EVOEDIT: A SELF-EVOLVING AGENT FRAMEWORK FOR COMPLEX MULTI-STEP DEPENDENT IMAGE EDITING

Anonymous authors
Paper under double-blind review

ABSTRACT

With the rapid advancement of generative AI, image editing models have become increasingly powerful and gained significant attention. Despite this progress, existing models and agent systems often remain fragile and struggle to execute complex, especially chain-dependent, image editing tasks. While this vulnerability is partly due to poor planning and a reliance on step-by-step verification, a more fundamental limitation is their inability to learn from accumulated interaction history, causing them to persistently repeat past mistakes. To address these critical bottlenecks, we present EvoEdit, a novel intelligent agent framework designed to solve complex chain-dependent image editing problems, built on the ComfyUI platform. EvoEdit introduces three core innovations: Multi-Path Exploration (MPE), which concurrently explores diverse execution trajectories to identify a high-quality plan; a Recursive Verification mechanism, which backtracks through the execution history to pinpoint the earliest fault responsible for chain failures; and a Dual-Dimensional Self-Evolving mechanism, which dynamically synthesizes macroscopic planning rules and refines microscopic step executions by learning from accumulated experience. Together, these components improve the stability, physical plausibility, and visual harmony of complex interdependent editing workflows. Experimental results demonstrate that EvoEdit significantly outperforms Nano Banana, a strong closed-source image editing model, achieving a 20.68% improvement in average score and a 31.55% increase specifically on three-chain dependent tasks.

1 INTRODUCTION

The rapid development of visual generation systems has yielded remarkable performance in the field of image editing (Wang et al., 2023; Ma et al., 2024; Li et al., 2024; Wu et al., 2025; Yang et al., 2025; Sheynin et al., 2024; Zhang et al., 2023; Wang et al., 2025; Ye et al., 2025; Bai et al., 2026), encompassing both open-source diffusion-based editing models (Zhang et al., 2023; Brooks et al., 2023; Zhao et al., 2024; Yu et al., 2025; Zhuang et al., 2025; Ge et al., 2024; Xiao et al., 2024; Fu et al., 2024; Zhang et al., 2025; StepFun AI, 2025; Deng et al., 2025) and closed-source multimodal systems (OpenAI, 2025; Anil et al., 2023; Google, 2025). However, existing open-source general-purpose generative models still face a series of challenges, often struggling to faithfully accomplish highly complex editing tasks (Wang et al., 2025; Ye et al., 2025; Bai et al., 2026). As a progressive step forward to overcome these limitations, various agentic frameworks and open-source systems (Liang et al., 2023; Gupta & Kembhavi, 2023; Shen et al., 2023; Liu et al., 2024; Gao et al., 2024; Zhu et al., 2025; Fang et al., 2025; Gal et al., 2024; Xue et al., 2025; Xu et al., 2025; Huang et al., 2025; Guo et al., 2025; Wang et al., 2024; Hu et al., 2025; Gupta et al., 2025; Zeng et al., 2026; Jia et al., 2025; Chang et al., 2026) have been proposed. By leveraging Large Language Models (LLMs) and Vision-Language Models (VLMs) as autonomous agents to decompose complex intents into manageable step-by-step operations, these modular systems have made successful attempts and achieved substantial progress in executing complex image editing requests.

Among these promising systems, a notable category comprises agentic architectures built upon ComfyUI (Gal et al., 2024; Xue et al., 2025; Xu et al., 2025; Huang et al., 2025; Guo et al., 2025). ComfyUI is an open-source platform designed to create and execute general-purpose generative workflows, encompassing text-to-image generation, image editing, and video generation. It provides

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

a node-based interface that allows users to flexibly construct visual generative pipelines represented as JSON formats according to their specific needs. While serving as a general-purpose framework, its highly modular design, extensive node library, and exceptional flexibility make it particularly well-suited for the customized demands of the image editing domain. Leveraging this powerful foundation, recent node-based agent systems within the Comfy series, such as ComfyMind (Guo et al., 2025), have demonstrated remarkable potential, even approaching the performance of strong closed-source models like GPT-Image-1 (OpenAI, 2025) on specific editing datasets (Huang et al., 2024).

Despite this, current image editing systems still struggle when tasked with chain-dependent instructions—where subsequent operations inherently rely on the execution and context of preceding steps. Specifically, existing agentic frameworks expose their fragility through three critical limitations: 1) **Poor planning**: they typically either break down complex instructions into excessively granular atomic operations, or attempt to execute the entire complex instruction in a single step, subsequently modifying the output only when errors occur. Consequently, they often fail to construct a globally insightful plan that accommodates the intricate relationships between sub-instructions for highly coupled tasks. 2) **Reliance on step-by-step verification**: while this approach performs well in simple tasks, it introduces redundancy and imposes unnecessary constraints on intermediate images, frequently failing to correctly evaluate intermediate states in multi-step dependent tasks. 3) **Persistent repetition of past mistakes**: as a more fundamental bottleneck, these systems fail to capture and utilize valuable trial-and-error feedback generated during runtime, causing them to discard experimental insights and persistently repeat past mistakes instead of dynamically adapting.

In this paper, we propose EvoEdit, an intelligent agent framework to address the above challenges. Our fundamental idea is to utilize Vision-Language Models (VLMs) as autonomous agents to dynamically strategize, execute, verify and self-evolve throughout the editing process. The core mechanisms of the system are: **Multi-Path Exploration (MPE).** EvoEdit launches multiple concurrent editing attempts with actively diversified global plans under the guidance of the planner and experience library. **Recursive Verification.** Within each path, it verifies the final result and backtracks through the executed steps to identify the earliest source of failure, avoiding unnecessary intermediate verification and modification. **Dual-Dimensional Self-Evolving.** Using historical execution data, EvoEdit evolves at both the macro level by improving global plans and avoiding flawed planning patterns, and the micro level by refining workflow choices for individual editing steps.

Our main contributions can be summarized as follows:

- We propose EvoEdit, a self-evolving agent framework for complex multi-step dependent image editing. By recording and reusing historical failure experiences, EvoEdit progressively improves its planning and execution strategies. To the best of our knowledge, it is the first image-editing framework to explicitly enable experience-driven self-evolution.
- We identify two complementary dimensions for improving agentic image-editing systems and jointly optimize these two dimensions through a dual-dimensional self-evolving mechanism.
- We demonstrate that EvoEdit substantially improves the performance of small open-source image-editing models, enabling them to approach or surpass strong closed-source models on complex dependent editing benchmarks. Additionally, results show that failure experiences help the system avoid similar errors.

2 Related Work

2.1 Instruction-based Image Editing Models

Recent advancements in diffusion models have significantly propelled the development of instruction-based image editing. Pioneering works like InstructPix2Pix (Brooks et al., 2023) demonstrated the feasibility of editing images through natural language instructions using paired synthetic data. Subsequent models, such as MGIE (Fu et al., 2024), Emu Edit (Sheynin et al., 2024), and AnyEdit (Yu et al., 2025), have further expanded editing capabilities by introducing multimodal guidance, multi-task training, and task-aware routing to handle diverse editing requests spanning attributes, objects, and styles. Notably, Google's closed-source Gemini 2.5 Flash Image, also referred to as

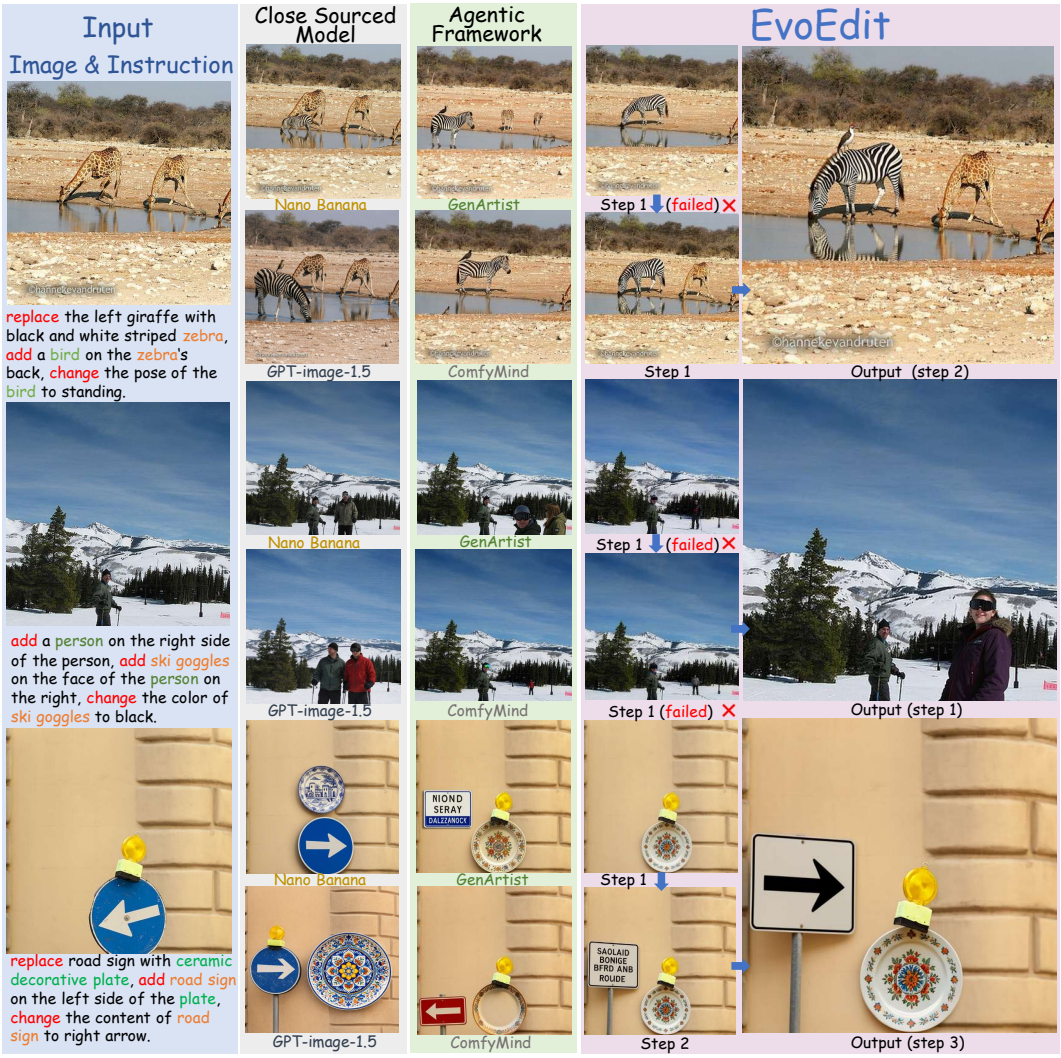

Figure 1: Qualitative comparison of EvoEdit with strong closed-sourced models and classic agentic frameworks on three-chain dependent complex instructions. The agentic frameworks are equipped with same base tools as EvoEdit (diffusion model, ComfyUI workflow, etc.) The rightmost column illustrates EvoEdit's unique planning ability, deciding steps to take by agents.

Nano Banana, has gained significant attention due to its remarkable capabilities in image generation and editing. While these foundational models exhibit strong performance in executing single-turn edits, their effectiveness remains limited when handling complex image editing scenarios, especially chain-dependent complex image editing tasks.

2.2 Agent-based Systems for Multi-step Image Editing

To bridge the gap between complex user intents and foundational editing models, recent literature has increasingly adopted Large Language Models (LLMs) and Vision-Language Models (VLMs) as autonomous agents. These agents decompose complex tasks into manageable sub-tasks and orchestrate a suite of editing tools. For instance, GenArtist (Wang et al., 2024) employs a tree-structured planning mechanism with step-by-step verification to unify image generation and editing, while other systems like IEAP (Hu et al., 2025), MIRA (Zeng et al., 2026), and CoSTA (Gupta et al., 2025) approach complex editing through composable programmatic operations, iterative perception-reasoning-action loops, and cost-sensitive toolpath exploration.

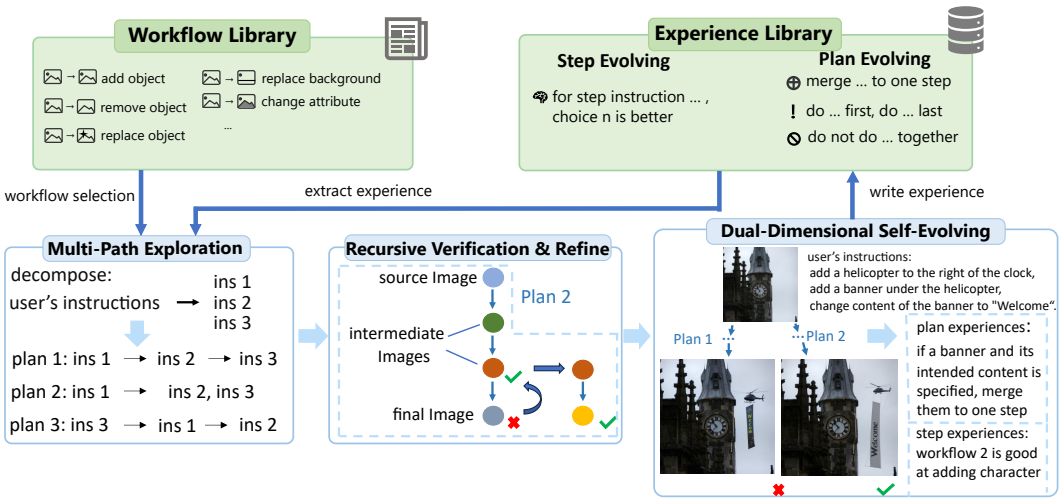

Figure 2: Overview of the EvoEdit framework.

Although these agentic workflows and frameworks have demonstrated significant progress in complex image editing, they still face substantial challenges and exhibit limited performance when tackling highly coupled, chain-dependent complex image editing tasks.

2.3 ComfyUI Platform and Workflow Generation

Among those agentic frameworks, a specialized and rapidly growing paradigm involves training agents directly within the ComfyUI platform. Several recent works, including ComfyGen (Gal et al., 2024), ComfyGPT (Huang et al., 2025), and ComfyUI-R1 (Xu et al., 2025), have attempted to automatically construct executable workflows to accomplish general-purpose tasks. ComfyMind (Guo et al., 2025) realizes the best performance on the ComfyBench benchmark among the Comfy-series frameworks.

Our work utilizes the ComfyUI platform and builds upon the SWI concept proposed by ComfyMind. However, previous works within the Comfy series have primarily focused on the mechanics of workflow generation and executing simple general-purpose tasks, encompassing basic image generation, image editing, and video generation. This study goes a step further by focusing specifically on solving the unique challenges of complex, chain-dependent image editing tasks.

3 Methodology: EvoEdit

3.1 Problem Formulation

We first define base-instructions as instructions that can be categorized into the following ten tasks and cannot be split into two or more: add object, delete object, change object, change color, change material, change content, change pose, change text, change background, and change style. Given a complex editing request, we decompose it into N base-instructions $T = \{T_1, T_2, \ldots, T_N\}$. We classify the task complexity into three levels based on their logical dependencies: *Level 1: Parallel* (logically independent operations), *Level 2: Two-chain* (e.g., $T_1 \rightarrow T_2$ forms a dependency chain, while T_3 remains independent), and *Level 3: Three-chain* (fully interdependent sequence $T_1 \rightarrow T_2 \rightarrow T_3$).

For chain-dependent tasks, the successful execution of T_i inherently relies on the semantic context established by T_{i-1}. Let S_i denote the evaluation score for an individual instruction T_i. To strictly account for the prerequisite nature of earlier steps, the overall performance score S_{chain} for an

N-instruction chain is formulated as the product of its constituent scores:

$$S_{\text{chain}} = \prod_{i=1}^{N} S_i \tag{1}$$

This multiplicative approach ensures that failure in any single step (resulting in $S_i = 0$) propagates to the entire chain, yielding a total score of 0 and accurately reflecting the inability to complete the dependent task sequence successfully. The problem and scoring are proposed by ComplexBench-Edit (Wang et al., 2025).

3.2 Multi-Path Exploration (MPE)

Formally, let a complex user instruction be denoted as I, which can be decomposed into a set of basic instructions $\{I_1, I_2, \ldots, I_N\}$. We define an execution plan p as an ordered sequence of instruction groups, mathematically representing both the execution order and the strategic merging of these basic instructions. For example, given $N = 3$, the diverse parallel plans instantiated by the system can be formulated as:

$$\begin{aligned} p_1 &: \{I_1\} \rightarrow \{I_2\} \rightarrow \{I_3\} \\ p_2 &: \{I_1\} \rightarrow \{I_2, I_3\} \\ p_3 &: \{I_3\} \rightarrow \{I_1\} \rightarrow \{I_2\} \end{aligned} \tag{2}$$

Here, p_1 represents a standard step-by-step sequential execution, p_2 merges I_2 and I_3 into a unified single-step execution to leverage parallel context, and p_3 alters the execution order to bypass rigid dependency traps.

Existing image editing frameworks often suffer from low success rates on complex, multi-step dependent tasks because of their poor planning. They typically use the plan $p : \{I_1\} \rightarrow \{I_2\} \rightarrow \{I_3\}$, which does not perform well under complex dependencies. To resolve this issue, we introduce the MPE (Multi-Path Exploration) framework as the core scheduling engine of EvoEdit. MPE initiates 3 paths ($|s| = 3,\ s = \{p_i\}$) concurrently, considering the balance between effectiveness and cost, which is discussed in Appendices A.1 and A.2.

MPE enforces diversity by explicitly requiring agents to generate different plans (e.g., p_1, p_2, p_3). By leveraging an experience library of historical experiences, the system guides these structural divergences, which is discussed in Section 3.4.

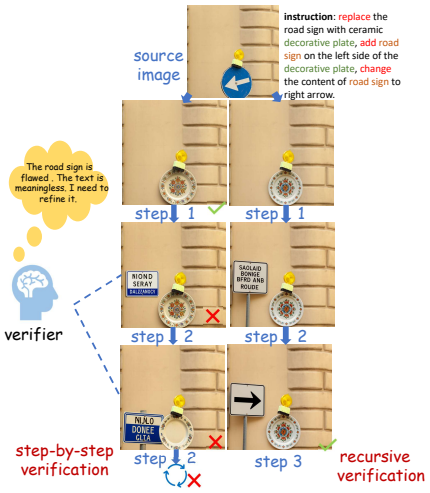

Figure 3: A comparison of recursive verification and step-by-step verification, keeping the tools the same.

3.3 Recursive Verification and Reasoning

Within each individual concurrent plan executed by the MPE framework, EvoEdit employs a pipeline driven by our Recursive Verification and Reasoning mechanism. The verification process starts after all executions are completed. An evaluator evaluates the result and scores the output. Once an error is identified, the system initiates the recursive verification process. It backtracks through intermediate steps and verifies the intermediate outputs based on the plan until the root error is found. Most existing agentic frameworks rely on step-by-step verifications. While this isolated approach performs exceptionally well in parallel multi-step tasks, it frequently initiates unnecessary assessments of intermediate images and traps the workflow in local failures when resolving multi-step dependent tasks. For example, in multi-agent systems like GenArtist, a VLM might reject a generated image during a specific sub-task, illustrated in Figure 3. Recursive verification addresses this issue by starting by evaluating the final output (get S_{p_i}). This approach effectively avoids unnecessary intermediate verifications and modifications, thereby mitigating the risk of trapping the agent in localized infinite loops.

The recursive verification procedure is summarized in the following pseudocode.

Input: editing plan with n executed steps and states $I_0, I_1, \ldots, I_n$
Output: verified final edited image
if Verify(I_n) succeeds **then**
    return I_n
else
    for $t = n-1, n-2, \ldots, 0$ **do**
        if Verify(I_t) succeeds **then**
            resume execution from step $t+1$
            rerun the remaining steps until step n
            return the new final edited image
        end if
    end for
end if

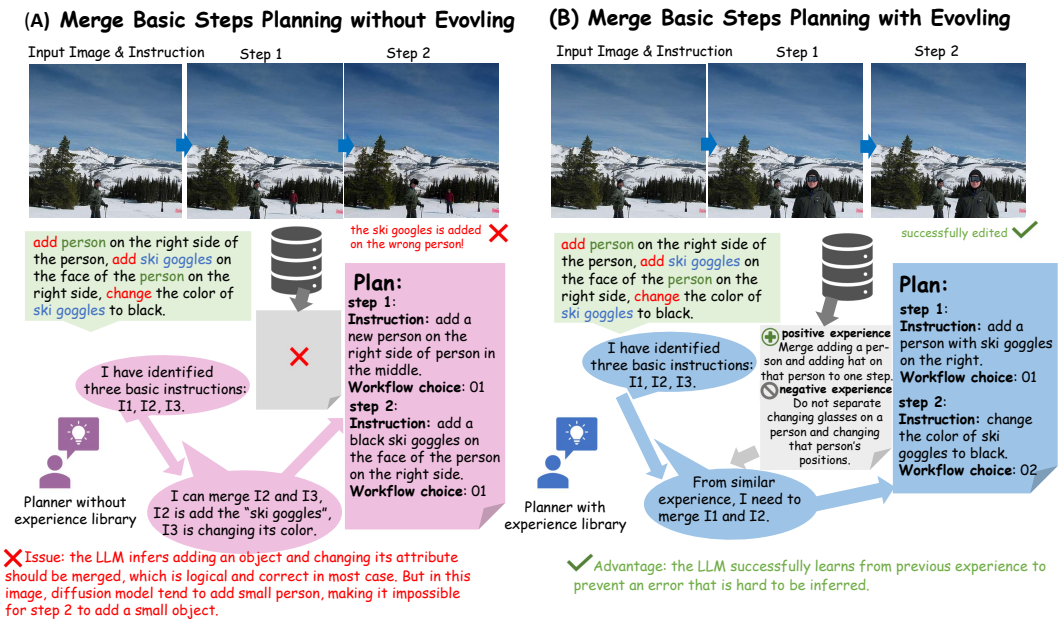

Figure 4: An illustration of the macro-evolving in dual-dimensional evolving.

3.4 Dual-Dimensional Self-Evolving

Previous works often repeat their past mistakes. To address this, we propose a mechanism that leverages past experiences. The MPE method features a strong ability to accumulate diverse execution data, which naturally facilitates a continuous self-improving mechanism. We introduce a Dual-Dimensional Self-Evolving process that allows EvoEdit to refine its editing capabilities across both macro and micro levels without requiring additional parameter fine-tuning; an example is illustrated in Figure 2.

Macro-Level Plan Evolution. At the macro level, evolution is driven by the comparative evaluation across the concurrent plans generated by MPE. When a significant score discrepancy ($|S_{p_1} - S_{p_2}| > S_{limit}$) among different structural plans is identified, a Plan Reasoner is triggered. This module analyzes why a plan ($p_{optimal}$) outperformed the others—synthesizing planning rules, such as identifying which decoupled steps should be merged together or determining optimal execution orders. These distilled insights are subsequently persisted into a global experience library. In future tasks, the global planner queries this library to bypass known architectural anti-patterns and actively scaffold historically optimized macro-plans; an example is shown in Figure 4. The S_{limit} is a hyperparameter, which is discussed in Appendix A.3.

Micro-Level Step Evolution. Concurrently, EvoEdit performs micro-level evolution within the inner Recursive Verification loop of an individual plan. By capturing the execution state difference ($S_{step1} > 0, S_{step2} = 0$) from the step that used two different workflows, our system records which specific ComfyUI workflow ($w_{optimal}$) is effective for specific steps. By continuously accumulating these microscopic execution heuristics, EvoEdit effectively learns to adopt the optimized tool choices for certain steps.

4 EXPERIMENTS

4.1 COMPARISON WITH IMAGE EDITING MODELS

In this section, we compare the performance of EvoEdit with other strong closed-source and open-source image editing models on the ComplexBench-Edit(Wang et al., 2025) dataset. The qualitative visualizations of some examples is in Figure 1. The quantitative evaluation is categorized by instruction complexity levels, specifically focusing on Parallel, Two-chain, and Three-chain dependencies. The score is calculated based on the method in 3.1, with each base instructions scoring $S_i \in [0, 5]$.

Table 1: Performance comparison on ComplexBench-Edit(Wang et al., 2025) across different instruction complexity levels. Here, "Parallel" indicates multi-step tasks that are mutually independent, while "Two-chain" and "Three-chain" denote tasks involving two and three interdependent sequential steps, respectively. Additionally, "Obj." denotes Object, "At." denotes Attribute, and "G." denotes Global. The score is calculated based on the method in 3.1.

Method	**Parallel**			**Two-chain**	**Three-chain**	**Avg.**
	Obj.	Obj.-At.	Obj.-At.-G.	Obj.-At.	Obj.-At.	
InstructPix2Pix (Brooks et al., 2023)	9.95	10.45	13.47	4.83	0.52	7.85
MagicBrush (Zhang et al., 2023)	18.29	14.65	11.90	9.70	0.57	11.02
UltraEdit (Zhao et al., 2024)	24.46	25.78	23.80	13.74	3.10	18.18
AnyEdit (Yu et al., 2025)	13.51	11.27	9.33	6.61	1.01	8.35
VAR-GPT (Zhuang et al., 2025)	0.86	0.39	3.80	0.54	0.00	1.12
SEED-LLAMA (Ge et al., 2024)	5.59	5.93	6.00	4.48	0.27	4.46
OmniGen (Xiao et al., 2024)	32.34	26.57	30.90	16.96	4.36	22.23
GoT (Fang et al., 2025)	20.09	14.85	16.42	11.76	0.93	12.81
ICEdit (Zhang et al., 2025)	32.97	26.96	27.07	21.80	8.09	23.38
Step1X-Edit (StepFun AI, 2025)	32.61	30.14	31.57	19.56	6.34	24.05
Gemini (Anil et al., 2023)	49.31	43.47	37.23	38.35	15.10	36.70
Gemini-CoT (Wang et al., 2025)	51.76	50.11	43.08	39.85	17.54	40.47
Nano Banana (Google, 2025)	42.95	42.47	49.61	37.17	36.39	41.78
EvoEdit (Ours)	**52.62**	**49.90**	**52.86**	**48.87**	**47.87**	**50.42**

As shown in Table 1, while most baselines perform reasonably well on simpler Parallel tasks, their performance collapses on complex, highly interdependent instructions. By contrast, EvoEdit demonstrates remarkable robustness, achieving high scores of 48.87 and 47.87 on Two-chain and Three-chain dependencies, respectively. By substantially outperforming strong end-to-end systems and advanced reasoning models (e.g., Gemini-CoT, Nano Banana), EvoEdit highlights the efficacy of our Dual-Dimensional Self-Evolving and MPE framework in mitigating poor planning.

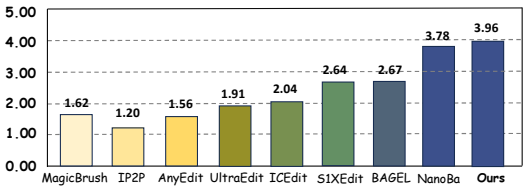

Figure 5: Comparison on Hybrid Task of ImgEdit (Ye et al., 2025) benchmark.

Furthermore, we evaluate our method on the ImgEdit benchmark (Ye et al., 2025), which encompasses a wide variety of image editing tasks ranging from simple single-step operations to complex multi-step instructions. Because our primary focus lies in resolving intricate task dependencies, we specifically benchmark our framework on the challenging "Hybrid" task category of ImgEdit. As illustrated in Figure 5, EvoEdit successfully surpasses leading state-of-the-art models, including the highly capable Nano Banana, reinforcing our method's superior generalization and efficacy in multi-step scenarios.

4.2 Comparison with Other Image Editing Agentic Workflows

To comprehensively evaluate our framework against existing agentic workflows, we benchmark EvoEdit against ComfyMind and GenArtist. ComfyMind (Guo et al., 2025) represents the previous state-of-the-art among the Comfy-series frameworks (e.g., ComfyAgent and ComfyUI-R1). For a fair comparison, we upgrade ComfyMind by equipping it with the new models and workflow formats proposed in EvoEdit. Meanwhile, GenArtist (Wang et al., 2024) is a prominent architecture that employs tree-structured planning and step-by-step verification. Because its original underlying tool library is outdated, we retain its high-level tree-planning algorithm and per-step verification but retrofit its bottom-level execution engine to use EvoEdit's robust ComfyUI tool library.

As illustrated in Figure 6, EvoEdit consistently achieves optimal performance across all task complexity levels. The performance trends reveal a striking pattern: while baseline methods exhibit severe degradation as task dependencies increase–EvoEdit maintains relatively stable high performance.

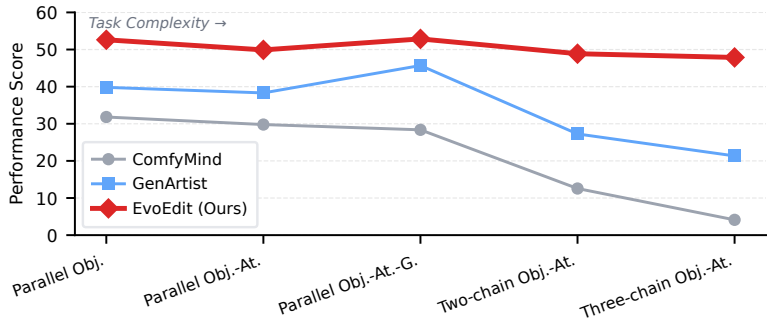

Figure 6: Performance comparison with tree-planning methods across task complexity levels on ComplexBench-Edit (Wang et al., 2025).

4.3 Ablation Study

To comprehensively evaluate the effectiveness of the key components in our proposed framework, we conduct an ablation study on the ComplexBench-Edit (Wang et al., 2025) benchmark. We focus on the three core modules that distinguish our method: the Multi-Path Exploration (MPE) architecture, the dual-dimensional self-evolving mechanism, and the recursive verification process. Because our evolving mechanism fundamentally relies on the diverse trajectories provided by MPE and the accurate fault attribution of recursive verification, our experimental design progressively removes these interconnected modules rather than evaluating evolving in isolation. The results are summarized in Table 2.

Table 2: Ablation study on ComplexBench-Edit (Wang et al., 2025) across different instruction complexity levels. "Obj." denotes Object, "At." denotes Attribute, and "G." denotes Global.

Evolving	**MPE**	**Rec. Verif.**	**Parallel**			**Two-chain**	**Three-chain**	**Avg.**
			Obj.	Obj.-At.	Obj.-At.-G.	Obj.-At.	Obj.-At.	
✓	✓	✓	**52.62**	**49.90**	**52.86**	**48.87**	**47.87**	**50.42**
-	✓	✓	52.34	47.30	51.25	39.94	36.52	45.47
-	-	✓	41.24	41.46	47.46	35.76	25.10	38.20
-	✓	-	40.28	33.60	43.76	37.64	34.31	37.92

As shown in the table, the full EvoEdit framework achieves the highest performance. Crucially, the most profound performance degradation occurs in the highly complex Two-chain and Three-chain categories when core modules are removed. One example of removing the evolving is shown in the figure 4. Disabling the evolving mechanism restricts the agent from generating an optimized plan and choosing most suitable workflow via accumulated experience, yielding notable drops in multi-step dependent scenarios. Further removing MPE causes a severe collapse in resolving dependencies,

validating its indispensability in mitigating poor planning. The significant difference between output images of poor planning and effective ones is shown in B.1. Overall, these results demonstrate that our proposed evolving mechanism, MPE, and recursive verification process collectively account for solving complex, multi-step dependent image editing tasks.

4.4 Qualitative Results

Some qualitative results are illustrated in Figure 1 and Figure 7. Specifically, these visual comparisons delineate the practical utility of plan evolving (macro-level evolving) and step evolving (micro-level evolving). For a fair comparison, the GenArtist baseline was equipped with identical workflows and underlying models; nevertheless, it failed to achieve comparable visual outcomes.

In the first example, plan evolving effectively utilizes historical experience to bypass the execution of adding a driver directly onto the tram—a flawed planning step that frequently introduces unnatural artifacts.

In the second example, EvoEdit utilizes step evolving to deduce and select the appropriate text-addition workflow. In contrast, ComfyMind relies exclusively on zero-shot LLM reasoning over tool descriptions. Because the workflow library lacked explicit user annotations defining the optimal tool for text generation, ComfyMind selected an incorrect workflow.

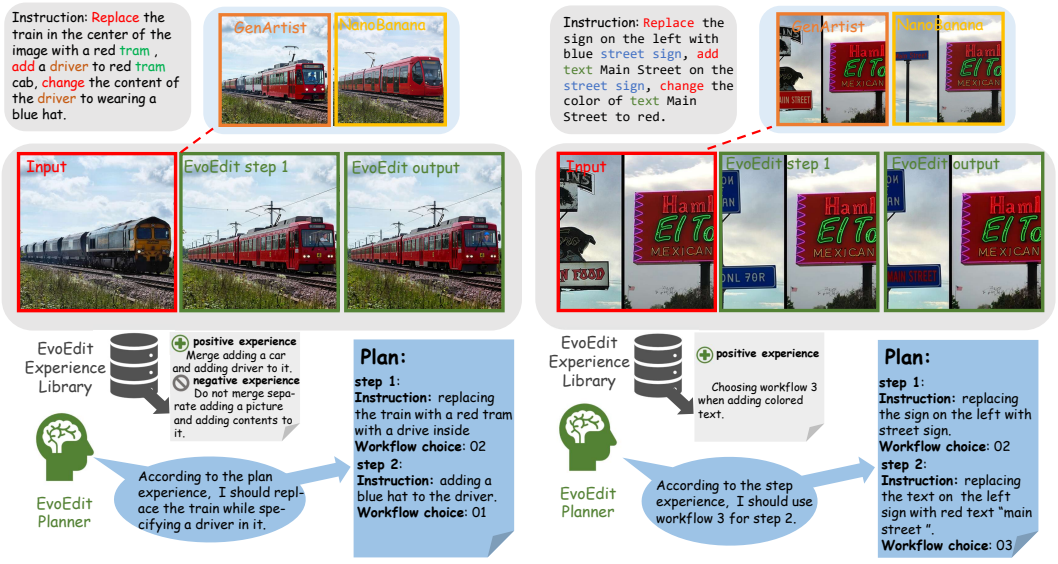

Figure 7: Illustrations of the macro-evolving and micro-evolving in dual-dimensional evolving.

5 Conclusion

In this work, we introduced EvoEdit, a self-evolving agent framework designed to address the challenges of complex, multi-step dependent image editing. By identifying the limitations of existing models and frameworks, we proposed a different execution paradigm. By leveraging multi-path exploration (MPE), recursive verification mechanism, and dual-dimensional self-Evolving strategy, EvoEdit well addresses complex multi-step dependent image editing tasks.

Evaluations on the challenging ComplexBench-Edit and ImgEdit benchmarks demonstrate that EvoEdit outperforms both strong closed-source models (e.g., Nano Banana) and classic open-source agentic workflows (e.g., ComfyMind, GenArtist). By establishing a self-improving methodology, we believe our work can provide valuable insights for future research for the design of highly autonomous and evolving visual agents.

Reproducibility Statement

Our anonymous repository, including the source code, workflow configurations, and experimental results, is available at `https://anonymous.4open.science/r/EvoEdit-38EC`.

References

Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, et al. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Xuehai Bai, Xiaoling Gu, Akide Liu, Hangjie Yuan, YiFan Zhang, and Jack Ma. MCIE: Multimodal llm-driven complex instruction image editing with spatial guidance. In *AAAI Conference on Artificial Intelligence*, 2026.

Tim Brooks, Aleksander Holynski, and Alexei A. Efros. InstructPix2Pix: Learning to follow image editing instructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Zewei Chang, Zheng-Peng Duan, Jianxing Zhang, Chun-Le Guo, Siyu Liu, Hyungju Chun, Hyunhee Park, Zikun Liu, and Chongyi Li. PerTouch: Vlm-driven agent for personalized and semantic image retouching. In *AAAI Conference on Artificial Intelligence*, 2026.

Chaorui Deng, Deyao Zhu, Kunchang Li, Chenhui Gou, Feng Li, Zeyu Wang, Shu Zhong, Weihao Yu, Xiaonan Nie, Ziang Song, Guang Shi, and Haoqi Fan. Emerging properties in unified multimodal pretraining. *arXiv preprint arXiv:2505.14683*, 2025.

Rongyao Fang, Chengqi Duan, Kun Wang, Linjiang Huang, Hao Li, Shilin Yan, Hao Tian, Xingyu Zeng, Rui Zhao, Jifeng Dai, Xihui Liu, and Hongsheng Li. GoT: Unleashing reasoning capability of multimodal large language model for visual generation and editing. In *Advances in Neural Information Processing Systems*, 2025.

Tsu-Jui Fu, Wenze Hu, Xianzhi Du, William Yang Wang, Yinfei Yang, and Zhe Gan. Guiding instruction-based image editing via multimodal large language models. In *International Conference on Learning Representations*, 2024.

Rinon Gal, Adi Haviv, Yuval Alaluf, Amit H. Bermano, Daniel Cohen-Or, and Gal Chechik. ComfyGen: Prompt-adaptive workflows for text-to-image generation. *arXiv preprint arXiv:2410.01731*, 2024.

Zhi Gao, Yuntao Du, Xintong Zhang, Xiaojian Ma, Wenjuan Han, Song-Chun Zhu, and Qing Li. CLOVA: A closed-loop visual assistant with tool usage and update. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Yuying Ge, Sijie Zhao, Ziyun Zeng, Yixiao Ge, Chen Li, Xintao Wang, and Ying Shan. Making LLaMA see and draw with SEED tokenizer. In *International Conference on Learning Representations*, 2024.

Google. Gemini 2.5 flash model card. `https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-2-5-Flash-Model-Card.pdf`, 2025.

Litao Guo, Xinli Xu, Luozhou Wang, Jiantao Lin, Jinsong Zhou, Zixin Zhang, Bolan Su, and Ying-Cong Chen. ComfyMind: Toward general-purpose generation via tree-based planning and reactive feedback. In *Advances in Neural Information Processing Systems*, 2025.

Advait Gupta, NandaKiran Velaga, Dang Nguyen, and Tianyi Zhou. Costa*: Cost-sensitive toolpath agent for multi-turn image editing. *arXiv preprint arXiv:2503.10613*, 2025.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Yujia Hu, Songhua Liu, Zhenxiong Tan, Xingyi Yang, and Xinchao Wang. Image editing as programs with diffusion models. *arXiv preprint arXiv:2506.04158*, 2025.

Oucheng Huang, Yuhang Ma, Zeng Zhao, Mingrui Wu, Jiayi Ji, Rongsheng Zhang, Zhipeng Hu, Xiaoshuai Sun, and Rongrong Ji. ComfyGPT: A self-optimizing multi-agent system for comprehensive comfyui workflow generation. *arXiv preprint arXiv:2503.17671*, 2025.

Yuzhou Huang, Liangbin Xie, Xintao Wang, Ziyang Yuan, Xiaodong Cun, Yixiao Ge, Jiantao Zhou, Chao Dong, Rui Huang, Ruimao Zhang, and Ying Shan. SmartEdit: Exploring complex instruction-based image editing with multimodal large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Qifei Jia, Yu Liu, Yajie Chai, Xintong Yao, Qiming Lu, Yasen Zhang, Runyu Shi, Ying Huang, and Guoquan Zhang. Lego-Edit: A general image editing framework with model-level bricks and mllm builder. *arXiv preprint arXiv:2509.12883*, 2025.

Baiqi Li, Zhiqiu Lin, Deepak Pathak, Jiayao Li, Yixin Fei, Kewen Wu, Tiffany Ling, Xide Xia, Pengchuan Zhang, Graham Neubig, and Deva Ramanan. GenAI-Bench: Evaluating and improving compositional text-to-visual generation. In *European Conference on Computer Vision*, 2024.

Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, Yun Wang, Linjun Shou, Ming Gong, and Nan Duan. TaskMatrix.AI: Completing tasks by connecting foundation models with millions of apis. *arXiv preprint arXiv:2303.16434*, 2023.

Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Ziheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, and Wenhai Wang. ControlLLM: Augment language models with tools by searching on graphs. In *European Conference on Computer Vision*, 2024.

Yiwei Ma, Jiayi Ji, Ke Ye, Weihuang Lin, Zhibin Wang, Yonghan Zheng, Qiang Zhou, Xiaoshuai Sun, and Rongrong Ji. I2EBench: A comprehensive benchmark for instruction-based image editing. In *Advances in Neural Information Processing Systems*, 2024.

OpenAI. GPT-Image-1. `https://platform.openai.com/docs/models/gpt-image-1`, 2025.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving ai tasks with chatgpt and its friends in hugging face. In *Advances in Neural Information Processing Systems*, 2023.

Shelly Sheynin, Adam Polyak, Uriel Singer, Yuval Kirstain, Amit Zohar, Oron Ashual, Devi Parikh, and Yaniv Taigman. Emu edit: Precise image editing via recognition and generation tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

StepFun AI. Step1X-Edit: Practical image editing. *arXiv preprint arXiv:2504.17761*, 2025.

Chenglin Wang, Yucheng Zhou, Qianning Wang, Zhe Wang, and Kai Zhang. ComplexBench-Edit: Benchmarking complex instruction-driven image editing via compositional dependencies. In *ACM Multimedia*, 2025.

Su Wang, Chitwan Saharia, Ceslee Montgomery, Jordi Pont-Tuset, Shai Noy, Stefano Pellegrini, Yasumasa Onoe, Sarah Laszlo, David J. Fleet, Radu Soricut, Jason Baldridge, Mohammad Norouzi, Peter Anderson, and William Chan. Imagen editor and editbench: Advancing and evaluating text-guided image inpainting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Zhenyu Wang, Aoxue Li, Zhenguo Li, and Xihui Liu. GenArtist: Multimodal llm as an agent for unified image generation and editing. In *Advances in Neural Information Processing Systems*, 2024.

Yongliang Wu, Zonghui Li, Xinting Hu, Xinyu Ye, Xianfang Zeng, Gang Yu, Wenbo Zhu, Bernt Schiele, Ming-Hsuan Yang, and Xu Yang. KRIS-Bench: Benchmarking next-level intelligent image editing models. In *Advances in Neural Information Processing Systems*, 2025.

Shitao Xiao, Yueze Wang, Junjie Zhou, Huaying Yuan, Xingrun Xing, Ruiran Yan, Shuting Wang, Tiejun Huang, and Zheng Liu. OmniGen: Unified image generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Zhenran Xu, Yiyu Wang, Xue Yang, Longyue Wang, Weihua Luo, Kaifu Zhang, Baotian Hu, and Min Zhang. ComfyUI-R1: Exploring reasoning models for workflow generation. *arXiv preprint arXiv:2506.09790*, 2025.

Xiangyuan Xue, Zeyu Lu, Di Huang, Zidong Wang, Wanli Ouyang, and Lei Bai. ComfyBench: Benchmarking llm-based agents in comfyui for autonomously designing collaborative ai systems. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Ling Yang, Bohan Zeng, Jiaming Liu, Hong Li, Minghao Xu, Wentao Zhang, and Shuicheng Yan. EditWorld: Simulating world dynamics for instruction-following image editing. In *ACM Multimedia*, 2025.

Yang Ye, Xianyi He, Zongjian Li, Bin Lin, Shenghai Yuan, Zhiyuan Yan, Bohan Hou, and Li Yuan. Imgedit: A unified image editing dataset and benchmark. In *Advances in Neural Information Processing Systems*, 2025.

Qifan Yu, Wei Chow, Zhongqi Yue, Kaihang Pan, Yang Wu, Xiaoyang Wan, Juncheng Li, Siliang Tang, Hanwang Zhang, and Yueting Zhuang. AnyEdit: Mastering unified high-quality image editing for any idea. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Ziyun Zeng, Hang Hua, and Jiebo Luo. MIRA: Multimodal iterative reasoning agent for image editing. *arXiv preprint arXiv:2511.21087*, 2026.

Kai Zhang, Lingbo Mo, Wenhu Chen, Huan Sun, and Yu Su. MagicBrush: A manually annotated dataset for instruction-guided image editing. In *Advances in Neural Information Processing Systems*, 2023.

Zechuan Zhang, Ji Xie, Yu Lu, Zongxin Yang, and Yi Yang. In-context edit: Enabling instructional image editing with in-context generation in large scale diffusion transformer. In *Advances in Neural Information Processing Systems*, 2025.

Haozhe Zhao, Xiaojian (Shawn) Ma, Liang Chen, Shuzheng Si, Rujie Wu, Kaikai An, Peiyu Yu, Minjia Zhang, Qing Li, and Baobao Chang. UltraEdit: Instruction-based fine-grained image editing at scale. In *Advances in Neural Information Processing Systems*, 2024.

Kaiwen Zhu, Jinjin Gu, Zhiyuan You, Yu Qiao, and Chao Dong. An intelligent agentic system for complex image restoration problems. In *International Conference on Image and Representation*, 2025.

Xianwei Zhuang, Yuxin Xie, Yufan Deng, Liming Liang, Jinghan Ru, Yuguo Yin, and Yuexian Zou. VARGPT: Unified understanding and generation in a visual autoregressive multimodal large language model. *arXiv preprint arXiv:2501.12327*, 2025.

A SUPPLEMENTARY QUANTITATIVE EXPERIMENTS

A.1 BALANCING GLOBAL PLANNING AND LOCAL REFINEMENT ITERATIONS

In addition to determining the optimal number of concurrent paths, it is critical to find the optimal trade-off between macro-level exploration (the number of global plans) and micro-level exploitation (the maximum number of allowed refine iterations per step). We evaluate various combinations of global planning attempts and local refine limits. The results reveal that over-allocating refine iterations can trap the agent in localized loops without yielding further improvements, whereas sufficient global plan diversity combined with a moderate refine limit achieves the optimal equilibrium between task success rate and computational efficiency. f global plans and maximum refine limits.

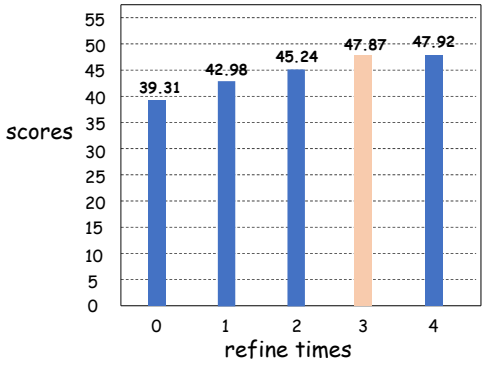

Figure 8: Score improvement when increasing refine times (plan times = 3, experience = 1).

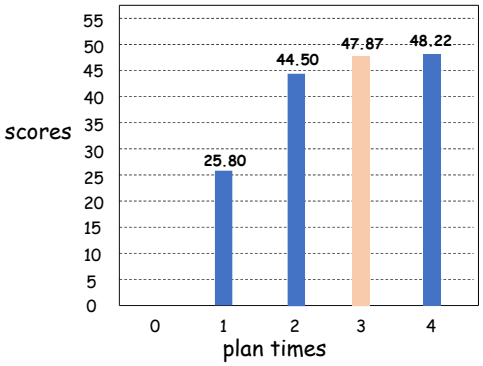

Figure 9: Scores improvement when increasing plan times (refine times = 3, experience = 1).

As observed in our preliminary small-scale experiments, regardless of the *plan times*, when the *refine times* within a single plan exceeds 3, the additional refinement iterations are largely futile. The agent tends to get trapped in failure loops and unable to constructively correct errors. Figure 8 validates this observation, showing that score improvements beyond 3 refinement iterations are negligible and fall within the bounds of random variance. Subsequently, while fixing the *refine times* at 3, we evaluated the impact of varying *plan times* (Figure 9). The results demonstrate that when *plan times* exceeds 3, the further performance gains are minimal and can primarily be attributed to the increased probabilistic chance of randomly sampling a correct trajectory. Note that both sets of experiments were conducted with the self-evolving mechanism enabled.

A.2 EXPLORATION OF THE BEST NUMBER OF PLAN EXPERIENCE ADVICE.

Table 3: Balancing planning and experience extraction.

Plan Times	Refine Times	Experiences	Three-chain
3	3	3	43.84
3	3	2	45.84
3	3	1	47.87
3	3	0	36.52

As shown in Table 3, while keeping both *plan times* and *refine times* at 3, we evaluated the impact of the number of provided *experiences*. Interestingly, when the number of experiences exceeds 1, the performance exhibits a downward trend. This decline can be primarily attributed to the fact that when the reasoning model receives multiple related positive and negative suggestions simultaneously, it tends to suffer from prompt hallucination and context confusion, rendering it unable to effectively isolate and adopt the optimal advice. Therefore, providing exactly one experience (consisting of one positive suggestion and one negative suggestion) yields the best results.

With these findings, we conclusively determined our fundamental hyperparameters: *plan times*, *refine times*, and the number of *experiences*. The determination followed a specific sequential order: we first established the optimal *refine times*, as preliminary small-scale experiments indicated that this parameter is relatively independent and does not fluctuate synergistically with changes in *plan times* or *experiences*. Next, given a single baseline experience, we optimized the *plan times*. Finally, with the previous parameters locked, we finalized the optimal number of *experiences*.

A.3 Exploration of Learning Rate

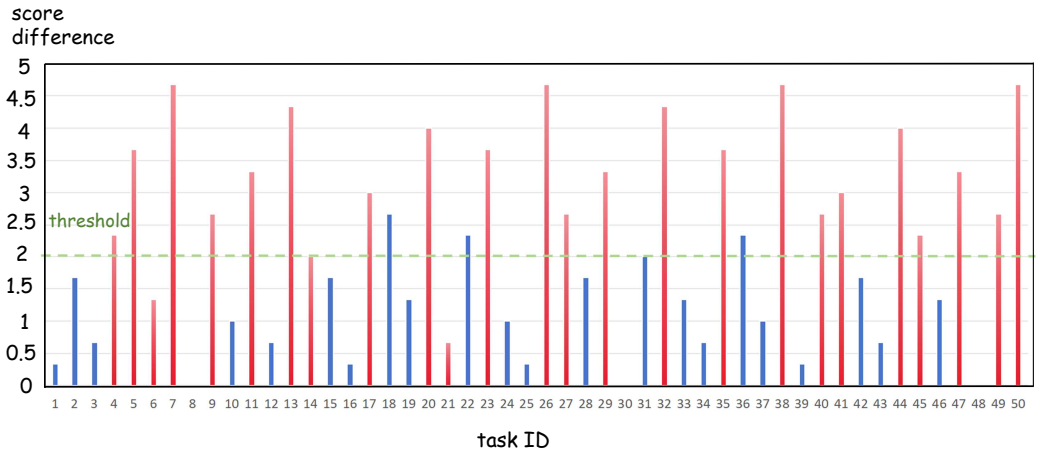

Figure 10: The horizontal axis represents the task index t_1, t_2, etc. The vertical axis denotes the score difference between the plans ($S_{t_i,diff} = max(p_1, p_2, p_3) - min(p_1, p_2, p_3)$). Red columns indicate instances where human annotators identified significant differences between plan outputs, whereas blue columns represent instances with minor visual discrepancies.

To prevent non-essential computational resource consumption and to avoid contaminating the experience library with low-value insights, we establish a stringent threshold for storing experiences. The system is constrained to trigger the experience extraction module only when the average score difference among the concurrent plans reaches a predetermined threshold. Given that each step is evaluated on a scale of 1 to 5 and the final score is the average across all steps, we conclusively set this extraction threshold to a score difference of 2 ($S_{difference} = 2$).

The rationale for this threshold is empirically justified in Figure 10, which illustrates that the boundary $S_{difference} = 2$ effectively segregates the red columns (significant differences) from the blue columns (minor differences). Although a minimal number of red columns fall below this threshold, such deviations can primarily be attributed to inherent preference discrepancies between VLM evaluators and human annotators. Consequently, this threshold operates as a robust filter, ensuring that only substantial methodological improvements are recorded as reusable experiences.

A.4 Latency and Computational Cost Analysis

Table 4: Latency and Computational Cost on a single three-chain dependent task.

Method	GPU time	Avg. total time per task	VLM API cost (USD)	Computational cost (USD)	Total cost (USD)
EvoEdit (Ours)	49s	2m35s	0.056	0.005	0.061
Nano Banana	-	very low	0.040	-	0.040
ComfyMind (updated)	1m02s	3m25s	0.129	0.007	0.136
GenArtist (ComfyUI)	1m22s	3m03s	0.057	0.009	0.066

As demonstrated in Table 4, when tackling three-chain dependent tasks, our method outperforms existing open-source agentic frameworks across latency, average token consumption, and overall cost. Furthermore, when compared against the strong closed-source model Nano Banana, although EvoEdit exhibits noticeably higher latency and an approximate 40% increase in cost (excluding the operational cost of the local RTX 4090 GPU, which is essentially negligible), our framework achieves a performance improvement of approximately 31.55%. We consider trading this degree of latency and computational cost for higher task success rates to be an acceptable compromise for complex image editing applications.

Additionally, Table 4 indicates that the structural balance between MPE and local refinement successfully manages efficiency. By employing concurrent multi-path exploration, the system adopts the numerous refinement iterations within a single path, which effectively curtails the overall latency. Concurrently, the self-evolving mechanism controls the total number of generated plans. By providing effective historical suggestions to the planner, the evolving module prevents the system from blindly initiating numerous exploratory paths through random combinatorial instructions, thereby controlling the overall token cost. The synergy of these three mechanisms enables our framework to reach competitive performance comparable to strong closed-source models.

B Additional Qualitative Visualizations & Examples

B.1 Visual Comparisons of different plans

Complementing the primary results in Section 4, we present broader qualitative comparisons of different plans generated by EvoEdit.

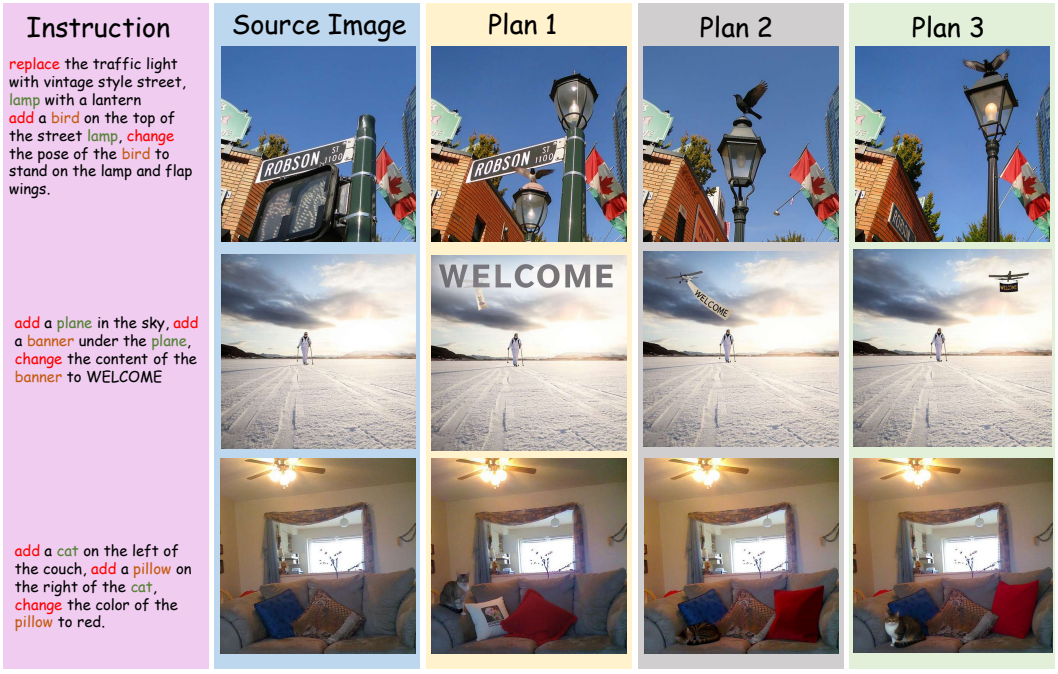

Figure 11: Differences between different plan(path) when solving the same task.

B.2 Recursive Verification Traces

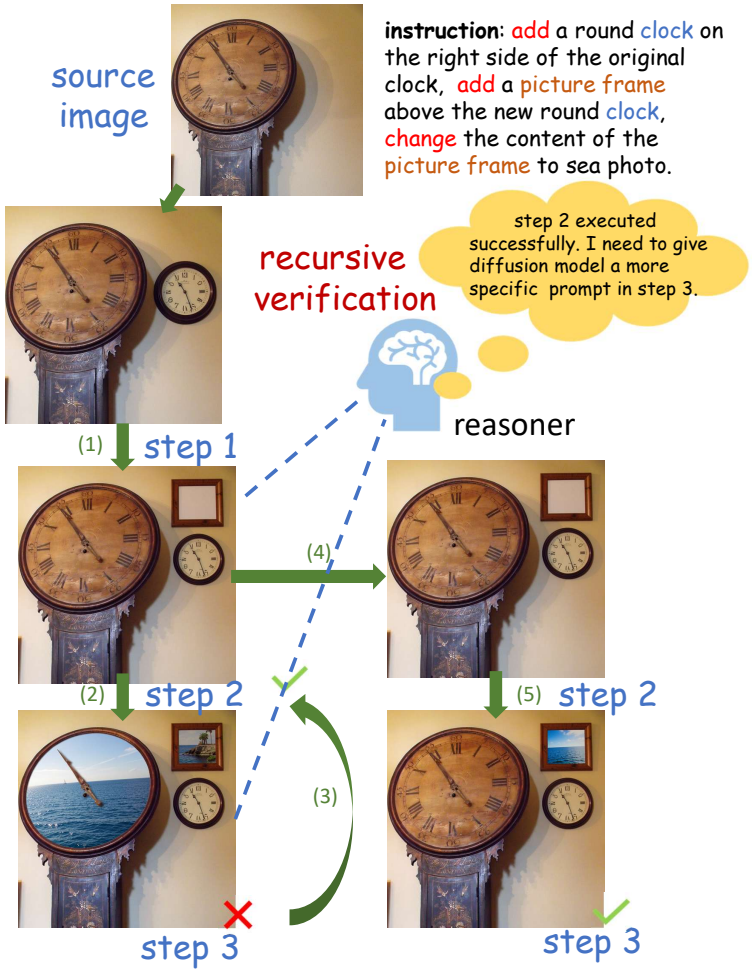

Figure 12: A verification trace is illustrated. The green arrow and numbers in brackets indicate the time sequence of operations.

To illustrate the internal reasoning of our framework, Figure 12 visualizes a recursive trace of the pipeline in action. It demonstrates the initial plan, the detection of targeted failures during verification, the subsequent local backtracking process, and the ultimately refined successful step.

C Implementation Details

C.1 Preparation

EvoEdit operates as an agent framework communicating with a local ComfyUI backend via HTTP and WebSocket protocols. All experiments were conducted on a single workstation equipped with one NVIDIA RTX 4090 GPU. Concurrent attempts in the MPE framework were efficiently scheduled and executed in parallel utilizing the advantage of queuing prompt of ComfyUI platform. The Base Model includes various open-source text clip model and diffusion model.

C.2 Base Diffusion Models and Text Encoding Configurations

To make the experimental setup reproducible, Table 5 summarizes the model components used by the ComfyUI workflows. The table reports the base diffusion model or checkpoint, VAE, text encoder, ControlNet modules, and captioning or auxiliary models for each workflow.

Table 5: Base diffusion models, text encoding models, and auxiliary components used in the ComfyUI workflows.

Workflow	Base / diffusion model	Checkpoint	VAE	CLIP / text encoder	ControlNet	Caption / other model
01	flux-2-klein-9b-fp8	–	flux2-vae	qwen_3_8b_fp8mixed	–	–
02	flux-2-klein-9b-fp8	–	flux2-vae	qwen_3_8b_fp8mixed	–	–
03	flux-2-klein-9b-fp8	–	flux2-vae	qwen_3_8b_fp8mixed	–	Florence-2-large-PromptGen-v2.0
04	–	majicmix Realistic_v7	vaeFtMse 840000 EmaPruned	from checkpoint	control_v11p_sd15_lineart	–
05	–	v1-5-pruned-emaonly	vaeFtMse 840000 EmaPruned	from checkpoint	control_v11f1p_sd15_depth; control_v11p	–
06	flux-2-klein-9b-fp8	–	flux2-vae	qwen_3_8b_fp8mixed	–	–

C.3 Semantic Workflow Interface and Workflow Library

Building on prior works based on ComfyUI, such as ComfyMind (Guo et al., 2025), SWI is a mechanism to convert natural language instructions into executable workflows. The workflow templates are constructed as follows:

Workflow	**Usage**	**Input_Parameters**
workflow01	add object, delete object, ...	image01, instruction01.
workflow02	change pose, change background.	image01, instruction01.
workflow03	replace object(reference).	image01, image02, instruction01.
...	...	...

Table 6: Workflow library that can be constructed and updated by users.

Operation	Available Workflow	Input_Parameters
add object	workflow01, workflow04, ...	image01, instruction01.
change pose	workflow02, ...	image01, instruction01.
replace object(reference)	workflow03, ...	image01, image02, instruction01.
...	...	...

Table 7: Workflow library generated for agents.

C.4 System Architecture

Horizontal Architecture. Our framework operates on a concurrent multi-path execution mechanism. For a given editing task, the scheduling engine initiates multiple exploratory paths to enforce structural diversity. Within each individual path, the system employs a systematic vertical architecture, which is detailed in the subsequent paragraph. Upon the completion of all concurrent paths, the generated outputs are independently evaluated and scored. Ultimately, these evaluated scores, alongside their corresponding execution plans, are forwarded to the Plan Reasoner to facilitate experience extraction and synthesis.

Vertical Architecture. Within each distinct path, the editing process is driven through a five-stage internal pipeline. Stage 1 (Global Planning) decomposes the complex user instruction into actionable steps. Stage 2 (Workflow Construction) translates these sequential steps into executable workflows through SWI. Stage 3 (Execution) sends the workflows to the ComfyUI backend to generate intermediate outputs. Stage 4 (Verification and Refinement) conducts visual evaluations, initiating recursive backtracking to isolate the earliest point of error if failures are detected. Finally, Stage 5 (Step Reasoner) diagnoses the identified failure and proposes concrete rectifications, such as adjusting workflow choices or textual instructions.

Workflow and Experience Library. The atomic workflows utilized by the platform are persistently stored as individual JSON files. To ensure efficient modular management and rapid retrieval, the aggregated meta-information of all available workflows is maintained within a centralized JSON file. Similarly, the historical experiences deduced by the reasoning modules are systematically serialized and stored in JSON formats, enabling the system to seamlessly query past insights during subsequent planning phases.

C.5 Hyperparameter and VLM Settings

The **hyperparameters** *plan times*, *refine times*, and *experiences* are designed as flexible variables that can be dynamically adjusted based on task complexity. In our primary experiments, we set *plan times* = 3, *refine times* = 3, and *experiences* = 1. This configuration represents an empirically derived optimal balance between performance and computational overhead, as extensively discussed and validated in Sections A.1 and A.2. Furthermore, the **hyperparameters** governing the *evolving* mechanism specifically dictate the criteria for the *write experience* operation. As discussed in Section A.3, the system triggers experience extraction only when the score gap between distinct exploratory paths reaches a threshold of 2, ensuring that only substantial methodological improvements are recorded.

Regarding the underlying **vision-language models**, the Step Reasoner and Plan Reasoner modules are powered by GPT-5.4, as these tasks demand advanced reasoning capabilities to accurately deduce and synthesize effective experiences. Conversely, the Verifier module utilizes GPT-4o. Compared to reasoning, verification imposes lower cognitive demands but requires significantly higher call frequencies. Thus, GPT-4o serves as an effective compromise, optimizing both cost and performance. Furthermore, during the execution and planning phases, we employ the `text-embedding-3` model to retrieve semantically similar instructions from historical data, ensuring precise and contextual extraction of relevant experiences.

D LIMITATIONS

Despite the improvements achieved by EvoEdit, the system still exhibits specific limitations. First, our framework utilizes advanced reasoning models such as GPT-4o and GPT-5.4, meaning that the overall effectiveness inherently depends on the capabilities of these underlying models. Second, the entire system is built upon the ComfyUI platform, which introduces a certain technical threshold that might hinder its widespread application among general users. Third, our focus on complex multi-step tasks may limit the system's efficiency in simpler editing scenarios. Finally, the powerful image editing capabilities demonstrated by our system could potentially cause negative impacts if intentionally misused for generating deceptive content.

E SYSTEM PROMPTS

E.1 GLOBAL PLANNING PROMPTS

Planner Prompt

```
You are an expert ComfyUI AI Agent workflow planner.
Your task is to analyze the provided source image and the user's editing instruction, and generate a step-by-step global editing plan.

User Instruction: "{instruction}"

Your plan should be broken down into steps. For each step, you must decide what action to perform.
The available actions are:
- **Object level**: Add object, Delete object, Change object
- **Attribute level**: Change color, Change material, Change content, Change pose, Change text
- **Global level**: Change background, Change style

{additional_requirements}

CRITICAL INSTRUCTION: Do NOT split a "Change object" action into separate "Delete object" and "Add object" actions. Use "Change object"
directly when replacing an existing object with another one.
The "instruction" for each step should be concise. Do not add unnecessary requirements beyond the user's original instruction unless absolutely
necessary.But you must accurately describe the position of the object in the picture.

 ## Output Format
The return format should be followed by the JSON object wrapped in <json> </json>.
{{
   "analysis": "Brief visual analysis of the image and the task",
   "remaining_step": "Number of remaining steps, an integer.",
   "plan": [
      {{
         "step": 1,
         "action": "The action to perform for this step. Must be one of the available actions (e.g., Add object).",
         "instruction": "The instruction to guide the diffusion model."
      }},
      {{
         "step": 2,
         "action": "The action to perform for this step. Must be one of the available actions (e.g., Change background).",
         "instruction": "The instruction to guide the diffusion model."
      }}
   ]
}}

 additional_requirements = (
            f"CRITICAL INSTRUCTION: Your generated plan MUST be different from these previous plans we have tried:
{previous_plans_str}.\n"
            f"Please explore a different approach. You can change the plan by:\n"
            f"Merging steps: For example, if the instruction involves adding a new object and then adding a mark/sign to the newly added object,
merge these two steps into a single step. When merging steps, do NOT include unnecessary details from previous plans that are not explicitly
requested in the instruction. Your sole goal is to fulfill the instruction.\n"
            f"Changing orders: but don't do this unless it's necessary to change the order."
            f"{experience_str}"
```

Figure 13: verifier prompt.

E.2 RECURSIVE VERIFICATION PROMPTS

The criteria provided to the VLM to independently verify the success of a generated step. This includes logic for checking spatial relationships, color alignments, and ensuring no missing constraints, which fundamentally drives the recursive backtracking.

Verifier Prompt

```
You are evaluating if an image successfully followed the user editing instruction: '{instruction}'.

You are provided with two images:
1. The first image is the original source image.
2. The second image is the generated (edited) image.

First, analyze the images visually to determine:
1. Has the editing instruction '{instruction}' been properly executed in the edited image compared to the original image? Your verification should be strict. When adding new objects, check if their spatial relationship with existing objects is reasonable.
CRITICAL INSTRUCTION FOR VERIFICATION: DO NOT check or evaluate specific text content, spelling, or fonts in the image UNLESS the user's instruction EXPLICITLY asks to modify or add text!

If the editing is not successful, you must then determine which step in the plan failed.
Here is the step-by-step editing plan that was executed:
{plan}

CRITICAL INSTRUCTION: If you determine that the editing failed, you MUST identify the EARLIEST (FIRST) step in the sequence that failed. The failure of an early step often causes subsequent steps to fail, so always find the first incorrect step!

The return format should be followed by the JSON object wrapped in <json> </json>.
{{
   "passed": true_or_false,
   "reason": "Detailed explanation of the visual verification comparing both images",
   "failed_step": "The integer step number of the FIRST step that failed, or null if all passed",
   "step_failed_reason": "Brief reason for why the specific step failed, or null if all passed"
}}
"""
```

Figure 14: verifier prompt.

Back Verifier Prompt

```
You are evaluating if a specific step in an image editing process was successful.
You are evaluating Step {step} and all steps before it from the following step-by-step editing plan:
{plan}

You are provided with two images:
1. The first image is the original source image.
2. The second image is the generated image for Step {step}.

Analyze the images visually to determine:
1. Have all the editing instructions up to and including Step {step} been properly executed? Your verification should be strict. When adding new objects, check if their spatial relationship with existing objects is reasonable.
CRITICAL INSTRUCTION FOR VERIFICATION: DO NOT check or evaluate specific text content, spelling, or fonts in the image UNLESS the user's instruction EXPLICITLY asks to modify or add text!
2. Is the non-edit region appropriately preserved according to the intentions of these steps?

The return format should be followed by the JSON object wrapped in <json> </json>.
{{
   "passed": true_or_false,
   "step_failed_reason": "If passed is false, a brief and direct reason how THIS specific Step {step} failed its instruction (do not mention previous steps). If passed is true, return null."
}}
```

Figure 15: back verifier prompt.

E.3 STEP AND PLAN REASONER PROMPTS

PLan Reasoner Prompt

```
You are an expert AI planning and reasoning agent specializing in evaluating image editing pipelines.
Below are {num_plans} different execution plans attempted for the same task, along with their respective evaluation scores:


{plans_text}


Plan {best_plan_idx} received the highest score of {max_score}.
Please analyze these plans and provide both a positive suggestion and a negative suggestion for the plan agent to improve its
future planning.
The suggestions should be concise and broadly applicable.


CRITICAL INSTRUCTION:
1. Both the `positive_suggestion` and `negative_suggestion` MUST be general, which can be applied to other scenarios.
2. The `positive_suggestion` can describe which steps should be merged into a single step, or which steps should be done first.
   Example: "if the instruction involves adding a new object and then adding a sign to the newly added object, merge these two
steps into a single step."


 ## Output Format
The return format should be followed by the JSON object wrapped in <json> </json>.
{{
  "positive_suggestion": "...",
  "negative_suggestion": "..."
}}
```

Figure 16: plan reasoner prompt.

Step Reasoner Prompt

```
You are an expert image editing prompt engineer and root cause analyzer for a ComfyUI AI Agent.
Your task is to analyze why the executed editing step failed to produce the correct image according to its instruction, and
provide actionable suggestions for the refiner.
Specifically, think about why the diffusion model failed at this specific step. Remember that the input to the diffusion model
at this step is exactly the text provided in the "instruction" field of that step.
A possible reason for failure is that the instruction for this step was not clear enough for the diffusion model.

You are provided with two images to assist your analysis:
1. The image BEFORE executing the failed step.
2. The resulting image AFTER executing the failed step.

The step that was executed between the two images:
{failed_step}

Your goal is to suggest a new "instruction" for this failed step.

The return format should be followed by the JSON object wrapped in <json> </json>.
{{
   "new_instruction": "Based on the failure analysis, provide a revised, more explicit and clearer instruction for this specific
step that the diffusion model can execute correctly."
}}
```

Figure 17: step reasoner prompt.

E.4 CHAIN EVALUATOR PROMPT

Chain Evaluator Prompt

You are an extremely strict, detail-oriented, and professional general image editing quality evaluator. You will assess an image editing task and assign a final score between 0 and 5 for EACH instruction.
Input Information:
1. Original Image
2. Edited Image
3. Editing Instructions details (provided as a JSON list above).

CRITICAL INSTRUCTION: The Edited Image is the result of executing all the "Editing Instructions". You must evaluate the success of EACH instruction individually. DO NOT penalize the score of one instruction for valid changes caused by other instructions.

You must evaluate each instruction based on the FOUR UNIFIED DIMENSIONS: Task Accuracy, Physical Plausibility, Visual Integration, and Cleanliness & Preservation.

Output Format
The return format should be followed by a JSON LIST of objects wrapped in ```json```. Each object corresponds to an instruction in the same order.

```
[
 {
   "instruction": "<instruction text>",
   "final_score": <0-5>,
   "reasoning": "Detailed justification..."
 }
]
```

Figure 18: chain evaluator prompt.