.4x)	0.58 (2.9x)	0.35 (7x)		**0.00** (~2x)	**0.96** (1.8x)	**0.88** (2.4x)	**0.78** (26x)
Gemma3-12B	**Block**	0.08	**0.92**	**0.77**	0.38	**Graph**	0.13	0.86	0.71	0.35
Qwen2.5-VL-7B		0.27	0.81	0.66	0.41		0.23	0.80	0.72	0.36
MiniCPM-V-2-6		0.16	0.81	0.50	0.19		0.30	0.69	0.59	0.08
GPT-4o-mini		0.07	**0.92**	0.60	**0.43**		0.09	0.92	0.83	0.75
Llama3.2-11Bt		0.06	0.07	0.20	NA		0.66	0.00	NA	0.00
Llama-VL-TUG (ours)		**0.03** (2x)	**0.71** (10x)	0.35 (1.8x)	NA		**0.04** (16.5x)	**0.40** (~40x)	NA	**0.15** (~15x)
Gemma3-12B	**C4**	0.55	0.51	0.21	NA	**Packet**	0.23	0.00	NA	0.00
Qwen2.5-VL-7B		0.68	0.36	0.18	NA		0.51	0.00	NA	0.00
MiniCPM-V-2-6		0.76	0.18	0.01	NA		0.51	0.00	NA	0.00
GPT-4o-mini		0.37	0.64	**0.41**	NA		0.25	0.00	NA	0.00
Llama3.2-11B		0.57	0.60	0.36	0.56		**0.00**	0.59	NA	0.34
Llama-VL-TUG (ours)		**0.03** (19x)	0.79 (1.3x)	0.66 (1.8x)	0.71 (1.3x)		**0.00** (1x)	0.78 (1.3x)	NA	0.39 (1.1x)
Gemma3-12B	**Class**	0.86	0.86	**0.86**	0.86	**Sequence**	0.10	**0.93**	NA	0.64
Qwen2.5-VL-7B		0.32	0.72	0.45	0.70		**0.00**	0.67	NA	0.45
MiniCPM-V-2-6		0.92	0.09	0.02	0.00		0.57	0.54	NA	0.35
GPT-4o-mini		**0.03**	**0.98**	**0.86**	**0.93**		0.02	0.90	NA	**0.72**
Llama3.2-11B		0.86	0.25	0.21	0.18		0.73	0.34	0.19	0.00
Llama-VL-TUG (ours)		**0.00** (~6x)	0.86 (3.4x)	0.63 (3x)	0.29 (1.6x)		0.16 (4.6x)	0.52 (1.5x)	0.31 (1.6x)	0.19 (~19x)
Gemma3-12B	**Flowchart**	**0.00**	**0.97**	**0.88**	0.81	**State**	**0.03**	0.76	0.61	0.52
Qwen2.5-VL-7B		0.33	0.74	0.70	0.46		0.44	0.52	0.38	0.34
MiniCPM-V-2-6		0.31	0.69	0.44	0.11		0.24	0.68	0.29	0.05
GPT-4o-mini		0.04	0.95	**0.88**	**0.85**		**0.03**	**0.81**	**0.65**	**0.56**

Table 3: Human evaluation scores for Image2Code task on the Real-world images corpus **D3**. Improvement factor of **LLama-VL-TUG** after fine-tuning over the vanilla variant is indicated in parentheses. Except for Compilation Error **CEr**, all other columns show F1 score. * **CEr:** Compilation Error; **CBl:** Correct Blocks (blocks/nodes/classes/headers); **CEd:** Correct Edges; **CLE:** Correct Labeled Edges; **CAM:** Correct Attributes & Methods (for Class Diagrams); **CBi:** Correct Bits (for Packet Diagrams); NA: Not Applicable

mini on Graph and Packet diagrams and outperforms all other baselines by a significant margin on all other diagrams. Similar results are obtained on the remaining 5 metrics: BLEU, SACREBLEU, METEOR, chrF, and BLEURT (see App. Table 8). The fine-tuned model improves the ROUGE-L scores of Llama3.2-11B-Vision-Instruct by 2.75x and over GPT-4o-mini by 1.88x on this task. Major improvements are observed in Block (3.48x) and Flowchart (5.45x) over GPT-4o-mini. Similar results are observed in the **Image2Description** task. Fig 2(c) shows that **LLama-VL-TUG** has the best all-round ROUGE-L performance. The model outperforms GPT-4o-mini and all other baselines by a significant margin on State, Sequence, Graph, Class, C4, Block and Flowchart diagrams and is competitive on Packet diagrams (more results in App. Table 9). The fine-tuned model improves the average ROUGE-L scores of Llama3.2-11B-Vision-Instruct by 1.37x and outperforms GPT-4o-mini by 1.38x.

Human Evaluation: Table 3 summarizes the results of human evaluation for Image2Code performance on Block and C4 diagrams. We report F1 scores for evaluating model performance in generating correct structural components—blocks/nodes/classes/headers, edges, labeled edges, attributes and methods, and bits—from real-world hand-drawn images (detailed results in App. Table 10). For scoring block-type structures, both node detection and correct block name recognition are required. Edges are scored only when both the source and target nodes match (node-edge-node triplet), and labeled edges additionally require the correct label. This strict scoring allows us to implicitly capture different types of errors, such as OCR/text recognition failures, node detection errors, and edge routing mistakes. Our findings show that **LLama-VL-TUG** achieves the lowest percentage of compilation errors, with a 9.8x reduction compared to Llama3.2-11B-Instruct. Moreover, fine-tuning on **TechING** significantly boosts the F1 scores over Llama3.2-11B-Instruct: 7.7x on blocks, 2.23x on edges, and 10.9x on labeled edges. None of the baseline models correctly identified the diagram type for Packet Diagrams. Since Packet Diagrams are structurally distinct from regular graph-type diagrams, we assigned zero scores

for incorrect generations. Notably, only the fine-tuned model was able to accurately detect and generate Packet Diagrams. In graph diagrams, **LLama-VL-TUG** demonstrates strong performance, correctly generating all nodes, edges, and labeled edges. The evaluation also highlights strengths and weaknesses of baseline models: for instance, GPT-4o-mini performs well on blocks, edges, and attributes in class and state diagrams but shows mediocre performance on blocks and bit labels in packet diagrams, while Gemma3-12B reliably generates blocks across diagram types but struggles with accurate edge generation.

Ablation Study: Role of self-supervision tasks in fine-tuning is shown in Fig. 3. **LLama-VL-TUG** is shown to outperform the ablated variant on real-world hand-drawn corpus **D3** in the Image2Code task across nearly all diagram types on ROUGE-L score. Most notably, our model achieves the highest score on block and state diagrams, indicating that self-supervision aids in understanding structurally rich and semantically diverse visual elements. Interestingly, for some categories like sequence and graph, the gap between ablated models and **LLama-VL-TUG** is smaller, indicating that these tasks may be less dependent on the self-supervision signals but still benefit overall from the full training pipeline, as the representations learned via self-supervision can transfer across diagram types and help in enhancing performance.

An additional ablation direction would involve isolating the contribution of individual primary tasks. However, preliminary experiments with continual training—where the model was trained sequentially on Image2Code, Description2Code, and self-supervised image enhancement—revealed that performance on earlier tasks remained stable even after training on subsequent tasks. This suggested that the primary tasks cultivate overlapping capabilities, rendering task-specific ablations unlikely to yield meaningful insights. Consequently, we did not pursue this ablation study. More details and results in App. J

Discussion and Error Analysis: Real-world diagrams have noise and incompleteness, and a lot of variations in the user styles, leading to errors in the code generated by models. Resolving this can potentially require a huge amount of data for training. We have explored the use of image augmentation and found it to be useful for improving model performance. Although **LLama-VL-TUG** was trained using synthetic images in **TechING** only, it

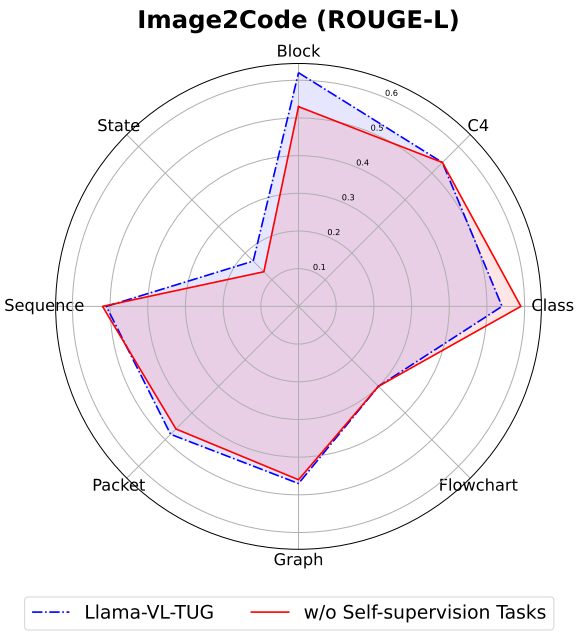

Figure 3: Ablation Study results (Real-World: **D3**)

demonstrated good generalization over real-world images (also see Limitations). Further, we find that most of the existing VLMs can reliably generate only graph-like diagrams, while for other types, the Mermaid code often fails to compile and includes hallucinated elements. The performance on edges is notably weak, with labeled edges performing even worse, including in sequence diagrams where messages act as labeled edges. Models also frequently generate the wrong diagram type, most notably in the case of packet diagrams, where all models fail to detect or generate them. Notably, only after fine-tuning does our model successfully detect and generate packet diagrams.

6 Conclusion

We address a low-resource setting and introduce a large synthetic corpus **TechING** with different complexity levels (Easy, Medium, Hard) to enhance the technical image understanding of VLMs. We benchmark various open 10B parameter models and closed source GPT-4o-mini model. We showed that fine-tuning on a pretrained VLM (resulting in **LLama-VL-TUG**) using a synthetic corpus via various tasks helps to improve real- world hand-drawn technical diagram understanding. Both automated and human evaluations show the efficacy of our approach compared to existing baseline models. In the future, we plan to continue to grow the corpus of real-world hand-drawn images to capture more variations in data (also see limitations).

Limitations

Corpus Size: In this paper, we made initial steps towards solving a hard problem due to large variability and domain specificity by introducing a large, diverse corpus. Since the synthetic images in **TechING** corpus were generated through a randomized strategy with few random templates, this may not capture all the possible variations that humans may introduce while creating diagrams by hand. To address these, we added 545 real-world hand-drawn diagrams to the corpus for evaluation. However, we anticipate encountering significantly greater variation in real-world scenarios. For example, the style of arrows, the way nodes and blocks are drawn, and even the diagram type vary widely in the real world. This may likely contribute to the decrease in performance, as the model is not specifically trained on all possible variations that are observed in the real-world images. Nevertheless, we plan to continue growing the corpus of real world hand-drawn images to capture more variations.

It is also difficult to manually evaluate large diagrams generated by VLMs; thus, we restricted the human evaluation on images with not more than 10-15 structures/blocks.

Synthetic Corpus: In this paper, we used a programmatic approach for generating synthetic images, as this guarantees consistency and correctness of technical diagrams. In our initial experiments, we tried diffusion and CLIP based models for technical image generation; however, these did not perform well and further research is required in training these models, which we plan to address in the future.

Real-world diagrams can have a lot of variations. For example, the style of arrows, the way nodes and blocks are drawn, and even the diagram type vary widely in the real world. In this work, we covered a subset of variations. Further work is needed to generate many variations of the same user-drawn image to increase the variety and create a large-size training corpus automatically. Given the challenges of manually evaluating large diagrams – where models may alter the diagram type or introduce spurious content – we deliberately limit human evaluation to diagrams with at most 10–15 structures or blocks. This controlled scope ensures consistency, reliability, and feasibility in the evaluation process.

Moreover, our dataset does not encompass the full range of highly irregular or informal hand-drawn sketches encountered in practice. While extremely messy diagrams remain challenging, the system generally produces a near-correct structural backbone that significantly reduces manual effort compared to diagram reconstruction from scratch. Future work should focus on improving robustness to more degraded visual inputs commonly found in engineering workflows.

Training Strategy: We experimented with a common finetuning strategy in a zero-shot setting for fine-tuning the VLM and did not explore complex techniques of context engineering, few-shot learning, etc., which we would like to explore in future work. We also acknowledge that fine-tuning all available open-source models could provide a more comprehensive comparison. However, due to computational constraints, we focused on fine-tuning a single model. Despite this limitation, our experiments demonstrate the effectiveness of the corpus through the strong performance of our fine-tuned model. Future work can explore fine-tuning additional models as more computational resources become available. Moreover, we would also like to explore other model architectures, such as mixture of experts VLMs.

Tasks: Further, in this paper, we experimented with a few tasks only (e.g., Image2Code, Description2Code, and Code2Description). While there can be several other tasks that could be used for training, we focused on the ones that we found practically relevant.

We deliberately adopt a simple, unified pipeline for diagram understanding to clearly evaluate the effectiveness of our corpus and training approach. This keeps the current study focused, while leaving systematic comparisons with other agentic pipelines like DiagramAgent (Wei et al., 2025) and exploration of hybrid or tool-augmented methods for future work.

Ethical Considerations

To the best of our knowledge, we do not see any direct societal harm from this work. In fact, this work aims to solve a problem that could benefit society at large.

References

Kriti Aggarwal, Aditi Khandelwal, Kumar Tanmay, Owais Khan Mohammed, Qiang Liu, Monojit Choudhury, Hardik Chauhan, Subhojit Som, Vishrav Chaud-

hary, and Saurabh Tiwary. 2023. Dublin: Visual document understanding by language-image network. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 693–706.

Harsh Agrawal, Karan Desai, Yufei Wang, Xinlei Chen, Rishabh Jain, Mark Johnson, Dhruv Batra, Devi Parikh, Stefan Lee, and Peter Anderson. 2019. nocaps: novel object captioning at scale. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE.

Qihang Ai, Jiafan Li, Jincheng Dai, Jianwu Zhou, Lemao Liu, Haiyun Jiang, and Shuming Shi. 2024. Advancement in graph understanding: A multimodal benchmark and fine-tuning of vision-language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7485–7501, Bangkok, Thailand. Association for Computational Linguistics.

et al. Aishwarya Kamath. 2025. Gemma 3 technical report.

Ahmad Montaser Awal, Guihuan Feng, Harold Mouchère, and Christian Viard-Gaudin. 2011. First experiments on a new online handwritten flowchart database. volume 7874, pages 1–10.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. 2025. Qwen2.5-vl technical report.

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, Thomas Unterthiner, Daniel Keysers, Skanda Koppula, Fangyu Liu, Adam Grycner, Alexey Gritsenko, Neil Houlsby, Manoj Kumar, Keran Rong, Julian Eisenschlos, Rishabh Kabra, Matthias Bauer, Matko Bošnjak, Xi Chen, Matthias Minderer, Paul Voigtlaender, Ioana Bica, Ivana Balazevic, Joan Puigcerver, Pinelopi Papalampidi, Olivier Henaff, Xi Xiong, Radu Soricut, Jeremiah Harmsen, and Xiaohua Zhai. 2024. Paligemma: A versatile 3b vlm for transfer.

Shreyanshu Bhushan, Eun-Soo Jung, and Minho Lee. 2024. Unveiling the power of integration: Block diagram summarization through local-global fusion. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13837–13856, Bangkok, Thailand. Association for Computational Linguistics.

Shreyanshu Bhushan and Minho Lee. 2022. Block diagram-to-text: Understanding block diagram images by generating natural language descriptors. In *Findings of the Association for Computational Linguistics: AACL-IJCNLP 2022*, pages 153–168, Online only. Association for Computational Linguistics.

Xiaohui Chen, Satya Narayan Shukla, Mahmoud Azab, Aashu Singh, Qifan Wang, David Yang, ShengYun Peng, Hanchao Yu, Shen Yan, Xuewen Zhang, and Baosheng He. 2024. Compcap: Improving multimodal large language models with composite captions.

Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollar, and C. Lawrence Zitnick. 2015. Microsoft coco captions: Data collection and evaluation server.

Zhi-Qi Cheng, Qi Dai, and Alexander G. Hauptmann. 2023. Chartreader: A unified framework for chart derendering and comprehension without heuristic rules. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 22202–22213.

Claudio Filipi Goncalves dos Sants, Felype de Castro Bastos, Ana Claudia Akemi Matsuki de Faria, Jose Victor Nogueira Alves da Silva, Valeska de Sousa Uchoa, and Decio Goncalves de Aguiar Neto. 2023. Visual question answering: A survey on techniques and common trends in recent literature.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. 2017. Making the v in vqa matter: Elevating the role of image understanding in visual question answering.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

Drew A. Hudson and Christopher D. Manning. 2019. Gqa: A new dataset for real-world visual reasoning and compositional question answering.

Samira Ebrahimi Kahou, Vincent Michalski, Adam Atkinson, Akos Kadar, Adam Trischler, and Yoshua Bengio. 2018. Figureqa: An annotated figure dataset for visual reasoning.

Aniruddha Kembhavi, Michael Salvato, Eric Kolve, Minjoon Seo, Hannaneh Hajishirzi, and Ali Farhadi. 2016. A diagram is worth a dozen images. *ArXiv*, abs/1603.07396.

Jia Li, Lijie Hu, Zhixian He, Jingfeng Zhang, Tianhang Zheng, and Di Wang. 2024a. Text guided image editing with automatic concept locating and forgetting.

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

Xin Li, Yunfei Wu, Xinghua Jiang, Zhihao Guo, Mingming Gong, Haoyu Cao, Yinsong Liu, Deqiang Jiang, and Xing Sun. 2024b. Enhancing visual document understanding with contrastive learning in large visual-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15546–15555.

Xue Li, Yiyou Sun, Wei Cheng, Yinglun Zhu, and Haifeng Chen. 2025. Chain-of-region: Visual language models need details for diagram analysis. In *The Thirteenth International Conference on Learning Representations*.

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

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. 2024. Mmbench: Is your multi-modal model an all-around player? In *European conference on computer vision*, pages 216–233. Springer.

Haoyu Lu, Wen Liu, Bo Zhang, Bingxuan Wang, Kai Dong, Bo Liu, Jingxiang Sun, Tongzheng Ren, Zhuoshu Li, Hao Yang, Yaofeng Sun, Chengqi Deng, Hanwei Xu, Zhenda Xie, and Chong Ruan. 2024. Deepseek-vl: Towards real-world vision-language understanding.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. 2022. Chartqa: A benchmark for question answering about charts with visual and logical reasoning.

Meta. 2024. Llama3.2.

Nitesh Methani, Pritha Ganguly, Mitesh M. Khapra, and Pratyush Kumar. 2020. Plotqa: Reasoning over scientific plots.

Stephan Oepen, Omri Abend, Jan Hajic, Daniel Hershcovich, Marco Kuhlmann, Tim O'Gorman, Nianwen Xue, Jayeol Chun, Milan Straka, and Zdenka Uresova. 2019. MRP 2019: Cross-framework meaning representation parsing. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 1–27, Hong Kong. Association for Computational Linguistics.

OpenAI. 2024a. Gpt-4o: Advancing multimodal intelligence. Accessed: 2025-04-14.

OpenAI. 2024b. Gpt-4o mini: advancing cost-efficient intelligence. Accessed: 2025-04-14.

Huitong Pan, Qi Zhang, Cornelia Caragea, Eduard Dragut, and Longin Jan Latecki. 2024. Flowlearn: Evaluating large vision-language models on flowchart understanding.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Juan A. Rodriguez, Abhay Puri, Shubham Agarwal, Issam H. Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. 2024. Starvector: Generating scalable vector graphics code from images and text.

Dustin Schwenk, Apoorv Khandelwal, Christopher Clark, Kenneth Marino, and Roozbeh Mottaghi. 2022. A-okvqa: A benchmark for visual question answering using world knowledge.

Thibault Sellam, Dipanjan Das, and Ankur Parikh. 2020. BLEURT: Learning robust metrics for text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7881–7892, Online. Association for Computational Linguistics.

Shreya Shukla, Prajwal Gatti, Yogesh Kumar, Vikash Yadav, and Anand Mishra. 2023. Towards making flowchart images machine interpretable. In *Document Analysis and Recognition - ICDAR 2023*, pages 505–521, Cham. Springer Nature Switzerland.

Shubhankar Singh, Purvi Chaurasia, Yerram Varun, Pranshu Pandya, Vatsal Gupta, Vivek Gupta, and Dan Roth. 2024. FlowVQA: Mapping multimodal logic in visual question answering with flowcharts. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 1330–1350, Bangkok, Thailand. Association for Computational Linguistics.

Simon Tannert, Marcelo G. Feighelstein, Jasmina Bogojeska, Joseph Shtok, Assaf Arbelle, Peter W. J. Staar, Anika Schumann, Jonas Kuhn, and Leonid Karlinsky. 2023. FlowchartQA: The first large-scale benchmark for reasoning over flowcharts. In *Proceedings of the 1st Workshop on Linguistic Insights from and for*

Multimodal Language Processing, pages 34–46, Ingolstadt, Germany. Association for Computational Linguistics.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. 2025. Gemma 3 technical report.

Shaowei Wang, Lingling Zhang, Longji Zhu, Tao Qin, Kim-Hui Yap, Xinyu Zhang, and Jun Liu. 2024. Cogdqa: Chain-of-guiding learning with large language models for diagram question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13969–13979.

Jingxuan Wei, Cheng Tan, Qi Chen, Gaowei Wu, Siyuan Li, Zhangyang Gao, Linzhuang Sun, Bihui Yu, and Ruifeng Guo. 2025. From Words to Structured Visuals: A Benchmark and Framework for Text-to-Diagram Generation and Editing . In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13315–13325, Los Alamitos, CA, USA. IEEE Computer Society.

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

Yuan Yao, Tianyu Yu, Ao Zhang, Chongyi Wang, Junbo Cui, Hongji Zhu, Tianchi Cai, Haoyu Li, Weilin Zhao, Zhihui He, Qianyu Chen, Huarong Zhou, Zhensheng Zou, Haoye Zhang, Shengding Hu, Zhi Zheng, Jie Zhou, Jie Cai, Xu Han, Guoyang Zeng, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Minicpm-v: A gpt-4v level mllm on your phone.

Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. 2014. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics (TACL)*, 2:67–78.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. 2024. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9556–9567.

Rowan Zellers, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. From recognition to cognition: Visual commonsense reasoning.

Chenshuang Zhang, Chaoning Zhang, Mengchun Zhang, In So Kweon, and Junmo Kim. 2024. Text-to-image diffusion models in generative ai: A survey.

Appendix

Table of Contents

List of Tables

List of Figures

A Related work

The domain of technical image understanding has recently started gaining attention, and researchers have focused on tasks like diagram understanding and reasoning (Kembhavi et al. (2016)), diagram summarization Bhushan and Lee (2022); Bhushan et al. (2024), and Question Answering (Masry et al. (2022); Methani et al. (2020); Kahou et al. (2018)). Accordingly, various datasets have been proposed. CBD (Bhushan and Lee, 2022) was created by web crawling from different search engines for the summarization task of block diagrams; it contained a total of 502 samples. FlowchartQA (Tannert et al., 2023) comprised of 1M programmatically created flowchart images using graphviz[3] by selecting random names and edge labels. The dataset contained 6M questions on geometry and topology using a range of templates. BD-EnKo (Bhushan et al., 2024), a multilingual, mostly synthetic summarization dataset of 47k images of the English language (91 real world images) was generated by prompting GPT-3.5 to give Mermaid code of various types and random themes of diagrams. The summary was also generated by GPT-3.5 by providing the generated code as a prompt. FlowVQA (Singh et al., 2024), a QA dataset, used real-world examples of WikiHow articles, Instructables DIY blogs, and FloCo (Shukla et al., 2023) to create structured summaries and then again converting the summary into Mermaid code by GPT-4. Questions were also generated by GPT-4 by giving a summary and Mermaid code and a template question. This dataset contained 2,272 images and 22,413 questions. Ai et al. (2024) introduced a graph dataset with 3,929 English and 2,747 Chinese image-text pairs, where images were scraped from search engines, and text in the form of questions with candidate answers were generated by GPT-4V. DoCo (Li et al., 2024b) employed Contrastive Learning in the pretraining phase and then using the encoding in the finetuning phase of an LLM. CoG-DQA (Wang et al., 2024) CoG-DQA leverages LLMs to guide diagram parsing tools (DPTs) through the guiding chains, enhancing the precision of diagram parsing for the QA task.

The FlowLearn dataset (Pan et al., 2024) is designed to improve the understanding of flowcharts, with a particular emphasis on scientific contexts. It consists of two subsets: Scientific Flowcharts (3,858 diagrams) curated from real-world sources, and Simulated(through Python scripts) Flowcharts (10,000 diagrams) generated to augment coverage. While FlowLearn provides valuable resources for studying flowchart comprehension, it is limited in scope as it focuses exclusively on a single diagram type, namely flowcharts. Additionally, it contains no hand-drawn real-world images, relying only on clean, structured diagrams, which reduces its suitability for evaluation on real-world hand-drawn images. Another similar yet distinctive domain related to our work involves the use of SVG for image generation, as seen in approaches like Rodriguez et al. (2024) and Wu et al. (2024). While their method targets simple images and employs SVG code as an intermediate representation, our work focuses on the understanding of technical diagrams using Mermaid code.

ChartReader (Cheng et al., 2023) focuses on understanding numerical information from charts, rather than structural or diagrammatic content, and supports chart types such as bar, line, and pie charts in tasks like Chart-to-Table, ChartQA, and Chart Summarization. CHAIN-OF-REGION (Li et al., 2025) improves VLM performance on scientific diagrams by decomposing diagrams into regions using computer vision techniques. In contrast, our work focuses on diagram regeneration and editing via an intermediate code representation for structural understanding across diverse diagram types.

DiagramGenBenchmark and DiagramAgent (Wei et al., 2025) introduce text-to-diagram generation, producing structured, editable diagrams from text, primarily covering flowcharts, model architecture diagrams, and mind maps, targeting a completely different set of diagrams from our work.

Table 1 summarizes various available datasets along with **TechING**. As can be seen, **TechING** covers more diagram types than existing works and includes a much larger set of hand-drawn real-world images.

B TechING Image Examples

We include the following diagram types in our work: Block, C4, Class Flowchart, Graph, Packet, Sequence, and State diagrams. We provide examples of each diagram type (both synthetic and hand-drawn versions) in Fig. 4 and Fig. 5.

C TechING Creation

Fig. 6 and Fig. 7 illustrate the dataset creation pipelines for **D1** (Image-Mermaid Code-

[3]https://graphviz.org/

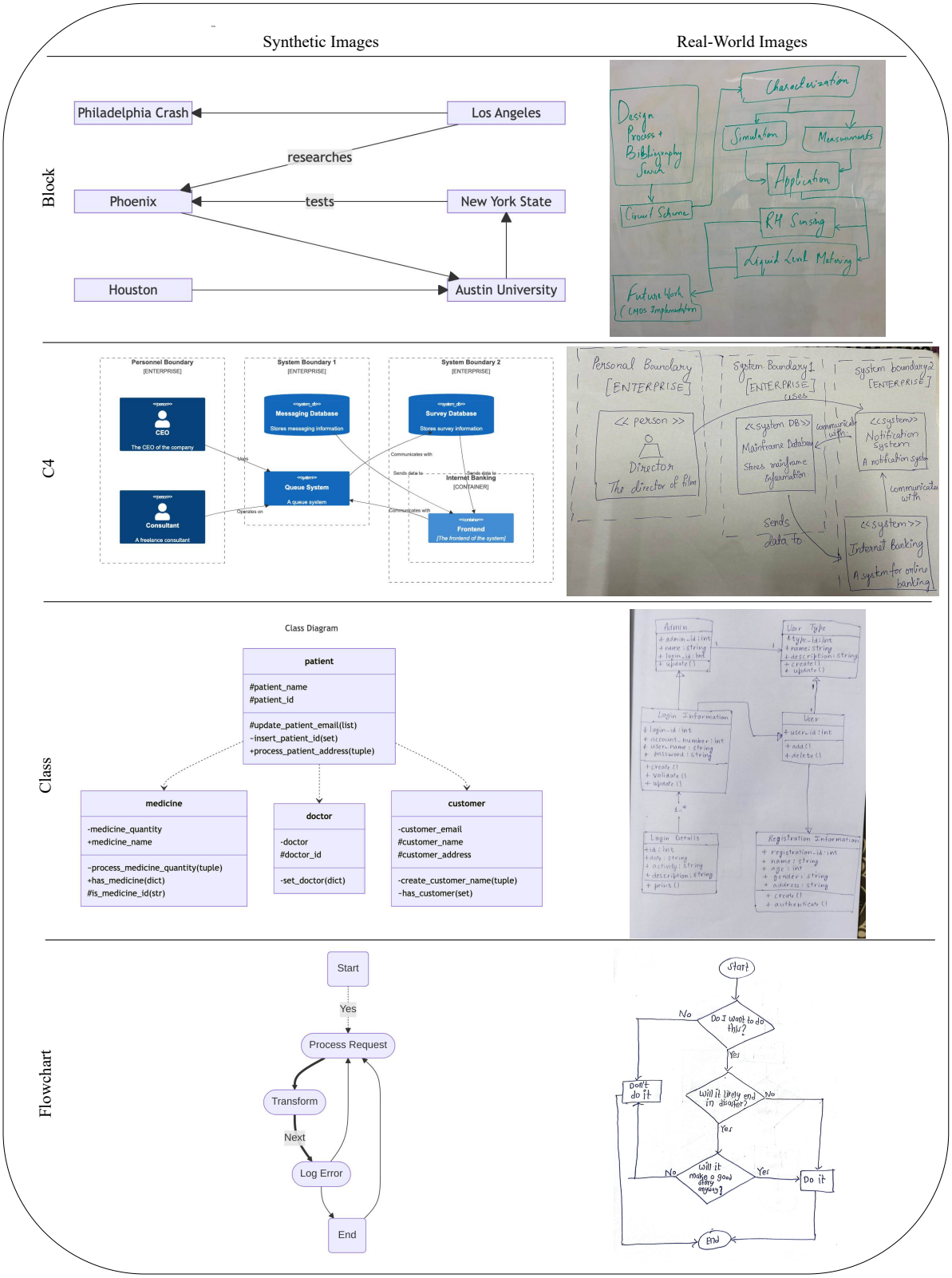

Figure 4: Synthetic and Real-World Examples for Block, C4, Class, Flowchart Diagrams

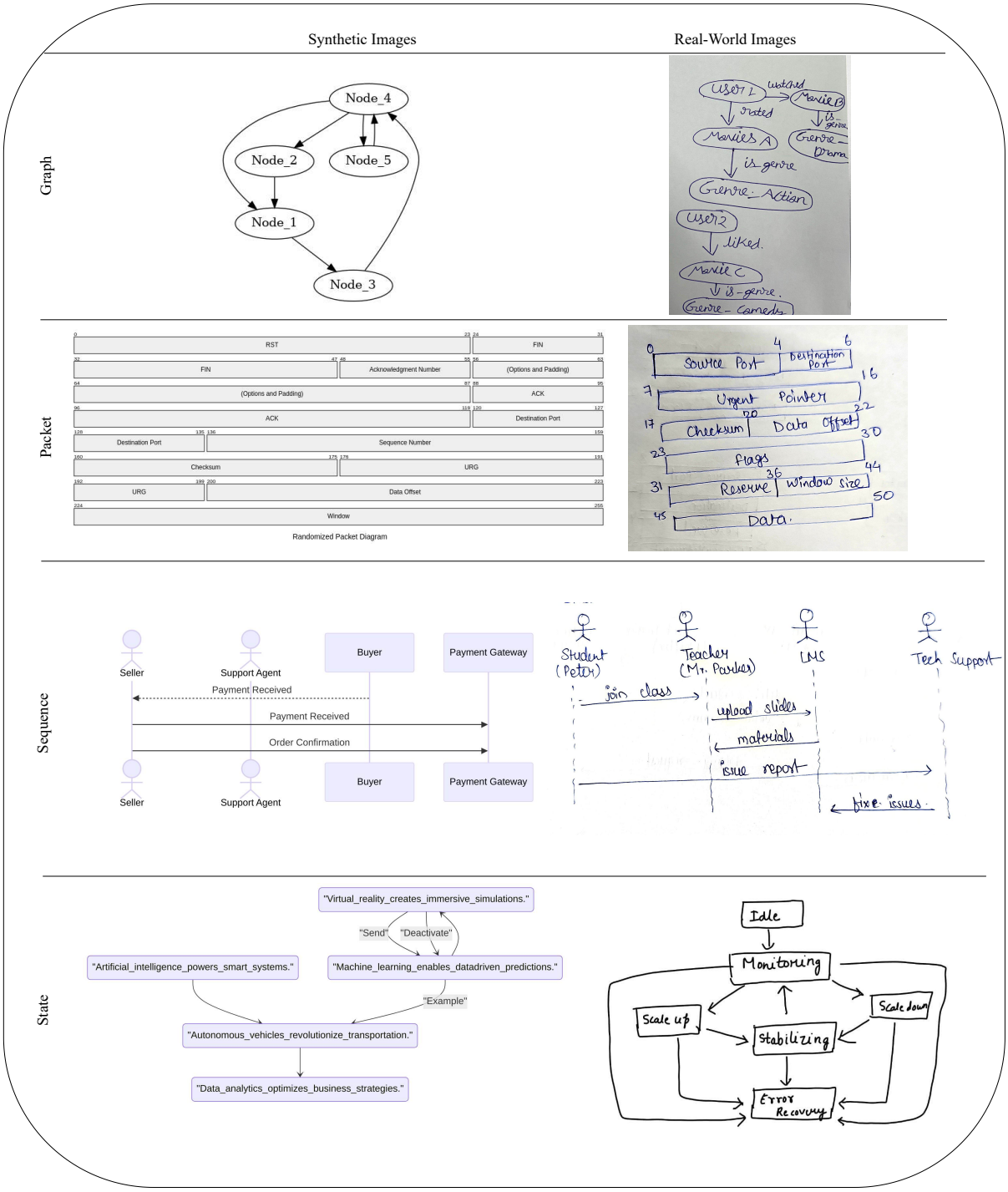

Figure 5: Synthetic and Real-World Examples for Graph, Packet, Sequence, State Diagrams

Description Corpus) and **D2** (Image-Enhancement Corpus), providing a visual overview of the step-by-step processes involved in generating each corpus.

D TechING Statistics

Table 4 provides the detailed statistics about the various diagram types in the dataset across different difficulty levels (Easy, Medium, and Hard). For each diagram type, the table reports the minimum, mean ± standard deviation, and maximum values for several structural properties, including the number of edges, blocks/classes, attributes, headers, and code length. These statistics highlight the variation in structural complexity across both diagram categories and difficulty levels. For instance, diagrams in the Medium and Hard categories generally exhibit a higher number of edges and blocks, along with longer code lengths, indicating more intricate layouts and richer content. Conversely, Easy diagrams tend to have simpler structures with fewer components, making them less challenging for both model training and evaluation. This detailed breakdown provides a comprehensive view of the dataset's distribution and structural diversity, which is crucial for understanding model performance across different levels of complexity.

E Tasks Descriptions and Examples

The primary task is described in detail in the main paper. Fig. 8 shows the formulation of different Primary Tasks utilizing **TechING**. Fig. 9, 10, 11 and 12 provides examples of each of the primary tasks, Image2Code, Description2Code, Image2Description, and Image Enhancement via Prompt. The self-supervision tasks, which are used exclusively during training to improve fine-grained alignment and structural understanding, are outlined below.

Image Enhancement via Description: In this task, the model is given an image along with a textual description of the target image, and must produce code that reflects the enhanced description, effectively adding missing components to align with the full target diagram. By leveraging both visual and textual cues, the model learns fine-grained alignment between the image, description, and code, enabling it to enhance incomplete diagrams and generate more complete and accurate code representations. This training signal also enables the model to recover missing parts in diagram components and relationships by grounding textual edits in the visual structure of the input image. An example of the task is shown in Fig. 13.

Code Enhancement via Prompt: The model is given existing Mermaid code along with an enhancement prompt and must update the code accordingly. This encourages the model to understand structural modifications in the provided instructions via prompt and apply contextual edits, which is crucial for handling incremental edits and refinements during diagram generation and editing. Fig. 14 shows a representative example of the task.

Code Enhancement via Description: The model receives a Mermaid code snippet along with a natural language description of the target image and is tasked with enhancing the code to accurately reflect the changes present in the description. This objective requires the model to interpret textual description of the image and apply the corresponding structural modifications to the code. As a result, the task strengthens code completion and generation capabilities, ensuring the model can effectively bridge gaps between natural language descriptions and structured code representations. An illustration of the task can be seen in Fig. 15.

Positive/Negative Image–Code Pair Q&A: The model predicts whether a given image–code pair constitutes a valid match or a mismatch. This binary classification objective requires the model to jointly reason over visual and structural code representations, enabling it to distinguish semantically consistent pairs from incorrect or misaligned ones. By explicitly learning to identify mismatches, the model develops a stronger notion of cross-modal alignment, which improves robustness to noisy or ambiguous inputs. Fig. 16 illustrates an example of the task.

Partial Match Image–Code Pair Q&A: The model determines whether the generated code partially matches the given image by capturing fine-grained sub-part relationships between visual elements and their corresponding code components. Instead of enforcing a complete one-to-one correspondence, this formulation encourages alignment between relevant subsets of the image and portions of the code, even when the match is incomplete. Through this process, the model learns to reason about partial correspondences between visual components and structured code representations, which is particularly beneficial for complex, noisy, or incomplete diagrams. Fig. 17 depicts how the task is performed through an example.

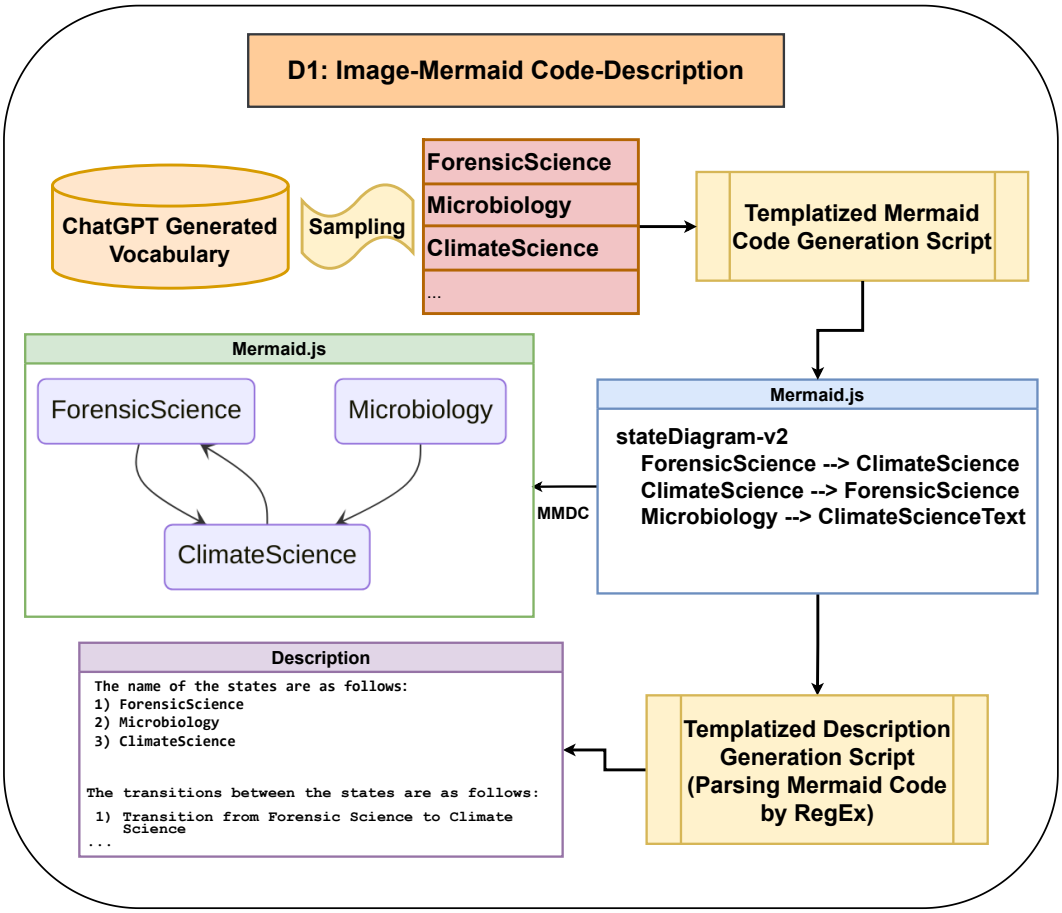

Figure 6: **D1**: Image-Mermaid Code-Description Corpus generation process.

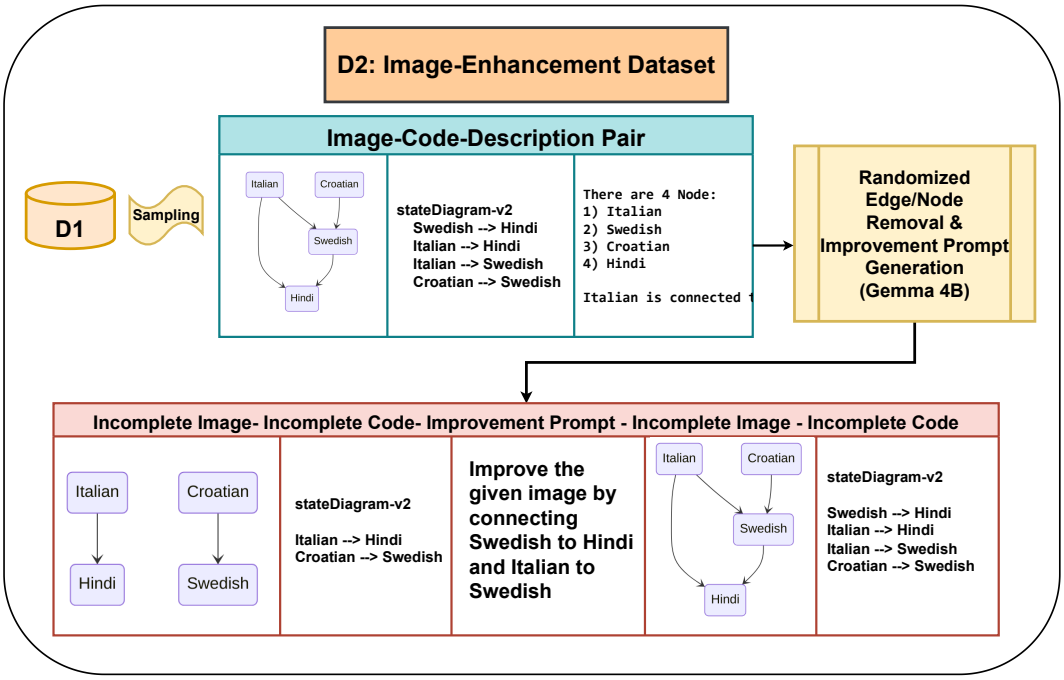

Figure 7: **D2**: Image Enhancement Corpus generation process

Diag Type	Level	Edges	Blocks/Classes	Attributes	Headers	Code Length
Block	Easy	(1, 1.00 ± 0.00, 1)	(2, 2.01 ± 0.11, 3)	-	-	(80, 142.6 ± 28.3, 237)
	Medium	(1, 4.94 ± 0.92, 6)	(4, 4.51 ± 0.78, 7)	-	-	(247, 439.2 ± 112.6, 772)
	Hard	(1, 6.16 ± 1.73, 9)	(4, 6.05 ± 1.44, 9)	-	-	(303, 568.6 ± 97.8, 971)
C4	Easy	(1, 1.00 ± 0.00, 1)	(1, 1.50 ± 0.50, 2)	-	-	(221, 314.9 ± 36.4, 415)
	Medium	(4, 4.77 ± 0.66, 6)	(3, 4.52 ± 0.88, 6)	-	-	(833, 1077.2 ± 91.6, 1352)
	Hard	(4, 4.77 ± 0.67, 6)	(3, 4.52 ± 0.869, 6)	-	-	(859, 1079.6 ± 91.0, 1355)
Class	Easy	(1, 1.00 ± 0.00, 1)	(2, 2.00 ± 0.00, 2)	(4, 4.67 ± 1.56, 10)	-	(198, 254.2 ± 46.8, 459)
	Medium	(3, 3.25 ± 0.66, 5)	(4, 4.25 ± 0.66, 6)	(9, 17.09 ± 3.59, 33)	-	(460, 726.8 ± 133.7, 1287)
	Hard	(5, 5.00 ± 0.00, 5)	(6, 6.00 ± 0.00, 6)	(14, 23.92 ± 2.80, 34)	-	(730, 1016.3 ± 89.2, 1291)
Flowchart	Easy	(3, 3.50 ± 0.50, 4)	(4, 4.50 ± 0.50, 5)	-	-	(173, 222.0 ± 29.7, 277)
	Medium	(4, 6.18 ± 1.40, 9)	(5, 6.00 ± 0.82, 7)	-	-	(196, 287.8 ± 50.7, 394)
	Hard	(4, 5.67 ± 1.41, 9)	(5, 5.93 ± 0.69, 7)	-	-	(198, 279.7 ± 45.8, 393)
Graph	Easy	(1, 1.68 ± 0.47, 2)	(3, 3.34 ± 0.57, 5)	-	-	(41, 109.0 ± 36.9, 234)
	Medium	(3, 3.76 ± 0.83, 5)	(2, 4.06 ± 0.93, 8)	-	-	(62, 141.8 ± 56.9, 344)
	Hard	(3, 5.64 ± 1.46, 7)	(2, 4.72 ± 1.36, 8)	-	-	(81, 131.2 ± 20.5, 213)
Packet	Easy	-	-	-	(2, 2.97 ± 0.17, 3)	(95, 120.8 ± 8.9, 142)
	Medium	-	-	-	(3, 4.99 ± 1.71, 8)	(114, 177.6 ± 49.6, 322)
	Hard	-	-	-	(6, 8.23 ± 2.14, 17)	(174, 258.7 ± 43.9, 419)
Sequence	Easy	(1, 1.44 ± 0.98, 4)	(2, 2.35 ± 0.76, 4)	-	-	(47, 85.7 ± 67.4, 268)
	Medium	(3, 3.50 ± 0.50, 4)	(2, 3.91 ± 0.32, 4)	-	-	(137, 232.1 ± 34.9, 366)
	Hard	(5, 6.60 ± 1.11, 8)	(6, 7.61 ± 1.10, 9)	-	-	(303, 422.0 ± 67.6, 538)
State	Easy	(1, 1.88 ± 0.86, 4)	(2, 3.40 ± 1.28, 8)	-	-	(32, 71.8 ± 30.0, 217)
	Medium	(5, 7.81 ± 1.78, 12)	(5, 10.61 ± 2.43, 19)	-	-	(118, 248.9 ± 73.5, 518)
	Hard	(5, 6.52 ± 0.94, 8)	(3, 8.33 ± 1.56, 13)	-	-	(441, 703.2 ± 119.8, 1093)

Table 4: Detailed statistics of different diagram types across difficulty levels. Each cell shows (min, mean ± std, max).

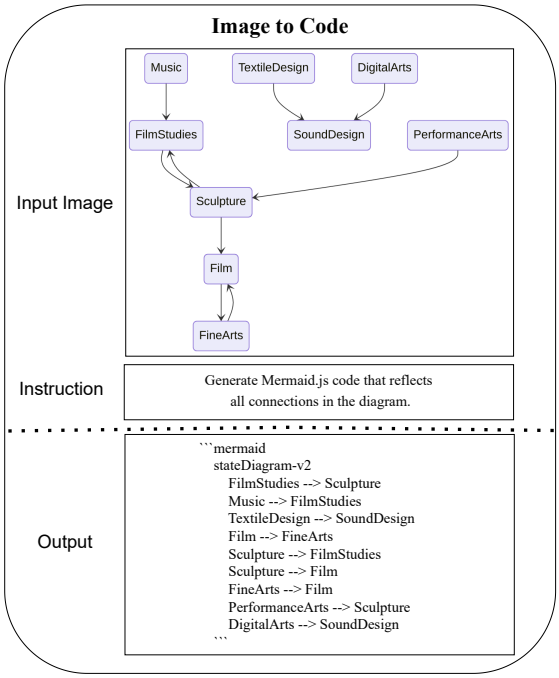

Figure 9: Image2Code Task Example

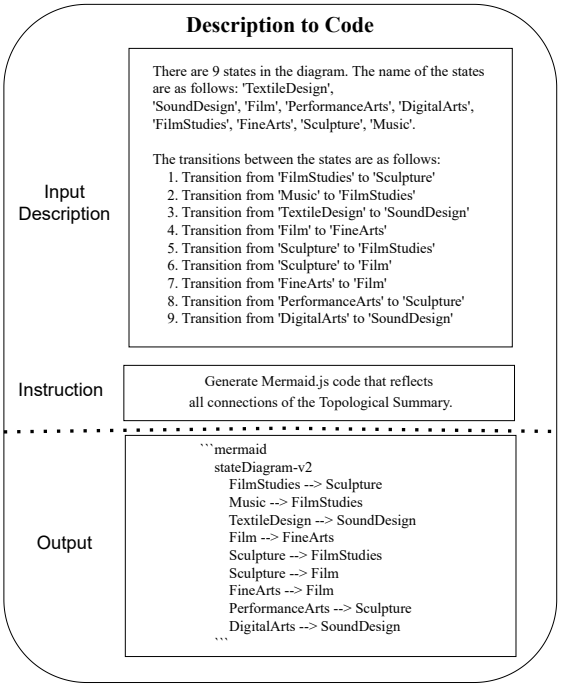

Figure 10: Description2Code Task Example

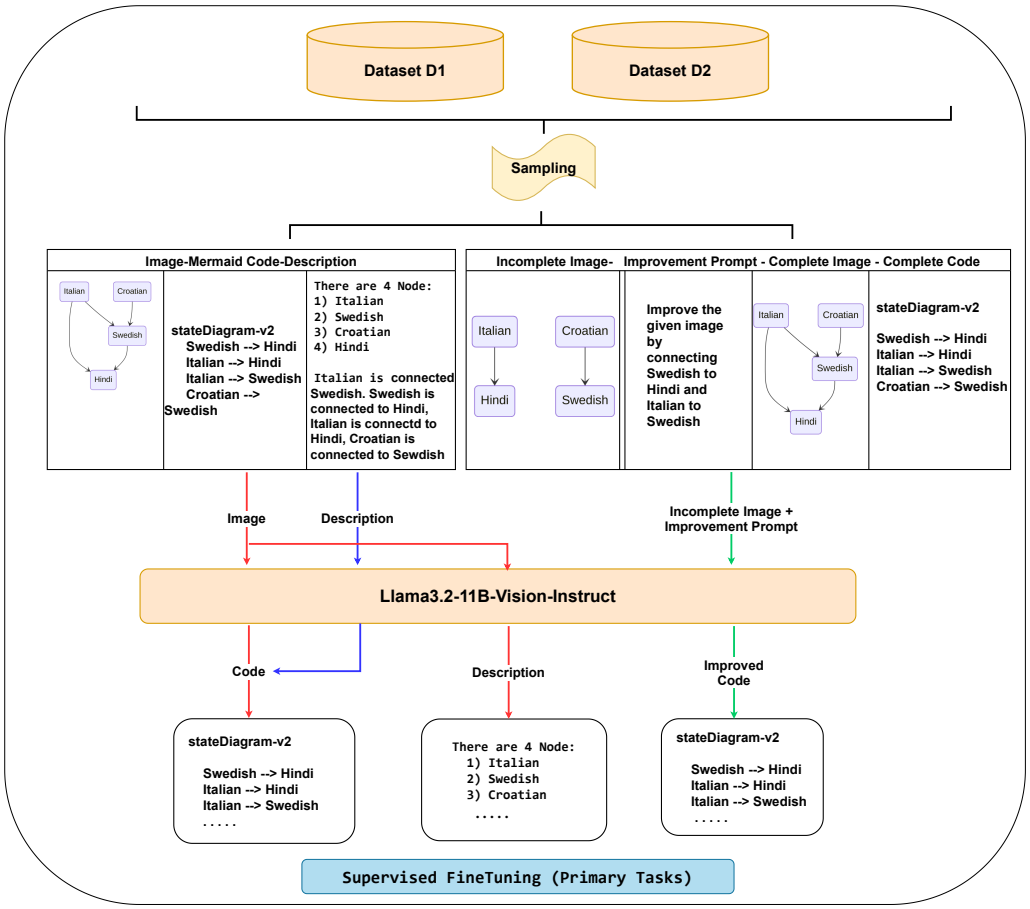

Figure 8: Diagram illustrating formulation of different Primary Tasks utilizing **TechING** for finetuning

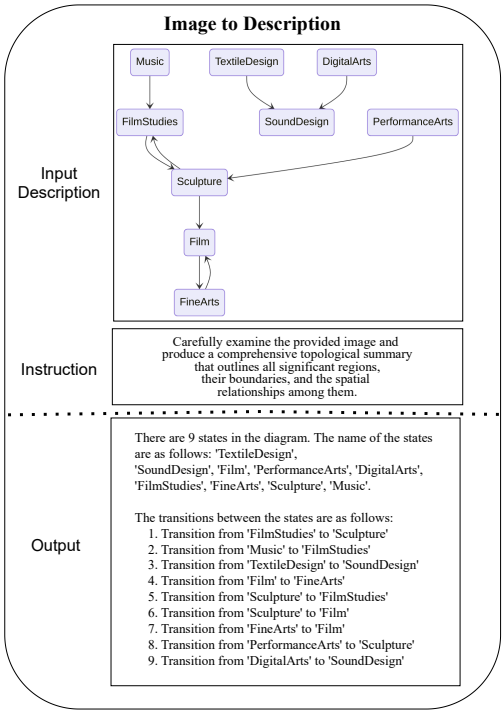

Figure 11: Image2Description Task Example

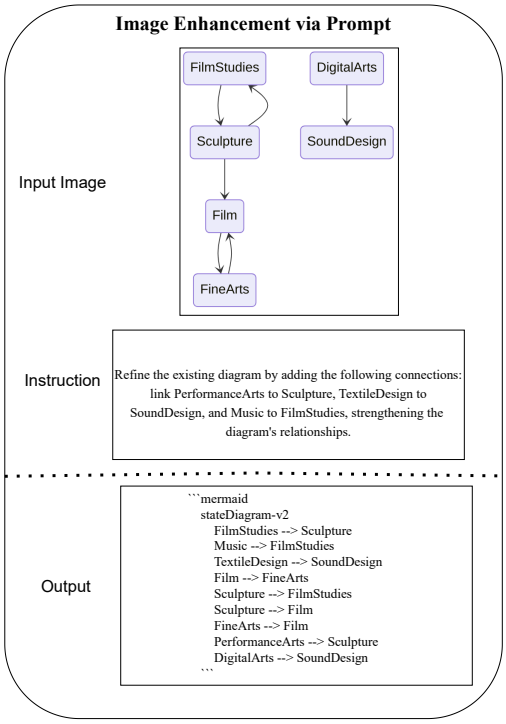

Figure 12: Example of Image Enhancement via Prompt

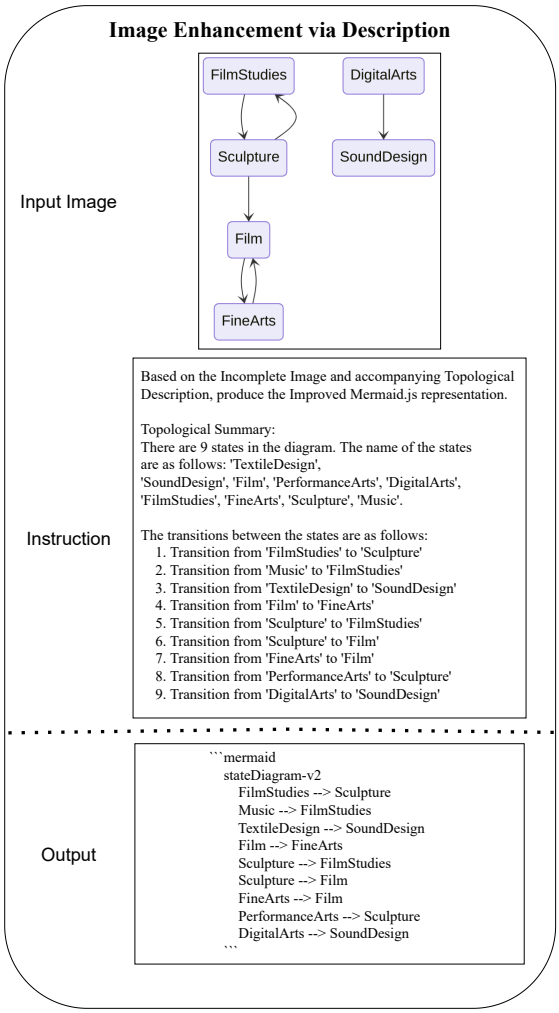

Figure 13: Example of Image Enhancement via Description

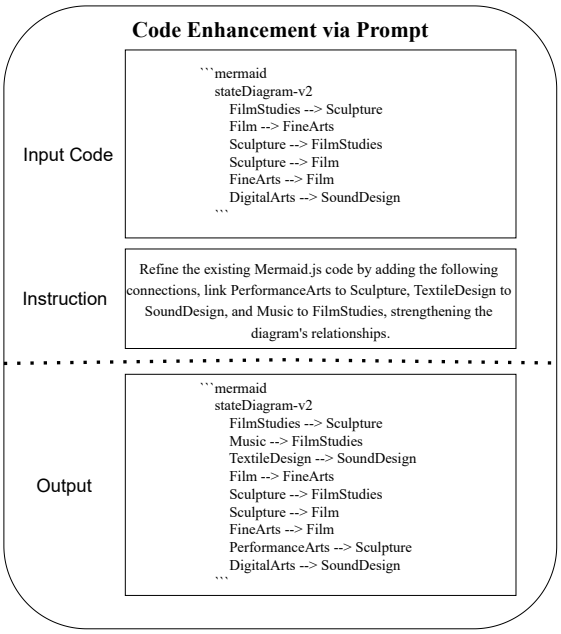

Figure 14: Example of Code Enhancement via Prompt

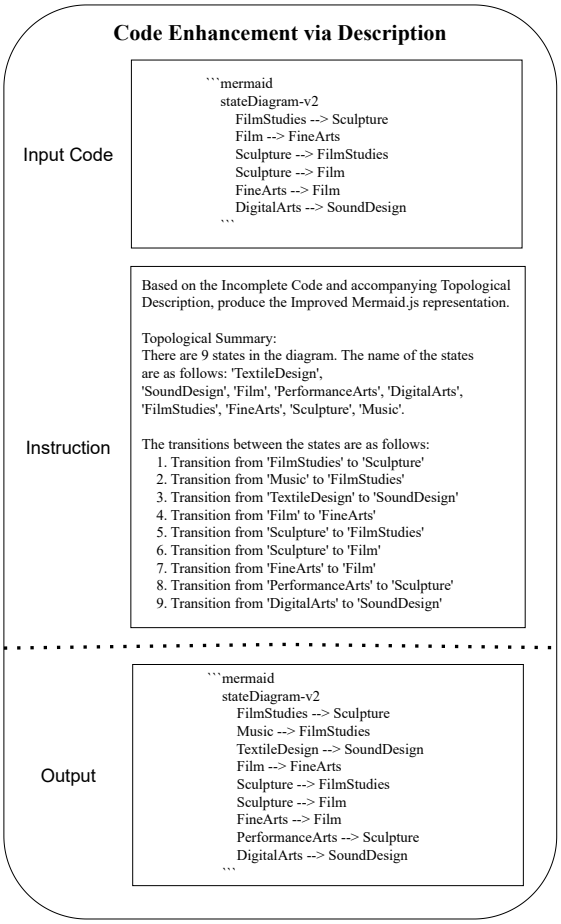

Figure 15: Example of Code Enhancement via Description

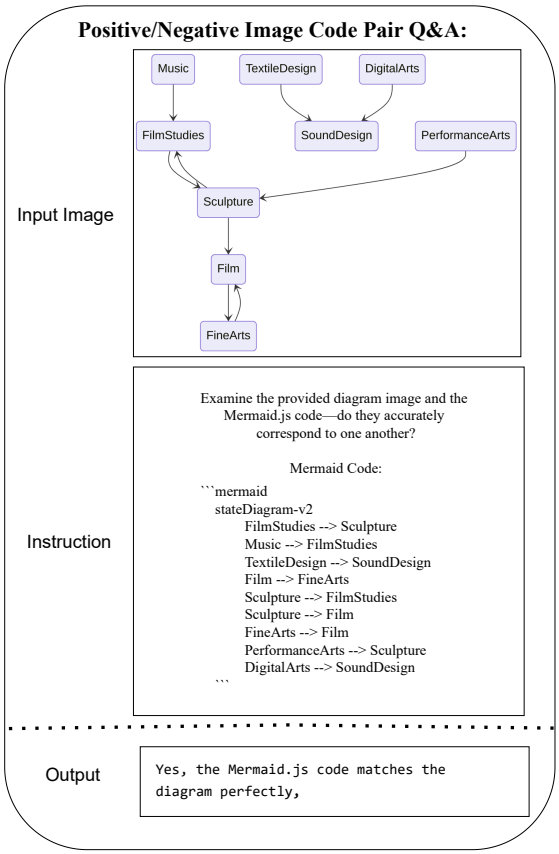

Figure 16: Example of Positive/Negative Image Code pair Q/A

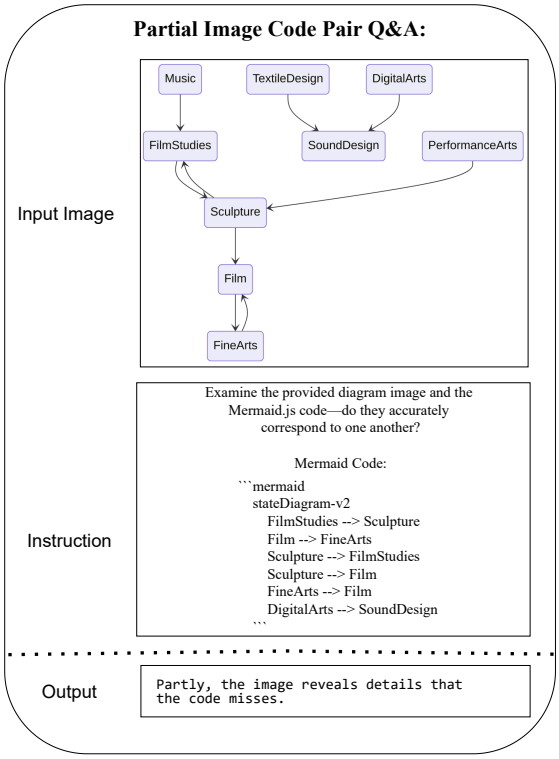

Figure 17: Example of Partial Image Code pair Q/A

F Baseline Models

We had used several baseline Multimodal LLMs which are of the similar parameter size of our base model Llama3.2-11B-Vision-Instruct (Meta, 2024). We experimented with MiniCPM-V-2-6(8B) (Yao et al., 2024), Qwen2.5-VL-7B-Instruct (Bai et al., 2023), Gemma3-12B-Instruction-Tuned (Aishwarya Kamath, 2025) and GPT-4o-mini (OpenAI, 2024b)

G Augmentations

Data augmentation is a strategy to artificially increase the diversity of the training dataset by applying transformations. The augmentation pipeline we employ plays a crucial role in improving the model's ability to generalize across real-world imaging conditions. By introducing a diverse set of transformations, including blur, noise, lighting changes, geometric distortions, and weather effects, the model is exposed to the kinds of variability commonly encountered in natural settings, such as those captured by camera sensors. For instance, motion blur and Gaussian noise simulate camera shake or low-light grain, while color jitter and brightness/contrast shifts mimic different lighting environments. Perspective transforms and random rotations help the model become invariant to viewpoint changes, making it robust to varied camera angles or object orientations. Similarly, occlusion-based augmentations like coarse or grid dropout encourage the model to learn from partial visual cues, improving resilience when parts of the image are obscured. Collectively, these augmentations ensure that the model doesn't overfit to clean, synthetic data but instead learns features that are stable and discriminative under real-world noise and viewpoint variations. For the augmentation procedure, we follow the steps shown in Algorithm 1. We use the Albumentations library `https://albumentations.ai/`. Fig. 18 illustrates some examples of augmentations used in the training process.

H Training and Hyperparameters Details

We performed supervised finetuning of Llama-3.2-11B-Vision-Instruct (Meta, 2024) using parameter-efficient finetuning method LoRA (Hu et al., 2021) using hyperparameters outlined in Table 5. Overall finetuning took 88 hours on 2 Nvidia A6000 GPUs.

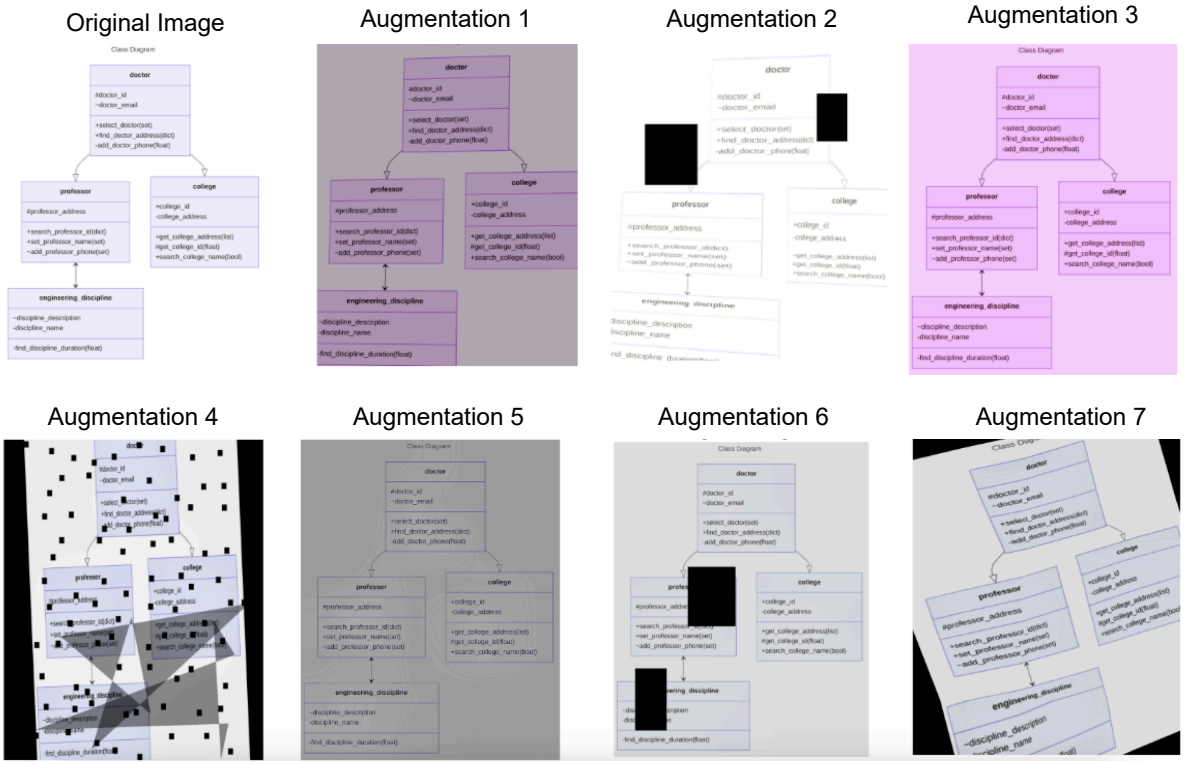

Figure 18: Examples of Augmentations used for keeping diversity and preventing overfitting.

Hyperparameters	Value
per_device_train_batch_size	1
gradient_accumulation_steps	1
learning_rate	2e-5
weight_decay	0.05
num_train_epochs	2
lr_scheduler_type	cosine
warmup_ratio	0.2
bf16	True
lora rank	32
lora_alpha	16
target_modules	QKV
lora_dropout	0.2
use_rslora	True

Table 5: Training Hyperparameters.

I Evaluation Metrics

We employ a combination of standard automatic language-based metrics and human evaluation metrics with custom structural heuristics to evaluate the performance of models across synthetic and real-world datasets as presented in Table 6.

Task	Metrics
Automatic Evaluation Image2Code (Synthetic) Description2Code (Synthetic) Image2Description (Synthetic) Image2code (Realworld)	BLEU SACREBLEU METEOR chrF BLEURT ROUGE-L
Human Evaluation Image2code (Realworld)	Compilation Error Correct Blocks Correct Edges Correct Labeled Edges Correct Attributes & Methods Correct Bits

Table 6: Evaluation metrics

I.1 Automatic Evaluation Metrics

For Image2Code, Description2Code, and Image2Description tasks on synthetic data, where the ground truth Mermaid code is clean and perfectly aligned with the diagram, we use established text similarity metrics i.e., BLEU, SACREBLEU, METEOR, chrF, BLEURT, and ROUGE-L. These metrics quantify lexical and semantic overlap between the generated and ground truth Code/Description, providing a baseline for textual accuracy.

Algorithm 1 Image Augmentation Pipeline

Input: Image I
Output: Augmented image I'
procedure AUGMENT(I)
    Select one (with probability $p = 0.7$):
        Motion Blur ($p = 0.5$)
        Median Blur (limit = 3, $p = 0.5$)
        Gaussian Blur (limit = [3, 5], $p = 0.5$)
    Select one (with probability $p = 0.9$):
        RGB Shift (limit = 80, $p = 0.8$)
        Hue/Saturation/Value Shift (limits: 15/25/20, $p = 0.8$)
    Apply Random Brightness/Contrast (brightness = 0.4, contrast = 0.2, $p = 0.7$)
    Select one (with probability $p = 0.3$):
        Rotate (limit = 30°, $p = 0.5$)
        Shift/Scale/Rotate (shift = 0.1, scale = 0.2, rotate = 45°, $p = 0.3$)
    Apply Grid Distortion ($p = 0.5$) with overall probability $p = 0.2$
    Apply Perspective Transform (scale = [0.01, 0.05], $p = 0.4$)
    Select one:
        Coarse Dropout (holes = 4, size = 16×16, $p = 0.3$)
        Grid Dropout (ratio = 0.2, $p = 0.1$)
    Select one (with probability $p = 0.4$):
        Random Rain ($p = 0.3$)
        Random Fog ($p = 0.3$)
        Random Shadow ($p = 0.8$)
    Optionally apply CLAHE ($p = 0.3$)
    Optionally apply Sharpen ($p = 0.3$)
    return Augmented image I'
end procedure

I.2 Human Evaluation Metrics

For the Image2Code task on Realworld Images Corpus **D3**, the evaluation focuses on both structural fidelity between the generated code and the original diagram. Correct Blocks/Correct Attributes and Methods/Correct Bits measures how accurately the model identifies and reproduces the diagram's components, depending on the diagram type. Correct Edges assesses the accuracy of the relationships or connections between these components. Correct Labeled Edges adds an additional layer by verifying whether the labels on the connections are also correctly generated. Finally, Compilation Error identifies whether the generated code is syntactically and semantically correct enough to compile by the Mermaid compiler without errors.

J Detailed Experimental Results

This section presents the complete set of experimental results in tabular form (Tables 7, 8, 9, and 10). These include performance metrics across all diagram types and models, complementing the summary findings discussed in the main paper. The tables provide a more granular view for in-depth analysis. Tables 7, 8, 9, and 10 present the detailed automatic evaluation results for the Image2Code, Description2Code, and Image2Description tasks on synthetic images, as well as the Image2Code task on real-world images, respectively.

Table 11 details the preliminary results obtained in a continual training setup where model trained sequentially on Image2Code, then Description2Code, and finally on a self-supervised image-enhancement task showed that performance on the previous primary tasks remained largely unchanged after training on next primary task. In the table, CTS1, CTS2, and CTS3 correspond to sequential stages of model training: CTS1 represents the model fine-tuned on the first primary task (Image2Code); CTS2 represents the model after subsequent fine-tuning on the Description2Code and Code2Description tasks; and CTS3 represents the model after final fine-tuning on the Image Enhancement task.

Diagram Type	Model	BLEU↑	SACREBLEU↑	METEOR↑	chrF↑	BLEURT↑	ROUGE-L↑
Block	**Llama3.2-11B-Vision-Instruct**	0.0059	1.9446	0.1823	27.9335	-0.6176	0.2278
	Llama-VL-TUG (ours)	**0.8676**	**86.7621**	**0.9432**	**94.6115**	**0.3255**	**0.8953**
	Gemma3-12B-Instruction-Tuned	0.0488	5.6815	0.2353	32.8237	-0.5744	0.2474
	Qwen2.5-VL-7B-Instruct	0.0148	3.0378	0.2211	24.8280	-0.6823	0.2275
	MiniCPM-V-2-6	0.0000	0.4034	0.0669	2.9787	-1.5357	0.0231
	GPT-4o-mini	0.0021	2.4243	0.1960	18.9216	-0.7492	0.2077
C4	**Llama3.2-11B-Vision-Instruct**	0.0525	5.5587	0.1569	37.4066	-0.4687	0.3580
	Llama-VL-TUG (ours)	**0.9585**	**95.8483**	**0.9845**	**98.9167**	**0.2863**	**0.9877**
	Gemma3-12B-Instruction-Tuned	0.0570	6.2295	0.2171	42.5174	-0.4602	0.4325
	Qwen2.5-VL-7B-Instruct	0.0657	6.9249	0.1978	39.0591	-0.5121	0.3557
	MiniCPM-V-2-6	0.0000	0.0322	0.0073	1.8335	-1.3834	0.0298
	GPT-4o-mini	0.1022	10.4744	0.2325	44.4395	-0.4793	0.4898
Class	**Llama3.2-11B-Vision-Instruct**	0.6317	63.1853	0.5364	75.3705	0.1994	0.5912
	Llama-VL-TUG (ours)	**0.9465**	**94.6490**	**0.9523**	**99.0225**	0.2890	**0.8181**
	Gemma3-12B-Instruction-Tuned	0.6328	63.2811	0.5355	75.4252	0.1044	0.6053
	Qwen2.5-VL-7B-Instruct	0.5923	59.2725	0.5419	73.4314	0.0562	0.5550
	MiniCPM-V-2-6	0.0001	0.2943	0.0604	6.2279	-1.2576	0.0460
	GPT-4o-mini	0.7975	79.7497	0.6845	89.2833	**0.3699**	0.6830
Flowchart	**Llama3.2-11B-Vision-Instruct**	0.0004	0.7182	0.1278	22.1123	-0.6942	0.2966
	Llama-VL-TUG (ours)	**0.9082**	**90.8177**	**0.9724**	**97.5900**	**0.2340**	**0.9864**
	Gemma3-12B-Instruction-Tuned	0.0000	1.5966	0.1806	22.3908	-0.6147	0.2900
	Qwen2.5-VL-7B-Instruct	0.0094	2.3684	0.1568	23.2609	-0.5725	0.2718
	MiniCPM-V-2-6	0.0000	0.5603	0.0482	6.8664	-1.1244	0.0517
	GPT-4o-mini	0.0003	1.0921	0.0704	13.3751	-1.0583	0.0437
Graph	**Llama3.2-11B-Vision-Instruct**	0.1080	14.7753	0.4323	62.3889	**0.2716**	0.6220
	Llama-VL-TUG (ours)	**0.6360**	**63.8309**	**0.7667**	**86.2771**	-0.0717	**0.7047**
	Gemma3-12B-Instruction-Tuned	0.1370	17.1570	0.4911	62.5730	0.0750	0.5705
	Qwen2.5-VL-7B-Instruct	0.0861	12.3727	0.4571	56.7718	-0.0477	0.5189
	MiniCPM-V-2-6	0.0000	2.4965	0.1444	7.8484	-1.2013	0.0544
	GPT-4o-mini	0.1672	18.5799	0.4880	57.8477	-0.0880	0.5753
Packet	**Llama3.2-11B-Vision-Instruct**	0.0298	3.2621	0.1502	24.7333	-0.8368	0.2480
	Llama-VL-TUG (ours)	**0.5190**	**51.8954**	**0.6775**	**68.7837**	**0.0244**	**0.5898**
	Gemma3-12B-Instruction-Tuned	0.0278	2.8232	0.0941	14.5225	-1.2830	0.1566
	Qwen2.5-VL-7B-Instruct	0.0450	4.8357	0.2034	29.4239	-0.7568	0.2976
	MiniCPM-V-2-6	0.0000	0.1241	0.0212	5.5846	-1.2374	0.0230
	GPT-4o-mini	0.0525	5.7991	0.2362	38.8445	-0.3666	0.3684
Sequence	**Llama3.2-11B-Vision-Instruct**	0.2053	21.3545	0.5646	65.3644	-0.1650	0.4853
	Llama-VL-TUG (ours)	**0.8236**	**82.3598**	**0.9555**	**96.2541**	0.0992	**0.9606**
	Gemma3-12B-Instruction-Tuned	0.2798	28.2498	0.6416	73.1667	**0.1889**	0.5860
	Qwen2.5-VL-7B-Instruct	0.3008	30.8903	0.6368	73.6891	0.1591	0.6002
	MiniCPM-V-2-6	0.0000	1.7125	0.1946	23.0416	-1.3419	0.0696
	GPT-4o-mini	0.1448	18.9303	0.5875	67.1868	-0.2169	0.4863
State	**Llama3.2-11B-Vision-Instruct**	0.4643	46.4642	0.4617	58.4349	-0.0451	0.4639
	Llama-VL-TUG (ours)	**0.8793**	**87.9324**	**0.7923**	**91.4462**	**0.4082**	**0.6738**
	Gemma3-12B-Instruction-Tuned	0.2088	21.3466	0.2781	40.4843	-0.3614	0.4356
	Qwen2.5-VL-7B-Instruct	0.2609	26.4496	0.3161	45.0595	-0.3306	0.4507
	MiniCPM-V-2-6	0.0003	0.3201	0.1123	3.5678	-1.3960	0.0091
	GPT-4o-mini	0.5100	51.0173	0.5167	61.4020	0.0565	0.5049

Table 7: Detailed results for Image2Code Generation (On Synthetic Eval Dataset)

Diagram Type	Model	BLEU↑	SACREBLEU↑	METEOR↑	chrF↑	BLEURT↑	ROUGE-L↑
Block	**Llama3.2-11B-Vision-Instruct**	0.0290	4.2783	0.2111	32.7885	-0.5828	0.2457
	Llama-VL-TUG (ours)	**0.8994**	**89.9374**	**0.9547**	**94.6979**	**0.7247**	**0.8440**
	Gemma3-12B-Instruction-Tuned	0.0663	6.7988	0.2753	34.8009	-0.5153	0.2684
	Qwen2.5-VL-7B-Instruct	0.0301	5.1130	0.2592	27.9851	-0.6851	0.2418
	MiniCPM-V-2-6	0.0168	3.5740	0.2355	22.2868	-0.7862	0.2055
	GPT-4o-mini	0.1080	10.9790	0.2684	30.7947	-0.5459	0.2426
C4	**Llama3.2-11B-Vision-Instruct**	0.0008	0.7302	0.0965	23.2782	-0.6560	0.2463
	Llama-VL-TUG (ours)	**0.9609**	**96.0880**	**0.9824**	**97.8606**	**0.4821**	**0.9711**
	Gemma3-12B-Instruction-Tuned	0.0093	0.9878	0.1259	23.6268	-0.6678	0.2728
	Qwen2.5-VL-7B-Instruct	0.0059	1.2289	0.1066	24.0475	-0.6474	0.2711
	MiniCPM-V-2-6	0.0129	1.9129	0.1401	21.1324	-0.7070	0.2614
	GPT-4o-mini	0.8229	82.2867	0.8841	88.4546	0.2825	0.8299
Class	**Llama3.2-11B-Vision-Instruct**	0.3238	32.3804	0.4820	60.4781	-0.3297	0.6389
	Llama-VL-TUG (ours)	**0.9656**	**96.5596**	**0.9889**	**98.5066**	**0.6291**	**0.9789**
	Gemma3-12B-Instruction-Tuned	0.3469	34.7061	0.3975	54.5017	-0.4732	0.5856
	Qwen2.5-VL-7B-Instruct	0.3491	34.9999	0.4559	61.5165	-0.3032	0.6486
	MiniCPM-V-2-6	0.2144	21.7403	0.3136	51.1385	-0.3994	0.5044
	GPT-4o-mini	0.9029	90.2938	0.9571	94.9272	0.5473	0.9547
Flowchart	**Llama3.2-11B-Vision-Instruct**	0.0014	1.2677	0.1303	22.7094	-0.5936	0.2582
	Llama-VL-TUG (ours)	**0.9656**	**96.5596**	**0.9889**	**98.5066**	**0.6291**	**0.9789**
	Gemma3-12B-Instruction-Tuned	0.0204	2.6764	0.1725	24.7587	-0.6932	0.2767
	Qwen2.5-VL-7B-Instruct	0.0067	2.0388	0.1526	22.7352	-0.5851	0.2788
	MiniCPM-V-2-6	0.0002	1.5939	0.1540	21.7691	-0.5673	0.2961
	GPT-4o-mini	0.1756	17.5553	0.4283	29.6297	-1.0116	0.1797
Graph	**Llama3.2-11B-Vision-Instruct**	0.1428	17.8969	0.5116	66.2555	0.1393	0.6608
	Llama-VL-TUG (ours)	0.8464	84.6445	0.8541	93.3428	0.7006	0.9403
	Gemma3-12B-Instruction-Tuned	0.1590	20.8206	0.5247	73.1211	0.2322	0.7391
	Qwen2.5-VL-7B-Instruct	0.0563	9.7717	0.4591	57.6757	-0.0484	0.5707
	MiniCPM-V-2-6	0.0115	5.3132	0.3373	50.4762	-0.3247	0.4465
	GPT-4o-mini	**0.9092**	**90.9247**	**0.9076**	**97.3794**	**0.9716**	**0.9990**
Packet	**Llama3.2-11B-Vision-Instruct**	0.0235	3.6337	0.1563	27.1367	-0.8113	0.2517
	Llama-VL-TUG (ours)	0.9580	95.8048	0.9902	97.0555	0.8043	0.9803
	Gemma3-12B-Instruction-Tuned	0.1243	12.7378	0.3533	45.9207	-0.3478	0.4548
	Qwen2.5-VL-7B-Instruct	0.0544	5.7997	0.2202	27.7765	-0.8272	0.2582
	MiniCPM-V-2-6	0.0556	5.8855	0.2378	34.0349	-0.5503	0.3326
	GPT-4o-mini	**1.0000**	**100.0000**	**1.0000**	**100.0000**	**0.8295**	**1.0000**
Sequence	**Llama3.2-11B-Vision-Instruct**	0.2793	28.1335	0.6721	74.8698	-0.3530	0.5829
	Llama-VL-TUG (ours)	**1.0000**	**100.0000**	**1.0000**	**100.0000**	**1.0068**	**1.0000**
	Gemma3-12B-Instruction-Tuned	0.4614	46.2925	0.7616	88.6449	0.2405	0.8791
	Qwen2.5-VL-7B-Instruct	0.3698	37.6440	0.6798	80.9233	0.1786	0.7054
	MiniCPM-V-2-6	0.2580	26.6537	0.5595	72.8661	-0.1069	0.5815
	GPT-4o-mini	0.8511	85.1115	0.8815	96.7221	0.8917	0.9997
State	**Llama3.2-11B**	0.2885	28.8777	0.2065	47.7941	-0.4619	0.4204
	Llama-VL-TUG (ours)	**0.9369**	**93.6941**	**0.9380**	**95.9338**	**0.7330**	**0.9699**
	Gemma3-12B-Instruction-Tuned	0.5186	51.8634	0.3189	79.6331	0.2222	0.8198
	Qwen2.5-VL-7B-Instruct	0.3800	38.0818	0.2994	58.7889	-0.2469	0.5391
	MiniCPM-V-2-6	0.2960	29.6741	0.2650	50.5644	-0.4541	0.4362
	GPT-4o-mini	0.9278	92.7769	0.9365	94.8294	0.7213	0.9582

Table 8: Detailed results for Description2Code Generation (On Synthetic Eval Dataset)

Diagram Type	Model	BLEU↑	SACREBLEU↑	METEOR↑	chrF↑	BLEURT↑	ROUGE-L↑
Block	**Llama3.2-11B-Vision-Instruct**	0.6919	69.1903	0.8296	82.1518	0.3238	0.7618
	Llama-VL-TUG (ours)	**0.8952**	**89.5213**	**0.9591**	**97.7836**	**0.7190**	**0.9561**
	Gemma3-12B-Instruction-Tuned	0.7717	77.1740	0.8765	88.7206	0.4529	0.8000
	Qwen2.5-VL-7B-Instruct	0.6644	66.4355	0.7662	80.0331	0.3056	0.7424
	MiniCPM-V-2-6	0.3366	33.6623	0.4848	45.9604	-0.8165	0.5542
	GPT-4o-mini	0.6916	69.1613	0.8112	83.1176	0.4294	0.7540
C4	**Llama3.2-11B-Vision-Instruct**	0.5537	55.3658	0.7376	73.4240	0.1010	0.6860
	Llama-VL-TUG (ours)	**0.9916**	**99.1557**	**0.9993**	**99.9638**	**0.7393**	**1.0000**
	Gemma3-12B-Instruction-Tuned	0.7599	75.9947	0.8790	87.2885	0.4770	0.8213
	Qwen2.5-VL-7B-Instruct	0.7079	70.7854	0.8340	84.7851	0.4182	0.8037
	MiniCPM-V-2-6	0.1583	15.8265	0.3620	36.0967	-0.4531	0.4934
	GPT-4o-mini	0.8198	81.9774	0.9085	90.8801	0.5639	0.8511
Class	**Llama3.2-11B-Vision-Instruct**	0.3256	32.5603	0.4865	50.6936	-0.2765	0.5287
	Llama-VL-TUG (ours)	**0.9746**	**97.4633**	**0.9809**	**98.0992**	**0.6531**	**0.9764**
	Gemma3-12B-Instruction-Tuned	0.6978	69.7829	0.6972	77.1858	0.2685	0.6276
	Qwen2.5-VL-7B-Instruct	0.7294	72.9380	0.7439	82.1311	0.3192	0.6571
	MiniCPM-V-2-6	0.3256	32.5603	0.4865	50.6936	-0.2765	0.5287
	GPT-4o-mini	0.8032	80.3173	0.8333	87.2117	0.3869	0.6836
Flowchart	**Llama3.2-11B-Vision-Instruct**	0.6598	65.9797	0.7943	82.5270	0.3285	0.8450
	Llama-VL-TUG (ours)	**0.9914**	**99.1419**	**0.9993**	**99.9588**	**0.8069**	**1.0000**
	Gemma3-12B-Instruction-Tuned	0.4695	46.9527	0.6673	73.1234	0.2282	0.9125
	Qwen2.5-VL-7B-Instruct	0.6823	68.2324	0.8161	84.3189	0.4034	0.8936
	MiniCPM-V-2-6	0.2970	29.7045	0.5195	47.7263	-0.9542	0.4732
	GPT-4o-mini	0.2728	27.2827	0.5799	50.0034	-0.9770	0.4993
Graph	**Llama3.2-11B-Vision-Instruct**	0.5314	53.1421	0.7467	73.3913	0.2576	0.6943
	Llama-VL-TUG (ours)	**0.8878**	**88.7774**	**0.9654**	**95.9720**	**0.7278**	**0.9436**
	Gemma3-12B-Instruction-Tuned	0.5901	59.0075	0.7873	77.8164	0.3555	0.7381
	Qwen2.5-VL-7B-Instruct	0.5296	52.9603	0.7175	73.5296	0.2960	0.7127
	MiniCPM-V-2-6	0.5314	53.1421	0.7467	73.3913	0.2576	0.6943
	GPT-4o-mini	0.5609	56.0883	0.7868	76.2206	0.2698	0.7030
Packet	**Llama3.2-11B-Vision-Instruct**	0.4044	40.4410	0.6921	74.9078	0.2912	0.5578
	Llama-VL-TUG (ours)	**0.6302**	**63.0240**	**0.7891**	**81.7360**	**0.4825**	**0.7232**
	Gemma3-12B-Instruction-Tuned	0.4553.	45.5292	0.6432	70.5237	0.2239	0.6365
	Qwen2.5-VL-7B-Instruct	0.3441	34.4130	0.6457	71.2078	0.1079	0.5171
	MiniCPM-V-2-6	0.4044	40.4410	0.6921	74.9078	0.2912	0.5578
	GPT-4o-mini	0.3828	38.2805	0.6982	75.0080	0.1336	0.5470
Sequence	**Llama3.2-11B-Vision-Instruct**	0.6893	68.9273	0.8780	84.3601	0.3644	0.7752
	Llama-VL-TUG (ours)	**0.8888**	**88.8756**	**0.9775**	**97.2891**	**0.6180**	**0.9603**
	Gemma3-12B-Instruction-Tuned	0.5279	52.7857	0.7633	79.9958	0.2875	0.7664
	Qwen2.5-VL-7B-Instruct	0.6505	65.0489	0.8374	82.4184	0.3669	0.7782
	MiniCPM-V-2-6	0.6893	68.9273	0.8780	84.3601	0.3644	0.7752
	GPT-4o-mini	0.7568	75.6781	0.9242	92.3463	0.5460	0.8144
State	**Llama3.2-11B-Vision-Instruct**	0.6220	62.1973	0.7418	75.4259	0.3177	0.6388
	Llama-VL-TUG (ours)	**0.8919**	**89.1866**	**0.9227**	**95.2614**	**0.6111**	**0.8635**
	Gemma3-12B-Instruction-Tuned	0.6292	62.9213	0.7348	75.3195	0.4053	0.6640
	Qwen2.5-VL-7B-Instruct	0.6082	60.8156	0.6896	76.0601	0.3481	0.6437
	MiniCPM-V-2-6	0.1269	12.6916	0.4379	28.6882	-0.8319	0.3387
	GPT-4o-mini	0.6634	66.3353	0.7468	79.3858	0.4126	0.6664

Table 9: Detailed results for Image2Description (On Synthetic Eval Dataset)

Diagram	Model	BLEU↑	SACREBLEU↑	METEOR↑	chrF↑	BLEURT↑	ROUGE-L↑
Block	**Llama3.2-11B-Vision-Instruct**	0.0211	4.0663	0.2446	28.1006	-0.8078	0.2623
	Gemma3-12B-Instruction-Tuned	0.0219	4.8489	0.2844	21.8190	-0.8729	0.2373
	Qwen2.5-VL-7B-Instruct	0.0200	4.8255	0.2931	22.1637	-0.8625	0.2505
	MiniCPM-V-2-6	0.0303	4.9937	0.3025	32.6082	-0.7513	0.1926
	GPT-4o-mini	0.0044	3.5151	0.2701	19.9548	-0.8435	0.2395
	Llama-VL-TUG (ours)	**0.4905**	**49.1096**	**0.7208**	**72.1252**	**-0.1484**	**0.6163**
	Llama3.2-w/o Self Supervision (Ablation)	0.4089	41.5985	0.6412	64.3320	-0.2587	0.5343
C4	**Llama3.2-11B-Vision-Instruct**	0.0261	3.4245	0.1522	31.3062	-0.7199	0.3490
	Gemma3-12B-Instruction-Tuned	0.0814	8.6327	0.2431	35.5166	-0.7242	0.3587
	Qwen2.5-VL-7B-Instruct	0.0338	4.3824	0.1820	31.1980	-0.7247	0.3519
	MiniCPM-V-2-6	0.0471	5.2783	0.2358	36.5629	-0.6841	0.2360
	GPT-4o-mini	0.0363	4.4410	0.1453	29.5557	**-0.7419**	0.3685
	Llama-VL-TUG (ours)	**0.4893**	**48.9480**	**0.6438**	65.7211	-0.2596	0.5371
	Llama3.2-w/o Self Supervision (Ablation)	0.4731	47.4111	0.6185	**66.3420**	-0.2626	**0.5412**
Class	**Llama3.2-11B-Vision-Instruct**	0.4109	41.4988	0.5741	68.3856	-0.2471	0.7046
	Gemma3-12B-Instruction-Tuned	0.3773	38.3370	0.5416	68.4976	-0.2581	0.6865
	Qwen2.5-VL-7B-Instruct	0.3402	34.3344	0.5597	66.7691	-0.3037	0.6625
	MiniCPM-V-2-6	0.0606	6.8319	0.2751	40.6963	-0.6234	0.2985
	GPT-4o-mini	**0.5313**	**53.2315**	**0.7255**	**80.8516**	**-0.0819**	**0.8068**
	Llama-VL-TUG (ours)	0.3378	34.1778	0.5708	62.9082	-0.3636	0.5375
	Llama3.2-w/o Self Supervision (Ablation)	0.3882	38.9145	0.6119	68.3929	-0.3260	0.5932
Flowchart	**Llama3.2-11B-Vision-Instruct**	0.0220	4.6773	0.2696	30.7875	-0.6497	0.4238
	Gemma3-12B-Instruction-Tuned	0.2980	29.8013	0.6134	46.4670	-0.1988	0.7449
	Qwen2.5-VL-7B-Instruct	0.1435	16.0567	0.4860	36.3134	-0.4691	0.5332
	MiniCPM-V-2-6	0.1602	16.4982	0.4953	36.2944	-0.4702	0.4470
	GPT-4o-mini	**0.3241**	**32.4087**	**0.6488**	**51.1517**	**-0.1293**	**0.7786**
	Llama-VL-TUG (ours)	0.0491	7.2717	0.3740	37.6756	-0.7727	0.3009
	Llama3.2-w/o Self Supervision (Ablation)	0.0475	7.2187	0.3743	37.8416	-0.7709	0.2990
Graph	**Llama3.2-11B-Vision-Instruct**	0.0027	3.4573	0.2706	37.3223	-0.6903	0.5086
	Gemma3-12B-Instruction-Tuned	**0.0330**	7.5894	0.3823	40.5827	-0.5487	0.5809
	Qwen2.5-VL-7B-Instruct	0.0276	**7.8834**	**0.4031**	39.7404	-0.5507	0.6109
	MiniCPM-V-2-6	0.0124	4.8452	0.3616	31.6561	-0.6496	0.3859
	GPT-4o-mini	0.0223	6.8887	0.3710	**40.6762**	**-0.5060**	**0.6607**
	Llama-VL-TUG (ours)	0.0047	3.8026	0.2654	34.4467	-0.6424	0.4685
	Llama3.2-w/o Self Supervision (Ablation)	0.0021	3.5804	0.2838	35.2435	-0.6092	0.4621
Packet	**Llama3.2-11B-Vision-Instruct**	0.0072	2.9514	0.2416	36.5142	-0.7099	0.3938
	Gemma3-12B-Instruction-Tuned	0.0137	3.5934	0.2763	38.3844	-0.6463	0.4247
	Qwen2.5-VL-7B-Instruct	0.0170	3.9062	0.2751	35.2018	-0.7856	0.3460
	MiniCPM-V-2-6	0.0113	2.3418	0.2312	29.6449	-0.7706	0.2127
	GPT-4o-mini	0.0071	3.4682	0.2533	39.6114	-0.6467	0.4199
	Llama-VL-TUG (ours)	**0.3479**	**34.7864**	**0.6074**	**60.0923**	**-0.3605**	**0.4838**
	Llama3.2-w/o Self Supervision (Ablation)	0.3256	32.5567	0.5945	58.5569	-0.3824	0.4566
Sequence	**Llama3.2-11B-Vision-Instruct**	0.2644	27.9428	0.5901	66.7506	-0.3793	0.6272
	Gemma3-12B-Instruction-Tuned	0.3429	35.1075	0.7068	76.7020	**-0.2194**	**0.7590**
	Qwen2.5-VL-7B-Instruct	0.2935	30.4405	0.6523	73.3296	-0.3141	0.6859
	MiniCPM-V-2-6	0.1913	20.5521	0.5234	59.4344	-0.4340	0.5117
	GPT-4o-mini	**0.4261**	**43.8398**	**0.7140**	**77.8157**	-0.2344	0.7504
	Llama-VL-TUG (ours)	0.1400	14.9873	0.4974	53.9343	-0.4537	0.5105
	Llama3.2-w/o Self Supervision (Ablation)	0.1275	14.1303	0.5016	53.7913	**-0.2194**	0.5196
State	**Llama3.2-11B-Vision-Instruct**	0.1187	14.5587	0.4231	50.3392	-0.5306	0.4635
	Gemma3-12B-Instruction-Tuned	0.2861	29.9928	0.6579	61.0823	-0.4059	0.5660
	Qwen2.5-VL-7B-Instruct	0.0922	12.3810	0.4776	46.7963	-0.5155	0.4990
	MiniCPM-V-2-6	0.0195	4.4052	0.3851	32.9893	-0.7931	0.2421
	GPT-4o-mini	**0.3910**	**39.4981**	**0.7483**	**71.9773**	**-0.2527**	**0.6688**
	Llama-VL-TUG (ours)	0.0510	6.5729	0.3668	34.2009	-0.9271	0.1671
	Llama3.2-w/o Self Supervision (Ablation)	0.0200	3.2555	0.2795	25.6581	-0.9875	0.1300

Table 10: Detailed results for Image2Code (On Real-world Corpus **D3**)

Diagram	Model	BLEU↑	SACREBLEU↑	CodeBLEU↑	METEOR↑	chrF↑	BLEURT↑	ROUGE-L↑
	Llama3.2-11B-Vision-Instruct	0.0649	6.7359	0.0649	0.1856	42.4385	-0.4850	0.4000
	Qwen2.5-VL-7B-Instruct	0.0297	3.6166	0.0297	0.1497	33.6653	-0.6413	0.3342
	MiniCPM-V-2-6	0.0000	0.0439	0.0000	0.0149	3.1508	-1.2448	0.0517
C4	GPT-4o-mini	0.1022	10.4743	0.1022	0.2325	44.4395	-0.4793	0.4897
	Llama3.2-11B-Vis-Inst-finetuned (CTS1)	0.9991	99.9065	0.9991	0.9997	99.9171	0.5515	0.9977
	Llama3.2-11B-Vis-Inst-finetuned (CTS1 + CTS2)	0.9994	99.9426	0.9994	0.9997	99.9373	**0.5535**	0.9988
	Llama3.2-11B-Vis-Inst-finetuned (CTS1 + CTS2 + CTS3)	**0.9996**	**99.9664**	**0.9996**	**0.9998**	**99.9646**	0.5534	**0.9990**
	Llama3.2-11B-Vision-Instruct	0.1614	16.1660	0.1614	0.3073	48.9088	-0.5158	0.3095
	Qwen2.5-VL-7B-Instruct	0.1620	16.2131	0.1620	0.3482	44.8345	-0.4934	0.3666
	MiniCPM-V-2-6	0.0006	0.4957	0.0006	0.0501	6.0572	-1.1414	0.0798
Flowchart	GPT-4o-mini	0.1632	16.3373	0.1632	0.3500	46.7228	-0.2477	0.3884
	Llama3.2-11B-Vis-Inst-finetuned (CTS1)	**0.8844**	**88.4441**	**0.8844**	**0.9451**	**94.3313**	**0.6049**	0.7234
	Llama3.2-11B-Vis-Inst-finetuned (CTS1 + CTS2)	0.8823	88.2370	0.8823	0.9429	94.1437	0.5934	0.7329
	Llama3.2-11B-Vis-Inst-finetuned (CTS1 + CTS2 + CTS3)	0.8824	88.2425	0.8824	0.9447	94.3292	0.5918	**0.7358**
	Llama3.2-11B-Vision-Instruct	0.5589	55.9227	0.5589	0.5546	66.3656	-0.0411	0.4999
	Qwen2.5-VL-7B-Instruct	0.2674	26.9351	0.2674	0.3504	41.8751	-0.5321	0.4237
	MiniCPM-V-2-6	0.0000	0.2293	0.0000	0.0484	2.5240	-1.4924	0.0053
State	GPT-4o-mini	0.5099	51.0172	0.5099	0.5167	61.4019	0.0564	0.5048
	Llama3.2-11B-Vis-Inst-Finetuned (CTS1)	**0.9416**	**94.1600**	**0.9416**	**0.8983**	**95.4549**	0.7688	**0.6952**
	Llama3.2-11B-Vis-Inst-Finetuned (CTS1 + CTS2)	0.9358	93.5819	0.9358	0.8939	95.1734	0.7651	0.6911
	Llama3.2-11B-Vis-Inst-Finetuned (CTS1 + CTS2 + CTS3)	0.9387	93.8797	0.9387	0.8951	95.2143	**0.7700**	0.6950
	Llama3.2-11B-Vision-Instruct	0.0068	2.5596	0.0068	0.2026	29.6349	-0.7379	0.2323
	Qwen2.5-VL-7B-Instruct	0.0138	3.0087	0.0138	0.2116	25.0251	-0.8202	0.2265
	MiniCPM-V-2-6	0.0000	0.5429	0.0000	0.0765	5.1334	-1.3276	0.0182
Block	GPT-4o-mini	0.0020	2.4243	0.0020	0.1959	18.9215	-0.7491	0.2076
	Llama3.2-11B-Vis-Inst-finetuned (CTS1)	**0.9473**	**94.7300**	**0.9473**	0.9885	**97.9700**	0.7667	**0.9157**
	Llama3.2-11B-Vis-Inst-finetuned (CTS1 + CTS2)	0.9460	94.6066	0.9460	**0.9863**	97.8143	0.7665	**0.9157**
	Llama3.2-11B-Vis-Inst-finetuned (CTS1 + CTS2 + CTS3)	0.9455	94.5507	0.9455	0.9882	97.8689	**0.7668**	0.9145
	Llama3.2-11B-Vision-Instruct	0.1852	19.4537	0.1852	0.5470	64.0124	-0.4667	0.4887
	Qwen2.5-VL-7B-Instruct	0.2851	29.0668	0.2851	0.6211	72.1925	-0.3264	0.6055
	MiniCPM-V-2-6	0.0000	1.1295	0.0000	0.0704	5.3291	-1.4662	0.0037
Sequence	GPT-4o-mini	0.1447	18.9302	0.1447	0.5875	67.1867	-0.2169	0.4862
	Llama3.2-11B-Vis-Inst-finetuned (CTS1)	**0.9974**	**99.7400**	**0.9974**	**0.9990**	**99.8600**	**1.0060**	**0.9981**
	Llama3.2-11B-Vis-Inst-finetuned (CTS1 + CTS2)	0.9945	99.4514	0.9945	0.9978	99.6812	1.0045	0.9959
	Llama3.2-11B-Vis-Inst-finetuned (CTS1 + CTS2 + CTS3)	0.9944	99.4404	0.9944	0.9976	99.6791	1.0034	0.9957
	Llama3.2-11B-Vision-Instruct	0.1312	16.2506	0.1312	0.4702	63.0973	-0.1654	0.6182
	Qwen2.5-VL-7B-Instruct	0.1078	13.6070	0.1078	0.4964	56.9693	-0.3456	0.5153
	MiniCPM-V-2-6	0.0000	2.0048	0.0000	0.1186	10.1976	-1.0992	0.0440
Graph	GPT-4o-mini	0.1672	18.5799	0.1672	0.4879	57.8477	-0.0880	0.5752
	Llama3.2-11B-Vis-Inst-finetuned (CTS1)	**0.8958**	**89.5800**	**0.8958**	**0.9206**	**94.4300**	**0.6287**	**0.7573**
	Llama3.2-11B-Vis-Inst-finetuned (CTS1 + CTS2)	0.8170	81.7045	0.8170	0.8202	91.5707	0.5667	0.7423
	Llama3.2-11B-Vis-Inst-finetuned (CTS1 + CTS2 + CTS3)	0.8184	81.8478	0.8184	0.8224	91.6472	0.5665	0.7433
	Llama3.2-11B-Vision-Instruct	0.3356	33.6372	0.3356	0.3351	60.1849	-0.2363	0.4422
	Qwen2.5-VL-7B-Instruct	0.8215	82.1546	0.8215	0.7081	88.6266	0.1243	0.6692
	MiniCPM-V-2-6	0.0000	0.0827	0.0000	0.0210	4.6216	-1.2969	0.0109
Class	GPT-4o-mini	0.7974	79.7497	0.7974	0.6844	89.2833	0.3698	0.6830
	Llama3.2-11B-Vis-Inst-finetuned (CTS1)	**0.9943**	**99.4300**	**0.9943**	0.9696	**99.8000**	0.6056	0.8574
	Llama3.2-11B-Vis-Inst-finetuned (CTS1 + CTS2)	0.9937	99.3746	0.9937	**0.9712**	99.7896	**0.6101**	**0.8653**
	Llama3.2-11B-Vis-Inst-finetuned (CTS1 + CTS2 + CTS3)	0.9935	99.3563	0.9935	0.9685	99.7913	0.6092	0.8518
	Llama3.2-11B-Vision-Instruct	0.0220	2.6485	0.0220	0.1539	26.2597	-0.6264	0.1512
	Qwen2.5-VL-7B-Instruct	0.0466	4.8793	0.0466	0.2385	35.6912	-0.6194	0.3911
	MiniCPM-V-2-6	0.0000	0.0651	0.0000	0.0150	6.6918	-1.1320	0.0144
Packet	GPT-4o-mini	0.0525	5.7991	0.0525	0.2361	38.8444	-0.3666	0.3684
	Llama3.2-11B-Vis-Inst-finetuned (CTS1)	0.5890	58.9013	0.5890	0.7892	80.2759	0.6375	0.9994
	Llama3.2-11B-Vis-Inst-finetuned (CTS1 + CTS2)	0.9405	94.0543	0.9405	**0.9616**	97.1664	**0.7972**	**0.9948**
	Llama3.2-11B-Vis-Inst-finetuned (CTS1 + CTS2 + CTS3)	**0.9432**	**94.3204**	**0.9432**	0.9535	**98.5849**	0.7397	0.9682

Table 11: Preliminary results for Image2Code Task in the Continual Training Setup explored in the initial stage of the project