

LEGO-EDIT: A GENERAL IMAGE EDITING FRAMEWORK WITH MODEL-LEVEL BRICKS AND MLLM BUILDER

Anonymous authors

Paper under double-blind review

Local Edit



Global Edit



Multi Steps Edit

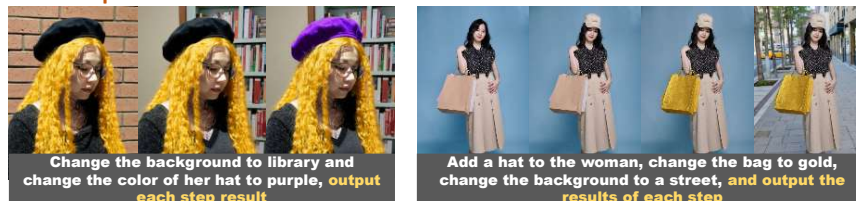


Figure 1: Editing examples of Lego-Edit.

ABSTRACT

Instruction-based image editing has garnered significant attention due to its direct interaction with users. However, real-world user instructions are immensely diverse, and existing methods often fail to generalize effectively to instructions outside their training domain, limiting their practical application. To address this, we propose Lego-Edit, which leverages the generalization capability of Multi-modal Large Language Model (MLLM) to organize a suite of model-level editing tools to tackle this challenge. Lego-Edit incorporates two key designs: (1) a model-level toolkit comprising diverse models efficiently trained on limited data and several image manipulation functions, enabling fine-grained composition of editing actions by the MLLM; and (2) a three-stage progressive reinforcement learning approach that uses feedback on unannotated, open-domain instructions to train the MLLM, equipping it with generalized reasoning capabilities for handling real-world instructions. Experiments demonstrate that Lego-Edit achieves state-of-the-art performance on GEdit-Bench and ImgBench. It exhibits robust reasoning capabilities for open-domain instructions and can utilize newly introduced editing tools without additional fine-tuning.

1 INTRODUCTION

Instruction-based image editing methods accept natural language instructions as input and modify the input image accordingly. It enables intuitive human-computer interaction, offering broad application potential. However, the significant diversity inherent in real-world editing instructions poses a substantial challenge for image editing systems in handling flexible user commands.

Existing approaches for instruction-based editing are broadly classified into two categories. End-to-end methods (Labs, 2024a; Liu et al., 2025) train a single generative model to learn both instruction comprehension and pixel mapping for editing implicitly. As shown in Fig. 2 (a), these methods are primarily constrained by the fixed instruction patterns within their training data, and consequently struggle to generalize well even with massive training datasets.

In contrast, agent-based schemes utilize MLLMs to explicitly interpret editing instructions and invoke editing tools to execute the requested modifications. Prior research (Wang et al., 2024; Xue et al., 2025) often relies on curated prompts to guide the agents, but such prompt-driven approaches lack a deep understanding of tools, impeding agents’ ability to organize them effectively. Subsequent studies (Guo et al., 2025) attempt to alleviate this burden by constructing predefined workflows as task-level tools for agents to invoke, as shown in Fig. 2 (b). However, this strategy inherently limits the framework’s capacity to handle instructions that deviate from the predefined workflows.

To address the challenge of processing flexible real-world instructions for image editing, we propose a novel framework, named Lego-Edit. It employs a fine-tuned MLLM as an agent, termed Builder. The Builder leverages its enhanced reasoning capability to organize a set of specialized pre-trained editing models, called Bricks, enabling precise execution of diverse editing instructions, as shown in Fig. 2 (c). It incorporates two key design innovations:

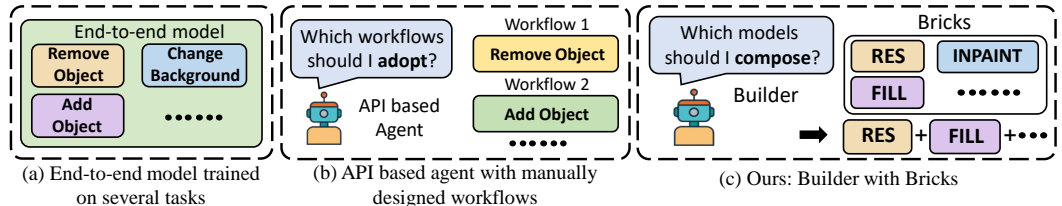


Figure 2: Comparison of end-to-end approach, API-based agent with curated workflows, and our method.

Model-Level Tools: We train and integrate a suite of specialized models as editing tools. This model-level design provides the Builder with high flexibility for composition and enables individual tools to achieve superior performance for their specific functions with less training data.

Three-Stage Progressive Reinforcement Learning Training Strategy: We first employ a Supervised Fine-Tuning (SFT) stage, followed by a Reinforcement Learning (RL) stage to train our Builder on specific editing tasks with the ground truth, establishing fundamental reasoning capabilities and tool usage knowledge. Subsequently, we conduct an additional RL stage utilizing abundant unlabeled instructions beyond the specific tasks, where a large-scale critic model provides feedback. This process enhances our Builder’s capability to handle flexible instructions.

Benefiting from these designs, our Lego-Edit reliably reasons about and executes flexible editing instructions and can integrate novel tools without additional training. Furthermore, it achieves state-of-the-art results on the GEdit-Bench and ImgBench benchmarks.

In summary, our principal contributions are threefold:

- We propose Lego-Edit, an instruction-based image editing framework that utilizes a reinforcement learning fine-tuned MLLM agent to coordinate model-level editing tools for executing flexible real-world instructions.
- We introduce a three-stage progressive reinforcement learning training strategy that provides feedback using unlabeled data, significantly enhancing the reasoning and tool composition capabilities of MLLM.

- Extensive experiments demonstrate that Lego-Edit achieves SOTA performance on GEdit-Bench and ImgBench. The framework also exhibits strong generalization in processing flexible open-domain instructions and could integrate new tools without retraining.

2 RELATED WORKS

Instruction-based Image Editing: Instruction-driven image editing, which emerged from Instruct-Pix2Pix (Brooks et al., 2023), provides a natural user interface for image editing. The majority of existing methods rely on diffusion models (Gao et al., 2025; Han et al., 2024; Hertz et al., 2022); while some methods incorporate Multi-modal Large Language Models (MLLMs) to achieve more precise edits like SmartEdit (Huang et al., 2024).

Models trained on massive datasets (e.g., FLUX (Labs, 2024a), HiDream (Cai et al., 2025), Step1X (Liu et al., 2025)) exhibit strong performance across various editing tasks. However, generalizing effectively across flexible editing instructions remains a critical challenge for them. Recently developed unified visual understanding and generation models (e.g., ILLUME++ (Huang et al., 2025a), GPT-4o (Hurst et al., 2024), Bagel (Deng et al., 2025), UniWorld (Lin et al., 2025)), trained on broader datasets encompassing multiple tasks like image captioning and image editing, demonstrate enhanced generalization capabilities yet are still constrained.

In contrast, our proposed approach achieves superior instruction generalization with minimal training data by orchestrating specialized editing tools via a fine-tuned MLLM.

MLLMs as Agents: Autonomous agents capable of utilizing tool calls have garnered significant research interest. Many approaches primarily leverage LLMs to invoke tools (Qin et al., 2023; Du et al., 2024; Zheng et al., 2024). With advances in MLLMs, considerable effort has focused on employing or fine-tuning MLLMs for multi-modal agent applications (Huang et al., 2025b).

Some researchers also explore MLLMs as agents for image generation and editing. ComfyAgent (Xue et al., 2025) adapts prompts to call ComfyUI tools via code execution, but this code-based invocation constrains its performance. ComfyMind (Guo et al., 2025) manually defines multiple pipelines for agent-driven tool use, ensuring high success rates but limiting operational flexibility.

Notably, our approach employs reinforcement learning to equip the MLLM with compositional tool-usage knowledge and reasoning ability. Combined with model-level tools, this framework achieves high success rates, superior editing performance, and robust generalization to diverse instructions.

3 METHOD

We introduce Lego-Edit, a framework designed for general instruction-based image editing. It uses Builder (an MLLM) to invoke Bricks (model-level tools) for flexibility and employs reinforcement learning (RL) to enhance the Builder’s reasoning and tool composition ability. We first outline the overall framework. Then, we describe the tool classification. Finally, we elaborate on the three-stage progressive reinforcement learning (RL) training strategy for the Builder.

3.1 OVERALL FRAMEWORK

As illustrated in Fig. 3, our system comprises: 1) the Builder, an MLLM reasoning agent denoted as $f(\theta)$ that generates workflows, where θ denotes the model parameters; 2) an Executor D that parses and executes workflows; and 3) the Bricks, an external model-level tool library $\mathcal{M} = \{M_1, \dots, M_N\}$ containing functions encapsulating models or logic processes, where N is the total number of tools.

Given an input pair (I, q) comprising the input image I and editing prompt q , the Builder $f(\theta)$, observing the state $s = (I, q)$, generates a reasoning trace, denoted as *Think*, and a JSON-formatted workflow g based on its strategy $\pi_\theta(g | s)$. This workflow $g = (V, E)$ is a tool invocation graph. The vertex set $V = \{M_1, \dots, M_K\}$ represents the selected tool instances, with each $M_i \in \mathcal{M}$ and $1 \leq K \leq N$; here, K is adaptively determined by the task complexity. The edge set $E \subseteq V \times V$ defines dependencies, where an edge (M_i, M_j) indicates that the input of M_j depends on the output of M_i . The Executor D then parses g , invokes the tools, and generates the edited image $I' = D(g, I)$.

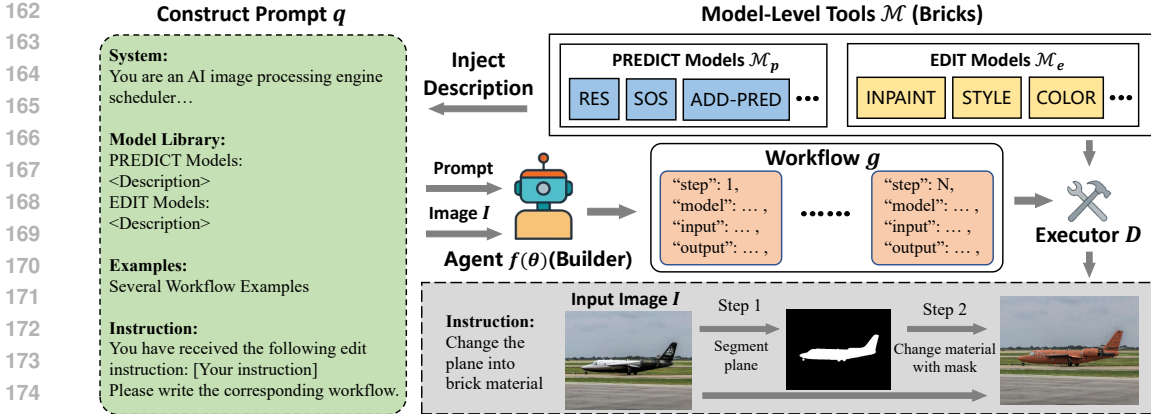


Figure 3: Overall framework of Lego-Edit. Given an instruction and an input image, the Builder generates a tool invocation workflow. The Executor then executes this workflow, calling corresponding tools to generate the edited output image.

3.2 MODEL-LEVEL EDITING TOOLS

We constructed a fine-grained library of model-level editing tools \mathcal{M} , where each tool represents a single model or function. Tools are categorized into two classes by whether they modify the image:

Predictive Models ($\mathcal{M}_p \subset \mathcal{M}$): Extract/process masks and regions to provide spatial constraints without altering pixels. Included tools are RES (segment specified objects), SOS (subject object segmentation), ADD-PRED (predict addition location), CAP-PRED (image captioning), INVERSE (invert the mask), and one additional tool.

Editing Models ($\mathcal{M}_e \subset \mathcal{M}$): Modify image content. Included tools are FILL (add object with given prompt or reference image), five specialized LoRA adapters trained on FILL: INPAINT (inpainting), POSE (human pose change), ENV (environment alteration), STYLE (style transfer), RCM (material/color change), and three additional tools.

To call a tool, the Builder needs to specify its name, input parameters, and output parameters. Complete details are in the supplementary material.

To prevent task confusion that can arise from joint training in end-to-end models (as in ICEdit (Zhang et al., 2025)), we train independent LoRA adapters for each editing model. And the Builder could precisely control edit scope using masks from \mathcal{M}_p , enabling more accurate editing.

3.3 THREE-STAGE PROGRESSIVE REINFORCEMENT LEARNING STRATEGY

To train the Builder $f(\theta)$ for effective tool composition, we employ a three-stage progressive RL strategy, gradually increasing task complexity and reducing reliance on ground truth data.

3.3.1 REINFORCEMENT LEARNING WITH GRPO

We first introduce the Group Relative Policy Optimization (GRPO) (Shao et al., 2024) algorithm utilized in stages 2 and 3. For a given input $s = (I, q)$, the policy π_θ samples G workflows $\{g_1, \dots, g_G\}$. Each workflow g_j receives a reward r_j . The relative advantage for each sample within the group is computed as:

$$A_j = \frac{r_j - \text{mean}(\{r_1, \dots, r_G\})}{\text{std}(\{r_1, \dots, r_G\})}, \quad j = 1, 2, \dots, G \quad (1)$$

The policy is updated by maximizing the GRPO objective:

$$\mathcal{J}(\theta) = \mathbb{E}_{s \sim \mathcal{Q}} \mathbb{E}_{\{g_j\}_{j=1}^G \sim \pi_{\theta, \text{old}}(\cdot|s)} \left[\frac{1}{G} \sum_{j=1}^G \min(r_j^{\text{ratio}} A_j, \text{clip}(r_j^{\text{ratio}}, 1 \pm \epsilon) A_j) - \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \right] \quad (2)$$

where \mathcal{Q} denotes the distribution of image-instruction pairs for sampling observation s , $\pi_{\theta_{\text{old}}}$ is the old policy before updating, π_{ref} is a fixed reference policy copied from the model’s initial state before any training, $r_j^{\text{ratio}} = \pi_{\theta}(g_j|s)/\pi_{\theta_{\text{old}}}(g_j|s)$, ϵ controls the clipping range, and β weights the KL regularization D_{KL} towards π_{ref} .

3.3.2 STAGE 1: SUPERVISED FINE-TUNING (SFT)

We adapt the Builder to image editing domain using SFT on data from several specific tasks. Each sample (I, q) is paired with expert-generated reasoning traces $Think_{\text{GT}}$ and ground truth workflows g_{GT} . The learning target is denoted as the concatenated sequence $l = [Think_{\text{GT}}, g_{\text{GT}}]$. The model is trained to minimize the negative log-likelihood:

$$L_{\text{SFT}} = - \sum_{t=1}^T \log p_{\theta}(l_t | I, q, l_{<t}) \quad (3)$$

where T is the total length of l , and p_{θ} is the model’s conditional next-token distribution.

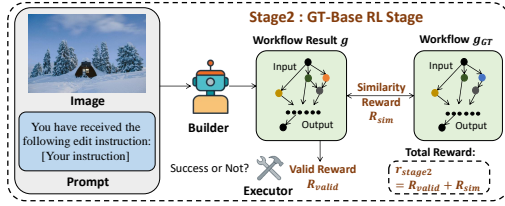


Figure 4: Illustration of the reward design adopted in Stage 2 (GT-based RL training).

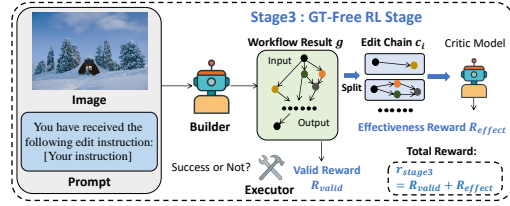


Figure 5: Illustration of the reward design adopted in Stage 3 (GT-free RL training).

3.3.3 STAGE 2: REFINEMENT WITH GT-BASED REWARDS

Building on SFT, Stage 2 continues training on specific tasks using only (I, q) pairs and g_{GT} as shown in Fig. 4. The workflow generated by Builder is denoted g . We adopt two rewards at this stage:

Valid Reward (R_{valid}): It penalizes non-executable workflows as follows:

$$R_{\text{valid}} = \begin{cases} 0, & \text{if } D \text{ successfully executes } g \\ -1, & \text{otherwise} \end{cases} \quad (4)$$

Similarity Reward (R_{sim}): It measures alignment between g and the expert workflow g_{GT} using hierarchical graph matching. The depth of nodes in g is calculated based on inverse topological order, which means the depth of the output node is 0. Matched nodes M are found per depth layer using the Hungarian algorithm based on node similarity with a threshold of 0.6. The node similarity sim_{node} is calculated by averaging the indicator of whether the same model is used and the proportion of identical parameters. R_{sim} combines node coverage and average matched node similarity:

$$R_{\text{sim}} = 0.5 \cdot \frac{|M|}{\max(|V_g|, |V_{g_{\text{GT}}}|)} + 0.5 \cdot \frac{1}{|M|} \sum_{(i,j) \in M} \text{sim}_{\text{node}}(i, j) \quad (5)$$

where V_g and $V_{g_{\text{GT}}}$ denote the node sets of the generated workflow g and the ground truth workflow g_{GT} respectively.

The total reward is: $r_{\text{stage2}} = R_{\text{valid}} + R_{\text{sim}}$.

3.3.4 STAGE 3: GENERALIZATION WITH GT-FREE CRITIC REWARDS

Stage 3 targets generalization to open-domain instructions using only (I, q) pairs. We employ the same valid reward R_{valid} as Eq. 4 and another effectiveness reward to provide feedback without a ground truth workflow as shown in Fig. 5:

Effectiveness Reward (R_{effect}): It uses an MLLM critic model C to assess semantic alignment between the workflow’s effect and instruction q . Workflows are decomposed into several editing chains

c_i , each of which contains only one editing model in \mathcal{M}_e to perform actual editing. The critic C would abstract each chain’s effect into a meta-edit description m_i . Then it evaluates the description set $\{m_i\}$ against q , which is formulated as:

$$(N_{\text{remove}}, N_{\text{add}}, A) = C(I, q, \{m_i\}) \quad (6)$$

Specifically, critic C must determine whether to remove existing editing chains or add new editing chains to better achieve the instruction. N_{remove} denotes the number of chains to remove, N_{add} denotes the number of chains to add and A denotes the content of added new chains. R_{effect} applies a penalty defined as follows:

$$R_{\text{effect}} = 1 - 0.5 \cdot (N_{\text{add}} + N_{\text{remove}}) \quad (7)$$

The total reward is: $r_{\text{stage3}} = R_{\text{valid}} + R_{\text{effect}}$.

4 EXPERIMENTS

We first demonstrate the model’s zero-shot capability, followed by extensive experiments validating the superiority of our framework on image editing benchmarks. Then we analyze the performance improvements and other advantages introduced by model-level tools, and finally demonstrate the Builder’s improved generalization and performance enabled by reinforcement learning.

4.1 IMPLEMENTATION DETAILS

Builder: Our Builder is based on MiMo-VL-7B (Team et al., 2025) and undergoes full-parameter fine-tuning in bf16 using a progressive three-stage curriculum: (1) 500 image-text pairs (instructions, thoughts and workflows), (2) 10K pairs (instructions and workflows), and (3) 50K pairs (instructions), all sourced from OmniEdit (Wei et al., 2024). The generation of thoughts and workflows, along with the critic model used in stage 3 training, are both based on Qwen2.5-VL-72B (Bai et al., 2025). Each stage trains for 1 epoch with AdamW ($\text{lr} = 1\text{e}-5$, $\beta = (0.9, 0.999)$, weight decay = 0.01), without warmup or decay (batch size 8, image size 448×448). Data construction details are provided in the supplementary material.

Predictive Tools: RES utilizes EVF-SAM (Zhang et al., 2024), trained from scratch on 200K MS COCO (Lin et al., 2014) samples and 8K HierText (Long et al., 2022) samples ($\text{lr} = 1\text{e}-4$, batch size 64, resolution 512×512 , 10K iterations) with BCE and Dice loss. SOS employs U2Net (Qin et al., 2020), trained on 5K DIS (Qin et al., 2022) with identical loss and optimization settings (batch size 24, 100 epochs, BCE loss only). Both models initialize without pretrained weights. ADD-PRED and CAP-PRED are built upon our Builder. CAP-PRED directly leverages the Builder’s native captioning capability. ADD-PRED is fine-tuned with LoRA adapters (rank=32) on 50K OmniEdit (Wei et al., 2024) samples for addition region prediction, formulated as bounding box generation between the source and target images, training uses 1 epoch and a learning rate of $1\text{e}-4$.

Editing Tools: We adopt ICEdit’s (Zhang et al., 2025) framework (LoRA fine-tuning at rank=32) but implement five specialized adapters for individual tasks, rather than the multi-task adapter. Each adapter is trained on 10K task-specific samples curated from OmniEdit (Wei et al., 2024) and MagicBrush (Zhang et al., 2023) via VIEScore (Ku et al., 2023) assessment, using a global batch size of 8 for 10K steps at 768×768 resolution. [We train these tools on two backbones, FLUX-1 Fill \(Labs, 2024b\) and Qwen-Image-Edit Wu et al. \(2025a\)](#). In addition, we incorporate the open-sourced LoRA of FLUX-Text (Lan et al., 2025) into our toolbox.

All the experiments utilized $8 \times$ NVIDIA H20 GPUs for training. We adopt DeepSpeed ZeRO-3 (Rajbhandari et al., 2020) to accelerate training.

4.2 EVALUATION SETTINGS

To ensure authoritative evaluation, we benchmark our method on two widely adopted datasets: GEdit-Bench (606 samples) (Liu et al., 2025) and ImgEdit-Bench (811 samples) (Ye et al., 2025), known for complex editing instructions and high-quality imagery. Following standard protocols, we employ VIEScore (Ku et al., 2023) executed by GPT-4o (Hurst et al., 2024) as our primary metric. To ensure fairness and reproducibility, we fix the random seed to 0 and perform single-shot inference for all evaluations. [We evaluate both the FLUX-Tools and Qwen-Tools variants of our system, while all ablation studies are performed using the FLUX-Tools variant.](#)

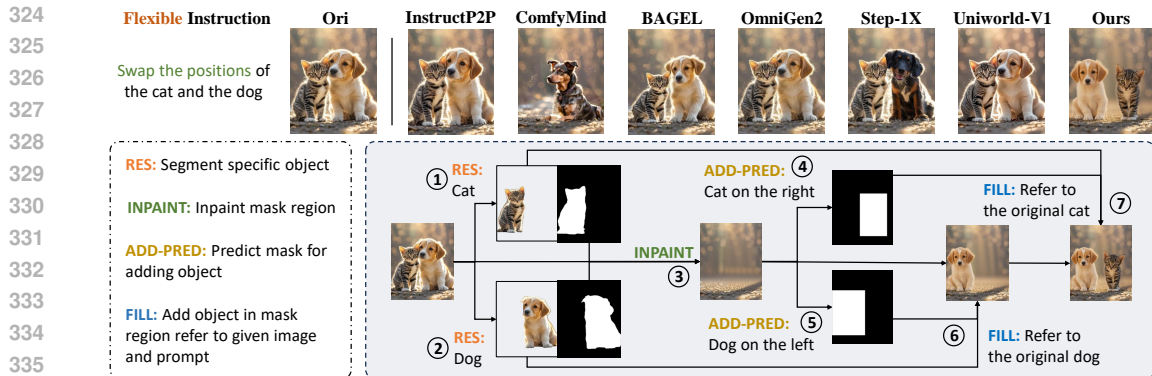


Figure 6: Comparison with other methods on complex edits (top) and our Tool composition workflow (bottom).

4.3 ZERO-SHOT CAPABILITY OF THE BUILDER

Zero-Shot Complex Edits with Flexible Tool Composition: Fig. 6 presents a visual comparison of editing results on flexible instructions, alongside the Builder’s tool composition process. For the “swap” instruction, although the Builder was not explicitly trained on this task, it effectively decomposes the instruction into atomic operations by first removing object A using RES and INPAINT, then inserting object B via ADD-PRED and FILL. This example shows its ability to compose specialized editing tools for flexible instructions, which enables complex edits beyond the reach of end-to-end or curated-pipeline models.

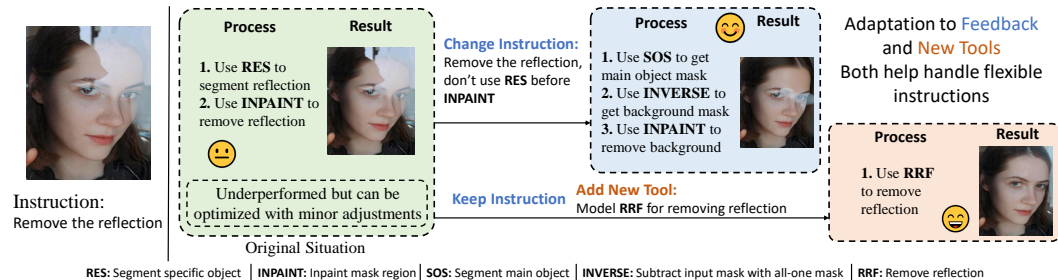


Figure 7: Example of Zero-Shot adaptation via feedback and Tool Insertion in reflection removal.

Zero-Shot Adaptation to Feedback and New Tools: Fig. 7 demonstrates the Builder’s adaptability to user feedback and new tools without retraining. For the reflection removal task, the Builder’s initial workflow (RES and INPAINT) failed because RES could not segment reflections effectively. Users can provide feedback, such as “don’t use RES before INPAINT” to prevent it. The Builder will revise its workflow: it uses SOS for foreground segmentation, INVERSE to acquire the background, and then INPAINT to remove part of the reflection. Additionally, users can introduce a dedicated reflection-removal tool (RRF), which the Builder readily adopts to solve the task effectively. This illustrates the system’s adaptability to extend capabilities by integrating new tools or incorporating user feedback, all without modifying the Builder.

4.4 COMPARISON WITH STATE-OF-THE-ART

GEdit-Bench: As shown in Tab. 1, the FLUX-Tools variant achieves competitive overall performance (G.O = 6.88), while the Qwen-Tools variant achieves the highest performance across all three metrics, outperforming recent SOTA methods Qwen-Image-Edit and SeedEdit-4.0. Fig. 8 further illustrates the FLUX-Tools variant’s strength in fine-grained tasks such as color change and material replacement. This precision stems from the Builder’s ability to compose RES, enabling localized edits while preserving non-target regions (Sec. 4.6). Compared to traditional API agents like ComfyMind that rely on pre-scripted pipelines, Lego-Edit gains notable advantages through its capable Builder and flexible orchestration of specialized Tools.

Generalization Beyond Training Tasks: In the first two training stages, the Builder is trained on five tasks (Background Change, Color Alteration, Subject Addition, Subject Removal, and Subject Replacement). During evaluation, it generalizes effectively to six additional tasks. This generalization is enabled by the Builder’s flexible composition of Tools, which allows it to tackle untrained tasks. As shown in Fig. 8, the FLUX-Tools variant achieves strong results on most untrained tasks, demonstrating robust generalization beyond its training scope.

Table 1: Quantitative evaluation on GEdit-Bench-EN. All metrics are reported as higher-is-better (↑).

Methods	G.SC↑	G.PQ↑	G.O↑
Instruct-P2P	3.58	5.49	3.68
MagicBrush	4.68	5.66	4.52
AnyEdit	3.18	5.82	3.21
OmniGen	5.96	5.89	5.06
Step1X-Edit	7.13	7.00	6.44
BAGEL	7.36	6.83	6.52
UniWorld-V1	4.93	7.43	4.85
ComfyMind	2.67	5.93	2.61
OmniGen2	-	-	6.42
FLUX.1 Kontext	-	-	6.26
Qwen-Image-Edit-2509	<u>8.15</u>	<u>7.86</u>	<u>7.54</u>
SeedEdit-4.0	<u>8.17</u>	<u>7.66</u>	<u>7.44</u>
Ours-FLUX-Tools	6.45	7.45	6.88
Ours-Qwen-Tools	8.42	7.90	7.84

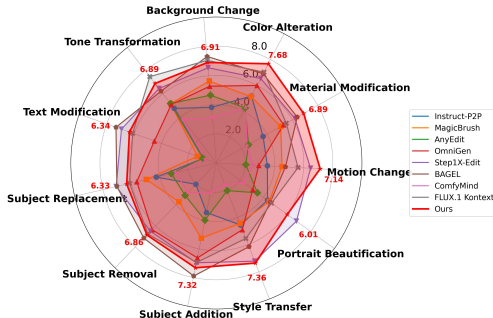


Figure 8: VIEScore of Each Sub-task in GEdit-Bench-En of FLUX-Tools variant. All the results are evaluated by GPT-4o.

Table 2: Quantitative evaluation on ImgEdit-Bench. All metrics are reported as higher-is-better (↑).

Methods	Add	Adjust	Extract	Replace	Remove	Style	Action	Hybrid	Background	Overall↑
MagicBrush (Zhang et al., 2023)	2.72	1.47	1.31	1.89	1.57	2.49	1.39	1.80	2.03	1.85
Instruct-P2P (Brooks et al., 2023)	2.29	1.79	1.33	1.93	1.49	3.54	1.51	1.48	1.67	1.89
AnyEdit (Yu et al., 2025)	3.12	2.66	1.82	2.71	2.34	3.27	3.31	2.07	2.37	2.63
UltraEdit (Zhao et al., 2024)	3.63	3.01	2.02	3.13	1.71	3.69	3.57	2.33	3.31	2.93
Step1X-Edit (Liu et al., 2025)	3.90	3.13	1.87	3.45	2.61	4.44	3.43	2.52	3.19	3.17
BAGEL (Deng et al., 2025)	3.55	3.30	1.56	3.38	2.44	4.24	4.29	2.55	3.22	3.17
UniWorld-V1 (Lin et al., 2025)	3.86	3.70	2.23	3.49	3.54	4.22	3.44	3.13	2.76	3.37
ComfyMind (Guo et al., 2025)	1.45	3.14	2.21	3.43	2.81	2.66	2.74	0.57	2.26	2.63
OmniGen2 (Wu et al., 2025b)	3.57	3.06	1.77	3.74	3.20	4.81	4.68	2.52	3.57	3.44
FLUX.1 Kontext (Batifol et al., 2025)	3.76	3.45	2.15	3.98	2.94	3.78	4.38	2.96	4.26	3.52
Qwen-Image-Edit-2509 (Wu et al., 2025a)	<u>4.32</u>	<u>4.36</u>	<u>4.04</u>	<u>4.64</u>	<u>4.52</u>	<u>4.84</u>	<u>4.71</u>	<u>3.39</u>	<u>4.37</u>	<u>4.35</u>
SeedEdit-4.0 (Seedream et al., 2025)	4.17	4.35	3.87	4.43	<u>4.66</u>	4.70	4.68	<u>3.51</u>	<u>4.49</u>	4.32
Ours-FLUX-Tools	4.03	3.84	2.47	3.41	3.42	4.48	4.04	3.20	3.41	3.59
Ours-Qwen-Tools	<u>4.19</u>	<u>4.41</u>	<u>4.17</u>	<u>4.58</u>	<u>4.70</u>	<u>4.86</u>	<u>4.74</u>	<u>3.60</u>	4.22	<u>4.39</u>

ImgEdit-Bench: Our FLUX-Tools variant maintains top performance on ImgEdit-Bench, while the Qwen-Tools variant achieves the highest overall score (4.39) among all compared methods including recent SOTA methods Qwen-Image-Edit and SeedEdit-4.0. The detailed results are shown in Tab. 2. Crucially, our Qwen-Tools variant dominates the most challenging Hybrid Editing sub-task (3.60). This success validates our proposition that the Builder can parse composite instructions into atomic sub-tasks and dynamically generate workflows to coordinate specialized Tools.



Figure 9: Qualitative results compared to other methods.

Qualitative Results: As illustrated in Fig. 9, our method outperforms other approaches in both edit accuracy and visual realism. Edits are well-aligned with the intended regions and maintain high perceptual quality. More comparative results are provided in the supplementary material.

4.5 ABLATION STUDY ON TOOLS

Necessity of Task-Specialized Tools. To validate Lego-Edit’s design, we compare its task-specialized architecture with a unified alternative using identical settings. Three separate LoRA adapters are trained on 10K samples per task (Color Alteration, Style Transfer, Tone Transfer), while the unified model uses a combined 30K dataset. As shown in Tab. 3, specialized models outperform the unified one (e.g., 6.83 vs. 5.94 in color alteration). Increasing LoRA rank in the unified model brings no gain. Qualitative results (provided in supplementary material) reveal frequent task confusion in the unified setup, highlighting the importance of specialization for editing fidelity.

Table 3: Comparison between Multi-task LoRA and Single-task LoRA on GEdit-Bench-EN, we report the G_O metric.

Model	Rank	Color Alteration	Style Transfer	Tone Transfer
Multi-task	32	5.94	5.47	5.56
Multi-task	64	5.42	5.43	5.57
Single-task	32	6.83	6.75	6.63

Table 4: Comparison between separately adopted and composed adopted tools on GEdit-Bench-EN.

Task	Compose Pattern	G_O
Color Alteration	RCM	6.83
	RCM + RES	7.68
Material Modification	RCM	6.08
	RCM + RES	6.89
Style Transfer	STYLE	6.75
	STYLE + CAP-PRED	7.36

4.6 ABLATION STUDIES ON BUILDER

Table 5: Performance is evaluated by execution success rate, indicating error-free syntax, and VIEScore, assessing editing quality. Success rate comprises Simple %Pass on original instructions and Complex %Pass on three GPT-4o-generated variants per instruction.

Agent Model	Simple %Pass	Complex %Pass	G_SC↑	G_PQ↑	G_O↑
GPT-4o	75.2	52.1	4.28	6.53	5.10
MiMo-VL-7B-Base	56.8	44.9	3.11	4.66	3.59
MiMo-SFT	75.7	53.8	4.39	7.01	5.22
MiMo-RL w/ GT	100	81.9	5.49	7.35	6.16
MiMo-RL w/o GT	100	99.1	6.45	7.45	6.88
Qwen3-VL-2B-Base	49.6	39.8	2.74	6.01	3.12
Qwen3-SFT	68.3	47.1	4.17	6.58	4.99
Qwen3-RL w/ GT	94.8	73.7	5.33	7.32	6.09
Qwen3-RL w/o GT	100	98.5	6.40	7.41	6.85

Effectiveness of Reinforcement Learning Training. Ablations on GEdit-Bench in Tab. 5 show the effectiveness of our progressive RL training. Starting from Builder-SFT, which outperforms basemodel MiMo-VL-7B and powerful MLLM GPT-4o on simple and complex success rates and VIEScores, subsequent RL training with ground truth (Builder-RL w/ GT) achieves 100% simple and 81.9% complex success rate with better VIEScores. Final GT-free RL training (Builder-RL w/o GT) maintains 100% simple success, boosts complex success to 99.1%, and achieves the highest VIEScores. Furthermore, we adopt Qwen3-VL-2B as the Builder base model and observe that, although its initial capability is weaker than MiMo, after the same three-stage training pipeline its performance becomes comparable to MiMo, highlighting the strong effectiveness and robustness of our training methodology.

Latency Discussion. We evaluate our system on GEdit-Bench using a single H20 GPU. The MiMo-based Builder takes 3.5 seconds, and the slowest Tool takes 2.7 seconds, resulting in a total pipeline latency of approximately 6.9 seconds. In comparison, the Qwen3-VL-based Builder takes 1.9 seconds, with the slowest Tool still at 2.7 seconds, yielding a total latency of approximately 5.2 seconds. for the end-to-end method, Bagel takes 25 seconds, Step1X-Edit takes 21.3 seconds, Omnigen2 takes 22.6 seconds and FLUX-Kontext takes 7.4 seconds, all under identical settings.

Effectiveness of Tool Composition. To evaluate the impact of explicit Tool composition in Lego-Edit, we examine its performance across GEdit-Bench sub-tasks. As shown in Tab. 4, integrating RES segmentation masks with RCM increases G_O for color alteration and material modification, highlighting the benefits of RES’s spatial control. Similarly, incorporating CAP-PRED text descriptions into STYLE boosts G_O in style transfer, as textual cues enforce semantic alignment between stylized outputs and the source image.

5 LIMITATIONS

For most suboptimal outcomes, the primary limiting factor was the capability of the toolset itself. We summarize two major sources of failure: insufficient tool performance and toolset incompleteness.

5.1 INSUFFICIENT TOOL PERFORMANCE

Insufficient tool performance refers to cases where the Builder reasonably invokes tools, but the failure of a specific tool leads to an incorrect outcome. As illustrated in Fig. 10, our instruction is "set the clock to 12:10". Neither our framework, Qwen-Image-Edit, nor SeedEdit-4.0 could accomplish this task. For our framework, the Builder gives a clear workflow: it first calls the RES model to segment and remove the hour hand, then predicts the position of the hand at 12:10 and inserts it. The left part of ours in Fig. 10 shows that the process fails because the RES model incorrectly extracts the entire clock face. We fine-tune the RES model with a few hour-hand examples to improve hand identification ability. As shown in the right part of ours in Fig. 10, once the hour hand is correctly segmented and removed, the subsequent steps succeed, ultimately fulfilling the instruction.

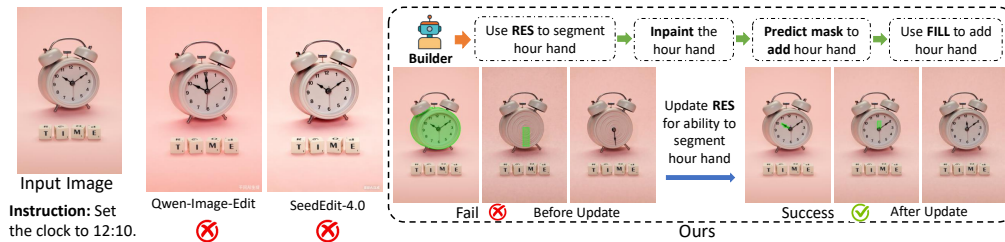


Figure 10: Example of editing failures caused by insufficient tool performance.

5.2 TOOLSET INCOMPLETENESS

Toolset incompleteness occurs when available tools cannot fully satisfy a user instruction, so the Builder produces the closest possible workflow, but results remain limited. As illustrated in Fig. 11, we instructed the model to extend the image to achieve a balanced composition. Both Qwen-Image-Edit and SeedEdit-4.0 failed to follow the instruction, while our framework produced a slightly better result. As shown in the left part of ours in Fig. 11, given the current toolset, the Builder correctly inferred the downward direction but overextended the image by a factor of 1.5, producing more padding than necessary. To address this, we trained a LoRA based on Builder on 2k examples to predict precise extension ratios in four directions. The Builder invokes this tool to accurately estimate the required extension ratio and ultimately satisfies the instruction.

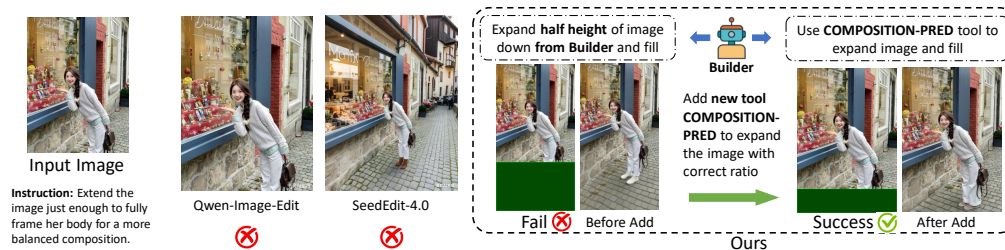


Figure 11: Example of editing failures caused by toolset incompleteness.

6 CONCLUSION

In this paper, we introduce Lego-Edit, a novel agent-based framework for instruction-based image editing. It employs an RL-finetuned MLLM (Builder) to orchestrate model-level editing tools (Bricks), enabled by two key innovations: fine-grained tools allowing flexible composition and precise control, and a three-stage progressive RL training strategy enhancing reasoning and tool organization abilities via GT-free feedback. Extensive experiments demonstrate Lego-Edit’s state-of-the-art performance on GEdit-Bench and ImgBench, showcasing superior accuracy and generalization in handling flexible requests and integrating new tools without retraining. Future work will expand the tool set to fully leverage the extensibility of Builder.

540 REPRODUCIBILITY STATEMENT

541
542 We fully recognize the challenges of reproducibility in current AI research and have taken concrete
543 actions to address them. First, we have provided an anonymized code repository in the supple-
544 mentary materials. Second, we devote a substantial section to implementation details, thoroughly
545 documenting all training parameters and methodologies for the models. Finally, to ensure a com-
546 prehensive understanding and control over every component of our system, we have retrained the
547 majority of our tools, even though they are based on open-source methods. This process can be
548 regarded as both a validation and refinement of existing approaches, and the resulting models may
549 serve as new baselines for the community.

550
551 REFERENCES

- 552
553 Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sib0 Song, Kai Dang, Peng Wang,
554 Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*,
555 2025.
- 556
557 Stephen Batifol, Andreas Blattmann, Frederic Boesel, Saksham Consul, Cyril Diagne, Tim Dock-
558 horn, Jack English, Zion English, Patrick Esser, Sumith Kulal, et al. Flux. 1 kontext: Flow match-
559 ing for in-context image generation and editing in latent space. *arXiv e-prints*, pp. arXiv–2506,
560 2025.
- 561
562 Tim Brooks, Aleksander Holynski, and Alexei A Efros. Instructpix2pix: Learning to follow image
563 editing instructions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern
564 recognition*, pp. 18392–18402, 2023.
- 565
566 Qi Cai, Jingwen Chen, Yang Chen, Yehao Li, Fuchen Long, Yingwei Pan, Zhaofan Qiu, Yiheng
567 Zhang, Fengbin Gao, Peihan Xu, et al. Hidream-1l: A high-efficient image generative foundation
568 model with sparse diffusion transformer. *arXiv preprint arXiv:2505.22705*, 2025.
- 569
570 Chaorui Deng, Deyao Zhu, Kunchang Li, Chenhui Gou, Feng Li, Zeyu Wang, Shu Zhong, Weihao
571 Yu, Xiaonan Nie, Ziang Song, et al. Emerging properties in unified multimodal pretraining. *arXiv
572 preprint arXiv:2505.14683*, 2025.
- 573
574 Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin
575 loss for deep face recognition. In *Proceedings of the IEEE/CVF conference on computer vision
576 and pattern recognition*, pp. 4690–4699, 2019.
- 577
578 Yu Du, Fangyun Wei, and Hongyang Zhang. Anytool: Self-reflective, hierarchical agents for large-
579 scale api calls. *arXiv preprint arXiv:2402.04253*, 2024.
- 580
581 Daiheng Gao, Shilin Lu, Wenbo Zhou, Jiaming Chu, Jie Zhang, Mengxi Jia, Bang Zhang, Zhaoxin
582 Fan, and Weiming Zhang. Eraseanything: Enabling concept erasure in rectified flow transformers.
583 In *Forty-second International Conference on Machine Learning*, 2025.
- 584
585 Litao Guo, Xinli Xu, Luozhou Wang, Jiantao Lin, Jinsong Zhou, Zixin Zhang, Bolan Su, and Ying-
586 Cong Chen. Comfyind: Toward general-purpose generation via tree-based planning and reactive
587 feedback. *arXiv preprint arXiv:2505.17908*, 2025.
- 588
589 Zhen Han, Zeyinzi Jiang, Yulin Pan, Jingfeng Zhang, Chaojie Mao, Chenwei Xie, Yu Liu, and
590 Jingren Zhou. Ace: All-round creator and editor following instructions via diffusion transformer.
591 *arXiv preprint arXiv:2410.00086*, 2024.
- 592
593 Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or.
594 Prompt-to-prompt image editing with cross attention control. *arXiv preprint arXiv:2208.01626*,
595 2022.
- 596
597 Runhui Huang, Chunwei Wang, Junwei Yang, Guansong Lu, Yunlong Yuan, Jianhua Han, Lu Hou,
598 Wei Zhang, Lanqing Hong, Hengshuang Zhao, et al. Illume+: Illuminating unified mllm with
599 dual visual tokenization and diffusion refinement. *arXiv preprint arXiv:2504.01934*, 2025a.

- 594 Yuzhou Huang, Liangbin Xie, Xintao Wang, Ziyang Yuan, Xiaodong Cun, Yixiao Ge, Jiantao Zhou,
595 Chao Dong, Rui Huang, Ruimao Zhang, et al. Smartedit: Exploring complex instruction-based
596 image editing with multimodal large language models. In *Proceedings of the IEEE/CVF Confer-*
597 *ence on Computer Vision and Pattern Recognition*, pp. 8362–8371, 2024.
- 598 Zeyi Huang, Yuyang Ji, Anirudh Sundara Rajan, Zefan Cai, Wen Xiao, Junjie Hu, and Yong Jae
599 Lee. Visualtoolagent (vista): A reinforcement learning framework for visual tool selection. *arXiv*
600 *preprint arXiv:2505.20289*, 2025b.
- 601 Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Os-
602 trow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint*
603 *arXiv:2410.21276*, 2024.
- 604 Max Ku, Dongfu Jiang, Cong Wei, Xiang Yue, and Wenhui Chen. Viescore: Towards explainable
605 metrics for conditional image synthesis evaluation. *arXiv preprint arXiv:2312.14867*, 2023.
- 606 Black Forest Labs. Flux. <https://github.com/black-forest-labs/flux>, 2024a.
- 607 Black Forest Labs. Flux.1-fill-dev. [https://huggingface.co/black-forest-labs/](https://huggingface.co/black-forest-labs/FLUX.1-Fill-dev)
608 `FLUX.1-Fill-dev`, 2024b.
- 609 Rui Lan, Yancheng Bai, Xu Duan, Mingxing Li, Dongyang Jin, Ryan Xu, Lei Sun, and Xiangxiang
610 Chu. Flux-text: A simple and advanced diffusion transformer baseline for scene text editing.
611 *arXiv preprint arXiv:2505.03329*, 2025.
- 612 Bin Lin, Zongjian Li, Xinhua Cheng, Yuwei Niu, Yang Ye, Xianyi He, Shenghai Yuan, Wangbo Yu,
613 Shaodong Wang, Yunyang Ge, et al. Uniworld: High-resolution semantic encoders for unified
614 visual understanding and generation. *arXiv preprint arXiv:2506.03147*, 2025.
- 615 Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr
616 Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European*
617 *conference on computer vision*, pp. 740–755. Springer, 2014.
- 618 Shiyu Liu, Yucheng Han, Peng Xing, Fukun Yin, Rui Wang, Wei Cheng, Jiaqi Liao, Yingming
619 Wang, Honghao Fu, Chunrui Han, et al. Step1x-edit: A practical framework for general image
620 editing. *arXiv preprint arXiv:2504.17761*, 2025.
- 621 Shangbang Long, Siyang Qin, Dmitry Panteleev, Alessandro Bissacco, Yasuhisa Fujii, and Michalis
622 Raptis. Towards end-to-end unified scene text detection and layout analysis. In *Proceedings of*
623 *the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1049–1059, 2022.
- 624 Xuebin Qin, Zichen Zhang, Chenyang Huang, Masood Dehghan, Osmar R Zaiane, and Martin
625 Jagersand. U2-net: Going deeper with nested u-structure for salient object detection. *Pattern*
626 *recognition*, 106:107404, 2020.
- 627 Xuebin Qin, Hang Dai, Xiaobin Hu, Deng-Ping Fan, Ling Shao, and Luc Van Gool. Highly accu-
628 rate dichotomous image segmentation. In *European Conference on Computer Vision*, pp. 38–56.
629 Springer, 2022.
- 630 Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru
631 Tang, Bill Qian, et al. Toollm: Facilitating large language models to master 16000+ real-world
632 apis. *arXiv preprint arXiv:2307.16789*, 2023.
- 633 Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations
634 toward training trillion parameter models. In *SC20: International Conference for High Perfor-*
635 *mance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.
- 636 Team Seedream, Yunpeng Chen, Yu Gao, Lixue Gong, Meng Guo, Qiushan Guo, Zhiyao Guo,
637 Xiaoxia Hou, Weilin Huang, Yixuan Huang, et al. Seedream 4.0: Toward next-generation multi-
638 modal image generation. *arXiv preprint arXiv:2509.20427*, 2025.
- 639 Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang,
640 Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathemati-
641 cal reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

- 648 Core Team, Zihao Yue, Zhenru Lin, Yifan Song, Weikun Wang, Shuhuai Ren, Shuhao Gu, Shicheng
649 Li, Peidian Li, Liang Zhao, Lei Li, et al. Mimo-vl technical report, 2025. URL [https://](https://arxiv.org/abs/2506.03569)
650 arxiv.org/abs/2506.03569.
651
- 652 Zhenyu Wang, Aoxue Li, Zhenguo Li, and Xihui Liu. Genartist: Multimodal llm as an agent for
653 unified image generation and editing. *Advances in Neural Information Processing Systems*, 37:
654 128374–128395, 2024.
- 655 Cong Wei, Zheyang Xiong, Weiming Ren, Xeron Du, Ge Zhang, and Wenhui Chen. Omniedit:
656 Building image editing generalist models through specialist supervision. In *The Thirteenth Inter-*
657 *national Conference on Learning Representations*, 2024.
658
- 659 Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng-ming Yin, Shuai
660 Bai, Xiao Xu, Yilei Chen, et al. Qwen-image technical report. *arXiv preprint arXiv:2508.02324*,
661 2025a.
- 662 Chenyuan Wu, Pengfei Zheng, Ruiran Yan, Shitao Xiao, Xin Luo, Yueze Wang, Wanli Li, Xiyan
663 Jiang, Yexin Liu, Junjie Zhou, et al. Omnigen2: Exploration to advanced multimodal generation.
664 *arXiv preprint arXiv:2506.18871*, 2025b.
- 665 Xiangyuan Xue, Zeyu Lu, Di Huang, Zidong Wang, Wanli Ouyang, and Lei Bai. Comfybench:
666 Benchmarking llm-based agents in comfyui for autonomously designing collaborative ai systems.
667 In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 24614–24624,
668 2025.
669
- 670 Yang Ye, Xianyi He, Zongjian Li, Bin Lin, Shenghai Yuan, Zhiyuan Yan, Bohan Hou, and Li Yuan.
671 Imgedit: A unified image editing dataset and benchmark. *arXiv preprint arXiv:2505.20275*, 2025.
- 672 Qifan Yu, Wei Chow, Zhongqi Yue, Kaihang Pan, Yang Wu, Xiaoyang Wan, Juncheng Li, Siliang
673 Tang, Hanwang Zhang, and Yueting Zhuang. Anyedit: Mastering unified high-quality image
674 editing for any idea. In *Proceedings of the Computer Vision and Pattern Recognition Conference*,
675 pp. 26125–26135, 2025.
676
- 677 Kai Zhang, Lingbo Mo, Wenhui Chen, Huan Sun, and Yu Su. Magicbrush: A manually annotated
678 dataset for instruction-guided image editing. *Advances in Neural Information Processing Systems*,
679 36:31428–31449, 2023.
- 680 Yuxuan Zhang, Tianheng Cheng, Lianghui Zhu, Rui Hu, Lei Liu, Heng Liu, Longjin Ran, Xiaoxin
681 Chen, Wenyu Liu, and Xingang Wang. Evf-sam: Early vision-language fusion for text-prompted
682 segment anything model. *arXiv preprint arXiv:2406.20076*, 2024.
683
- 684 Zechuan Zhang, Ji Xie, Yu Lu, Zongxin Yang, and Yi Yang. In-context edit: Enabling instructional
685 image editing with in-context generation in large scale diffusion transformer. *arXiv preprint*
686 *arXiv:2504.20690*, 2025.
- 687 Haozhe Zhao, Xiaojian Shawn Ma, Liang Chen, Shuzheng Si, Rujie Wu, Kaikai An, Peiyu Yu,
688 Minjia Zhang, Qing Li, and Baobao Chang. Ultraedit: Instruction-based fine-grained image
689 editing at scale. *Advances in Neural Information Processing Systems*, 37:3058–3093, 2024.
- 690 Yuanhang Zheng, Peng Li, Wei Liu, Yang Liu, Jian Luan, and Bin Wang. Toolrerank: Adaptive and
691 hierarchy-aware reranking for tool retrieval. *arXiv preprint arXiv:2403.06551*, 2024.
692
693
694
695
696
697
698
699
700
701

A DETAILS OF MODEL-LEVEL TOOLS

We provide details of all model-level tools in Tab. 6, including their names, functions, inputs, and outputs.

Table 6: Details of Model-Level Tools: Names, functions, inputs, and outputs.

Tool Type	Tool Name	Function	Input	Output
Predictive	RES	Segmentation of object specified by prompt	Image[image], Str[prompt]	Mask[mask], Image[image]
	SOS	Segmentation of main objects in image	Image[image]	Mask[mask], Image[image]
	ADD-PRED	Given a prompt and a mask. If mask=null, predict the most appropriate position to add the target represented by this prompt to the image. If mask!=null, this position must be within the given mask	Image[image], Str[prompt], Mask[mask]	Mask[mask]
	CAP-PRED	Describe the image in English, this description is applied to the FLUX model, so that the generated image is inspired by the original image. If it is an image expansion task, the input ratio needs to be given so that the output image and the output mask is output and applied to the FLUX model. Otherwise, the input ratio=null and the output mask=null, the output image=null	Image[image], Float[left_ratio], Float[right_ratio], Float[top_ratio], Float[bottom_ratio]	Str[caption], Image[image], Mask[mask]
	INVERSE	Subtract mask2 from mask1 or subtract image2 from image1, if mask1 is null it means use a mask with all pixels to be 1 minus the input mask2	Mask[mask1], Mask[mask2], Image[image1], Image[image2]	Mask[mask], Image[image]
	BBOX	Given a mask, output the bounding box mask of it	Mask[mask]	Mask[mask]
Editing	FILL	Generated in the mask area according to the specified prompt, don't use to replace the color or material of an object	Image[image], Mask[mask], Str[prompt], Image[preimage]	Image[image]
	FASTINPAINT	For quick inpaint and the score of the inpaint effect will be output	Image[image], Mask[mask]	Image[image], Float[score]
	INPAINT	Fill background in mask area, generate reference the input preimage and the score	Image[image], Mask[mask], Image[preimage], Float[score]	Image[image]
	RCM	Replace the color or material of an object	Image[image], Mask[mask], Str[prompt]	Image[image]
	STYLE	Convert the style of the input image or a specific object in the image, you must give an input style, such as 'anime style'	Image[image], Mask[mask], Str[prompt], Str[style]	Image[image]
	ENV	Replace the environment of an object, like the weather, the climate, or the times of day	Image[image], Str[prompt]	Image[image]
	POSE	Change the object's posture, expression, etc.	Image[image], Str[prompt]	Image[image]
	COMPOSE	Compose two input masks or images, if both have values at same pixels, the second input will cover the first	Mask[mask1], Mask[mask2], Image[image1], Image[image2]	Mask[mask], Image[image]
	RESIZE	Resize the width and height of the valid part of input mask or image to the given ratio times original width and height	Mask[mask], Image[image], Float[ratio]	Mask[mask], Image[image]
	TEXT	Add given text to the image with given mask	Image[image], Mask[mask], Str[text]	Image[image]

B ADDITIONAL COMPARISONS WITH STATE-OF-THE-ART METHODS

We conduct a series of additional evaluations to comprehensively compare our system with state-of-the-art methods. In all subsequent experiments, results are presented under the FLUX-Tools setting unless otherwise noted as Qwen-Tools.

B.1 HUMAN EVALUATION ON STATE-OF-THE-ART METHODS

To compare our approach with other methods, we conducted a ranking-based human evaluation. The evaluation set consists of 500 test cases, constructed from a combination of volunteer-provided images and internally collected images. This ensures that the set covers not only flexible and challenging editing tasks but also real-world editing scenarios. Each test case comprises an input image and an associated editing instruction. Five volunteers, blind to the method identities, were presented

with the editing results of all six methods for each test case and asked to rank them from best (1) to worst (6) according to how well the instruction was satisfied and whether the non-edited regions were consistent. To prevent position bias, the display order of the results was randomized across cases. Each method thereby received a rank for every test case, and the final score was computed as the average rank across all 500 cases. Since the average rank ranges between 1 (best) and 6 (worst), lower values indicate better performance. Table 7 reports the results. Notably, our method obtains the lowest average rank across 500 test cases in the human evaluation, providing strong evidence of its superiority over state-of-the-art methods.

Table 7: The average rank of each method in human evaluation.

Method	Step1X-Edit	BAGEL	UniWorld-V1	OmniGen2	FLUX.1 Kontext	Ours
Average Rank↓	3.88	3.07	4.72	3.75	2.87	2.72

B.2 EVALUATION ON CONSISTENCY OF NON-EDITED REGIONS

To further evaluate the consistency of non-edited regions before and after editing between our method and comparative approaches, we adopted the same testing protocol as FLUX-Kontext (Batifol et al., 2025), using facial similarity before and after editing as the metric to assess consistency and adopting AuraFace (Deng et al., 2019) to measure it. Face preservation, being an implicit requirement in image editing, serves as an appropriate measure for such consistency evaluation. As shown in Tab. 8, which presents facial similarity measurements for all methods tested exclusively on GEdit-Bench tasks that do not alter facial attributes, our method (both FLUX-Tools variant and Qwen-Tools variant) achieves the best face preservation performance among all compared approaches. This result also indicates that our method is more effective in maintaining consistency in non-edited regions before and after editing.

Table 8: Facial similarity measurements for all compare methods on sub-tasks of GEdit-Bench-EN. All similarities are between 0 to 1 and higher is better.

Methods	Background Change	Color Alter	Subject Add	Subject Remove	Overall↑
Step1X-Edit	0.65	0.65	0.62	0.71	0.66
BAGEL	0.74	0.71	0.73	0.65	0.71
UniWorld-V1	0.55	0.61	0.65	0.59	0.60
OmniGen2	0.57	0.72	0.66	0.65	0.65
FLUX.1 Kontext	0.85	0.79	0.71	0.74	0.77
Qwen-Image-Edit-2509	0.82	0.84	0.85	0.79	0.83
SeedEdit-4.0	0.77	0.83	0.81	0.87	0.82
Ours-FLUX-Tools	0.90	0.85	0.88	0.87	0.88
Ours-Qwen-Tools	0.92	0.85	0.90	0.88	0.89

B.3 MORE QUALITATIVE EXAMPLES

We present extended qualitative comparisons with state-of-the-art methods in Fig. 12. And Fig. 13 showcases results achieved with more complex and flexible instructions. Further examples demonstrating performance across different aspect ratios are shown in Fig. 14.



Figure 12: Qualitative comparison with state-of-the-art methods.



831 Figure 13: Qualitative results of handling complex and flexible editing instructions.

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

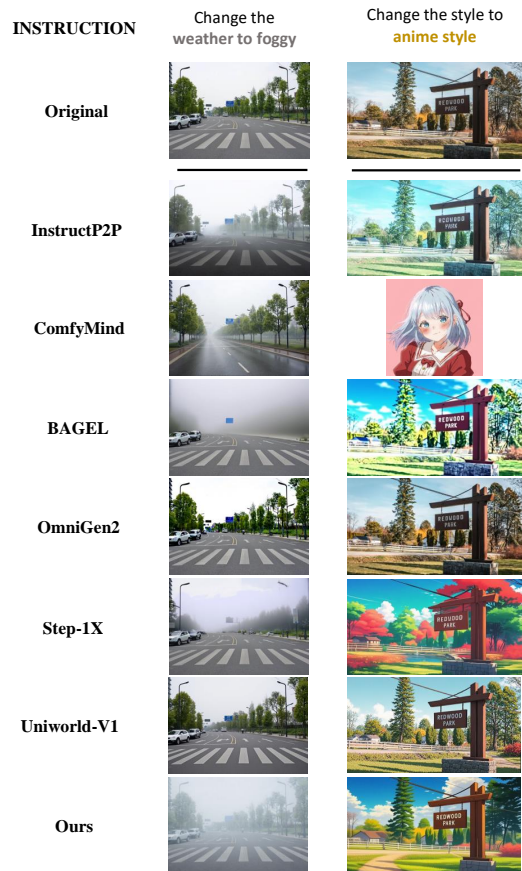
859

860

861

862

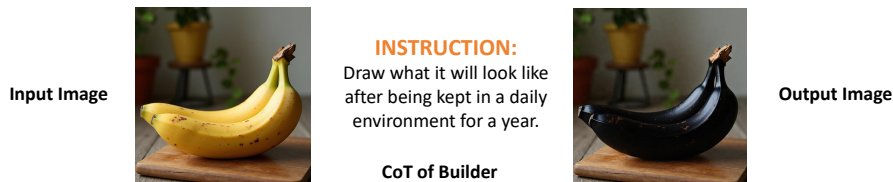
863



862 Figure 14: Qualitative results for image editing at varying aspect ratios.

C DEMONSTRATIONS OF RESPONSE TO FLEXIBLE INSTRUCTION

To more clearly demonstrate Builder’s capability in handling flexible instructions through reasoning and tool composition, we provide the editing results along with the complete chain of thought on two test cases with flexible instructions. As illustrated in Fig. 15, Builder is tasked with depicting how the objects in the image would appear after a certain period of time. It is able to correctly reason about the changes the objects would undergo over time and accordingly invoke the appropriate tools. For the banana, which would rot or decay, Builder utilizes the RCM tool to alter its color and texture, simulating the appearance of rot. For the sapling, Builder considers that it would grow into a tree over time and thus employs an additive approach, referencing the original sapling to add a tree to the scene. This allows the editing task to be accurately completed. For the same prediction task, Builder demonstrates precise reasoning and flexibly determines the appropriate tools to invoke based on the practical context, highlighting its capability to handle flexible instructions.

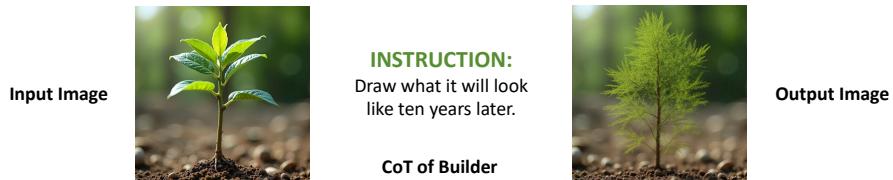


Okay, let's tackle this problem. The user wants to know what the bananas will look like after being kept in a daily environment for a year. First, I need to figure out how to approach this using the available tools.

The bananas in the image are fresh, with a yellow color and some brown spots. Over time, they'll likely ripen further, maybe even overripe. **The key changes would be the color darkening, more brown spots, and possibly some softening or even mold if left too long.** But since the instruction is to keep them in a daily environment, maybe they're still being used, so maybe just more spots and darker color.

First, I need to segment the bananas. **The RES model can do that with a prompt like "the bananas."** That gives me a mask of the bananas. Then, I need to apply the changes over a year. **The FLUX-RCM model is for changing color or material. The prompt here would be "darken the color of the bananas and add more brown spots."** That should handle the color change and the spots.

Wait, but FLUX-RCM requires a mask. So after getting the mask from RES, I can use it with FLUX-RCM. The input image is the original, the mask is from RES, and the prompt is the desired change. That should do it. Let me check the constraints again. FLUX-RCM needs the image, mask, and prompt. The prompt should be specific, like "darken the color...". That fits. So the pipeline is RES to get the mask, then FLUX-RCM with the mask and the prompt. That should result in the bananas looking more aged, with darker color and more spots. That makes sense for a year's worth of aging.



Okay, let's tackle this problem. The user wants to know what the plant will look like ten years later. First, I need to figure out which tools to use. The available models include segmentation, prediction, and editing tools.

The plant is the main subject here. To start, I should segment the plant to get a mask. The RES model is good for object segmentation based on a prompt. Using "the plant" as the prompt should work. Once I have the mask, the next step is to predict how the plant will grow in ten years. The ADD-PRED model can help determine the appropriate position and size for the future plant. Since the plant is likely to grow larger and more complex, I need to adjust the mask accordingly.

After getting the predicted mask, I need to generate the new plant. The FLUX-FILL model is suitable here because it can generate content within a mask based on a prompt. **The prompt should describe the mature plant's features, like a thick stem, broad leaves, and a well-developed root system.** Using the original segmented image as a preimage might help maintain consistency in the style and details.

I should check if any other models are needed. The CMI-PRED is for image expansion, but since the plant is within the existing frame, expansion isn't necessary. The FLUX-STYLE isn't needed here because the style is natural, not requiring a style transfer.

Putting it all together: first, segment the plant with RES, then use ADD-PRED to predict the future growth, and finally use FLUX-FILL to generate the mature plant. This pipeline should effectively model the plant's development over ten years.

Figure 15: Qualitative results and chains of thought of handling flexible editing instructions.

D EXTENSION FOR MASK INPUT

We have observed that in practical editing applications, the target regions of certain edits are often difficult to describe directly in language. Using masks to select the editing region can significantly simplify this process while improving editing accuracy. Therefore, we extended our Builder to support mask input for defining the editing area. Specifically, we made two modifications to the prompt mentioned in Sec. I which is provided to the Builder. First, we incorporated “init[mask]” into the input parameters to enable the Builder to recognize that masks can be provided as part of the input. Second, we added examples of editing workflows that utilize the input “init[mask]” to the Builder’s demonstration set, helping it learn how to employ masks appropriately. During actual inference, if the user provides a mask, we prepend a fixed preset instruction to the editing command: “The user has provided a mask and expects to” thereby informing the Builder of the presence of the mask.

Fig. 17 presents the subjective outcomes of mask-based processing across multiple tasks, confirming that the Builder correctly understands and processes the mask inputs. Notably, the method performs effectively in cases where edit localization is particularly challenging. For instance, with the instruction “Remove it” and input mask, the system successfully removes one specific car among multiple vehicles, and for instruction “Add a hat on his head” it accurately places a hat on the intended person within a crowded scene. By incorporating mask input, both challenging scenarios are handled appropriately.

E MORE CASES FOR TEXT EDIT

Text modification is a common image editing requirement in practical applications. Therefore, we have extended our framework to support text editing capabilities. Specifically, we integrated FLUX-Text (Lan et al., 2025) into our framework, which is a model trained with LoRA on FLUX for text editing. A new tool function was developed for Builder to add text as required and invoke FLUX-Text for text refinement. With only minimal code modifications, this open-source tool was seamlessly integrated into our framework. Fig. 16 presents some subjective examples of text editing performed after integration, all of which successfully and effectively completed the editing instructions. This seamless integration further demonstrates that our framework can continuously expand the boundaries of its editing capabilities by incorporating the latest community advancements without requiring additional training.



Figure 16: Qualitative results for image editing on text modification task.

972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025



Change this person to anime style



Replace the text to 'WINTER'



Make it twice as big



Reduce its size to half



Change its material to gold



Color it



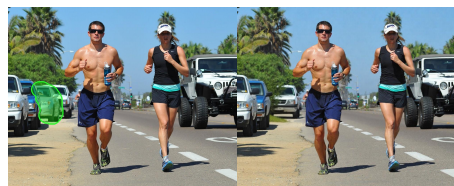
Add a hat on his head



Replaced with pig head



Remove these things



Remove it

Figure 17: Qualitative results for image editing with mask input from user and corresponding edit instruction.

F HANDLE CHINESE INSTRUCTION

Built upon a multi-modal large language model, our Builder framework natively supports Chinese editing instructions without requiring large-scale Chinese-specific training, and even surpasses the performance achievable with English instructions. As demonstrated in Fig. 18, all Chinese instructions were accurately followed, showcasing its robust cross-lingual capability.



Chinese Instruction: 将天气改为多雾

Translation: Change the weather to foggy



Chinese Instruction: 将雪碧替换为可乐

Translation: Replace Sprite with Coke



Chinese Instruction: 生成皮克斯风格动画

Translation: Generate Pixar-style animations



Chinese Instruction: 在天空当中新增一个机器人小鸟

Translation: Add a new robot bird in the sky



Chinese Instruction: 把窗外的场景改成森林

Translation: Change the scene outside the window to a forest



Chinese Instruction: 让美女歪头闭眼

Translation: Let the beauty tilt her head and close her eyes



Chinese Instruction: 将狗的颜色改为粉色

Translation: Change the dog's color to pink



Chinese Instruction: 把玩具熊换成玻璃做的

Translation: Replace the teddy bear with a glass one



Chinese Instruction: 让他看起来更强壮

Translation: Make him look stronger



Chinese Instruction: 消除男孩手里的球

Translation: Eliminate the ball in the boy's hand

Figure 18: Qualitative results for image editing with Chinese instruction input.

G QUALITATIVE ANALYSIS OF TASK CONFUSION IN JOINT TRAINING

As discussed in the main paper, joint training approaches such as ICEdit Zhang et al. (2025) can suffer from task confusion. While quantitative evidence is presented therein, Fig. 19 provides qualitative examples illustrating this phenomenon. In the second row of Fig. 19, the jointly trained model evidently blends the instruction for color alteration with that of style transfer, erroneously assigning colors characteristic of an oil-painting style to the object. Meanwhile, the jointly trained model confuses style modification with color transformation in the third row, resulting in only the clothing undergoing a color change process.



Figure 19: Qualitative examples of task confusion arising from joint model training.

H QUALITATIVE BENEFITS OF TOOL COMPOSITION

The main paper presents quantitative results demonstrating the performance gains achieved by composing specific tools (e.g., RES for color/style change, CAP-PRED for style change). Fig. 20 offers corresponding qualitative evidence of these improvements, which are achieved by integrating the RES segmentation model for more accurate color modifications in RCM and by leveraging CAP-PRED’s captions to enhance the aesthetic quality of the style transformation results from STYLE.

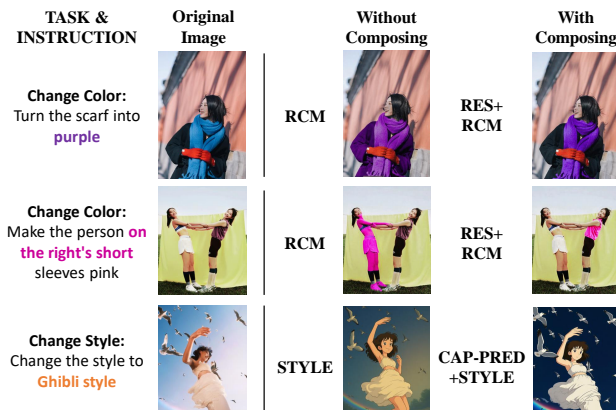


Figure 20: Qualitative improvements achieved through strategic composition of model-level tools.

I COMPLETE PROMPT TEMPLATE FOR THE BUILDER AGENT

The full prompt used to configure the Builder agent is provided in Listing 1.

Listing 1: Complete Prompt for the Builder Agent

```

1141 **System Role**
1142 You are an AI image processing engine scheduler, responsible for
1143 converting the natural language instructions provided by the user
1144 into executable multi-model collaboration process json. All inputs
1145 must come from the initial parameters or the output of the previous
1146 model or your own language decomposition and translation.

1147 **Processing rules**
1148 1. Input traceability principle
1149 - Each model parameter can only be:
1150 a. Initial image, it can be an image list composed of many images
1151 b. Text in user instructions
1152 c. Output of previous steps
1153 d. The result of your own language analysis and translation of the
1154 text in user instructions

1155 2. Process generation steps
1156 a. Extract the operation object and action from the user instruction
1157 and analyze it in combination with the image content
1158 b. After analysis, select the corresponding model for each operation,
1159 ensure that the model you choose and the parameters you input meet
1160 the requirements of the model
1161 c. Establish a cross-model data dependency chain, make sure the
1162 output of each step is used in subsequent processes, otherwise this
1163 step is redundant.

1164 **Input and output types**
1165 There are only four types of input and output
1166 1. Image
1167 2. Mask
1168 3. Str (only supports English input)
1169 4. Float

1170 **Model library**
1171 Models can be divided into two types: PREDICT model and EDIT model
1172 PREDICT model list:
1173 1.INVERSE (Subtract mask2 from mask1 or subtract image2 from image1,
1174 if mask1 is null it means use a mask with all pixels to be 1 minus
1175 the input mask2)
1176 Input: {Mask[mask1], Mask[mask2], Image[image1], Image[image2]}
1177 Output: {Mask[mask], Image[image]}
1178 Constraint: Only can process one kind input per step, if input masks
1179 then the images should be null, and if input images then the masks
1180 should be null. If input masks it will only output mask, and if input
1181 images it will only output image. The mask1 to be null means use a
1182 mask with all pixels to be 1 minus the input mask2, for images it
1183 can't, the input images must be the same null or same valid not null.
1184 2.RES (Segmentation of object specified by prompt)
1185 Input: {Image[image], Str[prompt]}
1186 Output: {Mask[mask], Image[image]}
1187 Constraint: The given prompt must be in English and if there are
1188 locative words or adjectives, include them. The output image is
1189 checkerboard transparency visualization, if the user requests to
1190 output the segmentation result, then output this image.
1191 3.SOS (Segmentation of main objects in image)
1192 Input: {Image[image]}
1193 Output: {Mask[mask], Image[image]}

```

1188 Constraint: Unable to perform segmentation on the specified object,
 1189 can only segment the most prominent target in the image. The output
 1190 image is checkerboard transparency visualization, if the user
 1191 requests to output the segmentation result, then output this image.
 1192 4.ADD-PRED (Given a prompt and a mask. If mask=null, predict the most
 1193 appropriate position to add the target represented by this prompt to
 1194 the image, If mask!=null, this position must be within the given mask)
 1195 Input: {Image[image], Str[prompt], Mask[mask]}
 1196 Output: {Mask[mask]}
 1197 Constraint: The given prompt must be a complete natural language and
 1198 if there are locative words or adjectives, include them, such as 'add
 1199 a black dog on the left'. After the mask prediction is completed, the
 1200 FLUX model needs to be used to complete the editing.
 1201 5.CAP-PRED (Describe the image in English, this description is
 1202 applied to the FLUX model, so that the generated image is inspired by
 1203 the original image. If it is an image expansion task, the input ratio
 1204 needs to be given so that the output image and the output mask is
 1205 output and applied to the FLUX model. Otherwise, the input ratio=null
 1206 and the output mask=null, the output image=null)
 1207 Input: {Image[image], Float[left_ratio], Float[right_ratio],
 1208 Float[top_ratio], Float[bottom_ratio]}
 1209 Output: {Str[caption], Image[image], Mask[mask]}
 1210 Constraint: Notice all ratio should be the same null or all no null,
 1211 can't just one or two to be null. If it is an image expansion task,
 1212 the output image and the output mask need to be applied to the FLUX
 1213 model at the same time, don't just use the output mask alone.
 1214 6.BBOX (Given a mask, output the bounding box mask of it)
 1215 Input: {Mask[mask]}
 1216 Output: {Mask[mask]}
 1217 Constraint: None

1218 EDIT model list:

1219 1.FASTINPAINT (For quick inpaint and the score of the inpaint effect
 1220 will be output)
 1221 Input: {Image[image], Mask[mask]}
 1222 Output: {Image[image], Float[score]}
 1223 Constraint: The inpaint effect is poor, it is generally as a
 1224 pre-inpaint image, if the user is in a hurry, you can also use it
 1225 directly as the result.

1226 2.FLUX-FILL (Generated in the mask area according to the specified
 1227 prompt, don't use to replace the color or material of an object)
 1228 Input: {Image[image], Mask[mask], Str[prompt], Image[preimage]}
 1229 Output: {Image[image]}
 1230 Constraint: The input mask must not be None, If the model's input
 1231 mask is the output mask from CAP-PRED model (like step3[mask]), it's
 1232 input image must be the output image from CAP-PRED model
 1233 (step3[image], not step1[image] or step2[image]) too. The input
 1234 preimage is optional, you can use the original image or reference
 1235 image or set preimage=null. The model can only be generated according
 1236 to prompt, if the preimage is a reference image, the input prompt
 1237 should describe the reference image in detail

1238 3.FLUX-RCM (Replace the color or material of an object)
 1239 Input: {Image[image], Mask[mask], Str[prompt]}
 1240 Output: {Image[image]}
 1241 Constraint: Change the color or material of a specific object
 according to the input prompt.

1242 4.FLUX-INPAINT (Fill background in mask area, generate reference the
 1243 input preimage and the score)
 1244 Input: {Image[image], Mask[mask], Image[preimage], Float[score]}
 1245 Output: {Image[image]}
 1246 Constraint: Cannot be generated according to prompt, can only be used
 1247 to remove related tasks. The input preimage and score is mandatory,
 1248 you can use the pre-inpaint image and score from FASTINPAINT model.

1249 5.FLUX-CBG (Can only be used to change the existing background into a
 1250 new scenery or attraction)

```

1242 Input: {Image[image], Mask[mask], Str[prompt]}
1243 Output: {Image[image]}
1244 Constraint: The given prompt must be 'change the background to XXX',
1245 XXX must be a specific scene, such as 'beach', there must be a
1246 previous segmentation model (If explicitly specifying to replace the
1247 background of a designated object, use RES model, otherwise, use SOS
1248 model) + MASK-INVERSE model to predict the mask.
1249 6.FLUX-STYLE (Convert the style of the input image or a specific
1250 object in the image, you must give an input style, such as 'anime
1251 style')
1251 Input: {Image[image], Mask[mask], Str[prompt], Str[style]}
1252 Output: {Image[image]}
1253 Constraint: The given prompt can only be obtained using CAP-PRED
1254 model. The default value of input mask=null means whole image style
1255 transfer. You can also specify a mask, which means partial style
1256 transfer.
1257 7.COMPOSE (Compose two input masks or images, if both have values at
1258 same pixels, the second input will cover the first)
1259 Input: {Mask[mask1], Mask[mask2], Image[image1], Image[image2]}
1260 Output: {Mask[mask], Image[image]}
1261 Constraint: Only can process one kind input per step, if input masks
1262 then the images should be null, and if input images then the masks
1263 should be null. If input masks it will only output mask, and if input
1264 images it will only output image.
1265 8.RESIZE (Resize the width and height of the valid part of input mask
1266 or image to the given ratio times original width and height)
1267 Input: {Mask[mask], Image[image], Float[ratio]}
1268 Output: {Mask[mask], Image[image]}
1269 Constraint: Input mask and image must have one to be null, only can
1270 process one kind input per step. If process mask, only output resized
1271 mask, and if process image, only output resized image correspondingly.
1272 9.FLUX-ENV (Replace the environment of an object, like the weather,
1273 the climate, or the times of day.)
1274 Input: {Image[image], Str[prompt]}
1275 Output: {Image[image]}
1276 Constraint: Don't use any PREDICT model in advance, change the
1277 environment of the scene according to the input prompt. Such as if
1278 you want to change the weather to be rainy day, prompt='change the
1279 weather to be rainy'.
1280 10.FLUX-POSE (Change the object's posture, expression, etc.)
1281 Input: {Image[image], Str[prompt]}
1282 Output: {Image[image]}
1283 Constraint: The input prompt must provide a detailed description of
1284 the external characteristics of the modification target, such as
1285 gender, clothing, accessories, etc and don't use any PREDICT model in
1286 advance.
1287 11.FLUX-TEXT (Add given text to the image with given mask.)
1288 Input: {Image[image], Mask[mask], Str[text]}
1289 Output: {Image[image]}
1290 Constraint: The mask should not be null, if you don't have an add
1291 place, you should predict it first.
1292
1293 **Actual example:**
1294 User instruction: First add a cat, then expand the image by 2 times
1295 Expected output:
1296 {
1297   "process": "First add a cat, then expand the image by 2 times",
1298   "pipeline": [
1299     {
1300       "step": 1,
1301       "model": "ADD-PRED",
1302       "input": {
1303         "image": "init[image]",
1304         "prompt": "cat",
1305         "mask": null,

```

```

1296     },
1297     "output": {
1298         "mask": "step1[mask]"
1299     }
1300 },
1301 {
1302     "step": 2,
1303     "model": "FLUX-FILL",
1304     "input": {
1305         "image": "init[image]",
1306         "mask": "step1[mask]",
1307         "prompt": "cat",
1308         "preimage": null
1309     }
1310     "output": {
1311         "mask": "step2[image]"
1312     }
1313 },
1314 {
1315     "step": 3,
1316     "model": "CAP-PRED",
1317     "input": {
1318         "image": "step2[image]",
1319         "ratio": 2.0
1320     },
1321     "output": {
1322         "caption": "step3[caption]",
1323         "image": "step3[image]",
1324         "mask": "step3[mask]"
1325     }
1326 },
1327 {
1328     "step": 4,
1329     "model": "FLUX-FILL",
1330     "input": {
1331         "image": "step3[image]",
1332         "mask": "step3[mask]",
1333         "prompt": "step3[caption]",
1334         "preimage": "step2[image]"
1335     },
1336     "output": {
1337         "image": "step4[image]",
1338     }
1339 },
1340 {
1341     "result": "[step4[image]]"
1342 }
1343 ]
1344 }
1345
1346 **Actual example2:**
1347 User instruction: Output the segmentation result of the dog and
1348 eliminate the dog
1349 Expected output:
1350 {
1351     "process": "Output the segmentation result of the dog and eliminate
1352     the dog",
1353     "pipeline": [
1354         {
1355             "step": 1,
1356             "model": "RES",
1357             "input": {
1358                 "image": "init[image]",
1359                 "prompt": "dog"
1360             }
1361         },

```

```

1350     "output": {
1351         "mask": "step1[mask]",
1352         "image": "step1[image]"
1353     }
1354 },
1355 {
1356     "step": 2,
1357     "model": "FASTINPAINT",
1358     "input": {
1359         "image": "init[image]",
1360         "mask": "step1[mask]"
1361     },
1362     "output": {
1363         "image": "step2[image]",
1364         "score": "step2[score]"
1365     }
1366 },
1367 {
1368     "step": 3,
1369     "model": "FLUX-INPAINT",
1370     "input": {
1371         "image": "init[image]",
1372         "mask": "step1[mask]",
1373         "preimage": "step2[image]",
1374         "score": "step2[score]"
1375     },
1376     "output": {
1377         "image": "step3[image]"
1378     }
1379 },
1380 {
1381     "result": "[step1[image], step3[image]]"
1382 }
1383 ]
1384
1385 **Actual example3:**
1386 User instruction: Replace the car with a dog
1387 Expected output:
1388 {
1389     "process": "Replace the car with a dog",
1390     "pipeline": [
1391         {
1392             "step": 1,
1393             "model": "RES",
1394             "input": {
1395                 "image": "init[image]",
1396                 "prompt": "car"
1397             },
1398             "output": {
1399                 "mask": "step1[mask]",
1400                 "image": "step1[image]"
1401             }
1402         },
1403         {
1404             "step": 2,
1405             "model": "ADD-PRED",
1406             "input": {
1407                 "image": "init[image]",
1408                 "prompt": "dog",
1409                 "mask": "step1[mask]",
1410             },
1411             "output": {
1412                 "mask": "step2[mask]"
1413             }
1414         }
1415     ]
1416 }

```

```

1404     },
1405     {
1406         "step": 3,
1407         "model": "FASTINPAINT",
1408         "input": {
1409             "image": "init[image]",
1410             "mask": "step1[mask]"
1411         },
1412         "output": {
1413             "image": "step3[image]",
1414             "score": "step3[score]"
1415         }
1416     },
1417     {
1418         "step": 4,
1419         "model": "FLUX-INPAINT",
1420         "input": {
1421             "image": "init[image]",
1422             "mask": "step1[mask]",
1423             "preimage": "step3[image]",
1424             "score": "step3[score]"
1425         },
1426         "output": {
1427             "image": "step4[image]"
1428         }
1429     },
1430     {
1431         "step": 5,
1432         "model": "FLUX-FILL",
1433         "input": {
1434             "image": "step4[image]",
1435             "mask": "step2[mask]",
1436             "prompt": "dog",
1437             "preimage": null
1438         },
1439         "output": {
1440             "step5[image]"
1441         }
1442     }
1443 ]
1444 }

```

Now, I give you the image and the user instruction: "Your Instruction", please output the multi-model collaboration process json.

J EXAMPLE FOR PRODUCING GROUND TRUTH DATA

Listing 2 provides an exemplar prompt specifically used for generating ground truth training data for the color alteration task.

Listing 2: Prompt Example for Generating Ground Truth Data for the Color Alteration Task

```

1455 **System Role**
1456 You are an AI image processing engine scheduler, responsible for
1457 converting the natural language instructions provided by the user
    into executable multi-model collaboration process json. All inputs

```

1458 must come from the initial parameters or the output of the previous
 1459 model or your own language decomposition and translation.
 1460

1461 ****Processing rules****
 1462 1. Input traceability principle
 1463 - Each model parameter can only be:
 1464 a. Initial image, it can be an image list composed of many images
 1465 b. Text in user instructions
 1466 c. Output of previous steps
 1467 d. The result of your own language analysis and translation of the
 1468 text in user instructions

1469 2. Process generation steps
 1470 a. Extract the operation object and action from the user instruction
 1471 and analyze it in combination with the image content
 1472 b. After analysis, select the corresponding model for each operation,
 1473 ensure that the model you choose and the parameters you input meet
 1474 the requirements of the model
 1475 c. Establish a cross-model data dependency chain, make sure the
 1476 output of each step is used in subsequent processes, otherwise this
 1477 step is redundant.

1478 ****Input and output types****
 1479 There are only four types of input and output
 1480 1. Image
 1481 2. Mask
 1482 3. Str (only supports English input)
 1483 4. Float

1484 ****Model library****
 1485 Models can be divided into two types: PREDICT model and EDIT model
 1486 PREDICT model list:
 1487 1.RES (Segmentation by object specified by prompt)
 1488 Input: {Image[image], Str[prompt]}
 1489 Output: {Mask[mask], Image[image]}
 1490 Constraint: The given prompt must be in English and if there are
 1491 locative words or adjectives, include them. The output image is
 1492 checkerboard transparency visualization, if the user requests to
 1493 output the segmentation result, then output this image.
 1494 2.SOS (Segmentation of main objects in image)
 1495 Input: {Image[image]}
 1496 Output: {Mask[mask], Image[image]}
 1497 Constraint: Unable to perform segmentation on the specified object,
 1498 can only segment the most prominent target in the image. The output
 1499 image is checkerboard transparency visualization, if the user
 1500 requests to output the segmentation result, then output this image.

1501 EDIT model list:
 1502 1.FLUX-RCM (Replace the color or material of an object)
 1503 Input: {Image[image], Mask[mask], Str[prompt]}
 1504 Output: {Image[image]}
 1505 Constraint: Change the color or material of a specific object
 1506 according to the input prompt.

1507 ****Actual example****
 1508 Describe:
 1509 1. For the color alteration task, simply consider what type of
 1510 transformation is needed and then write the corresponding prompt.
 1511 Example1:
 User instruction: Turn the teddy bear gray
 Expected output:
 {
 "process": "Turn the teddy bear gray",
 "pipeline": [
 {

```

1512     "step": 1,
1513     "model": "RES",
1514     "input": {
1515         "image": "init[image]",
1516         "prompt": "The teddy bear."
1517     },
1518     "output": {
1519         "mask": "step1[mask]",
1520         "image": "step1[image]"
1521     }
1522 },
1523 {
1524     "step": 2,
1525     "model": "FLUX-RCM",
1526     "input": {
1527         "image": "init[image]",
1528         "mask": "step1[mask]",
1529         "prompt": "Change the plush teddy bear to gray."
1530     },
1531     "output": {
1532         "image": "step2[image]"
1533     }
1534 },
1535 {
1536     "result": "[step2[image]]"
1537 }
1538 ]
1539 }
1540 Example2:
1541 User instruction: Turn the horse orange
1542 Expected output:
1543 {
1544     "process": "Turn the horse orange",
1545     "pipeline": [
1546         {
1547             "step": 1,
1548             "model": "RES",
1549             "input": {
1550                 "image": "init[image]",
1551                 "prompt": "The horse."
1552             },
1553             "output": {
1554                 "mask": "step1[mask]",
1555                 "image": "step1[image]"
1556             }
1557         },
1558         {
1559             "step": 2,
1560             "model": "FLUX-RCM",
1561             "input": {
1562                 "image": "init[image]",
1563                 "mask": "step1[mask]",
1564                 "prompt": "Change the color of the horse to orange."
1565             },
1566             "output": {
1567                 "image": "step2[image]"
1568             }
1569         }
1570     ],
1571     {
1572         "result": "[step2[image]]"
1573     }
1574 }
1575 ]

```

```

1566 Now, I give you the image and the user instruction:
1567 "<INSTRUCTION_TO_REPLACE>"
1568 Please output the thinking process in <think> </think>. During the
1569 thinking process
1570 1. Don't mention that you have seen the Actual example.
1571 2. Carefully think why this collaboration process is used (you can
1572 find hints in the 'Describe' section of the Actual example).
1573 Then output the multi-model collaboration process json in <answer>
1574 </answer>.

```

We utilize the prompt in Listing 2 to generate data via Qwen2.5-VL-72B, and an example of the produced data is presented below in Listing 3.

Listing 3: Example of Ground Truth Data for the Color Alteration Task

```

1581 <think>\nOkay, let's start by understanding the user's request. They
1582 want to change the color of the woman's dress in the artwork to dark
1583 red. First, I need to locate the dress in the image. Since the dress
1584 is part of a painting, I can't assume it's the main subject, so using
1585 the SOS model might not be effective. Instead, the RES model is more
1586 appropriate here because it allows segmentation based on a specific
1587 prompt. \n\nI'll use the RES model with the prompt \"woman's dress\"
1588 to get the mask of the dress. Once I have the mask, the next step is
1589 to apply the FLUX-RCM model, which is designed for changing colors or
1590 materials. The prompt for FLUX-RCM should be \"Change the color of
1591 the dress to dark red.\" This should replace the original color with
1592 the specified dark red while maintaining the dress's texture and
1593 details.\n\nI need to make sure that the mask from the RES model is
1594 correctly applied in the FLUX-RCM step. Also, checking the
1595 constraints of FLUX-RCM to confirm that it requires the image, mask,
1596 and the color change prompt. Since the user only wants the dress
1597 color changed, there's no need for additional steps like inpainting
1598 or adjusting the background. This two-step process should efficiently
1599 achieve the desired edit.\n</think>
1600 {
1601   "process": "Change the color of the woman's dress in the artwork to
1602   dark red",
1603   "pipeline": [
1604     {
1605       "step": 1,
1606       "model": "RES",
1607       "input": {
1608         "image": "init[image]",
1609         "prompt": "woman's dress"
1610       },
1611       "output": {
1612         "mask": "step1[mask]",
1613         "image": "step1[image]"
1614       }
1615     },
1616     {
1617       "step": 2,
1618       "model": "FLUX-RCM",
1619       "input": {
1620         "image": "init[image]",
1621         "mask": "step1[mask]",
1622         "prompt": "Change the color of the dress to dark red."
1623       },
1624       "output": {
1625         "image": "step2[image]"
1626       }
1627     }
1628   ]
1629 }

```

1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673

```
    {  
      "result": "[step2[image]]"  
    }  
  ]  
}
```