OmniDiagram: Advancing Unified Diagram Code Generation via Visual Interrogation Reward

Haoyue Yang[1,*], Xuanle Zhao[1,*], Xuexin Liu[1,*], Feibang Jiang[2], Yao Zhu[3,†]
[1] Institute of Automation, Chinese Academy of Sciences
[2] University of Chinese Academy of Sciences [3] Zhejiang University
yanghaoyue2024@ia.ac.cn, ee_zhuy@zju.edu.cn

Abstract

The paradigm of programmable diagram generation is evolving rapidly, playing a crucial role in structured visualization. However, most existing studies are confined to a narrow range of task formulations and language support, constraining their applicability to diverse diagram types. In this work, we propose OmniDiagram, a unified framework that incorporates diverse diagram code languages and task definitions. To address the challenge of aligning code logic with visual fidelity in Reinforcement Learning (RL), we introduce a novel visual feedback strategy named Visual Interrogation Verifies All (VIVA). Unlike brittle syntax-based rules or pixel-level matching, VIVA rewards the visual structure of rendered diagrams through a generative approach. Specifically, VIVA actively generates targeted visual inquiries to scrutinize diagram visual fidelity and provides fine-grained feedback for optimization. This mechanism facilitates a self-evolving training process, effectively obviating the need for manually annotated ground truth code. Furthermore, we construct M3^{2}Diagram, the first large-scale diagram code generation dataset, containing over 196k high-quality instances. Experimental results confirm that the combination of SFT and our VIVA-based RL allows OmniDiagram to establish a new state-of-the-art (SOTA) across diagram code generation benchmarks. Code, datasets, and models are available at https://github.com/Haoyue-Yang/OmniDiagram.

1 Introduction

Multimodal code generation has rapidly evolved beyond text-centric methods to encompass the direct manipulation and editing of visual inputs. While Large Language Models (LLMs) have significantly advanced traditional code synthesis, the rise of Multimodal Large Language Models (MLLMs) enables the processing of unstructured diagrams in raster formats (e.g., PNG), (Liu et al., 2024; Yang et al., 2024). This shift addresses a critical real-world demand for a unified intelligence that can bridge complex visual information with executable code across diverse tasks and languages.

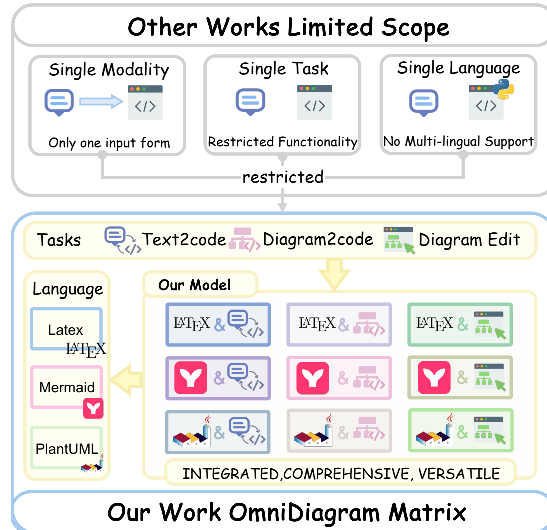

Figure 1: Overcoming the barriers of single-modality: The comprehensive landscape of OmniDiagram.

Despite meaningful explorations in diagram code generation, existing approaches generally suffer from limited versatility. They tend to be highly specialized, typically addressing singular tasks or supporting only a narrow set of programming languages. For instance, StarFlow (Bechard et al., 2025) focuses exclusively on JSON output, neglecting diverse diagrammatic languages. Similarly, while JanusCoder (Sun et al., 2025) attempts to broaden task integration across Text-to-Code and Diagram-to-Code scenarios, it relies solely on Supervised Fine-Tuning (SFT). This reliance constrains the model's capacity for visual alignment and robust code execution. Other works, such as MSRL (Chen et al., 2025a) and RLRF (Rodriguez et al., 2025), introduce RL with a multimodal reward system. However, they remain tailored for specific image-to-code tasks, lacking the flexibility

[*]Equal contribution.
[†]Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 16430–16452
July 2-7, 2026 ©2026 Association for Computational Linguistics

required for broader task modalities.

In this work, we focus on diagram code generation and propose OmniDiagram, a unified framework supporting diverse task modalities and code languages, as illustrated in Figure 1. To unify the verification of critical structural details across heterogeneous tasks and enhance the visual fidelity of the rendered diagrams, we introduce Visual Interrogation Verifies All (VIVA). Drawing from cognitive principles (Flavell, 1979), VIVA utilizes a question-driven verification strategy. Specifically, for each instance, the reward model formulates several critical questions conditioned on the instruction and input images (if available). Subsequently, the rollout code is executed and rendered into visual images, and the reward model assesses visual fidelity by answering questions conditioned on the generated images. This fine-grained, question-based feedback loop enables a continuous self-evolutionary process, allowing OmniDiagram to iteratively refine its performance across three core tasks.

To address the scarcity of diagram code generation datasets, we construct M3^{2}Diagram, the first large-scale multimodal corpus structured as a 3×3 task-language matrix. It covers LaTeX, Mermaid, and PlantUML across Diagram-to-Code, Diagram-Editing, and Text-to-Code tasks. All instances are synthesized via a top-down pipeline to ensure code quality and visual diversity. Following rigorous filtering and verification, we curate the final dataset. To ensure robust evaluation, we cluster and partition the data into a training set and an evaluation benchmark, designated as M3^{2}Bench. Additionally, every sample in M3^{2}Bench undergoes expert human verification to guarantee correctness and reliability. Experimental results on diagram code generation benchmarks, such as the Mermaid subset of VisPlotBench (Ni et al., 2025) and M3^{2}Bench, demonstrate the superiority of OmniDiagram, consistently surpassing both task-specific and open-source models. In summary, our contributions are as follows:

- We propose VIVA, a novel generative visual feedback mechanism that serves as an instance-specific reward signal for RL. Grounded in the philosophy that every sample deserves meticulous questioning, VIVA generates visual inquiries to evaluate the rendered execution of rollout code, thereby enhancing visual fidelity.
- We construct M3^{2}Diagram, the first large-scale omni-multimodal dataset with 196k samples, alongside M3^{2}Bench, a 1.7k verified benchmark for rigorous evaluation.
- We propose OmniDiagram, the first omni diagram code generation model. Experimental results demonstrate that OmniDiagram surpasses existing open-source models significantly.

2 Related Work

2.1 Multimodal Code Generation

Prior research has introduced various specialized multimodal code generation methods for specific domains, such as Web2Code (Yun et al., 2024) and ChartCoder (Zhao et al., 2025c). Recent MLLMs for code generation begin to adopt the Omni paradigm, unifying architectures for multiple code generation and editing (Sun et al., 2025; Zhao et al., 2025a). Specific tasks include Text-to-Code refinement (Rahman et al., 2025; Jain et al., 2025) and vector graphics synthesis (Yang et al., 2025a). Conversely, Diagram-to-Code research employs synthetic alignment and massive datasets to enhance structural robustness in domains like UML and flowcharts (Bates et al., 2025; He et al., 2025; Bechard et al., 2025; Chai et al., 2025; Singh et al., 2024). For instruction-driven editing, recent benchmarks and frameworks (Zhao et al., 2025b; Chen et al., 2025b) leverage rendering-aware RL to ensure execution fidelity.

2.2 Reinforcement Learning for Code

While Reinforcement Learning from Human Feedback (RLHF) has been pivotal for alignment, recent trends shift towards RLAIF (Lee et al., 2023) and environmental feedback to improve scalability. Following DeepSeek-R1 (Guo et al., 2025), applying RL to enhance multimodal code generation has become a widely adopted paradigm. Specifically, to optimize reasoning paths, Step-Text2Vis (Luo et al., 2025) employs Step-wise Direct Preference Optimization (Step-DPO) (Lai et al., 2024) for rationale refinement. In parallel, RLRF (Rodriguez et al., 2025) proposes utilizing visual feedback with Group Relative Policy Optimization (GRPO) (Shao et al., 2024) for SVG generation. Similarly, DaVinci (Xingchen et al.), MSRL (Chen et al., 2025a), and ChartMaster (Tan et al., 2025) leverage GRPO with multimodal rewards to align visual fidelity with code execution.

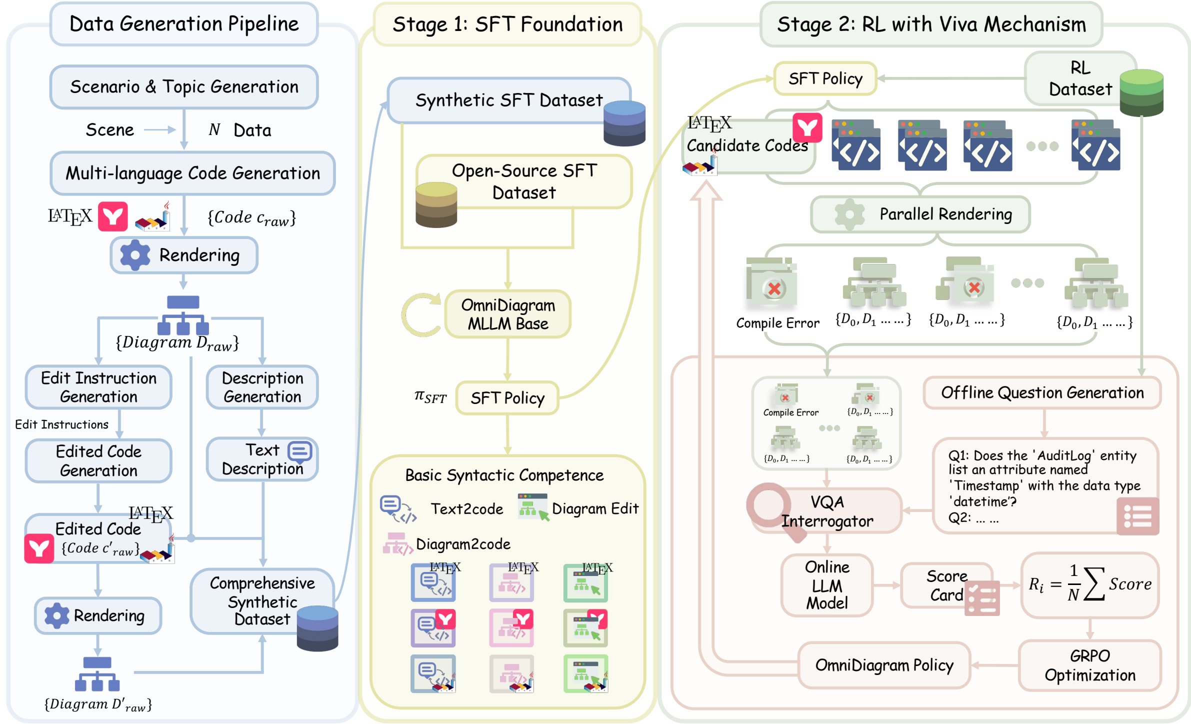

Figure 2: Overview of the OmniDiagram methodology. The framework illustrates the end-to-end flow from scalable data synthesis to model training, highlighting the two-stage optimization strategy combining an SFT foundation with the proposed VIVA-guided RL mechanism.

3 Method

The overall framework of OmniDiagram is illustrated in Figure 2. Our pipeline comprises three integral stages: (1) data generation (Section 3.2.1), (2) Supervised Fine-Tuning (SFT) to establish foundational syntactic competence (Section 3.3.1), and (3) evolutionary RL driven by the VIVA mechanism to align code with visual fidelity (Section 3.3.2).

3.1 Task Definition

The Omni framework is designed to address diverse code generation tasks. In this work, we specifically focus on diagrammatic code, encompassing LaTeX, Mermaid, and PlantUML. Let $\mathcal{I}$, $\mathcal{T}$, and $\mathcal{E}$ denote the visual diagram, textual description, and editing instruction, respectively. The model $\mathcal{M}_\theta$ maps an input subset $\mathcal{X}$ to the executable code $\mathcal{C}$:

$$\mathcal{C} = \mathcal{M}_\theta(\mathcal{X}), \quad \mathcal{X} \subseteq \{\mathcal{I}, \mathcal{T}, \mathcal{E}\} \tag{1}$$

This formulation encompasses three core tasks: *Diagram-to-Code* ($\mathcal{X} = \{\mathcal{I}\}$), *Diagram Editing* ($\mathcal{X} = \{\mathcal{I}, \mathcal{E}\}$) and *Text-to-Code* ($\mathcal{X} = \{\mathcal{T}\}$), . By unifying these mappings, OmniDiagram supports both initial diagram synthesis and iterative refinement within a single architecture.

3.2 Dataset Composition

To address the scarcity of large-scale and diverse diagrammatic code datasets, we construct a comprehensive dataset, **M3^{2}Diagram**, comprising 196k samples for SFT and RL training. Specifically, it combines 31k open-source samples and 165k high-quality samples synthesised by us. To ensure code correctness, we validate the alignment between code and images through rendering (tools are detailed in Appendix A.3) and subsequent visual filtering. Figure 3 shows that the dataset maintains a balanced distribution across the three languages.

3.2.1 Data Synthesis

To address the requirements of diagrammatic code generation tasks, we construct a 3×3 task-language matrix encompassing LaTeX, Mermaid, and PlantUML across three primary tasks: *Diagram-to-Code*, *Diagram-Editing*, and *Text-to-Code*. To ensure the training data accurately reflects real-world complexity, we implement diversity constraints from the initial generation phase, requiring each language to cover approximately 15 distinct diagram types with a particular emphasis on flowchart-related variants. The detailed classification of these types is provided in Table 4 of Appendix A.1.

Following (Yang et al., 2025b), we adopt a top-down, scenario-driven approach by sampling

topics and scenarios to ensure semantic diversity. Leveraging Gemini-2.5-Flash, our core pipeline follows the sequence: `topic` $\rightarrow$ `scenario` $\rightarrow$ `structured data` $\rightarrow$ `code-image pairs`. Building on the foundation of image-code pairs, the *Diagram-Editing* task synthesizes editing instructions and revised code outputs, while *Text-to-Code* derives corresponding textual descriptions. We employ a rigorous error-correction loop with execution feedback to ensure correctness. We employ a rigorous error-correction loop with execution feedback to ensure correctness. Additionally, rendered images are verified to eliminate non-standard outputs. This process filters an initial pool of 300k candidates into 165k high-quality, executable samples. The prompts are provided in Appendix A.2.

To ensure a balanced distribution of difficulty and topological complexity of SFT and RL training, we adopt a stratified clustering strategy based on perceptual hashing. For the 196k samples, we categorize them according to their visual-structural features and partition them into SFT and RL training sets at a fixed ratio. This ensures that each subset remains uniformly distributed across different diagram types and difficulty levels, thereby mitigating structural bias in any specific partition. Furthermore, we extend this strategy across diverse topics to establish M3^{2}Bench, a benchmark for omni-diagram code generation. Through rigorous filtering of diagram types and visual diversity, we curate 1.7k high-quality evaluation samples designed to comprehensively assess model diagram code generation capacity. Please refer to the Appendix A.4 for detailed data distribution statistics.

While recent findings from Qwen3-VL suggest that the thinking mode leads to performance degradation in multimodal code generation (Bai et al., 2025a), we seek to reinvestigate its potential specifically for diagram code generation. To this end, we distill forward reasoning trajectories from Gemini-2.5-Flash and reformulate the input queries and ground-truth code to construct 77k reasoning-enriched samples. These are combined with 196k direct-generation samples to explore the impact of reasoning on diagram generation capabilities.

3.2.2 Open-Source Data Curation

To enhance the model's generalization capabilities, we augment our synthetic corpus with 31k samples derived from open-source datasets. Specifically, we incorporate a subset from Cosyn-400k (Yang et al., 2025b) for the *Image-to-LaTeX* and *Image-to-Mermaid* tasks. To ensure data integrity, all code snippets are standardized and validated through a rendering pipeline. For the *Text-to-Mermaid* task, we integrate data from the Mermaid set of Viscoder2 (Ni et al., 2025), similarly verifying these text-code pairs before inclusion.

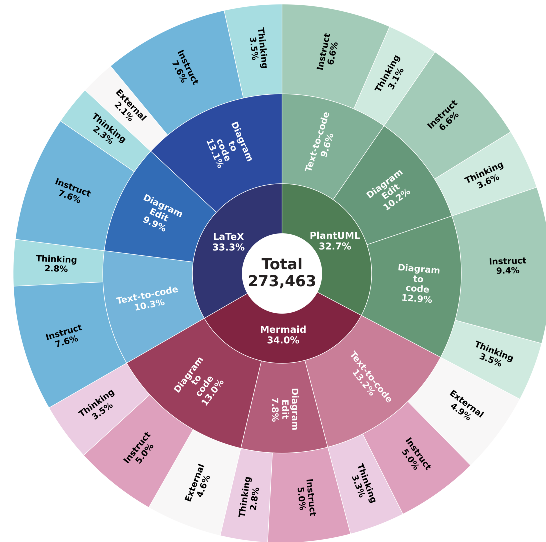

Figure 3: Breakdown of the 196k-sample M3^{2}Diagram dataset, supplemented by 77k reasoning-enriched samples, categorized across languages, tasks, and data sources.

3.3 Model Training

3.3.1 Supervised Finetuning (SFT)

Since general open-source VLMs exhibit limited proficiency in diagram code generation, we employ SFT as a preliminary stage. This establishes the fundamental diagram code generation capacity and guarantees valid candidates for subsequent refinement. The model is optimized via a standard next-token prediction objective,

$$\mathcal{L}(\theta) = -\mathbb{E}_{(x,y)}\left[\sum_{t=1}^{T} \log P(y_t \mid x, y_{<t}; \theta)\right], \qquad (2)$$

where (x, y) denotes the query and target response.

3.3.2 VIVA Reinforcement Learning (RL)

While previous diagram code generation works rely solely on SFT for optimization, we propose utilizing RL to further enhance the executability and visual fidelity of the generated code. However, utilizing Reinforcement Learning with Verifiable Rewards (RLVR) for code generation is challenging due to verification difficulties and the lack of visual fidelity guarantees. Current visual

feedback approaches typically fall into two categories: utilizing fixed prompt templates or computing global visual similarity. The former is heavily constrained by the evaluator model's capacity and is susceptible to prompt hacking. The latter, by over-relying on global similarity, often results in misaligned optimization objectives, biasing the model towards superficial structural resemblance at the expense of fine-grained detail accuracy. This issue is further compounded by the unique challenges of Omni tasks: the structural diversity of *Text-to-Code*, where multiple valid layouts preclude the use of a single reference image, and the non-bijective nature of *Diagram-to-Code*, where visually identical outputs can stem from distinct code snippets. Inspired by the question-driven analytical inspection mechanism employed by humans when performing complex, constructive, and reasoning-intensive tasks (Flavell, 1979), such as drawing, programming, and engineering review, we argue that effective evaluation should emulate how humans verify complex tasks—not through holistic similarity, but by systematically checking structural and semantic constraints through targeted questions.

Building upon this paradigm, we introduce VIVA, a unified reward mechanism driven by Visual Question Answering (VQA). Departing from passive scoring, VIVA assesses model outputs using instance-specific interrogative queries to evaluate the topological and semantic integrity of the rendered images. Specifically, for each RL training sample, we generate multiple visual questions derived from the input queries. A reward model then answers these questions based on the rendered images of the rollout code to evaluate its accuracy.

Also, to enhance the training efficiency and robustness, we decouple question generation from answer verification. We offline generate fine-grained questions per sample, specifically designed such that an answer of Yes corresponds to correct visual fidelity, as demonstrated in Appendix B. During training, a evaluation model interrogates each rollout candidate in an online manner. Notably, we incorporate intermediate scoring to reward partially correct outputs, yielding a smoother feedback signal that more accurately reflects the model's performance. A detailed theoretical analysis of this reward stability is provided in Section 3.4. This approach preserves the essential benefits of visual feedback while enabling task- and language-agnostic unification across the entire Omni diagram code generation framework. By emphasizing logical consistency aligned with user intent rather than strict global imitation, VIVA rewards more diverse rollouts during RL.

We employ GRPO for optimization (Shao et al., 2024). For each input x, we generate G candidate codes. These candidates are first rendered into images. Those that fail to compile are immediately assigned a reward of 0, while successfully rendered images are evaluated online by a VQA model against the pre-defined questions. Finally, candidates are ranked based on their VQA scores to compute the group-normalized advantage $\hat{A}_{i,t}$ for the i-th response:

$$\hat{A}_{i,t} = \frac{R_i - \text{mean}(\{R_j\}_{j=1}^{G})}{\text{std}(\{R_j\}_{j=1}^{G})}. \tag{3}$$

The policy model π_θ is optimized by maximizing the objective function,

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D},\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|x)} \Bigg[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\Big(\min\Big(r_{i,t}(\theta)\hat{A}_{i,t}, \text{clip}\left(r_{i,t}(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_{i,t}\Big)\Big)\Bigg], \tag{4}$$

where the probability ratio $r_{i,t}(\theta)$ is defined as:

$$r_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t} \mid x, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid x, o_{i,<t})}. \tag{5}$$

To provide holistic feedback, we employ a composite reward mechanism. The final reward R_i for the i-th candidate is defined as a weighted sum of VIVA and format rewards,

$$R_i = \alpha \cdot \underbrace{\left(\frac{1}{N}\sum_{k=1}^{N} S(\text{VQA}(o_i, q_k))\right)}_{R_{\text{VIVA}}} + (1-\alpha)\cdot R_{\text{fmt}} \tag{6}$$

The VIVA reward R_{VIVA} denotes the average VIVA score derived from N instance-specific questions. The format reward $R_{\text{fmt}} \in \{0, 1\}$ incentivizes strict adherence to the required code structures. Crucially, if the candidate code o_i fails to render, R_{VIVA} is set to 0 to penalize the non-execution rollouts.

3.4 Proof the Stability of VIVA Rewards

The effectiveness of policy gradient-based RL is profoundly influenced by the statistical variance of the reward signal. To provide theoretical insight into the stability of VIVA, we analyze its variance properties by modeling the total reward R_{acc} as the average of N graded QA scores $X_k \in [0, 1]$.

Model	Diagram-to-Code						Diagram Editing			Text-to-Code					
	M3²Bench			CoSyn$_{\text{Diagram}}$			M3²Bench			M3²Bench			VisPlot$_{\text{Mermaid}}$		
	Exec(%)	S_{vis}	S_{code}	Exec(%)	S_{vis}	S_{code}	Exec(%)	S_{pres}	S_{task}	Exec(%)	S_{vis}	S_{task}	Exec(%)	S_{vis}	S_{task}
Closed-Source Models															
GPT-5-mini	77.6	62.8	27.0	76.5	59.8	21.9	66.6	52.5	63.5	87.3	65.1	81.6	57.5	34.2	50.5
Gemini-3-Flash	82.4	73.6	31.7	83.4	72.2	37.1	86.5	77.8	82.0	91.4	72.4	87.7	90.1	58.4	80.2
Open-Source VLMs															
InternVL3-8B	65.9	35.1	17.1	69.7	35.9	12.6	46.1	22.6	34.3	42.6	24.4	34.8	45.0	19.1	28.2
InternVL3.5-8B	57.4	30.6	12.0	73.1	37.6	15.9	50.9	21.9	34.7	43.4	27.4	37.2	64.9	34.7	48.4
InternVL3.5-14B	74.8	40.9	21.7	81.8	42.1	20.2	52.3	23.2	37.6	44.2	25.6	36.8	63.4	35.2	49.0
Qwen3-VL-32B	79.8	58.0	30.0	84.5	61.1	40.3	60.1	45.6	51.8	68.5	49.1	64.5	67.2	40.4	55.1
InternVL3.5-38B	57.0	34.3	21.5	70.9	42.2	21.9	49.6	28.1	40.1	46.6	29.0	40.2	63.4	33.8	45.8
Qwen2.5-VL-72B	84.6	55.0	29.0	89.7	57.3	34.5	66.9	36.8	54.0	68.7	46.9	61.1	60.3	31.0	46.0
Qwen2.5-VL-3B	46.2	18.2	11.8	47.3	17.4	6.3	29.1	7.6	13.3	32.6	17.9	22.8	51.2	22.4	31.9
OmniDia-3B (SFT)	88.6	69.5	54.1	87.2	66.1	54.0	70.1	54.4	62.2	86.5	64.0	81.0	82.4	45.8	64.3
OmniDia-3B (RL)	93.0	72.2	51.6	93.7	67.1	57.5	**75.5**	**59.0**	64.8	**90.3**	70.8	**85.0**	**88.6**	49.4	64.5
Qwen2.5-VL-7B	62.0	32.7	23.0	62.3	28.3	22.7	41.2	17.3	26.2	46.1	28.1	35.1	57.3	25.6	37.3
OmniDia-7B (SFT)	92.9	74.1	**55.6**	89.5	69.9	50.9	70.1	54.3	62.1	88.7	70.6	83.5	84.0	45.3	62.6
OmniDia-7B (RL)	**94.3**	**75.5**	51.3	**96.0**	**74.5**	**60.6**	73.3	57.2	**65.2**	89.2	**71.8**	84.2	86.3	**51.0**	**66.9**

Table 1: Main results across three tasks. We compare models on our proposed **M3²Bench** and existing datasets, including the diagram set of **CoSyn** and the Mermaid set of **VisPlotBench**. All metrics are 0-100. Best results are bolded among open-source models.

Unlike conventional binary rewards (0 or 1) that follow a high-variance Bernoulli distribution, VIVA's graded scoring function allows for intermediate values, which mathematically ensures that the single-item variance satisfies $\text{Var}[X_k] \leq \mu_k(1-\mu_k)$.

Furthermore, for analytical tractability, we model the N instance-specific questions using an equicorrelation structure with correlation factor ρ. Under this assumption, the variance of the aggregated reward is given by:

$$\text{Var}[R_{\text{acc}}] = \frac{1}{N^2}\left(\sum_{k=1}^{N}\sigma_k^2 + 2\sum_{k<l}\rho\sqrt{\sigma_k^2\sigma_l^2}\right). \quad (7)$$

As derived in Appendix C, this multi-dimensional aggregation attenuates the impact of individual VQA uncertainty and provides a lower-variance advantage estimate A_i. Consequently, VIVA reduces the noise propagation to the policy gradient $\hat{g} = \nabla_\theta \log \pi_\theta(o_i \mid x)A_i$, thereby ensuring robust convergence and smoother optimization across diverse diagrammatic tasks.

4 Experiments

4.1 Implementation Details

We separate the M3²Diagram dataset into 8:1 as SFT and RL training sets. For SFT, we fine-tune Qwen2.5-VL-3B/7B-Instruct (Bai et al., 2025b) on the SFT dataset for two epochs on 8 H800 GPUs with a global batch size of 32. For RL, we apply GRPO to optimize the model with $G = 4$ and $\alpha = 0.9$. We first generate VIVA's queries offline via GPT-4.1-mini (OpenAI, 2025) and employing Qwen3-VL-32B (Bai et al., 2025a) as the reward model during training. This stage uses a global batch size of 128 across 8 GPUs for the policy. Our implementation leverages ms-swift (Zhao et al., 2025d) and EasyR1 (Sheng et al., 2024). More details are denoted in Appendix D.

4.2 Evaluation Settings

We evaluate the model on M3²Bench and other open-source test datasets, Cosyn-400k-Diagram eval set (Yang et al., 2025b) and VisplotBench-Mermaid (Ni et al., 2025). The M³Bench comprises 1.7k samples, evenly distributed across LaTeX, Mermaid, and PlantUML formats.

We compare our model against proprietary models, including GPT-5-mini and Gemini-3-flash (Google DeepMind, 2025), as well as a series of competitive open-source baselines such as the InternVL3 (Zhu et al., 2025) and InternVL3.5 (Wang et al., 2025) families (ranging from 8B to 38B), Qwen3-VL-32B (Bai et al., 2025a), and Qwen2.5-VL-72B (Bai et al., 2025b). To comprehensively assess generation quality, we employ a multi-dimensional evaluation framework leveraging GPT-4.1 as a judge to provide consistent and nuanced scoring. Specifically, for the Diagram-to-Code task, we decouple visual fidelity (S_{vis}) from code accuracy (S_{code}), with the latter assessed us-

Data	Model	Exec (%)	S_{vis}	S_{task}
M3²Bench	*Qwen2.5-Coder-3B*	66.4	40.0	46.3
	Viscoder2-3B	77.9 (↑11.5)	54.3 (↑14.3)	70.5 (↑24.2)
	Qwen2.5-VL-3B	32.6	17.9	22.8
	OmniDia-3B (SFT)	86.5 (↑53.9)	64.0 (↑46.1)	81.0 (↑58.2)
	OmniDia-3B (RL)	90.3 (↑57.7)	70.8 (↑52.9)	85.0 (↑62.2)
	Qwen2.5-Coder-7B	73.0	50.2	63.3
	Viscoder2-7B	83.6 (↑10.6)	63.0 (↑12.8)	78.8 (↑15.5)
	Qwen2.5-VL-7B	46.1	28.1	35.1
	OmniDia-7B (SFT)	88.7 (↑42.6)	70.6 (↑42.5)	83.5 (↑48.4)
	OmniDia-7B (RL)	89.2 (↑43.1)	71.8 (↑43.7)	84.2 (↑49.1)
VisPlotBench	*Qwen2.5-Coder-3B*	74.1	30.0	38.0
	Viscoder2-3B	76.3 (↑2.1)	43.0 (↑13)	59.0 (↑21)
	Qwen2.5-VL-3B	51.2	22.4	31.9
	OmniDia-3B (SFT)	82.4 (↑31.2)	45.8 (↑23.4)	64.3 (↑32.4)
	OmniDia-3B (RL)	88.6 (↑37.4)	49.4 (↑27.0)	64.5 (↑32.6)
	Qwen2.5-Coder-7B	77.9	39.0	53.0
	Viscoder2-7B	78.6 (↑0.7)	43.0 (↑4.0)	59.0 (↑6.0)
	Qwen2.5-VL-7B	57.3	25.6	37.3
	OmniDia-7B (SFT)	84.0 (↑26.7)	45.3 (↑19.7)	62.6 (↑25.3)
	OmniDia-7B (RL)	86.3 (↑29.0)	51.0 (↑25.4)	66.9 (↑29.6)

Table 2: Comparison on Text-to-Code tasks across M3²Bench and VisPlotBench. (↑x) (blue) denotes improvement of Viscoder2 over its Qwen-Coder baseline, while (↑x) (green) denotes OmniDia's improvement over its Qwen-VL baseline.

ing CrystalBLEU (Eghbali and Pradel, 2022). For the Diagram Editing task, we concurrently evaluate content preservation (S_{pres}) and instruction adherence (S_{task}). Regarding the Text-to-Code task, we measure visual correctness (S_{vis}) and task adherence (S_{task}), following the evaluation methodology defined in (Ni et al., 2025). Detailed scoring criteria and prompts are provided in Appendix E.2 and Appendix E.1, respectively.

4.3 Main Results

As shown in Table 1, OmniDiagram demonstrates leading performance in open-source logic diagram generation by consistently surpassing competitive baselines like the Qwen2.5-VL-72B across all tasks. Post-RL, OmniDiagram achieves performance comparable to, or in certain domains exceeding, proprietary models. These results demonstrate the efficacy of our unified framework in capturing the intricate structural and semantic nuances across diverse logic diagram formats. All the results validate the efficiency of our M3²Diagram dataset and the VIVA training strategy.

The comparison between the SFT baseline and the final RL-enhanced models further validates our proposed two-stage training paradigm. While the SFT stage provides a robust foundation for multi-format diagram generation, the subsequent stage with VIVA mechanism delivers substantial improvements. Specifically, the execution rate boosts significantly as the RL process penalizes non-renderable rollouts, effectively pruning invalid syntax. Concurrently, the VIVA-based visual feedback loop directly optimizes visual perceptual alignment, resulting in significant improvements in metrics such as S_{vis} and S_{fid}. As illustrated in Figure 4, the steady progression of the overall reward for both 3B and 7B models throughout the training process confirms the robustness of our RL strategy. This trend demonstrates that the framework effectively translates fine-grained visual feedback into superior generative performance.

We further compare the Text-to-Code performance of OmniDiagram against specialized code generation VLMs such as VisCoder2 (Ni et al., 2025). We only utilize the Mermaid set of M3²Bench and VisplotBench for fair comparison. This comparison is particularly challenging as VisCoder2 and its baselines are built upon dedicated code-centric LLM (Qwen2.5-Coder), whereas OmniDiagram originates from a general-purpose multimodal backbone (Qwen2.5-VL). As shown in Table 2, despite starting from a general-purpose backbone, OmniDiagram achieves greater performance gains than Viscoder2 relative to their respective baselines. For instance, at the 7B scale of M3²Bench, our model achieves a 43.1% leap in execution rate, far exceeding the 10.6% improvement of VisCoder2-7B. The results demonstrate the strong capability of the unified framework and VIVA mechanism in optimizing multimodal models for diagrammatic code synthesis.

Due to page limits, we provide a gallery of generated samples in Appendix F.1. We also conduct a systematic failure mode analysis in Appendix F.2 to investigate the model's limitations.

4.4 Ablation Studies

To verify the effectiveness of our design choices, we conduct three systematic ablation experiments focusing on the 3B model variant. All results are summarized in Table 3, representing the macro-average performance across all nine tasks.

Impact of Reasoning Trajectories. The integration of reasoning data introduces a task-specific performance bifurcation as shown in Table 3. Reasoning trajectories significantly bolster Diagram Editing performance, indicating that the reasoning process enhances the model instruction analy-

Configuration Strategy	Diagram-to-Code		Diagram Editing		Text-to-Code	
	Exec (%)	$\overline{S}_D$	Exec (%)	$\overline{S}_E$	Exec (%)	$\overline{S}_T$
Exp 1: Impact of Reasoning Trajectories						
SFT (w/o Reasoning Data)	88.6	61.8	70.1	58.3	86.5	72.5
SFT + RL (w/o Reasoning Data)	93.0	61.9	75.5	61.9	90.3	77.9
SFT (w/ Reasoning Data)	81.3	51.5	71.2	58.7	68.0	57.8
SFT + RL (w/ Reasoning Data)	89.3	55.4	77.6	62.6	88.1	76.0
Exp 2: Training Pipeline Strategy						
Pure RL (w/o SFT Stage)	30.2	19.6	28.7	8.9	34.1	25.4
Pure SFT (w/o RL Stage)	88.6	61.8	70.1	58.3	86.5	72.5
Full Pipeline	93.0	61.9	75.5	61.9	90.3	77.9
Exp 3: Reward Model Scale						
Reward by Qwen3-VL-32B	93.0	61.9	75.5	61.9	90.3	77.9
Reward by Qwen3-VL-30B-A3B	92.4	60.4	74.5	64.7	89.2	76.2

Table 3: Comprehensive ablation studies. We report the execution rate (**Exec (%)**) and mean scores ($\overline{\mathbf{S}}$) for each specific task category: Diagram-to-Code, Diagram Editing, and Text-to-Code.

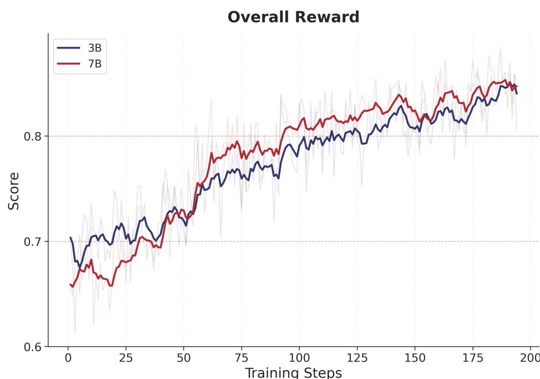

Figure 4: Progression of the overall reward during the RL training process for OmniDiagram.

sis capability. For other tasks, the extended context required for explicit reasoning may distract the model's attention from salient information, potentially leading to a degradation in performance. Detailed analysis is provided in Appendix G. This phenomenon could also be validated in Qwen3-VL (Bai et al., 2025a), where the performance of the instruct version surpasses the thinking version on multimodal code generation benchmarks.

Training Pipeline Strategy. We also conduct ablation studies on the training pipelines to validate the necessity of the SFT-to-RL strategy. As shown in Table 3, the RL-Only variant fails to converge effectively on omni-diagrammatic code tasks. Detailed analysis reveals that, in the absence of SFT, the model tends toward mode collapse, exclusively generating Mermaid code while ignoring the specific requirements provided in the instructions. This behavior results in negligible execution rates for the LaTeX and PlantUML test sets. We attribute this to the fact that models without SFT lack the capacity for rigorous instruction following. In contrast, our full pipeline (*w/ SFT + RL*) leverages SFT to establish foundational knowledge and RL to enforce logical discipline, achieving the best overall performance, which demonstrates that the SFT stage is essential for establishing a foundational omni-diagrammatic generation capability.

Robustness to Reward Model Scale. We evaluate the robustness of VIVA by employing Qwen3-VL-32B and Qwen3-VL-30B-A3B as reward models. As demonstrated in Table 3, the marginal performance discrepancy across these scales highlights the inherent robustness of our mechanism, suggesting it is relatively invariant to the capacity of the reward model. This suggests that the offline-generated visual questions, which provide essential visual grounding, are more pivotal to the optimization than the scale of the online feedback. By structuring the visual focus beforehand, VIVA lowers the discriminative burden on the reward model, enabling high-fidelity even with small verifiers.

5 Conclusion

In this work, we address the increasing demand for versatility in multimodal diagram code generation through three primary contributions. First, we introduce M3^{2}Diagram-196k, the first large-scale instruction-tuning dataset covering a 3×3 task-language matrix. Second, we propose VIVA, a

VQA-based reward mechanism built on the philosophy that "every sample deserves meticulous interrogation" via visual feedback. Third, we introduce OmniDiagram, an omni-model family that consistently surpasses competitive open-source baselines on M3^{2}Bench and various external diagrammatic benchmarks. We believe that both the M3^{2}Diagram dataset and the VIVA framework will significantly catalyze further developments in the field of unified multimodal code generation.

Limitation

From our perspective, our work has several limitations: (1) Reward Weighting: The weighting α between visual and format rewards is currently fixed. Exploring task-specific or dynamic adjustments could further optimize performance for complex diagram types. (2) Algorithmic Diversity: We primarily utilize GRPO for its efficiency. Comparing various RL paradigms like PPO or DPO within the VIVA framework would provide deeper insights into different optimization strategies. (3) Computational Overhead: The reliance on external models for data synthesis and evaluation involves significant resource investment. Future work could focus on more cost-effective methods to improve accessibility.

Ethic Statement

Our research utilizes publicly available open-source models and datasets, all of which are properly cited. By leveraging these widely recognized and vetted resources, we significantly mitigate the risk of generating harmful or toxic content. The diagram generation tasks focus on structured visualization and technical data, ensuring the outputs remain within professional and non-sensitive domains.

References

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, and 45 others. 2025a. Qwen3-vl technical report. *Preprint*, arXiv:2511.21631.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, and 1 others. 2025b. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Averi Bates, Ryan Vavricka, Shane Carleton, Ruosi Shao, and Chongle Pan. 2025. Unified modeling language code generation from diagram images using multimodal large language models. *Machine Learning with Applications*, page 100660.

Patrice Bechard, Chao Wang, Amirhossein Abaskohi, Juan Rodriguez, Christopher Pal, David Vazquez, Spandana Gella, Sai Rajeswar, and Perouz Taslakian. 2025. Starflow: Generating structured workflow outputs from sketch images. *arXiv preprint arXiv:2503.21889*.

Linzheng Chai, Jian Yang, Shukai Liu, Wei Zhang, Liran Wang, Ke Jin, Tao Sun, Congnan Liu, Chenchen Zhang, Hualei Zhu, and 1 others. 2025. Multilingual multimodal software developer for code generation. *arXiv preprint arXiv:2507.08719*.

Lei Chen, Xuanle Zhao, Zhixiong Zeng, Jing Huang, Liming Zheng, Yufeng Zhong, and Lin Ma. 2025a. Breaking the sft plateau: Multimodal structured reinforcement learning for chart-to-code generation. *arXiv preprint arXiv:2508.13587*.

Liangyu Chen, Yichen Xu, Jianzhe Ma, Yuqi Liu, Donglu Yang, Liang Zhang, Wenxuan Wang, and Qin Jin. 2025b. Charteditor: A reinforcement learning framework for robust chart editing. *arXiv preprint arXiv:2511.15266*.

Aryaz Eghbali and Michael Pradel. 2022. Crystalbleu: precisely and efficiently measuring the similarity of code. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–12.

John H Flavell. 1979. Metacognition and cognitive monitoring: A new area of cognitive–developmental inquiry. *American psychologist*, 34(10):906.

Google DeepMind. 2025. Gemini 3 flash. https://deepmind.google/models/gemini/flash/. Accessed: 2025-12-28.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Mengliang He, Jiayi Zeng, Yankai Jiang, Wei Zhang, Zeming Liu, Xiaoming Shi, and Aimin Zhou. 2025. Flow2code: Evaluating large language models for flowchart-based code generation capability. *arXiv preprint arXiv:2506.02073*.

Akriti Jain, Pritika Ramu, Aparna Garimella, and Apoorv Saxena. 2025. Doc2chart: Intent-driven zero-shot chart generation from documents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 34936–34951.

Xin Lai, Zhuotao Tian, Yukang Chen, Senqiao Yang, Xiangru Peng, and Jiaya Jia. 2024. Step-dpo: Step-wise preference optimization for long-chain reasoning of llms. *arXiv preprint arXiv:2406.18629*.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Kellie Ren Lu, Thomas Mesnard, Johan Ferret, Colton Bishop, Ethan Hall, Victor Carbune, and Abhinav Rastogi. 2023. Rlaif: Scaling reinforcement learning from human feedback with ai feedback.

Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, Li Zhang, Zhongqi Li, and Yuchi Ma. 2024. Exploring and evaluating hallucinations in llm-powered code generation. *arXiv preprint arXiv:2404.00971*.

Tianqi Luo, Chuhan Huang, Leixian Shen, Boyan Li, Shuyu Shen, Wei Zeng, Nan Tang, and Yuyu Luo. 2025. nvbench 2.0: Resolving ambiguity in text-to-visualization through stepwise reasoning. *arXiv preprint arXiv:2503.12880*.

Yuansheng Ni, Songcheng Cai, Xiangchao Chen, Jiarong Liang, Zhiheng Lyu, Jiaqi Deng, Kai Zou, Ping Nie, Fei Yuan, Xiang Yue, and 1 others. 2025. Viscoder2: Building multi-language visualization coding agents. *arXiv preprint arXiv:2510.23642*.

OpenAI. 2025. Introducing GPT-4.1 in the API. https://openai.com/index/gpt-4-1/. Accessed: 2025-12-28.

Mizanur Rahman, Md Tahmid Rahman Laskar, Shafiq Joty, and Enamul Hoque. 2025. Text2vis: A challenging and diverse benchmark for generating multimodal visualizations from text. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 31837–31862.

Juan A Rodriguez, Haotian Zhang, Abhay Puri, Aarash Feizi, Rishav Pramanik, Pascal Wichmann, Arnab Mondal, Mohammad Reza Samsami, Rabiul Awal, Perouz Taslakian, and 1 others. 2025. Rendering-aware reinforcement learning for vector graphics generation. *arXiv preprint arXiv:2505.20793*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

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. *arXiv preprint arXiv:2406.19237*.

Qiushi Sun, Jingyang Gong, Yang Liu, Qiaosheng Chen, Lei Li, Kai Chen, Qipeng Guo, Ben Kao, and Fei Yuan. 2025. Januscoder: Towards a foundational visual-programmatic interface for code intelligence. *arXiv preprint arXiv:2510.23538*.

Wentao Tan, Qiong Cao, Chao Xue, Yibing Zhan, Changxing Ding, and Xiaodong He. 2025. Chartmaster: Advancing chart-to-code generation with real-world charts and chart similarity reinforcement learning. *arXiv preprint arXiv:2508.17608*.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, and 1 others. 2025. Internvl3. 5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *arXiv preprint arXiv:2508.18265*.

ZENG Xingchen, Zhewei Su, Hengming Zhang, Juyong Jiang, Jiazhi Xia, and Wei Zeng. Davinci: Reinforcing visual-structural syntax in mllms for generalized scientific diagram parsing. In *The Fourteenth International Conference on Learning Representations*.

Cheng Yang, Chufan Shi, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran Xu, Xinyu Zhu, Siheng Li, Yuxiang Zhang, and 1 others. 2024. Chartmimic: Evaluating lmm's cross-modal reasoning capability via chart-to-code generation. *arXiv preprint arXiv:2406.09961*.

Yiying Yang, Wei Cheng, Sijin Chen, Xianfang Zeng, Fukun Yin, Jiaxu Zhang, Liao Wang, Gang Yu, Xingjun Ma, and Yu-Gang Jiang. 2025a. Omnisvg: A unified scalable vector graphics generation model. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Yue Yang, Ajay Patel, Matt Deitke, Tanmay Gupta, Luca Weihs, Andrew Head, Mark Yatskar, Chris Callison-Burch, Ranjay Krishna, Aniruddha Kembhavi, and 1 others. 2025b. Scaling text-rich image understanding via code-guided synthetic multimodal data generation. *arXiv preprint arXiv:2502.14846*.

Sukmin Yun, Rusiru Thushara, Mohammad Bhat, Yongxin Wang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, Haonan Li, Preslav Nakov, and 1 others. 2024. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *Advances in neural information processing systems*, 37:112134–112157.

Xuanle Zhao, Deyang Jiang, Zhixiong Zeng, Lei Chen, Haibo Qiu, Jing Huang, Yufeng Zhong, Liming Zheng, Yilin Cao, and Lin Ma. 2025a. Vincicoder: Unifying multimodal code generation via coarse-to-fine visual reinforcement learning. *arXiv preprint arXiv:2511.00391*.

Xuanle Zhao, Xuexin Liu, Haoyue Yang, Xianzhen Luo, Fanhu Zeng, Jianling Li, Qi Shi, and Chi Chen. 2025b. Chartedit: How far are mllms from automating chart analysis? evaluating mllms' capability via chart editing. *arXiv preprint arXiv:2505.11935*.

Xuanle Zhao, Xianzhen Luo, Qi Shi, Chi Chen, Shuo Wang, Zhiyuan Liu, and Maosong Sun. 2025c. Chartcoder: Advancing multimodal large language model for chart-to-code generation. *arXiv preprint arXiv:2501.06598*.

Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, and 1 others. 2025d. Swift: a scalable lightweight infrastructure for fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 29733–29735.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, and 1 others. 2025. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*.

A Dataset Details

A.1 Diagram Type Statistics

In this section, we present a quantitative breakdown of the diagram types generated during the Data Synthesis phase, detailing the proportional distribution of each category across the three supported syntaxes. As illustrated in Table 4, the dataset encompasses a diverse taxonomy: 12 types for LaTeX, 13 for Mermaid, and 20 for PlantUML. This extensive coverage is predominantly centered around flowcharts and logic-driven diagrams, ensuring the dataset's robustness and applicability to complex real-world scenarios.

A.2 Prompt Used for Data Generation

In this section, we provide the detailed prompt templates employed in our top-down data generation pipeline, taking the *Diagram-to-Mermaid-Mindmap* task as a representative example. To foster reproducibility and transparency, we present the prompts sequentially across the four key stages of our synthesis workflow: (1) Topic generation Figure 10, (2) Scenario design Figure 11, (3) structured Data synthesis (e.g., node and edge definitions) Figure 12, and (4) the final Code implementation Figure 13. Collectively, these prompts ensure both the semantic diversity and syntactic correctness of the OmniDiagram dataset.

A.3 Rendering Tools and Pipelines

To ensure high-fidelity visual outputs, we employ standardized rendering pipelines for each diagrammatic language. For **LaTeX**, we utilize the TeX Live distribution, compiling code via `pdflatex` and converting the output to high-resolution images using `Poppler` utilities. For **Mermaid**, we adopt the official `mermaid-cli` based on a headless Chromium browser to guarantee browser-consistent rendering. Finally, for **PlantUML**, we use the standard Java engine integrated with **Graphviz** to accurately compute complex node layouts and generate anti-aliased diagrams.

A.4 M3^{2}Bench Test Set Distribution

The M3^{2}Bench test set comprises a total of 17k high-quality samples, ensuring a rigorous evaluation of model performance across diverse diagramming languages and tasks. Specifically, the test set is distributed across three primary formats: LaTeX with 593 samples, Mermaid with 524 samples, and PlantUML with 582 samples. Each language category is further divided into three functional tasks to assess versatility: Diagram-to-Code (including 337 LaTeX, 280 Mermaid, and 280 PlantUML samples), Diagram Edit (comprising 128 LaTeX, 122 Mermaid, and 151 PlantUML samples), and Text-to-Diagram (consisting of 128 LaTeX, 122 Mermaid, and 151 PlantUML samples). As illustrated in Figure 5.

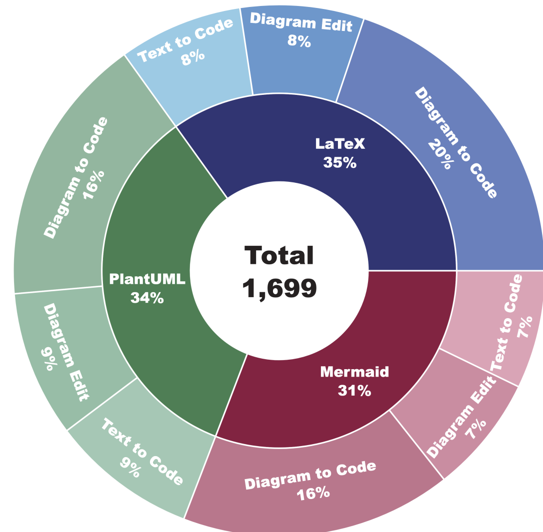

Figure 5: Statistical breakdown of tasks and diagram formats in M3^{2}Bench

B Visual Verification Showcase

To further illustrate the operational details of the VIVA reward mechanism, we present a series of qualitative examples across our three primary tasks. As discussed in the main text, VIVA deviates from traditional pixel-wise comparison by employing instance-specific "interrogative" probes. For every sample in the evaluation set, we generate ten fine-grained questions that scrutinize the rendered output from multiple dimensions: **topological structure** (connection logic), **semantic consistency** (textual accuracy), and **aesthetic attributes** (style, color, and shapes).

Figures 7, 8, and 9 demonstrate the diversity and specificity of these verification probes:

In the **Text-to-Code** task, Figure 7, the questions are designed to verify if the model has successfully translated complex textual descriptions into precise visual logic, such as identifying the correct shape for decision nodes or the specific color of a thick green outline. For the **Diagram-to-Code** task, Figure 8, the probes focus on structural fidelity, ensuring that hierarchical dependencies (e.g., the position of 'Web Servers' relative to the 'Load Bal-

LaTeX		Mermaid		PlantUML	
Diagram Type	**Ratio (%)**	**Diagram Type**	**Ratio (%)**	**Diagram Type**	**Ratio (%)**
Graph (Nodes & Edges)	28.6	Flowchart	12.9	Activity Diagram	45.9
Flowchart	13.8	Class Diagram	12.3	Sequence Diagram	23.1
Block Diagram	13.7	Kanban Board	10.6	Use Case Diagram	13.5
Timeline	10.0	Timeline	9.6	State Diagram	6.6
Finite State Machine	7.5	Entity Relationship	9.4	Deployment Diagram	3.5
Tree Diagram	6.5	Sequence Diagram	8.3	Regex Diagram	2.7
Unknown / Generic	5.7	Packet Diagram	7.7	Class Diagram	2.1
Chemical Structure	4.7	User Journey	7.3	Component Diagram	1.2
Free Body Diagram	3.2	Block Diagram	6.6	EBNF Diagram	0.9
Circuit Diagram	3.1	C4 Architecture	5.7	Network Diagram	0.4
Vector Graphics	1.6	GitGraph	5.0	*Others***	0.1
Others (Plots, Tables, etc.)	1.6	*Others**	4.6		

**Others for Mermaid include Mindmaps, Gantt, State, and BPMN diagrams.*

***Others for PlantUML include Entity-Relationship, SDL, Timing, Wireframe, and Ditaa diagrams.*

Table 4: Distribution of diagram types within the dataset. The table presents the percentage breakdown of each schema for LaTeX, Mermaid, and PlantUML. This demonstrates the dataset's coverage of diverse structural complexities ranging from scientific plots to software engineering diagrams.

ancer') and connection labels like 'API Calls' are preserved during the code generation and rendering process.

Finally, in the **Diagram Editing** task, Figure 9, VIVA serves as a safeguard for instruction following. It checks whether requested modifications—such as rounding rectangle corners or applying a soft blue fill—have been applied consistently while maintaining the integrity of the original graph's connectivity. By standardizing the ground-truth response to "Yes" for all probes, VIVA provides a task-agnostic scoring metric that facilitates stable Reinforcement Learning. Furthermore, the use of intermediate scoring ensures that models receiving partial credit for correctly rendered subcomponents are appropriately rewarded, leading to a smoother and more informative feedback signal.

C Detailed Theoretical Proof of Reward Stability

In policy gradient-based RL, the stability of gradient estimates largely depends on the statistical variance of the reward signal. For generative tasks with complex semantic and structural constraints, highly noisy rewards can significantly affect training stability and convergence. VIVA RL employs a graded reward design based on multiple QA constraints. This section analyzes the variance properties of this reward mechanism and its implications for policy optimization stability.

Modeling Graded QA Rewards as Random Variables: For a given input x and its candidate generation o_i, we construct N QA questions $\{q_k\}_{k=1}^N$ targeting topological and semantic constraints. The score of the k-th QA is defined as a random variable: $X_k \triangleq S(\mathrm{VQA}(o_i, q_k)) \in [0, 1]$, where $S(\cdot)$ is a graded scoring function that allows intermediate values (e.g., 0.5) to represent partial semantic satisfaction. The randomness mainly arises from visual rendering errors, VQA inference uncertainty, and ambiguity in semantic judgment. Moreover, we allow each QA to have its own mean and variance: $\mathbb{E}[X_k] = \mu_k, \mathrm{Var}[X_k] = \sigma_k^2$.

To capture the correlation induced by shared underlying semantic structure, we assume an equicorrelation structure for analytical convenience:

$$\mathrm{Cov}[X_k, X_l] = \rho\sqrt{\sigma_k^2\sigma_l^2}, \quad k \neq l, \quad \rho \in [0, 1), \tag{8}$$

where ρ measures the degree of dependency between QA scores.

Variance of Averaged Graded QA Rewards: The accuracy reward in VIVA is defined as the average of multiple QA scores: $R_{\mathrm{acc}} = \frac{1}{N}\sum_{k=1}^N X_k$. Using the variance–covariance decomposition, we first write

$$\mathrm{Var}[R_{\mathrm{acc}}] = \mathrm{Var}\left[\frac{1}{N}\sum_{k=1}^{N} X_k\right] = \frac{1}{N^2}\mathrm{Var}\left[\sum_{k=1}^{N} X_k\right]. \tag{9}$$

Next, applying the formula for the variance of a sum of correlated variables:

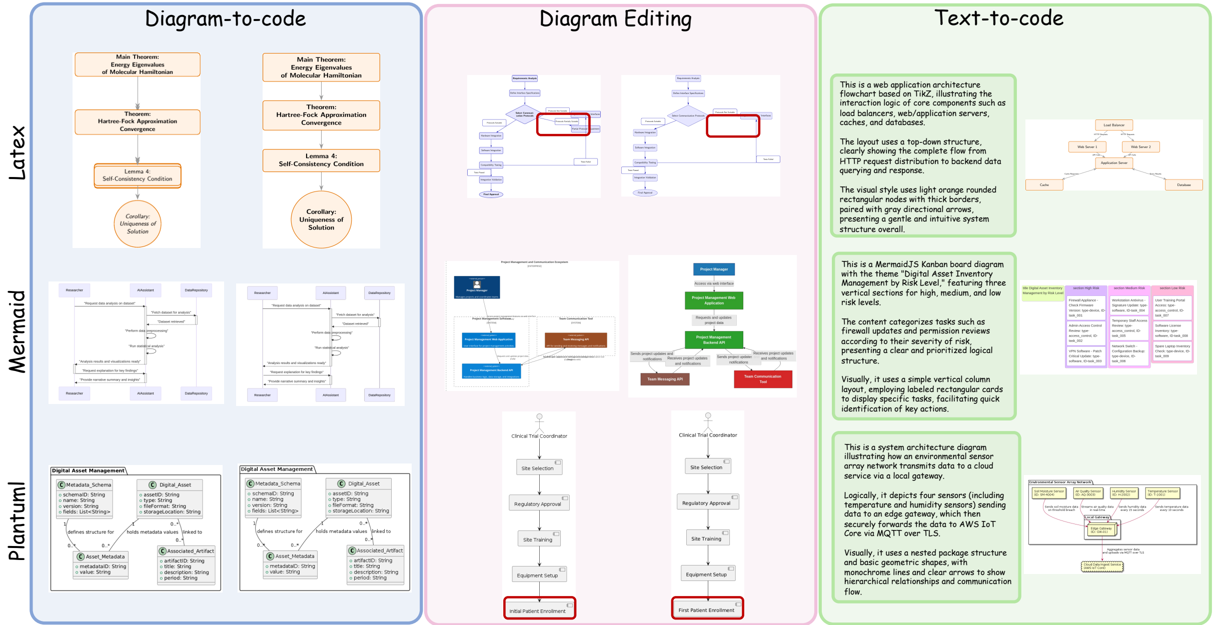

Figure 6: Qualitative showcase of our model across three modalities (LaTeX, Mermaid, PlantUML) and three tasks. **Left:** Diagram-to-Code reconstruction showing high semantic accuracy. **Middle:** Diagram Editing demonstrating precise local updates (highlighted in red). **Right:** Text-to-Code generation converting complex natural language descriptions into valid structural code.

$$\mathrm{Var}\left[\sum_{k=1}^{N} X_k\right] = \sum_{k=1}^{N} \mathrm{Var}[X_k] + 2\sum_{k<l} \mathrm{Cov}[X_k, X_l]. \quad (10)$$

Combining the two steps, we obtain:

$$\mathrm{Var}[R_{\mathrm{acc}}] = \frac{1}{N^2}\left(\sum_{k=1}^{N} \sigma_k^2 + 2\sum_{k<l} \rho\sqrt{\sigma_k^2\sigma_l^2}\right). \quad (11)$$

Upper Bound on Single QA Variance under Graded Rewards: For any random variable $X_k \in [0,1]$, the variance satisfies $\mathrm{Var}[X_k] \leq \mu_k(1-\mu_k)$, with equality if and only if X_k follows a Bernoulli distribution. Since graded rewards allow X_k to take intermediate values, the single QA variance is typically lower than that of binary rewards. Note that in cases where VQA outputs are highly uncertain or clustered near extremes, the variance may approach the upper bound, but it generally does not exceed the binary reward variance.

Implications for Policy Gradient Stability: In policy gradient methods, a single-step gradient estimate can be written as $\hat{g} = \nabla_\theta \log \pi_\theta(o_i \mid x)\, A_i$, where A_i is the advantage estimate constructed from the reward signal. Assuming $\nabla_\theta \log \pi_\theta(o_i \mid x)$ is bounded, the variance of the gradient estimator is positively correlated with the variance of the reward signal. Hence, VIVA rewards reduce variance at both the single QA level and the multi-QA aggregation level, suppressing the propagation of reward noise and providing more stable gradient signals for downstream policy optimization. Moreover, the GRPO normalization step further mitigates the impact of extreme rewards on the gradient, enhancing training stability.

D Efficiency and Computational Cost

The offline question generation for 25k samples was completed in approximately 1 hour using the GPT-4o-mini API with a concurrency of 10. All training was conducted on a cluster of 24 NVIDIA H800 GPUs. For the baseline configuration (w/o Reasoning), the SFT and RL stages require 12 and 20 hours for the 3B model, and approximately 16 and 26 hours for the 7B model, respectively. In contrast, the reasoning-enriched (w/ Reasoning) 3B model requires 16 hours for SFT and 30 hours for RL. During RL training, the Qwen3-VL-32B reward model is deployed in FP16 precision, occupying approximately 64 GB of VRAM per GPU. Notably, this reward model exhibits high inference efficiency, and its response latency is fully accounted for within the reported RL training durations. Regarding inference efficiency of the final model, Direct Mode maintains standard LLM latency, while Think Mode introduces a latency increase proportional to the reasoning trajectory length.

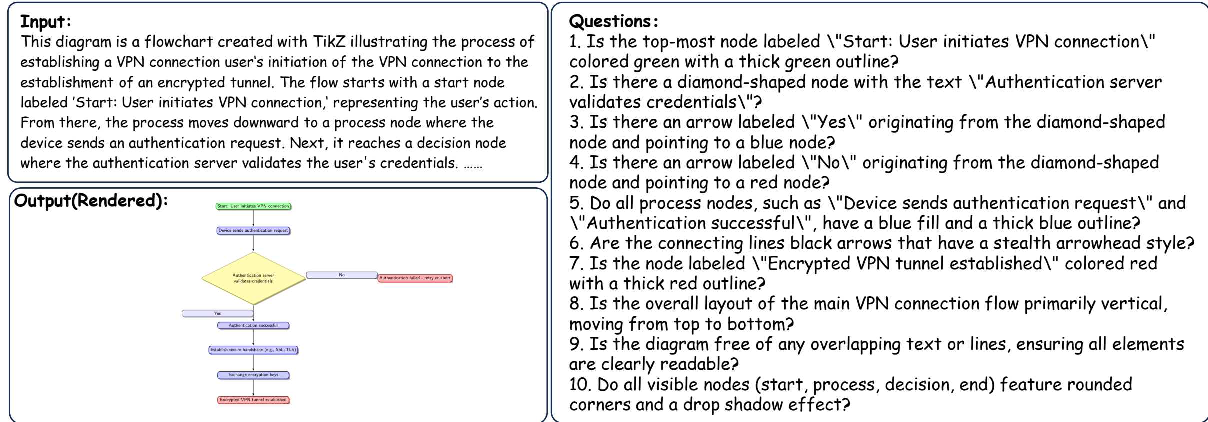

Figure 7: Qualitative example of visual verification questions for the Text-to-Code task.

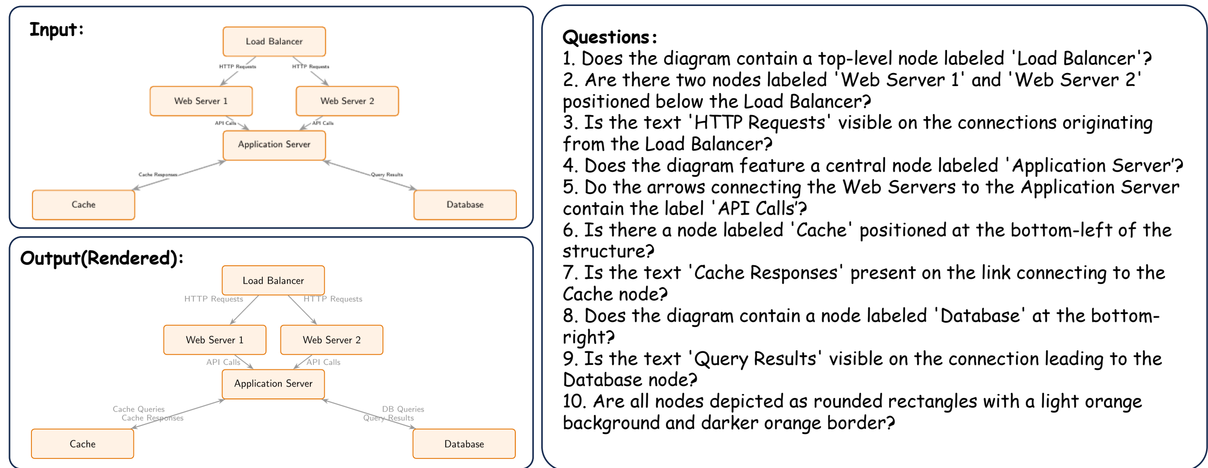

Figure 8: Qualitative example of visual verification questions for the Diagram-to-Code task.

E Evaluation

E.1 Prompt Used in Evaluation

To ensure reproducibility, we provide the exact system prompts used for our GPT-4.1-based evaluation. Figure 14 illustrates the prompts for **Diagram-to-Code**, focusing on Visual Fidelity (S_{vis}). Finally, Figure 15 presents the prompts for **Diagram Editing**, covering Content Preservation (S_{pres}) and Instruction Adherence (S_{task}). Figure 16 displays the prompts for **Text-to-Code** evaluation, assessing Visual Correctness (S_{vis}) and Task Adherence (S_{task}).

E.2 Evaluation Metrics

We utilize a GPT-4.1-based evaluation pipeline to simulate human judgment across diverse tasks, supplemented by deterministic code-level metrics. All scores are normalized to a scale of 0 to 100.

E.2.1 Diagram-to-Code Metrics

For the reverse engineering task, we decouple the structural rendering from the underlying code logic:

Visual Fidelity (S_{vis}): A composite metric evaluating the preservation of visual attributes (e.g., node shapes, layout orientation) and topological integrity (e.g., nodes, edges, and OCR content) between the original image and the reconstructed rendering.

Code Accuracy (S_{code}): Evaluates the syntactic and logical similarity of the generated code against the ground truth using *CrystalBLEU* (Eghbali and Pradel, 2022), which mitigates the bias of common code templates.

E.2.2 Diagram Editing Metrics

This task focuses on the precision of targeted modifications and the stability of unchanged components:

Instruction Adherence (S_{task}): Measures the suc-

Diagram-Editing

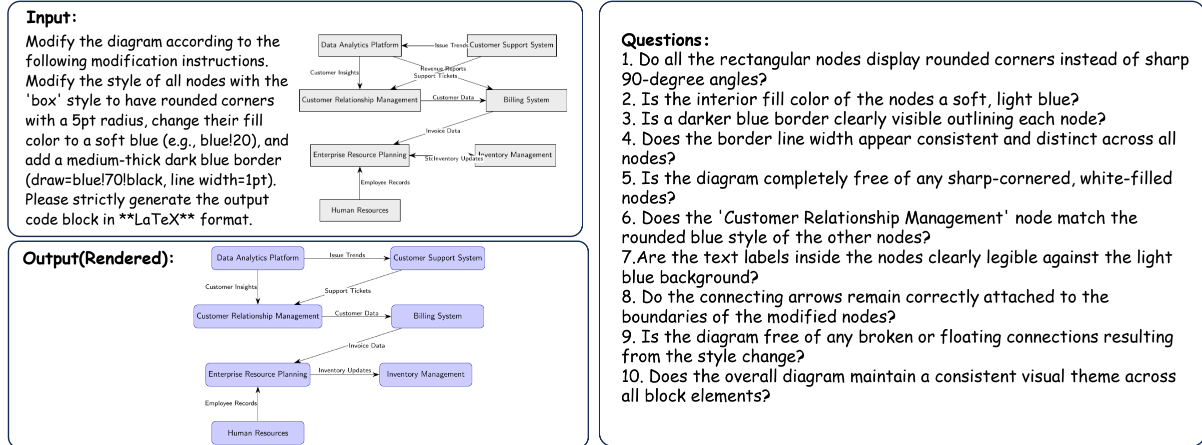

Figure 9: Qualitative example of visual verification questions for the Diagram Editing task.

cess rate of specific modifications requested by the user (e.g., color changes or node deletions), focusing strictly on the execution of the edit instruction.
Content Preservation (S_{pres}): Assesses the stability of regions not targeted by the edit. It penalizes structural collapse, hallucinations, or unintended semantic drifts in unmodified parts of the diagram.

E.2.3 Text-to-Code Metrics

Following the methodology established in *VisCoder2* (Ni et al., 2025), we evaluate the synthesis quality across two dimensions:
Visual Correctness (S_{vis}): Evaluates the alignment between the generated diagram's visual appearance and the ground truth, focusing on data distribution and geometric relationships. **Task Adherence (S_{task}):** Measures how accurately the model follows textual constraints (e.g., specific data points or plot types), independent of the final rendering's aesthetic quality.

F Inference Result of Omni Diagram

F.1 Examples of Ability

To further substantiate the quantitative results presented in the main text, we provide a comprehensive gallery of qualitative examples in Figure 6. This visualization demonstrates the versatility and robustness of our model across the three target diagrammatic languages: LaTeX, Mermaid, and PlantUML. The figure is stratified into three columns, each corresponding to a core task:
Diagram-to-Code: The left column compares the input ground truth images with the model's reconstructed outputs. The results exhibit exceptional visual fidelity and topological correctness, accurately recovering complex structures such as molecular energy levels, sequence interactions, and class hierarchies.
Diagram Editing: The middle column illustrates the model's capability to perform precise, localized modifications based on user instructions. As highlighted by the red bounding boxes, the model successfully executes specific edits (e.g., text updates, node recoloring) while preserving the integrity of the unmodified regions.
Text-to-Code: The right column showcases the model's ability to synthesize complex diagrams from dense, long-context natural language descriptions. The examples cover diverse scenarios, including system architectures, Kanban boards, and sensor networks, confirming the model's strong instruction-following capabilities in zero-shot generation.

F.2 Failure Mode Analysis

The error distribution across different model scales reveals clear boundaries in current diagram generation capabilities, and the specific error statistics are summarized in Table 5. For LaTeX tasks, reference errors represent the most frequent failure mode because the model often struggles to maintain consistency in diagrams with dense connections and complex coordinate systems. The transition from the 3B to the 7B model shows a significant reduction in these instances, which suggests that larger models possess a better capacity for the spatial memory required to link node identifiers across extensive code.

Conversely, syntax errors in PlantUML and Mermaid remain largely stagnant regardless of model

Prompts for Topic Generation

System Prompt:
You are an AI assistant that specializes in generating structured JSON data.
Your task is to generate a unique, engaging topic for each set of numbered keywords provided by the user.
Output Requirements:
1. You MUST return a single, valid JSON object.
2. The JSON object MUST contain exactly one key: "topics".
3. The value of "topics" MUST be a JSON array of strings.
4. Each string in the array should be a 2-3 sentence topic description corresponding to one set of keywords from the user input.
5. The number of strings in the array MUST EXACTLY match the number of keyword sets provided.

User Prompt:
Please generate topic descriptions for the following 3 characters. Return the result as a JSON object according to the system instructions.
1. Keywords: Name=Alex, Age=32, Profession=Software Engineer, Trait=innovative problem solving, Goal=to streamline a complex workflow
2. Keywords: Name=Jordan, Age=45, Profession=Product Manager, Trait=user-centric design, Goal=to map out a new user experience
3. Keywords: Name=Taylor, Age=28, Profession=Data Analyst, Trait=data-driven, Goal=to present data insights to stakeholders

Figure 10: The prompt template used for generating different topics to set scene limitations.

size, pointing to a persistent bottleneck in handling strict domain specific languages. These failures typically occur in diagrams featuring multi-layer nesting or highly recursive structures where the model fails to properly manage hierarchical blocks. When faced with extremely dense wiring or deep logic tiers, the output frequently violates language constraints, demonstrating that topological complexity remains a primary challenge for ensuring structural integrity during the generation process.

Language	Error Category	OmniDiagram (RL)	
		3B	7B
LaTeX	Reference Error	33	23
	Structural Error	16	10
	Syntax Error	15	13
	Math & Resource Error	8	9
	Dependency & Encoding	7	6
	Timeout	0	2
PlantUML	Syntax Error	83	83
	Logic Error	3	6
	System Error	0	2
Mermaid	Syntax Error	52	52
	Rendering & Resource Error	6	6
	Logic Error	5	5
	Structural Error	3	4
	Formatting Error	2	0

Table 5: Statistical summary of generation rrrors across diagram languages.

G Detailed Analysis of Reasoning Strategies

In this section, we provide a granular analysis of the trade-offs between reasoning-enhanced training and inference strategies, as presented in Table 6.

Internalization vs. Explicit Output. Our results demonstrate that the benefits of Chain-of-Thought data are best realized through *internalization* rather than explicit output. The *Mixed SFT* model in Direct mode achieves a higher Task Adherence score (S_{task} 63.0) compared to the *Pure SFT* baseline (61.0), confirming that the model learns to plan better diagrams from the reasoning data. However, explicitly outputting these thoughts acts as a disturbance during inference.

The "Syntax Tax". A critical discovery is the trade-off between semantic quality and syntactic robustness, which we term the "Syntax Tax." When valid, diagrams generated with explicit thinking exhibit superior visual details (Valid S_{vis} 59.23 vs. 55.32). However, the generation of extensive natu-

Prompts for Scenario Generation

System Prompt:
You are an expert in diagram design and have a broad knowledge of different topics.
User Prompt:
My scenario is: "`{scenario}`"
I want you to generate `{num_topics}` topics for a Mermaid "`{figure_type}`" that I will be interested in or I may see during my daily life given my scenario.
Here are the requirements:

1. Each topic is a high-level summary of the contents in the diagram with some design details, e.g., "a Sequence Diagram illustrating a user login process with two-factor authentication".
2. The topics should be diverse to help me generate varied diagrams. Each topic should be unique and not overlap with others.
3. The topics are strictly conditioned on the Mermaid diagram type. Please ensure the topics you provided can be best visualized in "`{figure_type}`".
4. All topics must be in English, even if the scenario is non-English.
5. List `{num_topics}` topics for "`{scenario}`" and separate them with a | character, e.g., topic1 | topic2 | ...... | topic`{num_topics}`.

Do not include any additional text at the beginning or end of your response.

Figure 11: The prompt template used for generating diverse diagram scenario based on user topics and specific diagram types.

ral language reasoning consumes the context window and disrupts the model's ability to maintain the strict syntactic structure required for diagrammatic code (e.g., LaTeX or Mermaid). This results in a precipitous drop in execution rate (from 84.73% to 69.47%), causing the overall performance to suffer despite higher potential visual quality.

Reinforcement Learning alignment. RL effectively mitigates robustness issues but does not alter the fundamental superiority of direct inference for this task. RL aligns the model with the compiler's requirements, boosting the execution rate to a state-of-the-art **90.84%** in Direct mode. While RL also improves the robustness of the Thinking mode, the structural instability of mixing long-form text with code remains, making Direct inference the optimal strategy for Diagram-to-Code generation.

Model Setting	Inf.	Rob.	Rendered (v)		Overall	
		Exec (%)	S_{vis}	S_{task}	$\mathbf{S}_{\text{vis}}$	$\mathbf{S}_{\text{task}}$
Qwen2.5-VL-3B	Direct	51.2	43.7	62.3	22.0	32.0
VisCoder2-3B	Direct	76.3	–	–	43.0	59.0
3B_SFT (190k)	Direct	82.4	55.1	77.9	45.8	64.3
3B_SFT (270k)	Direct	84.7	55.3	73.9	**47.0**	**63.0**
	Think	69.5	59.2	75.0	41.0	52.0
3B_SFT+RL (190k)	Direct	88.6	55.8	72.8	49.4	64.5
3B_SFT+RL(270k)	Direct	**90.8**	54.2	76.6	**49.0**	**70.0**
	Think	75.6	57.3	**76.7**	43.0	58.0

Table 6: Ablation study on data composition and inference strategies. **(v)** metrics are computed exclusively on successfully rendered images. Underlined values highlight superior visual quality. Overall score $S \approx$ Exec $\times$ Rendered.

Prompts for Structured Data Generation

System Prompt:
You are an expert in diagram design and have broad knowledge about various topics.

User Prompt:
My topic is: "`{topic}`"
I need structured data elements about "`{topic}`", which can be used to generate a Mermaid "`{figure_type}`".
Here are the requirements:

1. The data should be highly structured as a JSON object, with its schema tailored specifically for the "`{figure_type}`" syntax. For example, for a Class Diagram, the JSON should contain a list of 'classes' and a list of 'relationships'.
2. The data should be realistic, and the contents should be named using real-world entities. Do not use placeholder names like xxA, xxB, etc.
3. The elements should be concise and directly map to a meaningful diagram. Do not provide too many elements; just the key information.
4. All elements must be in English, even if the topic is non-English.
5. You can use the provided JSON templates to structure decision-based flows. If the topic is related to decision-making or conditional logic, please use or adapt the templates provided in the `<templates>` block.

Figure 12: The prompt template used for generating structured JSON data elements tailored to specific Mermaid diagram types and topics.

Prompts for Mermaid Mindmap Code Generation

System Prompt:
You are an expert in diagram design and good at writing Mermaid code to generate high-quality, visually appealing diagrams.

User Prompt:
My topic is: "`{topic}`". I have a JSON object of structured data about "`{scenario}`" which needs to be converted into a Mermaid "mindmap".
Here is the data (JSON format): `<data> {data} </data>`
Please write the complete Mermaid code to generate the diagram. Here are the requirements:
1. The code must be a valid and complete Mermaid mindmap script adhering strictly to syntax rules.
2. The code must start with "mindmap". The "mindmap" keyword is reserved.
3. **Indentation is the ONLY way to define the hierarchy**. All child nodes MUST be indented more deeply than their parent. Consistent indentation (e.g., 2 or 4 spaces) is required.
4. Do not include any additional text outside of the Mermaid code block.
5. The code must be self-contained within the "'`mermaid` ... "' block.

CRITICAL RULE: A mindmap can only have ONE SINGLE ROOT NODE.
- The first line of the mindmap defines the root.
- All other nodes MUST BE indented under this root node.
- Any line with zero indentation (nodes or styles) will be treated as a second root, causing a fatal error.

IMPORTANT: Styling Syntax and Placement
The safest way is to use `classDef` at the end and apply it via `:::`. **AVOID** placing `style` commands at the top level.
Negative Example (WRONG):
```
mindmap
Root
Another Root // <- FATAL ERROR. Not indented.
```
Positive Example (Valid Syntax):
```
mindmap
root((Main Topic)):::mainStyle
Sub-Topic 1
Detail A
Detail B:::detailStyle
%% Styles defined at the end
classDef mainStyle fill:#f0f8ff,stroke:#333,stroke-width:2px
classDef detailStyle fill:#lightgrey,stroke:#333
```

Figure 13: The prompt template used and applying structured JSON data into executable Mermaid Mindmap code.

Prompts for Diagram-to-code Evaluation

JUDGE_INSTRUCT_VISUAL_FIDELITY:
You are an excellent judge at evaluating the visual fidelity of a diagram reconstruction task. You will be giving scores on how faithfully the reconstructed diagram matches the original source image in terms of both structural logic and visual appearance.
The original diagram (source image) will be given to you as the first image.
The reconstructed diagram (rendered from model-generated code) will be given to you as the second image.
Please score how well the reconstructed diagram matches the original image on a scale from 0 to 100.
Scoring should be carried out based on the Visual Fidelity, which considers the following aspects:
First, determine the Topological Integrity by strictly evaluating the logical correctness, including the existence of all nodes, the accuracy of connections (edges), and the correctness of the text content (OCR). Second, evaluate the Visual Consistency by checking the preservation of visual attributes such as layout orientation (e.g., top-down vs. left-right), node shapes, line styles, and overall aesthetic alignment with the original image.
A score of 100 implies that the reconstructed diagram is a perfect digital twin of the original.
Please penalize any missing information, incorrect logic, or significant stylistic deviations.
After scoring from the above aspects, please give a final score. Do not write anything else. The final score is preceded by the [FINAL SCORE] token.
For example [FINAL SCORE]: 45

Figure 14: The prompt used for the Diagram-to-Code task evaluation.

Prompts for Diagram Editing Evaluation

JUDGE_INSTRUCT_ADHERENCE:
You are an excellent judge at evaluating the instruction adherence of a diagram editing task. You will be giving scores on how well the modified diagram executes the user's request.
The original diagram (before editing) will be given to you as the first image.
The modified diagram (after editing) will be given to you as the second image.
The user instruction describing the required change is provided below (begins from [EDIT INSTRUCTION] token).
Please score how well the modified diagram matches the instruction. Score it on a scale from 0 to 100.
Scoring should be carried out in the following aspect:
Instruction Adherence:
Determine strictly if the specific changes requested in the instruction were applied in the modified diagram.
Focus ONLY on the requested change (e.g., color change, node addition). Ignore side effects (e.g., layout shifts) as long as the requested change is present.
After scoring from the above aspect, please give a final score. Do not write anything else. The final score is preceded by the [FINAL SCORE] token.
For example [FINAL SCORE]: 40

JUDGE_INSTRUCT_PRESERVATION:
You are an excellent judge at evaluating the stability and preservation of a diagram editing task. You will be giving scores on how well the agent preserved the parts of the diagram that should not have changed. The original diagram (before editing) will be given to you as the first image.
The modified diagram (after editing) will be given to you as the second image.
The user instruction is provided below to help you identify what should have changed.
Please score how well the modified diagram preserves the unrequested content. Score it on a scale from 0 to 100.
Scoring should be carried out in the following aspects: Content Preservation:
Compare the nodes, text, and connections that were NOT mentioned in the instruction.
Check for layout stability and ensure the agent did not hallucinate new, unrequested nodes or break the visual structure.
A score of 100 implies perfect preservation of unrequested areas.
After scoring from the above aspect, please give a final score. Do not write anything else. The final score is preceded by the [FINAL SCORE] token.
For example [FINAL SCORE]: 30

Figure 15: The prompt used for the Diagram Editing task evaluation.

Prompts for Text-to-Code Evaluation

JUDGE_INSTRUCT_VIS:
You are an excellent judge at evaluating visualization plots between a model generated plot and the ground truth. You will be giving scores on how well it matches the ground truth plot.
The generated plot will be given to you as the first figure.
Another plot will be given to you as the second figure, which is the desired outcome of the user query, meaning it is the ground truth for you to reference. Please compare the two figures head-to-head and rate them.
Suppose the second figure has a score of 100, rate the first figure on a scale from 0 to 100.
Scoring should be carried out in the following aspect:
Plot correctness:
Compare closely between the generated plot and the ground truth...
After scoring from the above aspect, please give a final score. Do not write anything else. The final score is preceded by the [FINAL SCORE] token.
For example [FINAL SCORE]: 40

JUDGE_INSTRUCT_TASK:
You are an excellent judge at evaluating a visualization plot according to the given task. You will be giving scores on how well the plot image matches the task.
The generated plot will be given to you as an image.
Please score how well the plot matches the task. Score it on a scale from 0 to 100. Scoring should be carried out in the following aspect:
Task adherence: how the plot corresponds to the task given below (begins from [PLOT TASK] token)
After scoring from the above aspect, please give a final score. Do not write anything else. The final score is preceded by the [FINAL SCORE] token.
For example [FINAL SCORE]: 40

Figure 16: The prompt used for Text-to-Code task evaluation.