R2E-PROMPT: REASONING-TO-EXECUTABLE PROMPT GENERATION FOR IMPLICIT-INSTRUCTION IMAGE EDITING

Anonymous authors
Paper under double-blind review

ABSTRACT

We present R2E-Prompt, a training-free framework that formulates implicit-instruction image editing as a structured reasoning task in language space. Existing editors translate explicit commands faithfully but struggle with abstract conditions whose visual consequences are not directly named, leaving the entailed changes unrealized in the output. R2E-Prompt addresses this by moving reasoning entirely upstream of the editor: a VLM and an LLM produce, in a single pass, an explicit element-level edit specification consumable by any off-the-shelf instruction-based editor. The framework comprises three stages: *Target-State Reasoning* first grounds the source image in a source caption and then derives, via chain-of-thought reasoning, an explicit target caption that makes the implicit visual consequences of the instruction explicit; *Edit-Aware Scene Graph Transformation* parses the source and target captions into text-derived scene graphs and casts source-to-target editing as a structured prediction problem over scene-graph elements; and *Executable Prompt Generation* linearizes the labeled graph into a natural-language editing command. Without architectural modification or fine-tuning, R2E-Prompt achieves state-of-the-art performance among open-source baselines on the RISEBench and RE-Edit benchmark.

1 INTRODUCTION

Image editing is a fundamental task across creative content production, design, and visual communication. Recent instruction-based image editing models (Brooks et al., 2023; Zhang et al., 2023a; Huang et al., 2024) have made these workflows accessible to non-experts by translating natural-language commands into high-fidelity visual manipulations. In practice, however, users rarely enumerate every visual change they want; they describe their intent through implicit conditions, expecting the editor to infer the resulting visual state. For example, as illustrated in Figure 1, an image ``after many people have walked on it'' implies footprints in the snow rather than the people themselves, while a scene ``during the Jurassic period'' implies a living dinosaur rather than merely changing the museum's surroundings. Such implicit instructions cannot be fulfilled by surface-level word matching—they require *reasoning* about the visual consequences entailed by the user's condition. Existing editors, primarily optimized to treat explicit instructions as conditioning signals (Brooks et al., 2023; Zhang et al., 2023a), lack the *reasoning* needed to infer the visual consequences such instructions entail—producing edits that match the words of the instruction rather than the visual outcome it implies.

To equip editors with this reasoning capability, prior work follows one of two routes. One route embeds reasoning into the editor via reasoning-augmented finetuning or jointly trained vision-language architectures (Fu et al., 2024; Huang et al., 2024; Xu et al., 2025; Zhang et al., 2025; Ji et al., 2025; Bai et al., 2025a; Kou et al., 2026; Deng et al., 2025; Qin et al., 2025), coupling reasoning with a specific training architecture. The other keeps the editor frozen but reaches the target state through iterative update, running the editor repeatedly to decompose instructions or revise intermediate outputs (Zeng et al., 2025; Li et al., 2025). Consequently, the first approach requires fine-tuning the specific editor or its auxiliary modules, while the second remains training-free but reaches the target state through repeated invocations of the editor.

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.



Figure 1: Qualitative comparison with instruction-based image editing model Qwen-Image-Edit (Wu et al., 2025). On implicit instructions, Qwen-Image-Edit either reacts literally to surface words (*people*, *2 minutes*; left) without realizing the visual change they entail, or fails to infer the implicit consequence (*fully mixed*, *during the Jurassic period*; right). In contrast, R2E-Prompt reasons through these implicit consequences and produces edits aligned with the user's intent.

To overcome these limitations, we propose **Reasoning-to-Executable Prompt (R2E-Prompt)**, a single-pass training-free framework that tackles the implicit-instruction setting introduced in Figure 1, where the visual consequences of a command are entailed but left unstated. Our key insight is to formulate implicit-instruction interpretation as a *structured reasoning task in language space*, performed entirely upstream of the editor: R2E-Prompt unpacks the implicit instruction via chain-of-thought reasoning into a structured intermediate representation at the scene-element level, which is then rendered into a single editing command consumable by any off-the-shelf instruction editor.

Concretely, R2E-Prompt consists of three stages. *Target-State Reasoning* first grounds the source image in a textual caption and then infers, through structured language reasoning, an explicit caption of the intended final visual state implied by the user's implicit instruction. *Edit-Aware Scene Graph Transformation* lifts this caption-level information to the element level by parsing it into a structured scene graph (Dhamo et al., 2020; Zhang et al., 2022) and labels each element with an edit operation to form a unified *edit scene graph*. *Executable Prompt Generation* renders the results of each stage as an editing instruction prompt compatible with any off-the-shelf editor. With this pipeline, R2E-Prompt achieves state-of-the-art performance among open-source baselines on a reasoning-intensive editing benchmarks (Zhao et al., 2026; Ding et al., 2026) while remaining single-pass, training-free, and editor-agnostic.

In summary, our contributions are as follows:

- We propose **R2E-Prompt**, a training-free, single-pass framework that formulates implicit-instruction image editing as a structured-language reasoning task: VLM and LLM reasoning produce, in language space, an editor-executable specification that is consumed by any off-the-shelf instruction-based image editor without architectural modification.
- We propose *Target-State Reasoning*, which decouples implicit reasoning from the editor by performing it entirely in language space, anchoring downstream editing in an explicit target state.
- We propose *Edit-Aware Scene Graph Transformation*, which makes the source-to-target transition explicit at the element level through a scene graph representation, yielding an element-level edit specification.
- Extensive experiments on a reasoning-intensive image editing benchmark show that R2E-Prompt achieves state-of-the-art performance among open-source baselines.

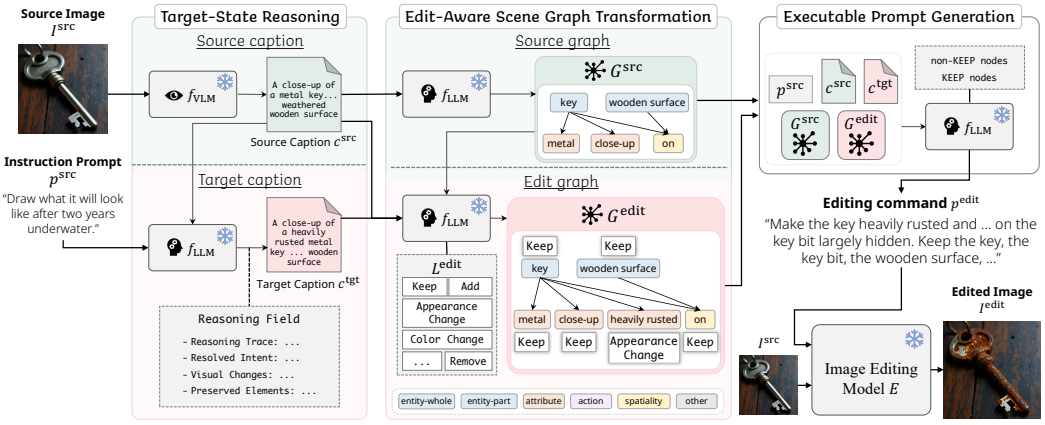


Figure 2: Overall pipeline of R2E-Prompt. Given a source image I^{src} and a source instruction prompt p^{src}, *Target-State Reasoning* grounds I^{src} in a source caption c^{src} via a frozen VLM, followed by an LLM-based reasoning step that, from $(c^{\text{src}}, p^{\text{src}})$, unpacks the implicit visual consequences of p^{src} into an explicit target caption c^{tgt}. *Edit-Aware Scene Graph Transformation* parses c^{src} into a scene graph G^{src} and, conditioned on $(c^{\text{src}}, c^{\text{tgt}})$, relabels each node with an edit operation $\ell \in L^{\text{edit}}$ to yield the edit scene graph G^{edit}. *Executable Prompt Generation* renders the graph pair $(G^{\text{src}}, G^{\text{edit}})$ together with $(c^{\text{src}}, c^{\text{tgt}}, p^{\text{src}})$ into an editing command p^{edit}, consumed by a frozen off-the-shelf editor E with I^{src} to produce the edited image I^{edit}. Our system prompt templates are provided in Appendix E.

2 Related Work

2.1 Instruction-based Image Editing

Instruction-based image editing lets users edit images via free-form natural language commands, removing the need for masks (Avrahami et al., 2022; Yang et al., 2022) or region annotations (Li et al., 2023). Early instruction-guided editors commonly relied on shallow text encoders such as CLIP (Radford et al., 2021), with short, literal editing commands serving as the dominant form of text guidance (Brooks et al., 2023; Zhang et al., 2023a; Kawar et al., 2023). Recent general-purpose editors further leverage VLMs (Bai et al., 2025c; Mistral AI, 2025) or LLMs (Yang et al., 2025) for multimodal instruction understanding and alignment (Liu et al., 2025; Wu et al., 2025; Labs et al., 2025; Labs, 2026). This shift allows modern editors to handle longer and more complex editing instructions beyond short literal commands. However, these editors are primarily trained on explicit image–text modification pairs and therefore remain optimized for instructions that directly specify the desired visual change. When the intended modification is only implicitly entailed by the instruction, requiring inference about the target state, these editors tend to fail.

These limitations motivate the incorporation of reasoning capabilities into image editing to infer implied target states beyond explicit instruction following.

2.2 Reasoning-based Image Editing

Building on reasoning capabilities from large language and vision-language models (Brown et al., 2020; Wei et al., 2022; Zhang et al., 2023b), recent work has incorporated such reasoning into image editing along two main directions. One line of work integrates reasoning directly into the editing architecture through joint optimization with the editor (Fu et al., 2024; Huang et al., 2024; Xu et al., 2025; Zhang et al., 2025; He et al., 2026), reasoning-augmented fine-tuning (Yin et al., 2025; Ji et al., 2025; Kou et al., 2026), or unified vision-language editing architectures (Bai et al., 2025a; Deng et al., 2025; Qin et al., 2025). While effective, these approaches tightly couple reasoning with specific editor architectures and training pipelines, limiting their compatibility across different editing frameworks. Meanwhile, another line of work keeps the editor frozen and delegates reasoning to external modules (Zeng et al., 2025; Li et al., 2025). These methods typically produce the target image

through iterative loops—generating an intermediate image, inspecting it, and re-editing—requiring multiple editor invocations per instruction and incurring substantial inference-time overhead.

In contrast, R2E-Prompt formulates implicit-instruction interpretation as a structured reasoning task in language space, performed upstream of the editor in a single pass and applicable to any off-the-shelf instruction-based editor without modification.

3 METHOD

Given a source image I^{src} and a source instruction prompt p^{src} whose intended visual changes are only implicitly entailed, R2E-Prompt aims to produce an editing instruction prompt p^{edit} that makes those changes explicit while specifying which source elements should be preserved. We define R2E-Prompt as a VLM/LLM-based training-free transformation F that maps $(I^{\text{src}}, p^{\text{src}})$ to p^{edit}; a frozen, off-the-shelf instruction-based image editor E then consumes p^{edit} without modification to produce the edited image I^{edit}:

$$p^{\text{edit}} = F(I^{\text{src}}, p^{\text{src}}), \qquad I^{\text{edit}} = E(I^{\text{src}}, p^{\text{edit}}), \tag{1}$$

where F factorizes into three stages running over frozen models, all of which reason in language space. First, *Target-State Reasoning* (Section 3.1) grounds I^{src} in a source caption c^{src} and infers from $(c^{\text{src}}, p^{\text{src}})$ a target caption c^{tgt} describing the intended final state. Next, *Edit-Aware Scene Graph Transformation* (Section 3.2) lifts this caption-level information into an element-level scene graph that specifies, for every source element, whether it should change, be removed, or be preserved. Finally, *Executable Prompt Generation* (Section 3.3) renders this graph—together with the upstream captions and original prompt—into a single editing command p^{edit} that can be executed by any instruction-based editor. The overall pipeline is illustrated in Figure 2. For examples of the system prompt of each process, please refer to Appendix E.

3.1 TARGET-STATE REASONING

Since p^{src} typically specifies a desired condition (e.g., "during the Jurassic period"; Figure 1) rather than the concrete visual transformations that satisfy it, we leverage a pretrained VLM and LLM to unpack its implicit visual consequences. Target-State Reasoning (TSR) operates in language space: it first grounds the source image I^{src} in a source caption c^{src}, then infers from c^{src} and p^{src} an explicit target caption c^{tgt} specifying the visual state that the edited image I^{edit} should attain.

Source caption. To anchor reasoning in language space, we leverage a frozen vision-language model to ground I^{src} in text:

$$c^{\text{src}} = f_{\text{VLM}}(I^{\text{src}}), \tag{2}$$

where f_{VLM} denotes the frozen VLM and is prompted to describe only what is clearly visible in I^{src}—entities, attributes, actions, and spatial layout—and to avoid speculation or unsolicited commentary.

Target caption. To generate the explicit target caption from the implicit visual consequences of p^{src}, we leverage the Chain-of-Thought (CoT) prompting (Wei et al., 2022). Our CoT-based approach structures the LLM's output into four distinct reasoning fields, which consist of: (i) a *Reasoning Trace* explains how the inferred target visual state is derived from c^{src} under p^{src}; (ii) a *Resolved Intent* summarizes the intended edit; (iii) a *Visual Changes* enumerates objects, attributes, states, actions, or relations that must change; (iv) a *Preserved Elements* identifies source elements unaffected by p^{src}. The target caption is emitted from this decomposition:

$$c^{\text{tgt}} = f_{\text{LLM}}(c^{\text{src}}, p^{\text{src}}), \tag{3}$$

where f_{LLM} is a frozen large language model. The resulting c^{tgt} specifies the visual state that the edited image I^{edit} should attain.

3.2 EDIT-AWARE SCENE GRAPH TRANSFORMATION

Target-State Reasoning yields c^{src} and c^{tgt} that explicitly describe the source and target visual states, but *describing* the two states is not the same as *specifying how to get from one to the other*. Since our goal is to produce a single editing command for the downstream editor, this transition—which

source elements should be preserved and which should be edited—must be made explicit at the element level, but a caption pair alone makes this difficult to express in a directly usable form. We therefore introduce a scene graph as a structured intermediate textual representation, where each entity, attribute, and relation is represented by a separate node that can be independently labeled with an edit operation. Given $(c^{\text{src}}, c^{\text{tgt}})$, Edit-Aware Scene Graph Transformation (ESGT) parses c^{src} into the source scene graph G^{src} and relabels each node with an edit operation, yielding the edit scene graph G^{edit} that turns the two state descriptions into an explicit specification of the source-to-target transition.

Source graph. To make this element-level labeling possible, we first parse c^{src} into a text-derived scene graph (Cho et al., 2024; Wu et al., 2026)—typed nodes connected by directed edges—so that each element of the source state becomes an isolable unit with a stable identity for downstream labeling. The LLM produces the source scene graph G^{src}:

$$G^{\text{src}} := (V^{\text{src}}, D^{\text{src}}) = f_{\text{LLM}}(c^{\text{src}}), \tag{4}$$

where V^{src} consists of N_v^{src} typed nodes $\{v_i\}_{i=1}^{N_v^{\text{src}}}$, each typed by one of six categories—`entity-whole`, `entity-part`, `attribute-`$\langle$`category`$\rangle$, `action`, `spatiality`, or `other`—with the attribute kind $\langle$`category`$\rangle$ ranging over color, material, state, and similar properties. D^{src} consists of N_d^{src} directed edges $\{d_i\}_{i=1}^{N_d^{\text{src}}}$, with each edge d_i anchoring a parent node to its child node (e.g., `entity-whole` to `attribute-`$\langle$`category`$\rangle$, or `entity-whole` to `entity-part`).

Edit graph. Given the source scene graph G^{src}, we make the source-to-target transformation explicit at the element level by assigning each source graph node an operation label that specifies how the node should be edited (or preserved) to realize the target state. Building on the whole-image editing primitives of Hu et al. (2026), we define a label space of fourteen operations partitioned by the scene-graph node type they target, adapted to element-level granularity (full specification in Appendix B). Three *universal* primitives (ADD, REMOVE, KEEP) apply to any node, while the remaining eleven are type-specific—covering entity-level (REPLACE, RESIZE, MOVE), attribute-level (COLOR/MATERIAL/APPEARANCE/EXPRESSION CHANGE), action-level (ACTION CHANGE), and scene-level (BACKGROUND/TONE/STYLE CHANGE) edits. We include KEEP as a dedicated label so that preserved facts are stated explicitly in the edit graph rather than left as the implicit complement of edited nodes.

Conditioning jointly on the source graph G^{src} (the structure to relabel) and the caption pair $(c^{\text{src}}, c^{\text{tgt}})$ describing the source and target states, the LLM produces the edit scene graph G^{edit}:

$$G^{\text{edit}} := (V^{\text{edit}}, D^{\text{edit}}, L^{\text{edit}}) = f_{\text{LLM}}(G^{\text{src}}, c^{\text{src}}, c^{\text{tgt}}), \tag{5}$$

where V^{edit} consists of N_v^{edit} typed nodes $\{v_i\}_{i=1}^{N_v^{\text{edit}}}$ drawn from the same six categories as in V^{src}; D^{edit} consists of N_d^{edit} directed edges $\{d_i\}_{i=1}^{N_d^{\text{edit}}}$ anchoring each parent node to its child node; and $L^{\text{edit}} = \{\ell_i\}_{i=1}^{N_v^{\text{edit}}}$ assigns an operation label ℓ_i to each node v_i. The resulting G^{edit} provides an element-level specification of the source-to-target edit, identifying which source nodes should change, be removed, or be preserved to realize the target state.

3.3 EXECUTABLE PROMPT GENERATION

The edit scene graph G^{edit} encodes the source-to-target edit specification element-wise, but the editor-facing prompt p^{edit} should take the form of a natural-language command that an instruction-based editor can directly consume, rather than the graph structure itself. Conditioning jointly on the graph pair $(G^{\text{src}}, G^{\text{edit}})$ (the source structure and its labeled edit counterpart) and the natural-language context $(c^{\text{src}}, c^{\text{tgt}}, p^{\text{src}})$, Executable Prompt Generation (EPG) renders the editor-facing prompt p^{edit}:

$$p^{\text{edit}} = f_{\text{LLM}}(G^{\text{src}}, c^{\text{src}}, G^{\text{edit}}, c^{\text{tgt}}, p^{\text{src}}). \tag{6}$$

This rendered prompt is formatted as a two-segment executable instruction: a modification segment that issues directives for all non-KEEP nodes (with verbs aligned to each operation label and source values drawn from G^{src} for change operations), followed by a closing sentence listing the preserved nodes. Entities touched by a non-KEEP node are excluded from the preservation list, preventing the editor from being asked to both change and retain the same object. EPG thereby completes the

Table 1: Quantitative comparison on RISEBench (Zhao et al., 2026). We report per-category accuracy (%) over the four reasoning types (*Temporal*, *Causal*, *Spatial*, *Logical*) and the *Overall* accuracy over all 360 samples. Models are grouped by access: closed-source proprietary systems and open-source editors. **Bold** and underline denote the best and second-best results within each column among open-source methods.

Method	Temporal ↑	Causal ↑	Spatial ↑	Logical ↑	Overall ↑
Closed-source models					
Gemini-2.5-Flash-Image (Nano Banana) (Google DeepMind, 2025)	29.4	48.9	37.0	18.8	33.9
GPT-Image-1 (OpenAI, 2025)	36.5	34.4	37.0	10.6	30.0
GPT-Image-1-mini (OpenAI, 2025)	25.9	31.1	33.0	9.4	25.3
Gemini-2.0-Flash-Exp (Google DeepMind, 2024)	9.4	16.7	23.0	4.7	13.9
Seedream-4.0 (Seedream et al., 2025)	17.6	13.3	11.0	7.1	12.2
Open-source models					
Think-Then-Generate (Kou et al., 2026)	20.0	<u>28.9</u>	15.3	**30.0**	<u>23.9</u>
EditThinker-8B (Li et al., 2025)	10.8	23.3	27.0	8.2	17.8
BAGEL (w/ CoT) (Deng et al., 2025)	11.8	27.8	21.0	1.2	15.8
FLUX.2-Dev (Labs, 2026)	9.4	25.6	17.0	1.2	13.6
FLUX.2-Klein-9B-KV (Labs, 2026)	7.1	12.2	23.0	5.9	12.5
Uni-CoT (Qin et al., 2025)	8.2	18.9	20.0	1.2	12.5
FLUX.2-Klein-4B (Labs, 2026)	9.4	10.0	15.0	7.1	10.6
FLUX.1-Kontext-Dev (Labs et al., 2025)	2.3	5.5	13.0	1.2	5.8
BAGEL (Deng et al., 2025)	3.5	4.4	14.0	0.0	5.8
Step1X-Edit (Liu et al., 2025)	0.0	2.2	2.0	3.5	1.9
MGIE (Fu et al., 2024)	1.2	1.1	2.0	1.2	1.4
Qwen-Image-Edit (Wu et al., 2025)	<u>21.2</u>	18.9	<u>31.0</u>	4.7	19.4
Qwen-Image-Edit + R2E-Prompt (**Ours**)	**35.3**	**44.4**	**33.0**	<u>11.8</u>	**31.4**

R2E-Prompt mapping by delivering p^{edit} as a natural-language editing prompt directly compatible with off-the-shelf instruction-based editors.

4 Experiments

4.1 Implementation Details

We instantiate the vision-language model f_{VLM} with Qwen3-VL-8B-Instruct (Bai et al., 2025b) and the large language model f_{LLM} with Qwen3-30B-A3B-Instruct (Yang et al., 2025), both run in bfloat16 with greedy decoding (temperature $= 0$, top-$p = 1.0$) to eliminate sampling randomness. The editor E is the frozen Qwen-Image-Edit-2511 (Wu et al., 2025), run with its default configuration (40 denoising steps, true CFG scale 4.0, distilled guidance scale 1.0, empty negative prompt). Each source image is fed at its native aspect ratio, with the longer edge capped at 1024 pixels. All experiments are conducted on a single NVIDIA RTX PRO 6000 GPU with PyTorch. Our pipeline takes on average 72.25 seconds per instance, including image-editing time (17.82 seconds with reasoning only). Further implementation details are provided in Appendix A.

4.2 Evaluation Setup

Benchmark. We follow the evaluation protocol proposed in RISEBench (Zhao et al., 2026), a benchmark specifically designed for reasoning-informed visual editing. RISEBench consists of 360 image–instruction pairs spanning four reasoning categories—*Temporal* (85), *Causal* (90), *Spatial* (100), and *Logical* (85)—each providing a source image and a source instruction prompt p^{src} whose intended visual changes are only implicitly entailed. We use the official release without modification, and apply the same R2E-Prompt configuration to all instances. We additionally evaluate on RE-Edit (Ding et al., 2026) (Appendix D).

Evaluation metrics. Following the official protocol of RISEBench, we adopt an LMM-as-a-Judge pipeline based on GPT-4.1 (Achiam et al., 2023) that scores each edited image along three sub-dimensions: *Instruction Reasoning*, *Appearance Consistency*, and *Visual Plausibility*. A sample is counted as successful only when all three sub-scores meet the per-dimension threshold defined by the benchmark, and the per-category accuracy is reported as the percentage of successful samples

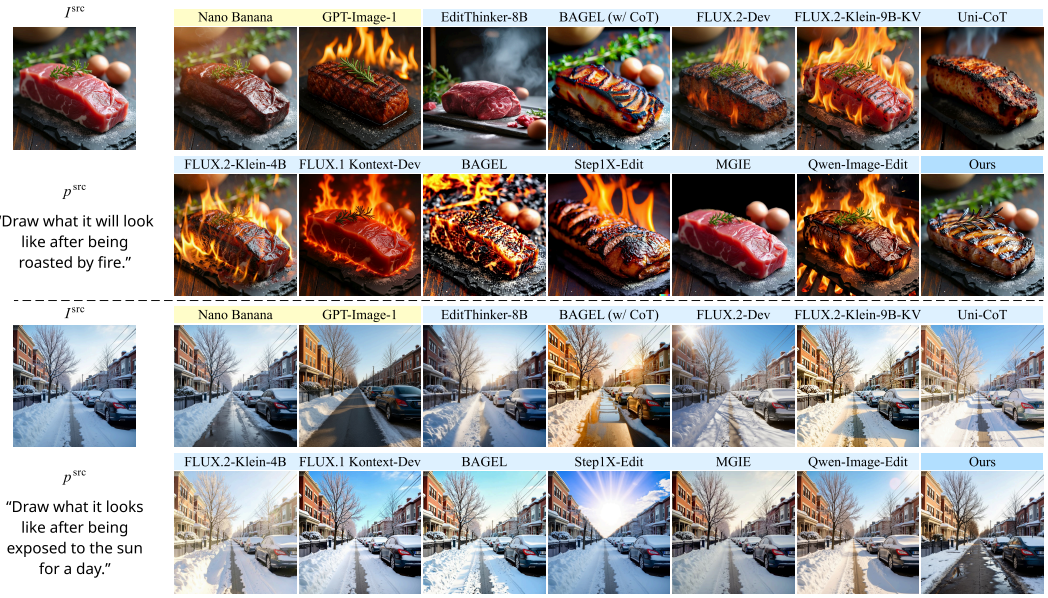


Figure 3: Qualitative comparison with state-of-the-art editing models. Closed-source and open-source models are highlighted in yellow and blue, respectively. For closed-source models, we include only Gemini-2.5-Flash-Image (Nano Banana) and GPT-Image-1, the two top-performing systems on RISEBench (Table 1). Also, we omit Think-Then-Generate (Kou et al., 2026) as its editing model weights are not publicly available.

within that category. We report per-category accuracy for the four reasoning types together with the *Overall* accuracy over all 360 samples, which serves as the primary metric for comparison.

4.3 Comparison with State-of-the-Art Editing Models

Quantitative comparison. Table 1 reports per-category and overall accuracy on RISEBench (Zhao et al., 2026). Applied on top of Qwen-Image-Edit (Wu et al., 2025), R2E-Prompt achieves the best overall accuracy of 31.4% among open-source methods, lifting the Qwen-Image-Edit baseline by 12.0 points (19.4% $\rightarrow$ 31.4%) without modifying the editor itself. The gains concentrate on instructions whose visual consequences must be inferred rather than named directly: R2E-Prompt outperforms all open-source baselines by a substantial margin on *Temporal* and *Causal* reasoning—categories that demand unpacking what an abstract condition entails—demonstrating that explicit target-state reasoning provides its strongest leverage precisely where the user's instruction is most implicit. Notably, despite operating purely as a training-free prompt-level intervention, R2E-Prompt achieves competitive performance with Gemini-2.5-Flash-Image (Google DeepMind, 2025) (33.9%) while outperforming several closed-source proprietary systems on the benchmark, substantially narrowing the open-source-to-closed-source gap.

Qualitative comparison. Figure 3 illustrates these patterns on two implicit instructions: ``after being roasted by fire'' and ``after being exposed to the sun for a day'', whose entailed final states are a properly cooked steak and melted snow, respectively. Most baselines fixate on surface-level words in the instruction rather than what those words entail. For ``roasted by fire,'' nearly every open-source baseline renders literal flames around an otherwise raw or partially cooked steak. For ``exposed to the sun for a day,'' the failure takes different forms: Step1X-Edit literally draws sun rays in the sky without affecting the snow, while BAGEL w/ CoT partially melts the road but leaves the snow on cars and rooftops untouched—capturing the consequence locally but failing to apply it consistently across the scene. The Qwen-Image-Edit baseline exhibits both failure modes: flames still surround the steak in the first case, and the snowy street remains nearly unchanged in the second.

Table 2: Ablation study on R2E-Prompt. *TSR* (Target-State Reasoning) produces the target caption c^{tgt}; *ESGT* (Edit-Aware Scene Graph Transformation) produces the edit scene graph G^{edit}. Executable Prompt Generation (EPG) is always retained as the final renderer—ablating TSR amounts to conditioning EPG without c^{tgt}, and ablating ESGT amounts to conditioning EPG without $(G^{\text{src}}, G^{\text{edit}})$. Row (d) is the full method. The symbol ↑ indicates that higher is better. The best and second-best results are highlighted in **bold** and underline, respectively.

#	Method	Temporal ↑	Causal ↑	Spatial ↑	Logical ↑	Overall ↑
(a)	Qwen-Image-Edit	21.2	18.9	<u>31.0</u>	4.7	19.4
(b)	Qwen-Image-Edit + TSR + EPG	**36.5**	<u>37.8</u>	28.0	<u>7.1</u>	<u>27.5</u>
(c)	Qwen-Image-Edit + ESGT + EPG	28.2	25.6	30.0	<u>7.1</u>	23.1
(d)	Qwen-Image-Edit + TSR + ESGT + EPG	<u>35.3</u>	**44.4**	**33.0**	**11.8**	**31.4**

Applying R2E-Prompt to the same Qwen-Image-Edit editor successfully addresses both failure modes: the steak appears properly grilled, with surface charring and no surrounding fire, and the street shows clear melting, with reduced snow on the road, cars, and rooftops. Among all baselines, Gemini-2.5-Flash-Image (Nano Banana) produces results of comparable quality. This visually confirms what Table 1 shows quantitatively—R2E-Prompt's target-state reasoning unpacks p^{src}'s implicit consequences, allowing an off-the-shelf editor to act on what the instruction *entails* rather than what it *literally says*.

4.4 Ablation Study

Table 2 disentangles the contributions of Target-State Reasoning (TSR) and Edit-Aware Scene Graph Transformation (ESGT), with Executable Prompt Generation (EPG) retained in every configuration as the final renderer. Table 2-(a) is the Qwen-Image-Edit baseline running directly on p^{src}, and Table 2-(d) is R2E-Prompt's full pipeline with both TSR and ESGT active. The partial settings remove one stage each: Table 2-(b) removes ESGT, so EPG conditions on the caption pair $(c^{\text{src}}, c^{\text{tgt}})$ with p^{src} without the scene graphs $(G^{\text{src}}, G^{\text{edit}})$; Table 2-(c) removes TSR's target reasoning, so the pipeline operates without the target caption c^{tgt}.

The two stages address complementary failure modes. TSR alone (b) drives the largest gains in *Temporal* and *Causal* reasoning, categories where unpacking the entailments of an abstract condition is the primary bottleneck. To isolate the contribution of ESGT, we compare (b) and (d), where TSR is held fixed. Adding ESGT improves Overall accuracy from 27.5% to 31.4%. In particular, with TSR alone (b), *Spatial* accuracy actually *drops* below the Qwen-Image-Edit baseline (28.0% vs. 31.0%), whereas adding ESGT raises it to 33.0%. This supports the role of ESGT in grounding the inferred target state to source objects, attributes, parts, and relations.

Conversely, ESGT without TSR (c) improves Overall accuracy over the baseline (23.1% vs. 19.4%) but remains substantially below the full pipeline, indicating that source-grounded structure alone is insufficient without reliable target-state inference. Overall, TSR and ESGT play complementary roles, with R2E-Prompt achieving its highest Overall accuracy when both stages are used together.

4.5 Editor-Agnostic Transferability

R2E-Prompt is designed as a prompt-level intervention applicable to any instruction-based editor. In Table 3, we verify this by applying the same R2E-Prompt pipeline to three additional open-source editors from the FLUX.2 family, reporting the *Overall* accuracy (per-category results and qualitative comparisons are in Appendix C). R2E-Prompt yields consistent and substantial gains across all three—FLUX.2-Klein-4B ($+11.9$ points), FLUX.2-Klein-9B-KV ($+12.8$ points), and FLUX.2-Dev (32B; $+16.1$ points)—with each base editor roughly doubling its baseline accuracy. The uniformity of these gains across backbones of different parameter scales (4B to 32B) indicates that R2E-Prompt's benefit does not hinge on any specific editor's architecture or capacity. Notably, FLUX.2-Dev with R2E-Prompt (29.7%) alone outperforms every prior open-source baseline in Table 1 and is second only to the R2E-Prompt + Qwen-Image-Edit configuration (31.4%), confirming that R2E-Prompt is plug-and-play and model-agnostic, applicable to any off-the-shelf instruction-based editor without modification.

Table 3: Editor-agnostic transferability of R2E-Prompt. We report *Overall* accuracy (%) when R2E-Prompt is applied as a training-free prompt-level intervention on three open-source base editors from the FLUX.2 family. The same pipeline is applied to each backbone without modification.

Method	Overall ↑
FLUX.2-Klein-4B (Labs, 2026)	10.6
FLUX.2-Klein-4B + R2E-Prompt (**Ours**)	**22.5**
FLUX.2-Klein-9B-KV (Labs, 2026)	12.5
FLUX.2-Klein-9B-KV + R2E-Prompt (**Ours**)	**25.3**
FLUX.2-Dev (Labs, 2026)	13.6
FLUX.2-Dev + R2E-Prompt (**Ours**)	**29.7**

Table 4: Average reasoning time comparison between R2E-Prompt (Ours) and EditThinker.

	EditThinker	R2E-Prompt
Reasoning Time (s) ↓	275.37	**17.82**

4.6 Reasoning Time Comparison

We compare the average reasoning time of R2E-Prompt and EditThinker (Li et al., 2025) on RISEBench—the closest editor-agnostic baseline, which reaches the target state through iterative editor invocations rather than a single pass. To isolate the overhead of the reasoning framework itself, we measure only the reasoning time before the image generation stage. As shown in Table 4, R2E-Prompt is approximately 15.5× faster than EditThinker while achieving substantially better editing performance (31.4% vs. 17.8% in Table 1). These results demonstrate that structured upstream reasoning can improve reasoning-intensive image editing efficiently by avoiding the iterative editor invocations that EditThinker relies on for refinement, while requiring no additional training.

4.7 User Study

We conduct a user study with 50 participants to complement the automated evaluation. Based on Table 1, we compare R2E-Prompt against Gemini-2.5-Flash-Image (Nano Banana), EditThinker, and BAGEL (w/ CoT). Participants evaluate these methods alongside R2E-Prompt on three criteria: *Instruction Following*, *Appearance Consistency*, and *Visual Plausibility*. R2E-Prompt outperforms all open-source baselines across all criteria and remains competitive with Gemini-2.5-Flash-Image in Instruction Following (45.2% vs. 49.6%) and Appearance Consistency (50.0% vs. 47.6%). The remaining gap in Visual Plausibility (30.8% vs. 41.6%) reflects the rendering limitations of the frozen downstream editor. Full protocol, interface, and results are in Appendix I.

5 Conclusion

In this work, we introduce R2E-Prompt, a training-free framework for reasoning-based image editing that bridges the gap between implicit user instructions and instruction-based image editors. R2E-Prompt shifts reasoning upstream of the editor, operating purely in language space to produce an explicit source-to-target specification for off-the-shelf editors. Our framework consists of three stages: *Target-State Reasoning* unpacks the implicit visual consequences of the instruction through chain-of-Thought reasoning; *Edit-Aware Scene Graph Transformation* parses the resulting caption pair into text-derived scene graphs and assigns each source element an edit operation via structured prediction; and *Executable Prompt Generation* renders it as a single executable editing command. Extensive experiments on RISEBench show that R2E-Prompt achieves state-of-the-art performance among open-source methods and matches or surpasses several closed-source proprietary systems, while consistently improving four different editor backbones without modifying the underlying models. These results confirm that R2E-Prompt is an effective, editor-agnostic approach to reasoning-intensive image editing.

AI USE STATEMENT

In this work, we used generative AI tools to assist with drafting and polishing the manuscript, including improvements to grammar, clarity, and organization. We have not used generative AI tools to develop research ideas, design or refine the research methodology or experiments, or interpret results. Additionally, the proposed R2E-Prompt framework itself uses pretrained generative models as components of the research pipeline, as detailed in Section 3 and Section 4. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses publicly available evaluation datasets and previously released models or model services. We do not collect, use, or release personally identifiable or sensitive personal information. Our user study collected only task-specific ratings of model outputs under predefined evaluation criteria; responses were analyzed and are reported only in aggregate, and no participant-level data were used to develop or adapt the method. R2E-Prompt is training-free and does not introduce any new training data. The study protocol and evaluation interface are described in Appendix I.

REPRODUCIBILITY STATEMENT

We provide the model variants, inference configurations, hardware setup, and other implementation details in Section 4.1 and Appendix A. The benchmark and evaluation protocol are described in Section 4.2, and the complete system prompts for every stage of R2E-Prompt are provided in Appendix E. These details specify the full inference and evaluation pipeline used to obtain the reported results.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Omri Avrahami, Dani Lischinski, and Ohad Fried. Blended diffusion for text-driven editing of natural images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Chengyu Bai, Jintao Chen, Xiang Bai, Yilong Chen, Qi She, Ming Lu, and Shanghang Zhang. Uniedit-i: Training-free image editing for unified vlm via iterative understanding, editing and verifying. *arXiv preprint arXiv:2508.03142*, 2025a.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025b.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report, 2025c. URL https://arxiv.org/abs/2502.13923.

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*, pp. 18392–18402, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Guoxuan Chen, Chufeng Xiao, Haoran Yang, Siyue Xie, Binxiao Huang, Ming Zhang, Cheuk Him Chau, Xinyu Fu, Yingzhao Lian, Tom SY Li, et al. Boogu-image-0.1: Boosting open-source unified multimodal understanding and generation. *arXiv preprint arXiv:2607.13125*, 2026.

Jaemin Cho, Yushi Hu, Jason Michael Baldridge, Roopal Garg, Peter Anderson, Ranjay Krishna, Mohit Bansal, Jordi Pont-Tuset, and Su Wang. Davidsonian scene graph: Improving reliability in fine-grained evaluation for text-to-image generation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ITq4ZRUT4a.

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

Helisa Dhamo, Azade Farshad, Iro Laina, Nassir Navab, Gregory D Hager, Federico Tombari, and Christian Rupprecht. Semantic image manipulation using scene graphs. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5213–5222, 2020.

Yixuan Ding, Wei Huang, Ruijie Quan, Xiaojuan Qi, and Yi Yang. Is this edit correct? a multi-dimensional benchmark for reasoning-aware image editing, 2026. URL https://arxiv.org/abs/2606.05172.

Tsu-Jui Fu, Wenze Hu, Xianzhi Du, William Wang, Yinfei Yang, and Zhe Gan. Guiding instruction-based image editing via multimodal large language models. In *International Conference on Learning Representations*, volume 2024, pp. 54820–54833, 2024.

Google DeepMind. Gemini 2.0 Flash. https://deepmind.google/models/gemini/flash/, December 2024. Accessed: 2024.

Google DeepMind. Gemini 2.5 Flash Image (Nano Banana). https://deepmind.google/models/gemini-image/flash/, August 2025. Accessed: 2025.

Qingdong He, Xueqin Chen, Chaoyi Wang, Yanjie Pan, Xiaobin Hu, Zhenye Gan, Yabiao Wang, Chengjie Wang, Xiangtai Li, and Jiangning Zhang. Reasoning to edit: Hypothetical instruction-based image editing with visual reasoning. In *ICML*, 2026.

Yujia Hu, Songhua Liu, Zhenxiong Tan, Xingyi Yang, and Xinchao Wang. Image editing as programs with diffusion models. *Advances in Neural Information Processing Systems*, 38:18405–18436, 2026.

Yuzhou Huang, Liangbin Xie, Xintao Wang, Ziyang Yuan, Xiaodong Cun, Yixiao Ge, Jiantao Zhou, Chao Dong, Rui Huang, Ruimao Zhang, et al. 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*, pp. 8362–8371, 2024.

Liya Ji, Chenyang Qi, and Qifeng Chen. Instruction-based image editing with planning, reasoning, and generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17506–17515, 2025.

Bahjat Kawar, Shiran Zada, Oran Lang, Omer Tov, Huiwen Chang, Tali Dekel, Inbar Mosseri, and Michal Irani. Imagic: Text-based real image editing with diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Siqi Kou, Jiachun Jin, Zetong Zhou, Ye Ma, Yugang Wang, Quan Chen, Peng Jiang, Xiao Yang, Jun Zhu, Kai Yu, et al. Think-then-generate: Reasoning-aware text-to-image diffusion with llm encoders. *arXiv preprint arXiv:2601.10332*, 2026.

Black Forest Labs. Flux.2. https://bfl.ai/models/flux-2, 2026. Accessed: March 2026.

Black Forest Labs, Stephen Batifol, Andreas Blattmann, Frederic Boesel, Saksham Consul, Cyril Diagne, Tim Dockhorn, Jack English, Zion English, Patrick Esser, Sumith Kulal, Kyle Lacey, Yam Levi, Cheng Li, Dominik Lorenz, Jonas Müller, Dustin Podell, Robin Rombach, Harry Saini, Axel Sauer, and Luke Smith. Flux.1 kontext: Flow matching for in-context image generation and editing in latent space, 2025. URL https://arxiv.org/abs/2506.15742.

Hongyu Li, Manyuan Zhang, Dian Zheng, Ziyu Guo, Yimeng Jia, Kaituo Feng, Hao Yu, Yexin Liu, Yan Feng, Peng Pei, et al. Editthinker: Unlocking iterative reasoning for any image editor. *arXiv preprint arXiv:2512.05965*, 2025.

Yuheng Li, Haotian Liu, Qingyang Wu, Fangzhou Mu, Jianwei Yang, Jianfeng Gao, Chunyuan Li, and Yong Jae Lee. Gligen: Open-set grounded text-to-image generation, 2023. URL https://arxiv.org/abs/2301.07093.

Shiyu Liu, Yucheng Han, Peng Xing, Fukun Yin, Rui Wang, Wei Cheng, Jiaqi Liao, Yingming Wang, Honghao Fu, Chunrui Han, Guopeng Li, Yuang Peng, Quan Sun, Jingwei Wu, Yan Cai, Zheng Ge, Ranchen Ming, Lei Xia, Xianfang Zeng, Yibo Zhu, Binxing Jiao, Xiangyu Zhang, Gang Yu, and Daxin Jiang. Step1X-Edit: A practical framework for general image editing. *arXiv preprint arXiv:2504.17761*, 2025.

Mistral AI. Mistral small 3.1. https://mistral.ai/news/mistral-small-3-1, March 2025.

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

Luozheng Qin, Jia Gong, Yuqing Sun, Tianjiao Li, Mengping Yang, Xiaomeng Yang, Chao Qu, Zhiyu Tan, and Hao Li. Uni-cot: Towards unified chain-of-thought reasoning across text and vision. *arXiv preprint arXiv:2508.05606*, 2025.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PmLR, 2021.

Team Seedream, Yunpeng Chen, Yu Gao, Lixue Gong, Meng Guo, Qiushan Guo, Zhiyao Guo, Xiaoxia Hou, Weilin Huang, Yixuan Huang, et al. Seedream 4.0: Toward next-generation multimodal image generation. *arXiv preprint arXiv:2509.20427*, 2025.

Lin Song, Wenbo Li, Guoqing Ma, Wei Tang, Bo Wang, Yuan Zhang, Yijun Yang, Yicheng Xiao, Jianhui Liu, Yanbing Zhang, et al. Joyai-image: Awaking spatial intelligence in unified multimodal understanding and generation. *arXiv preprint arXiv:2605.04128*, 2026.

Super Intelligence Team, Changhao Qiao, Chao Hui, Chen Li, Cunzheng Wang, Dejia Song, Jiale Zhang, Jing Li, Qiang Xiang, Runqi Wang, et al. Firered-image-edit-1.0 technical report. *arXiv preprint arXiv:2602.13344*, 2026.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, Yuxiang Chen, Zecheng Tang, Zekai Zhang, Zhengyi Wang, An Yang, Bowen Yu, Chen Cheng, Dayiheng Liu, Deqing Li, Hang Zhang, Hao Meng, Hu Wei, Jingyuan Ni, Kai Chen, Kuan Cao, Liang Peng, Lin Qu, Minggang Wu, Peng Wang, Shuting Yu, Tingkun Wen, Wensen Feng, Xiaoxiao Xu, Yi Wang, Yichang Zhang, Yongqiang Zhu, Yujia Wu, Yuxuan Cai, and Zenan Liu. Qwen-image technical report, 2025. URL https://arxiv.org/abs/2508.02324.

Shiyu Wu, Mingzhen Sun, Weining Wang, Yequan Wang, and Jing Liu. Visualprompter: Semantic-aware prompt optimization with visual feedback for text-to-image synthesis. In *International Conference on Learning Representations (ICLR)*, 2026.

Yingjing Xu, Jie Kong, Jiazhi Wang, Xiao Pan, Bo Lin, and Qiang Liu. Insightedit: Towards better instruction following for image editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2694–2703, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Binxin Yang, Shuyang Gu, Bo Zhang, Ting Zhang, Xuejin Chen, Xiaoyan Sun, Dong Chen, and Fang Wen. Paint by example: Exemplar-based image editing with diffusion models. *arXiv preprint arXiv:2211.13227*, 2022.

Yang Ye, Xianyi He, Zongjian Li, Shenghai Yuan, Zhiyuan Yan, Bohan Hou, Li Yuan, et al. Imgedit: A unified image editing dataset and benchmark. *Advances in Neural Information Processing Systems*, 38, 2026.

Fukun Yin, Shiyu Liu, Yucheng Han, Zhibo Wang, Peng Xing, Rui Wang, Wei Cheng, Yingming Wang, Aojie Li, Zixin Yin, et al. Reasonedit: Towards reasoning-enhanced image editing models. *arXiv preprint arXiv:2511.22625*, 2025.

Ziyun Zeng, Hang Hua, and Jiebo Luo. Mira: Multimodal iterative reasoning agent for image editing. *arXiv preprint arXiv:2511.21087*, 2025.

Dong Zhang, Lingfeng He, Rui Yan, Fei Shen, and Jinhui Tang. R-genie: Reasoning-guided generative image editing. *arXiv preprint arXiv:2505.17768*, 2025.

Kai Zhang, Lingbo Mo, Wenhu Chen, Huan Sun, and Yu Su. Magicbrush: A manually annotated dataset for instruction-guided image editing. *Advances in Neural Information Processing Systems*, 36:31428–31449, 2023a.

Zhongping Zhang, Huiwen He, Bryan A Plummer, Zhenyu Liao, and Huayan Wang. Complex scene image editing by scene graph comprehension. *arXiv preprint arXiv:2203.12849*, 2022.

Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. Multimodal chain-of-thought reasoning in language models. *arXiv preprint arXiv:2302.00923*, 2023b.

Xiangyu Zhao, Peiyuan Zhang, Kexian Tang, Xiaorong Zhu, Hao Li, Wenhao Chai, Zicheng Zhang, Renqiu Xia, Guangtao Zhai, Junchi Yan, et al. Envisioning beyond the pixels: Benchmarking reasoning-informed visual editing. *Advances in Neural Information Processing Systems*, 38, 2026.

R2E-PROMPT: REASONING-TO-EXECUTABLE PROMPT GENERATION FOR IMPLICIT-INSTRUCTION IMAGE EDITING

-APPENDIX-

A IMPLEMENTATION DETAILS

We provide implementation details not included in the main paper. All models in the R2E-Prompt pipeline are pretrained and kept frozen; no parameters are updated at any stage.

We evaluate R2E-Prompt on top of seven frozen off-the-shelf instruction-based image editors: the four editors used in the main paper and three additional recent editors evaluated in Appendix C. For fairness and reproducibility, all editors are run with publicly released implementations and default hyperparameters.

- **Qwen-Image-Edit-2511** (Wu et al., 2025): official release at `https://huggingface.co/Qwen/Qwen-Image-Edit-2511`. This is the main editor used in Table 1; we run it with 40 denoising steps, true CFG scale 4.0, and distilled guidance scale 1.0.
- **FLUX.2-Klein-9B-KV** (Labs, 2026): official release at `https://huggingface.co/black-forest-labs/FLUX.2-klein-9b-kv`. We run it with 4 denoising steps and guidance scale 1.0.
- **FLUX.2-Klein-4B** (Labs, 2026): official release at `https://huggingface.co/black-forest-labs/FLUX.2-klein-4B`. We run it with 4 denoising steps and guidance scale 1.0.
- **FLUX.2-Dev** (Labs, 2026): official release at `https://huggingface.co/black-forest-labs/FLUX.2-dev`. We run it with 50 denoising steps and guidance scale 4.0.

Additional editors (Appendix C).

- **Boogu-Image-Edit-Turbo** (Chen et al., 2026): official release at `https://huggingface.co/Boogu/Boogu-Image-0.1-Edit-Turbo`. We run it with 4 denoising steps and guidance scale 1.0.
- **JoyAI-Image-Edit-Plus** (Song et al., 2026): official release at `https://huggingface.co/jdopensource/JoyAI-Image-Edit-Plus-Diffusers`. We run it with 30 denoising steps and guidance scale 4.0.
- **FireRed-Image-Edit-1.1** (Team et al., 2026): official release at `https://huggingface.co/FireRedTeam/FireRed-Image-Edit-1.1`. We run it with 40 denoising steps and true CFG scale 4.0.

B EDIT OPERATION TAXONOMY

We provide the full specification of the fourteen edit operations referenced in Section 3.2. Table 5 lists each operation with its definition and an illustrative example, organized into five groups according to the type of scene-graph node that each operation targets.

Node-type partition. Because every source-graph node is assigned an operation label specifying how it should be edited (or preserved) to realize the target state, we partition the label space by scene-graph node type: for any given node, the admissible operations are restricted to those meaningful for its type. This reduces the effective per-node label space and provides inductive bias for the LLM-based label predictor. Three *universal* operations (ADD, REMOVE, KEEP) apply uniformly across node types—any node, whether it represents an entity, attribute, action, or spatial relation, may be introduced, removed, or preserved. The remaining eleven operations are type-specific:

- *Entity-targeting*
  (REPLACE, RESIZE, MOVE): edits acting on whole entities or their identity, scale, or spatial position.

Table 5: **Edit operation taxonomy.** The fourteen operations used by Edit-Aware Scene Graph Transformation, organized by the scene-graph node type they target. Three *universal* operations (ADD, REMOVE, KEEP) apply to any node type, while the remaining operations are type-specific. KEEP is included as a dedicated label so that preserved facts are stated explicitly in the edit graph rather than left implicit as the complement of edited nodes.

Operation	Definition
Universal (applies to any node type)	
ADD	Introduce a new entity, attribute, action, relation, or semantic element (e.g., add a car on the road).
REMOVE	Eliminate an existing element—entity, attribute, action, or relation (e.g., remove the sofa).
KEEP	Preserve a source-grounded fact as unchanged from the original state.
Entity-targeting (acts on `entity-whole`/`entity-part` nodes)	
REPLACE	Substitute one entity with a different entity (e.g., replace the coffee with an apple).
RESIZE	Adjust the scale or size of an entity (e.g., enlarge the clock).
MOVE	Change the spatial position of an entity (e.g., move the plane to the left).
Attribute-targeting (acts on `attribute-`⟨`category`⟩ nodes)	
COLOR CHANGE	Modify an entity's color attribute (e.g., change the shoes to blue).
MATERIAL CHANGE	Alter an entity's surface material or texture (e.g., change the sign to stone).
APPEARANCE CHANGE	Modify visual attributes such as patterns or accessories (e.g., make the cup have a floral pattern).
EXPRESSION CHANGE	Adjust an entity's facial expression (e.g., change the expression to smiling).
Action-targeting (acts on `action` nodes)	
ACTION CHANGE	Modify the pose or action of an instance (e.g., raise the boy's hands).
Scene-level (acts globally on the image)	
BACKGROUND CHANGE	Change the background scene to another (e.g., a forest background).
TONE TRANSFER	Change global atmosphere or lighting conditions (e.g., foggy weather, seasonal change).
STYLE CHANGE	Modify the overall visual style (e.g., cartoon).

- *Attribute-targeting*
  (COLOR/MATERIAL/APPEARANCE/EXPRESSION CHANGE): edits to entity properties along the four most common visual attribute axes invoked by editing instructions.
- *Action-targeting*
  (ACTION CHANGE): edits to pose or action.
- *Scene-level*
  (BACKGROUND/TONE/STYLE CHANGE):
  global edits applied to the image as a whole rather than to individual nodes.

Explicit preservation label. In addition to the thirteen edit operations, we include KEEP as a dedicated label for source-grounded facts that remain unchanged. Preservation can in principle be inferred as the complement of edited nodes, but attaching it as an explicit label is a convenient fit for our prompt-based setting. It lets the downstream Executable Prompt Generation stage separate source elements that are *explicitly* preserved from those that are simply *not mentioned* in the target caption—a distinction that off-the-shelf editors tend to be sensitive to, since regions not anchored by the prompt are more prone to drift during editing. The label also carries through the pipeline, so KEEP-labeled nodes are collected into the final command's preservation clause, providing a direct textual anchor for source-state retention.

Table 6: **Editor-agnostic transferability of R2E-Prompt on RISEBench (Zhao et al., 2026).** We report per-category accuracy (%) over the four reasoning types (Temporal, Causal, Spatial, Logical) and the Overall accuracy across seven image editor backbones. The same R2E-Prompt pipeline is applied to each backbone without modification. The better result within each backbone pair is highlighted in **bold**.

Method	Temporal ↑	Causal ↑	Spatial ↑	Logical ↑	Overall ↑
FLUX.2-Klein-4B (Labs, 2026)	9.4	10.0	15.0	**7.1**	10.6
FLUX.2-Klein-4B + R2E-Prompt (**Ours**)	**25.9**	**34.4**	**22.0**	**7.1**	**22.5**
FLUX.2-Klein-9B-KV (Labs, 2026)	7.1	12.2	23.0	5.9	12.5
FLUX.2-Klein-9B-KV + R2E-Prompt (**Ours**)	**27.1**	**42.2**	**24.0**	**7.1**	**25.3**
FLUX.2-Dev (Labs, 2026)	9.4	25.6	17.0	1.2	13.6
FLUX.2-Dev + R2E-Prompt (**Ours**)	**37.6**	**42.2**	**31.0**	**7.1**	**29.7**
Qwen-Image-Edit (Wu et al., 2025)	21.2	18.9	31.0	4.7	19.4
Qwen-Image-Edit + R2E-Prompt (**Ours**)	**35.3**	**44.4**	**33.0**	**11.8**	**31.4**
Boogu-Image-Edit-Turbo (Chen et al., 2026)	5.9	12.2	16.0	4.7	10.0
Boogu-Image-Edit-Turbo + R2E-Prompt (**Ours**)	**23.5**	**22.2**	**23.0**	**10.6**	**20.0**
JoyAI-Image-Edit-Plus (Song et al., 2026)	8.2	17.8	21.0	3.5	13.1
JoyAI-Image-Edit-Plus + R2E-Prompt (**Ours**)	**32.9**	**31.1**	**22.0**	**9.4**	**23.9**
FireRed-Image-Edit (Team et al., 2026)	14.1	24.4	23.0	1.2	16.1
FireRed-Image-Edit + R2E-Prompt (**Ours**)	**42.4**	**36.7**	**25.0**	**5.9**	**27.5**

Relation to IEAP taxonomy. The thirteen non-KEEP operations are adopted from IEAP (Hu et al., 2026), which defines them as atomic edit categories for a program-style editing framework: each category is realized by a dedicated LoRA adapter trained on a shared Diffusion Transformer (DiT) backbone, and a VLM-based agent programs sequences of these atomic operations to execute complex instructions. Our use of this vocabulary differs along three axes:

- *Element-level vs. whole-image granularity.* In IEAP, each atomic operation is applied to the whole image (or a region-of-interest) within a sequential step of the program. We reinterpret the same vocabulary as per-node labels assigned in parallel to typed scene-graph elements, so that a single instruction can simultaneously address multiple element-level edits via a labeled graph rather than be decomposed into a sequence of whole-image applications.
- *Explicit preservation label.* IEAP has no preservation primitive: unmentioned content is left implicit, and the framework relies on the editor's region-of-interest mechanism to localize changes. We add KEEP as a fourteenth label so that source-grounded facts that remain unchanged are stated explicitly in the edit graph, which the downstream renderer can then carry into the editing command as an explicit preservation clause.
- *Training-free, single-pass framework.* IEAP trains a dedicated LoRA adapter per category on a custom DiT backbone and executes edits through sequential VLM-programmed inference of these adapters. R2E-Prompt is training-free and single-pass: the per-node label assignment is produced once by a frozen LLM, and the resulting edit graph is linearized into a single command that any off-the-shelf instruction-based editor can consume without modification or fine-tuning.

Together, these distinctions repurpose IEAP's atomic editing vocabulary into a structured-prediction label space over typed scene-graph nodes—a formulation that the original whole-image, sequence-of-operations taxonomy does not directly support.

C GENERALIZATION ABILITY OF R2E-PROMPT

To further validate the editor-agnostic transferability of R2E-Prompt, we extend our evaluation to a broader set of image editors. Table 6 reports per-category and Overall accuracy for the three FLUX.2 variants previously summarized by Overall accuracy in Table 3, Qwen-Image-Edit evalu-

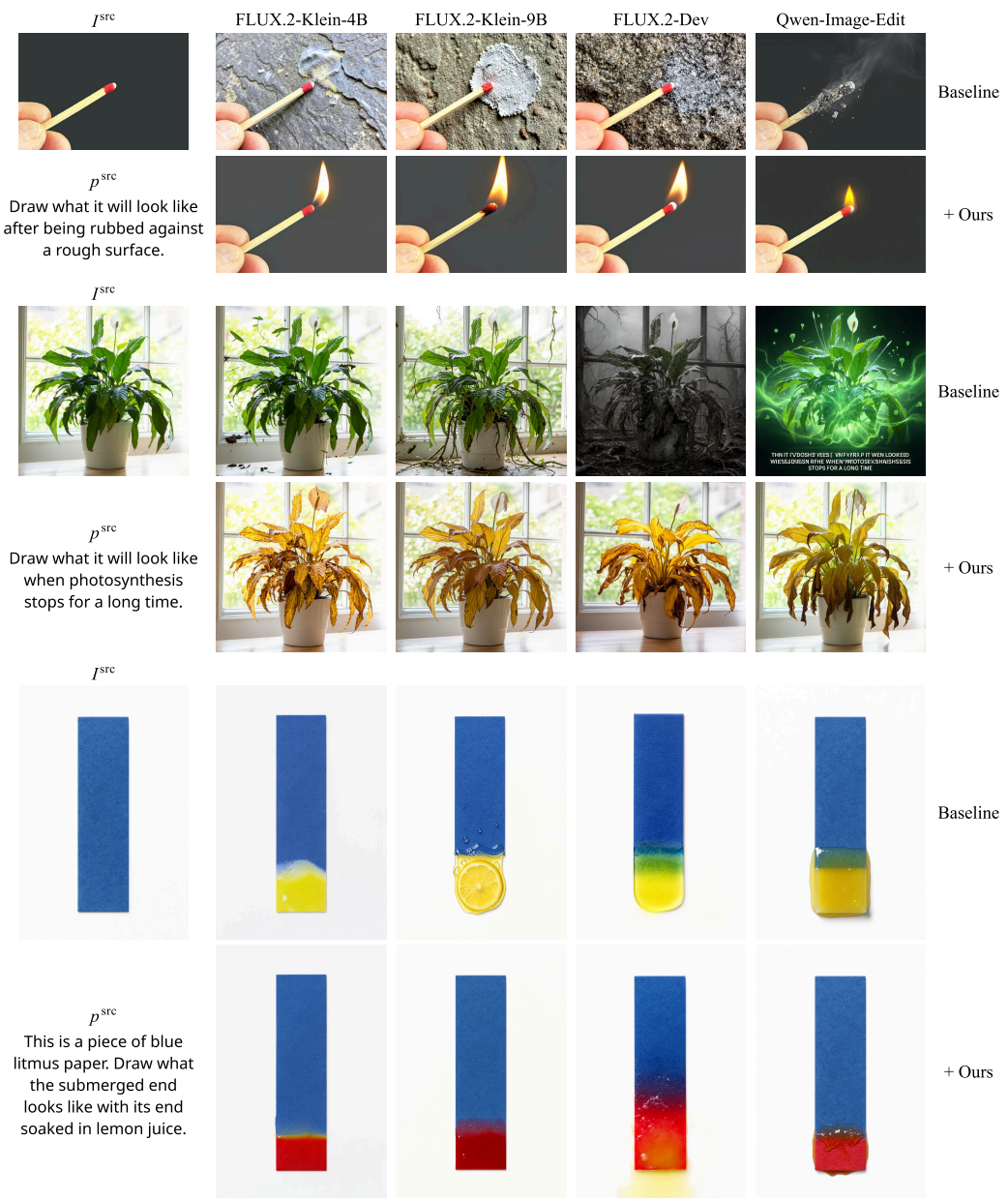


Figure 4: Qualitative comparison of applying R2E-Prompt to four different editing backbones (FLUX.2-Klein-4B, FLUX.2-Klein-9B-KV, FLUX.2-Dev, Qwen-Image-Edit) on three implicit-reasoning instructions from RISEBench. For each instruction, the top row shows the backbone's baseline output and the bottom row shows the output when R2E-Prompt is applied. Across backbones of different capacities and families, R2E-Prompt consistently converts implicit instructions into correct visual edits.

ated in Table 1, and three additional recent editors: FireRed-Image-Edit (Team et al., 2026), JoyAI-Image-Edit-Plus (Song et al., 2026), and Boogu-Image-Edit-Turbo (Chen et al., 2026). We apply the same R2E-Prompt pipeline and configuration to every editor without editor-specific adaptation. Figure 4 and Figure 5 further visualizes its transfer to multiple off-the-shelf editors, highlighting the consistent improvements across different editing backbones.

Quantitative analysis. The per-category breakdown reveals a consistent pattern across all evaluated image editors: R2E-Prompt's largest gains fall on *Causal* and *Temporal* reasoning—the

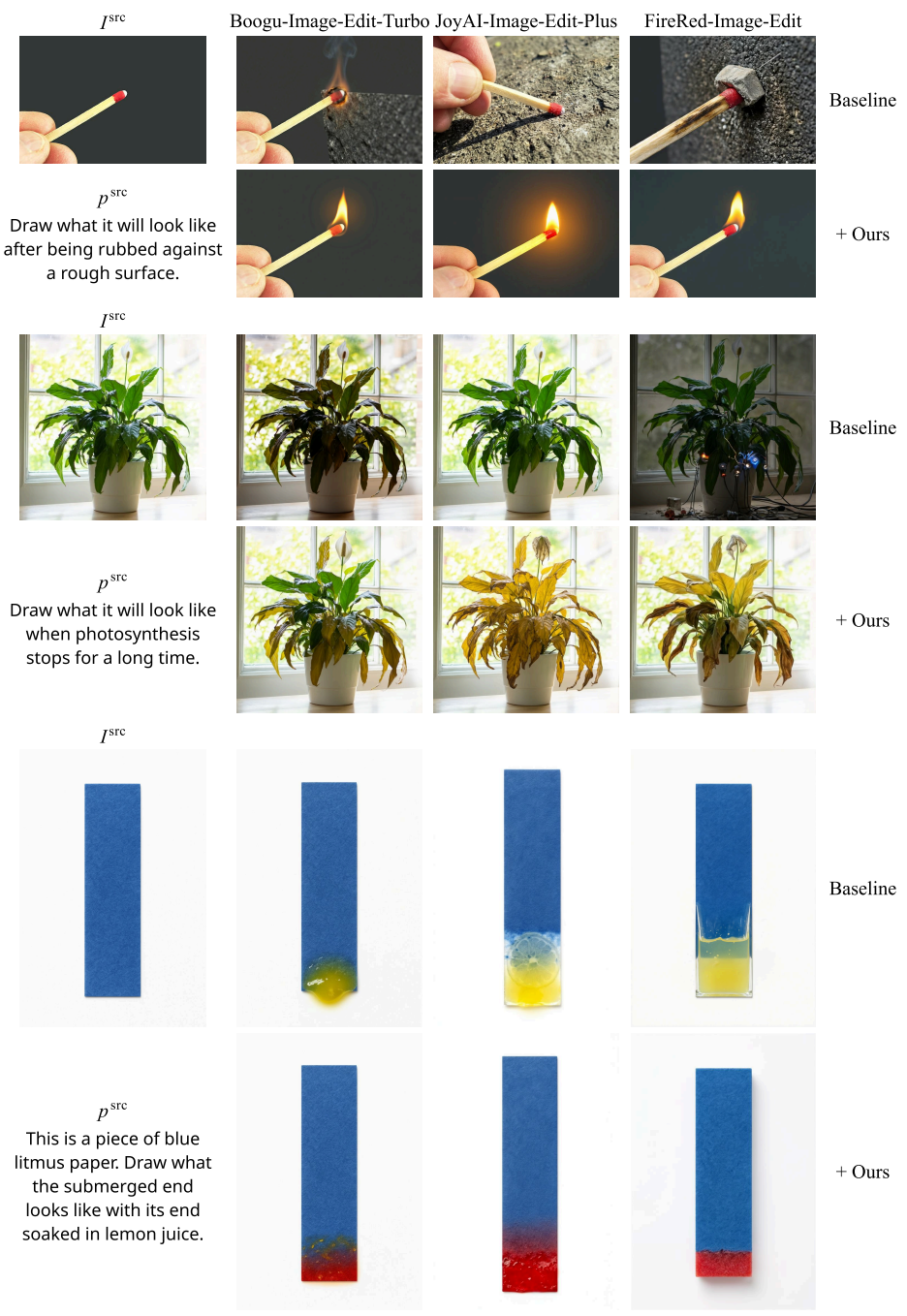


Figure 5: Qualitative comparison of applying R2E-Prompt to three different editing backbones (Boogu-Image-Edit-Turbo, JoyAI-Image-Edit-Plus, FireRed-Image-Edit) on three implicit-reasoning instructions from RISEBench. For each instruction, the top row shows the backbone's baseline output and the bottom row shows the output when R2E-Prompt is applied. Across backbones of different capacities and families, R2E-Prompt consistently converts implicit instructions into correct visual edits.

categories that most require unpacking what an abstract condition entails. On *Causal* reasoning alone, R2E-Prompt lifts FLUX.2-Klein-4B by +24.4 points (10.0% $\rightarrow$ 34.4%), FLUX.2-Klein-9B-KV by +30.0 points (12.2% $\rightarrow$ 42.2%), FLUX.2-Dev by +16.6 points (25.6% $\rightarrow$ 42.2%), and Qwen-Image-Edit by +25.5 points (18.9% $\rightarrow$ 44.4%); *Temporal* reasoning shows a similar pattern

($+16.5/+20.0/+28.2/+14.1$ points across the four backbones). *Spatial* and *Logical* categories—where success depends more on the editor's intrinsic capacity for spatial manipulation or discrete-object reasoning than on instruction interpretation—show smaller gains, with no degradation on any backbone. The three additional recent editors exhibit the same trend: R2E-Prompt consistently improves all three models, with particularly large gains on *Temporal* ($+28.3/+24.7/+17.6$ points) and *Causal* ($+12.3/+13.3/+10.0$ points) reasoning for FireRed-Image-Edit, JoyAI-Image-Edit-Plus, and Boogu-Image-Edit-Turbo, respectively. The pattern echoes the main paper's analysis: explicit target-state reasoning provides its strongest leverage on the categories where the user's instruction is most implicit, and this leverage transfers consistently across editor backbones regardless of capacity (4B to 32B within the FLUX.2 family) or model family (FLUX.2, Qwen-Image-Edit, FireRed-Image-Edit, JoyAI-Image-Edit-Plus, and Boogu-Image-Edit-Turbo).

Qualitative analysis. Figure 4 and Figure 5 illustrates the same R2E-Prompt pipeline applied to three implicit-reasoning instructions, each entailing a visual consequence that is not directly named.

"After being rubbed against a rough surface" (matchstick). The entailed consequence is the match igniting via friction. Without R2E-Prompt, the FLUX.2 backbones, JoyAI-Image-Edit-Plus, and FireRed-Image-Edit interpret "rough surface" literally and render damage to the surrounding scene, introducing cracks, debris, or rock fragments next to an unchanged match. Qwen-Image-Edit destroys the matchhead with smoke and ash, while Boogu-Image-Edit-Turbo produces a partially charred or smoldering matchhead without clearly rendering the intended ignition outcome. With R2E-Prompt applied, every backbone, regardless of family or capacity, lights the match with a flame while leaving the rest of the scene intact.

"When photosynthesis stops for a long time" (plant). The entailed consequence is the plant wilting and yellowing. Without R2E-Prompt, the failure modes diverge by backbone: FLUX.2-Klein-4B leaves the plant essentially unchanged, FLUX.2-Klein-9B-KV produces incongruous, dead-looking vines without yellowing the leaves, FLUX.2-Dev applies a global black-and-white transformation, and Qwen-Image-Edit hallucinates visual artifacts overlaid with garbled text. The additional editors similarly fail to express the intended long-term physiological consequence, largely retaining green foliage or introducing unrelated appearance changes. With R2E-Prompt applied, every backbone converges on the correct outcome: realistically yellowed, drooping foliage with the rest of the scene preserved.

"Blue litmus paper soaked in lemon juice" (chemistry). The entailed consequence is the submerged end turning red, following a basic acid–base reaction (blue litmus + acid $\rightarrow$ red). Without R2E-Prompt, every backbone produces a yellow stain instead—either treating "lemon juice" as a yellow color or rendering a literal lemon slice. With R2E-Prompt applied, every backbone correctly turns the submerged end red while preserving the rest of the paper as blue.

These cases highlight two complementary properties of R2E-Prompt's transferability. First, R2E-Prompt unpacks the implicit consequence into an explicit visual specification that is recoverable by editors of widely different architectures and capacities. Second, the convergence of edits across seven different backbones—each of which fails differently in its baseline—indicates that the reasoning capability provided by R2E-Prompt does not depend on any specific editor's internal architecture or pretrained capability; rather, R2E-Prompt supplies an external reasoning layer that any instruction-based editor can consume.

D Additional Evaluation on RE-Edit

Following the official RE-Edit protocol (Ding et al., 2026), we additionally evaluate the reasoning-aware image-editing capability of R2E-Prompt on 1,000 samples across five dimensions: physical, environmental, cultural, causal, and referential reasoning. For the five reasoning categories, we report the per-category pass rate (%). In addition, we report two general editing metrics on a 0–10 scale: Instruction Following (IF), which measures how well the output satisfies the explicit editing instruction, and Semantic Consistency (SC), which measures semantic content preservation. We compare against the three highest-performing publicly available models in Table 1 of the main paper—EditThinker (Li et al., 2025), BAGEL (w/CoT) (Deng et al., 2025), and FLUX.2-dev (Labs, 2026)—representing both reasoning-enhanced editors and a strong direct editor. R2E-Prompt outperforms these baselines across the reported reasoning dimensions, while also improving over its

Table 7: **Quantitative comparison on the RE-Edit benchmark.** We report performance across five reasoning dimensions (*Physical*, *Environmental*, *Cultural*, *Causal*, and *Referential*), together with the non-reasoning metrics IF and SC. Models are grouped by access into closed-source and open-source models. **Bold** and underline denote the best and second-best results within each column.

Method	Reasoning					Non-Reasoning	
	Physical ↑	Environmental ↑	Cultural ↑	Causal ↑	Referential ↑	IF ↑	SC ↑
Closed-source models							
Gemini-2.5-Flash-Image (Nano Banana) (Google DeepMind, 2025)	20.10	10.80	3.90	15.50	48.00	7.60	3.60
Seedream-4.0 (Seedream et al., 2025)	18.70	16.30	6.10	14.50	53.40	8.00	3.70
Open-source models							
EditThinker (Li et al., 2025)	7.01	9.36	0.00	3.00	17.16	4.56	2.14
BAGEL (w/ CoT) (Deng et al., 2025)	11.21	9.36	1.11	6.00	40.20	6.96	3.40
FLUX.2-Dev (Labs, 2026)	20.10	14.80	7.80	15.00	50.50	7.70	4.00
Qwen-Image-Edit (Wu et al., 2025)	21.00	12.80	3.30	13.00	50.00	7.00	3.80
Qwen-Image-Edit + R2E-Prompt (**Ours**)	**21.03**	**38.92**	**8.89**	**20.00**	**58.33**	**8.19**	**5.11**

```
Task: I will provide an input image. Generate an accurate caption for the
image.
The caption must describe only what is clearly visible in the image.
Do not include guesses, hidden details, explanations, or extra commentary.

Output the caption prefixed with the label ORIGINAL_CAPTION:

Output format:
ORIGINAL_CAPTION: <caption>

---

### Example 1
[input]
IMAGE

[output]
ORIGINAL_CAPTION: A woman standing on a rainy street while holding a
closed umbrella.

---

### Example 2
[input]
IMAGE

[output]
ORIGINAL_CAPTION: A standard traffic light at an intersection displaying
a green signal.
```

Figure 6: Source caption system prompt.

base model, Qwen-Image-Edit, and surpassing some proprietary models (Nano Banana and Seedream 4.0). The consistent improvements on RE-Edit indicate that the effectiveness of R2E-Prompt extends beyond RISEBench.

E System Prompt Templates

We provide the system prompt templates used by each LLM/VLM call in the R2E-Prompt pipeline. The five prompts correspond to the five inference calls in our framework: source captioning and target captioning in Target-State Reasoning (Figure 6, Figure 7); source scene graph extraction and edit scene graph transformation in Edit-Aware Scene Graph Transformation (Figure 8, Figure 9); and prompt rendering in Executable Prompt Generation (Figure 10). Each prompt is held constant across all instances and across all editor backbones evaluated in the main paper.

Source caption. The source caption prompt (Figure 6) instructs the VLM to describe what is clearly visible in the source image, without speculation or unsolicited commentary. The output is constrained to a single line tagged `ORIGINAL_CAPTION:`, with two in-context examples. This restriction is intentional: the source caption serves as the textual anchor that grounds all downstream reasoning, so we deliberately avoid inviting the VLM to introduce hidden details or interpretive content beyond visible facts.

Target caption. The target caption prompt (Figure 7) drives the chain-of-thought reasoning at the heart of Target-State Reasoning. Given the `ORIGINAL_CAPTION` and the user's `INSTRUCTION`, the LLM is asked to model the edit as an explicit state transition `ORIGINAL_STATE` $\rightarrow$ `RESULTING_STATE` and emit a structured four-field reasoning block: (i) a *Reasoning Trace* explaining how the instruction transforms the original scene; (ii) a *Resolved Intent* summarizing the intended edit beyond the literal instruction; (iii) a *Visual Changes* list enumerating the objects, at-

```
Task:
You are an implicit reasoning engine for instruction-based image editing.
I will provide several examples that include inputs and outputs.
Given an ORIGINAL_CAPTION and an INSTRUCTION, infer the final visual state of the image after applying the instruction to the
original scene.
The instruction may not always be clear, so reasoning may be required to interpret it correctly.
Model the edit as a state transition:
ORIGINAL STATE → (apply instruction) → RESULTING STATE
Focus on the resulting visual state while preserving elements that are not affected by the instruction.

---

### Reasoning and Generation Procedure

1. Write concise structured reasoning that explains how the instruction and original scene lead to the final visual state.
2. Use this reasoning to identify the intended edit, the required visual changes, and the preserved elements.
3. Then, generate the OUTPUT_CAPTION based on the resolved final visual state.

---

### STRUCTURED_REASONING Format

Use the following structure:
- Reasoning Trace:
- Resolved Intent:
- Visual Changes:
- Preserved Elements:

Keep the reasoning concise and focused on deriving the resulting visual state.
Do not include unnecessary details, speculative assumptions, or narrative explanations.
Do not generate a long step-by-step chain-of-thought; provide only concise reasoning summaries.

---

### Field Definitions

- Reasoning Trace: Provide a brief reasoning trace that explains how the instruction and original scene lead to the inferred
final visual state.
- Resolved Intent: Summarize the intended edit after reasoning. Do not merely repeat the instruction.
- Visual Changes: List the objects, attributes, states, actions, or relations that must change from the original state.
- Preserved Elements: List the source elements that should remain unchanged because they are not affected by the instruction.

---

### Caption Requirements

The OUTPUT_CAPTION must:
- Clearly describe the final visual state.
- Include key object states, such as color, shape, condition, action, or position.
- Make relationships between objects explicit when relevant.
- Describe only visual content that should be present in the inferred resulting image.

---

### Output Format

STRUCTURED_REASONING
- Reasoning Trace:
- Resolved Intent:
- Visual Changes:
- Preserved Elements:
OUTPUT_CAPTION: <final visual description of the resulting image>

---

### Output Requirements

- Output both STRUCTURED_REASONING and OUTPUT_CAPTION.
- Do not repeat the input.
- Be concrete, specific, and visually descriptive.
- Avoid vague or generic expressions.
- Make implicit effects explicit.
- Do not describe intermediate editing operations.
- Do not include extra commentary outside the required format.

---

### Example 1
[input]
ORIGINAL_CAPTION: A woman standing on a rainy street while holding a closed umbrella.
INSTRUCTION: Make her use the umbrella.

[output]
STRUCTURED_REASONING
- Reasoning Trace: Using an umbrella on a rainy street implies opening it and holding it above the person for rain protection.
- Resolved Intent: The woman should be using the umbrella as rain protection.
- Visual Changes: The umbrella changes from closed to open and is positioned above the woman.
- Preserved Elements: The woman, the rainy street, and the woman holding the umbrella remain unchanged.
OUTPUT_CAPTION: A woman standing on a rainy street while holding an open umbrella above herself.

---

### Example 2
[input]
ORIGINAL_CAPTION: A standard traffic light at an intersection displaying a green signal.
INSTRUCTION: Change the traffic light to indicate that vehicles must stop.

[output]
STRUCTURED_REASONING
- Reasoning Trace: The instruction is to change the traffic light to indicate a stop signal, which means changing its state
from green to red.
- Resolved Intent: The traffic light should display a red signal.
- Visual Changes: The traffic light signal changes from green to red.
- Preserved Elements: The standard traffic light and its position at the intersection remain unchanged.
OUTPUT_CAPTION: A standard traffic light at an intersection displaying a red signal.
```

Figure 7: Target caption system prompt.

tributes, states, actions, or relations that must change; and (iv) a *Preserved Elements* list recording the source elements unaffected by the instruction. From this structured decomposition, the LLM emits the final `OUTPUT_CAPTION` describing the inferred target state. The four-field decomposition makes the reasoning explicit and inspectable, and the *Preserved Elements* field foreshadows the explicit preservation later carried by the KEEP label.

Source scene graph. The source scene graph prompt (Figure 8) instructs the LLM to parse `ORIGINAL_CAPTION` into a typed scene graph G^{src}, expressed as two parallel blocks: `ORIGINAL_TUPLES` (the typed nodes V^{src}) and `ORIGINAL_DEPENDENCIES` (the directed edges D^{src} anchoring each node to its supporting node). The prompt enforces the six node categories used in our scene graph—`entity-whole`, `entity-part`, `attribute-`⟨`category`⟩, `action`, `spatiality`, and `other`—and constrains tuple extraction to facts explicitly stated in the caption, prohibiting guesses or hidden details. A tuple budget of up to 30 is imposed to keep the graph tractable for downstream relabeling.

Edit scene graph. The edit scene graph prompt (Figure 9) is the core of Edit-Aware Scene Graph Transformation. Conditioned jointly on $(G^{\text{src}}, c^{\text{src}}, c^{\text{tgt}})$—provided as `ORIGINAL_TUPLES`, `ORIGINAL_DEPENDENCIES`, `ORIGINAL_CAPTION`, and `OUTPUT_CAPTION`—the LLM pro-

```
Task:
I will provide several examples that include inputs and outputs.
Then, I will provide an ORIGINAL_CAPTION. Extract structured tuples and their dependencies from the
caption.

Extract only facts explicitly stated in the caption.
Do not include guesses, hidden details, or extra commentary.

---

### Tuple Requirements

Use semantic units rather than trivial fragments.
A semantic unit is a meaningful object, group, part, state, relation, displayed text, equation, row, panel,
or event that can be treated as one fact.

Extract the main entities, distinctive attributes, important states, meaningful relationships, actions,
displayed text, symbols, equations, labels, and other explicitly stated semantic units from the caption.

Tuple budget:
- Generate only as many tuples as needed to cover all meaningful information.
- Do not exceed 30 tuples.

Entity tuples:
- Use "entity - whole" for standalone objects or central semantic units.
- Use "entity - part" for parts of another entity. Include the parent entity name in the part name.

Attribute tuples:
- Use "attribute - <category>" for distinctive visible attributes or stated properties, such as color,
shape, size, material, state, type, text content, symbolic state, etc.
- Descriptive appearance phrases that contain verbs should also be treated as attributes when they
describe how an entity looks.
- Descriptions of location is not considered in this task.
- Attach each attribute to the exact entity or part it describes, not to a nearby parent entity.
- Format: attribute - <category> (<entity>, <value>)

Action tuples:
- Use "action -" for actions between entities.
- Format: action - (<subject>, <object>, <verb>)

Spatiality tuples:
- Use "Spatiality -" for meaningful spatial relationships.
- Format: Spatiality - (<entity1>, <entity2>, <preposition>)

Other tuples:
- Use "other" for style, count, text content, equation content, or global descriptors.

Do not generate duplicate tuples.
Renumber tuples consecutively starting from 1.

---

### Dependency Requirements

For each tuple, identify which tuple IDs it depends on.
- Attribute, action, and spatiality tuples depend on the entity tuples they reference.
- Entity - part tuples depend on their parent entity tuple.
- Entity - whole tuples with no parent dependency use 0.
- Multiple dependencies are separated by commas.
- The numbering of dependency lines must correspond one-to-one with the numbering of the extracted tuples.
- Every tuple ID must have exactly one dependency line, and no tuple ID should be skipped or added.

The number of dependency lines must equal the number of tuples.
```

```
---

### Output Format

ORIGINAL_TUPLES
id | tuple
ORIGINAL_DEPENDENCIES
id | dependencies

---

### Example 1
[input]
ORIGINAL_CAPTION: A woman standing on a rainy street while holding a closed
umbrella.

[output]
ORIGINAL_TUPLES
1 | entity - whole (woman)
2 | entity - whole (umbrella)
3 | entity - whole (street)
4 | attribute - state (woman, standing)
5 | attribute - state (umbrella, closed)
6 | attribute - weather (street, rainy)
7 | action - (woman, umbrella, hold)
8 | Spatiality - (woman, street, on)
ORIGINAL_DEPENDENCIES
1 | 0
2 | 0
3 | 0
4 | 1
5 | 2
6 | 3
7 | 1,2
8 | 1,3

---

### Example 2
[input]
ORIGINAL_CAPTION: A standard traffic light at an intersection displaying a
green signal.

[output]
ORIGINAL_TUPLES
1 | entity - whole (traffic light)
2 | entity - whole (intersection)
3 | entity - part (traffic light signal)
4 | attribute - type (traffic light, standard)
5 | Spatiality - (traffic light, intersection, at)
6 | action - (traffic light, traffic light signal, display)
7 | attribute - color (traffic light signal, green)
ORIGINAL_DEPENDENCIES
1 | 0
2 | 0
3 | 1
4 | 1
5 | 1,2
6 | 1,3
7 | 3
```

Figure 8: Source scene graph system prompt.

duces EDITED_TUPLES and EDITED_DEPENDENCIES in which every tuple carries an explicit <EDIT_LABEL> drawn from the operation taxonomy (Appendix B). The prompt enumerates the editing procedure as an eight-step decision sequence—comparing source and target captions, deciding relation changes, resolving the final entity inventory, and recomputing dependencies—so that label assignment is principled rather than ad hoc. Crucially, the prompt instructs the LLM to *edit, not rewrite from scratch*: tuples whose facts remain valid are preserved with the KEEP label, and the smallest valid edit set is preferred. This bias anchors the edit graph to the source structure and ensures that unchanged facts surface explicitly through the KEEP label.

Executable prompt generation. The prompt for Executable Prompt Generation (Figure 10) linearizes the edit scene graph into a natural-language editing command. The LLM is given the source-side blocks (ORIGINAL_CAPTION, ORIGINAL_TUPLES, ORIGINAL_DEPENDENCIES), the target-side blocks (OUTPUT_CAPTION, EDITED_TUPLES, EDITED_DEPENDENCIES), and the original INSTRUCTION. It reads each tuple's EDIT_LABEL as the primary signal for whether to issue an edit verb (ADD, REMOVE, REPLACE, etc.) or to include the tuple in the closing preservation sentence (KEEP). The output OUTPUT_INSTRUCTION follows a two-segment format: a sequence of imperative edit sentences (one per non-KEEP label) followed by a single KEEP sentence enumerating all preserved facts. This formatting choice carries the explicit preservation label from Appendix B through to the editor-facing command: preservation appears as an explicit clause rather than as the implicit complement of edit verbs.

F PIPELINE TRACES OF R2E-PROMPT

Figure 11 and Figure 12 provide pipeline traces that expose R2E-Prompt's intermediate representations across the full pipeline. Each figure shows two examples in which we trace the transformation from the user's source instruction prompt p^{src} through every intermediate produced by R2E-Prompt—the source and target captions (c^{src}, c^{tgt}), the source and edit scene graphs (G^{src}, G^{edit}), and the editing command p^{edit}—to the edited image I^{edit} produced by the frozen editor. The four examples span a range of reasoning patterns: a tadpole maturing into a frog, a progression of lunar phases, a soccer ball deflating after a puncture, and oil settling on the surface of water in a glass.

Taken together, the examples show that R2E-Prompt's pipeline behaves as intended. Target-State Reasoning correctly unpacks the implicit visual consequence of each instruction into the target caption c^{tgt}; Edit-Aware Scene Graph Transformation attaches an operation label to every source-graph node, identifying which elements should change, be removed, or be preserved; and Executable Prompt Generation linearizes the labeled graph into an editing command p^{edit} in which the implicit



```
Task:
I will provide several examples that include inputs and outputs.
Then, I will provide ORIGINAL_CAPTION, ORIGINAL_TUPLES, ORIGINAL_DEPENDENCIES, and OUTPUT_CAPTION. Transform
the original tuples and dependencies into EDITED_TUPLES and EDITED_DEPENDENCIES.

The editied scene graph must describe how the original scene graph changes into the final visual state
described by OUTPUT_CAPTION.

---

### General Requirements

- Use OUTPUT_CAPTION as the final authority for changed facts.
- Preserve unchanged grounded context from ORIGINAL_TUPLES when it remains compatible with OUTPUT_CAPTION.
- Edit tuple bodies and dependencies, not only labels.
- Do not add unsupported facts, guesses, hidden details, explanations, or duplicate tuples.
- Prefer the smallest valid edit set.
- Do not express absence unless OUTPUT_CAPTION explicitly states it.
- Each EDITED_TUPLES must end with an EDIT_LABEL as the final element:
  (..., <EDIT_LABEL>)

---

### Scene Graph Editing Procedure

1. Compare ORIGINAL_CAPTION and OUTPUT_CAPTION to identify what remains true, what changes, what is
contradicted, what is removed, and what is newly introduced.
2. Decide relation changes.
3. For each original or final relation, determine whether the relation is preserved, added, removed,
replaced, moved, or otherwise changed.
4. After relation decisions are fixed, decide the final entity inventory.
5. Determine which entities remain, which entities are newly introduced, which entities are removed, and
which entities are replaced.
6. After the final entity inventory is fixed, decide attributes, states, text, actions, counts, and other
dependent facts.
7. Complete and renumber EDITED_TUPLES from 1 to N.
8. Recompute EDITED_DEPENDENCIES using the final renumbered tuple IDs.

---

### Supported Edit Operations

- Add: Introduce a new object, person, attribute, relation, action, text, or semantic element into the image,
e.g.: add a car on the road
- Remove: Eliminate an existing object or element from the image, e.g.: remove the sofa in the image
- Color Change: Modify the color of a specific object, e.g.: change the color of the shoes to blue
- Material Change: Alter the surface material or texture of an object, e.g.: change the material of the sign
to stone
- Action Change: Modify the pose or action of an instance, e.g.: change the action of the boy to raising
hands
- Expression Change: Adjust the facial expression, e.g.: change the expression to smiling
- Replace: Substitute one object in the image with a different object, e.g.: replace the coffee with an
apple
- Background Change: Change the background scene to another, e.g.: change the background into a forest
- Appearance Change: Modify visual attributes such as patterns or accessories, e.g.: make the cup have a
floral pattern
- Move: Change the spatial position of an object within the image, e.g.: move the plane to the left
- Resize: Adjust the scale or size of an object, e.g.: enlarge the clock
- Tone Transfer: Change global atmosphere or lighting conditions, e.g.: foggy weather, seasonal change
- Style Change: Modify the overall visual style, e.g.: cartoon
- Keep: No change from original state

---

### Tuple Editing Requirements

- Edit, do not rewrite from scratch.
- Preserve original tuple wording, category, and granularity when the fact remains valid.
- If a tuple describes a final-state fact that differs from the original, modify the tuple body to describe
the final state and label it with the appropriate edit label.
- Do not edit the body of a removed tuple into a final-state fact.
- If an object is removed, keep the removed object tuple and its dependent attributes, actions, and
relations when they describe removed facts, and label them Remove.
- If an object is replaced, omit the obsolete object as a final-state entity and add or modify tuples for
the replacement object using Replace.
- If only an attribute changes, preserve the entity tuple and modify only the attribute tuple.
- If only a relation changes, preserve the involved entity tuples and modify, add, or remove the relation
tuple as needed.
- If a new relation is added, the participating final-state entities required to ground that relation must
also appear in EDITED_TUPLES.
- If a new relation or attribute is added between entities that remain unchanged, keep the entity tuples
labeled as Keep, and assign the appropriate edit label only to the relation or attribute tuple.
- Add tuples only for new final-state entities, attributes, counts, text, visible states, actions, or
spatial relations explicitly supported by OUTPUT_CAPTION.
- If a relation is removed but the participating entities remain in the final state, keep the entity tuples
as Keep and label only the relation tuple Remove.
- Every EDITED_TUPLES must end with exactly one high-level EDIT_LABEL.

---

### Dependency Requirements

For each EDITED_TUPLE, identify which final renumbered tuple IDs it depends on.
- Attribute, action, and spatiality tuples depend on the entity tuples they reference.
- Entity - part tuples depend on their parent entity tuple.
- Entity - whole tuples with no parent dependency use 0.
- Multiple dependencies are separated by commas.
- The numbering of dependency lines must correspond one-to-one with the numbering of the extracted tuples.
- Every tuple ID must have exactly one dependency line, and no tuple ID should be skipped or added.
- Added relation tuples must depend on all participating final-state entities.
- Removed relation tuples may depend on removed or kept parent entity tuples that remain in EDITED_TUPLES.
- Preserve original dependencies for removed tuples when their referenced parent tuples are also present in
EDITED_TUPLES.
- Preserve original dependencies only when the referenced edited tuples still exist and the grounding
relation remains valid.
- Recompute dependencies for any modified or newly added tuple.
- If a parent tuple is omitted or replaced, update the dependency to a valid edited tuple when possible.
- Do not create self-dependencies, duplicate dependency IDs, or references to omitted or non-existent tuples.

The number of dependency lines must equal the number of tuples.
```

```
---

### Output Format

EDITED_TUPLES
id | tuple
EDITED_DEPENDENCIES
id | dependencies

---

### Output Requirements

- Output only EDITED_TUPLES and EDITED_DEPENDENCIES.
- Do not repeat the input.
- Do not output ORIGINAL_CAPTION or OUTPUT_CAPTION.
- Do not include notes, explanations, intermediate reasoning, or change
summaries.

---

### Example 1
[input]
ORIGINAL_CAPTION: A woman standing on a rainy street while holding a
closed umbrella.
ORIGINAL_TUPLES
1 | entity - whole (woman)
2 | entity - whole (umbrella)
3 | entity - whole (street)
4 | attribute - state (woman, standing)
5 | attribute - state (umbrella, closed)
6 | attribute - weather (street, rainy)
7 | action - (woman, umbrella, hold)
8 | Spatiality - (woman, street, on)
ORIGINAL_DEPENDENCIES
1 | 0
2 | 0
3 | 0
4 | 1
5 | 2
6 | 3
7 | 1,2
8 | 1,3
OUTPUT_CAPTION: A woman standing on a rainy street while holding an open
umbrella above herself.

[output]
EDITED_TUPLES
1 | entity - whole (woman, Keep)
2 | entity - whole (umbrella, Keep)
3 | entity - whole (street, Keep)
4 | attribute - state (woman, standing, Keep)
5 | attribute - state (umbrella, open, Appearance Change)
6 | attribute - weather (street, rainy, Keep)
7 | action - (woman, umbrella, hold, Keep)
8 | Spatiality - (woman, street, on, Keep)
9 | Spatiality - (umbrella, woman, above, Add)
EDITED_DEPENDENCIES
1 | 0
2 | 0
3 | 0
4 | 1
5 | 2
6 | 3
7 | 1,2
8 | 1,3
9 | 2,1

...(Example 2).
```

Figure 9: Edit scene graph system prompt.

Table 8: **Comparison with prompt-engineering baselines on RISEBench (Zhao et al., 2026).** We report per-category accuracy (%) over the four reasoning types (Temporal, Causal, Spatial, Logical) and the Overall accuracy over all 360 samples. **Bold** denotes the best results.

Method	Temporal ↑	Causal ↑	Spatial ↑	Logical ↑	Overall ↑
(i)	11.8	18.9	27.0	3.5	15.8
(ii)	30.6	23.3	31.0	2.4	22.2
(iii)	28.2	33.3	28.0	8.2	24.7
R2E-Prompt (**Ours**)	**35.3**	**44.4**	**33.0**	**11.8**	**31.4**

consequence of p^{src} becomes fully explicit at the element level. Reading p^{edit} alone—without the original reasoning instruction—is sufficient to recover what the user implied: the editing commands shown spell out, in surface-level imperative form, the wholesale identity replacement (tadpole $\rightarrow$ frog), the fine-grained illumination change (moon), the new state attributes acquired after deflation (soccer ball), and the new compositional structure that emerges when oil is added (oil-on-water layer)—visual consequences that the original p^{src} only entailed. The resulting edited images I^{edit} confirm that this explicit editing command is sufficient for an off-the-shelf editor to realize the intended visual change.

```
Task:
I will provide several examples that include inputs and outputs.
Then, I will provide ORIGINAL_CAPTION, ORIGINAL_TUPLES, ORIGINAL_DEPENDENCIES, INSTRUCTION,
OUTPUT_CAPTION, EDITED_TUPLES, EDITED_DEPENDENCIES.

You must write a concise natural-language image editing instruction that transforms the visual state
described by ORIGINAL_CAPTION, ORIGINAL_TUPLES, and ORIGINAL_DEPENDENCIES into the final visual state
described by OUTPUT_CAPTION, EDITED_TUPLES, and EDITED_DEPENDENCIES.
Each tuple in EDITED_TUPLES includes an EDIT_LABEL as the final element inside parentheses.

---

### Procedure

1. Understand the original visual state.
2. Understand the final visual state.
3. Identify edited facts using EDIT_LABELs marked as Add, Remove, Color Change, Material Change, Action
Change, Expression Change, Replace, Background Change, Appearance Change, Move, Resize, Tone Transfer,
and Style Change.
4. Identify kept facts using EDIT_LABELs marked as Keep.
5. Write the edit instruction with both changed facts and kept facts.

---

### Understanding the Input

- Use ORIGINAL_CAPTION, ORIGINAL_TUPLES, and ORIGINAL_DEPENDENCIES to understand the starting visual
state.
- Use OUTPUT_CAPTION, EDITED_TUPLES, and EDITED_DEPENDENCIES to understand the final visual state.
- Use INSTRUCTION and OUTPUT_CAPTION only as additional context.
- Treat the final argument of each tuple in EDITED_TUPLES as its EDIT_LABEL.
- Interpret each EDIT_LABEL according to the definitions in Supported Edit Operations.
- Use EDIT_LABELs as the primary signal for detecting edited facts and kept facts.
- Use ORIGINAL_TUPLES only to recover previous values for changed facts when needed.

---

### Supported Edit Operations

- Add: Introduce a new object, person, attribute, relation, action, text, or semantic element into the
image, e.g.: add a car on the road
- Remove: Eliminate an existing object or element from the image, e.g.: remove the sofa in the image
- Color Change: Modify the color of a specific object, e.g.: change the color of the shoes to blue
- Material Change: Alter the surface material or texture of an object, e.g.: change the material of the
sign to stone
- Action Change: Modify the pose or action of an instance, e.g.: change the action of the boy to
raising hands
- Expression Change: Adjust the facial expression, e.g.: change the expression to smiling
- Replace: Substitute one object in the image with a different object, e.g.: replace the coffee with an
apple
- Background Change: Change the background scene to another, e.g.: change the background into a forest
- Appearance Change: Modify visual attributes such as patterns or accessories, e.g.: make the cup have
a floral pattern
- Move: Change the spatial position of an object within the image, e.g.: move the plane to the left
- Resize: Adjust the scale or size of an object, e.g.: enlarge the clock
- Tone Transfer: Change global atmosphere or lighting conditions, e.g.: foggy weather, seasonal change
- Style Change: Modify the overall visual style, e.g.: cartoon
- Keep: No change from original state

---

### Instruction Writing Principles

- Write the instruction as a direct imperative edit command.
- First describe the edited object, attribute, action, relation, style, background, or other edited
fact in sentence form.
- For Remove or Replace, explicitly state "remove" or "replace" when the tuple represents removal or
replacement.
- For Add, explicitly state "add" only when introducing a new object, person, text, or semantic element.
For added attributes, actions, or relations, describe the final fact naturally instead of forcing the
word "add".
- For Color Change, Material Change, Appearance Change, Action Change, Expression Change, Move, Resize,
Background Change, Tone Transfer, and Style Change, state the final value or final visual state clearly
and include the original value when it can be identified from ORIGINAL_TUPLES.
- Do not copy or paraphrase INSTRUCTION.
- Do not infer or assume any change that is not explicitly represented by a non-Keep EDIT_LABEL, even
if suggested by INSTRUCTION, ORIGINAL_CAPTION, or OUTPUT_CAPTION.
- After all edit sentences, write a Keep sentence that reflects every Keep-labeled tuple.
- If a tuple is labeled Keep, describe it only in the Keep sentence.

---

### Keep sentence rules

- The Keep sentence must start with "Keep".
- Include all facts from Keep-labeled tuples in natural language.
- Combine Keep facts into one concise sentence when possible.
- Do not mention tuple IDs, edit labels, or dependency structure.
```

```
---

### Output format

OUTPUT_INSTRUCTION: <instruction>

---

### Output requirements

- Use the section label OUTPUT_INSTRUCTION exactly.
- Output between 1 and 5 sentences as a single coherent instruction.
- The final sentence must be the Keep sentence when any Keep-labeled tuple
exists.
- Do not invent objects, attributes, relationships, or background details.
- Do not include information that contradicts EDITED_TUPLES.
- Do not describe processes or explanations.
- Do not output tuples, dependencies, captions, notes, or reasoning.

---

### Example 1
[input]
ORIGINAL_CAPTION: A woman standing on a rainy street while holding a closed
umbrella.
ORIGINAL_TUPLES
1 | entity - whole (woman)
2 | entity - whole (umbrella)
3 | entity - whole (street)
4 | attribute - state (woman, standing)
5 | attribute - state (umbrella, closed)
6 | attribute - weather (street, rainy)
7 | action - (woman, umbrella, hold)
8 | Spatiality - (woman, street, on)
ORIGINAL_DEPENDENCIES
1 | 0
2 | 0
3 | 0
4 | 1
5 | 2
6 | 3
7 | 1,2
8 | 1,3
OUTPUT_CAPTION: A woman standing on a rainy street while holding an open
umbrella above herself.
INSTRUCTION: Make her use the umbrella.
EDITED_TUPLES
1 | entity - whole (woman, Keep)
2 | entity - whole (umbrella, Keep)
3 | entity - whole (street, Keep)
4 | attribute - state (woman, standing, Keep)
5 | attribute - state (umbrella, open, Appearance Change)
6 | attribute - weather (street, rainy, Keep)
7 | action - (woman, umbrella, hold, Keep)
8 | Spatiality - (woman, street, on, Keep)
9 | Spatiality - (umbrella, woman, above, Add)
EDITED_DEPENDENCIES
1 | 0
2 | 0
3 | 0
4 | 1
5 | 2
6 | 3
7 | 1,2
8 | 1,3
9 | 2,1

[output]
OUTPUT_INSTRUCTION: Change the umbrella from closed to open. Position the
umbrella above the woman. Keep the woman, the umbrella, the street, the woman
standing, the rainy street, the woman holding the umbrella, and the woman on
the street.

...(Example 2).
```

Figure 10: Executable Prompt Generation system prompt.

G Comparison with Prompt-Engineering Baselines

We evaluate three prompt-engineering baselines to disentangle the contribution of our structured source-to-target reasoning from prompt engineering alone: (i) an LLM directly rewrites the instruction into a more explicit prompt; (ii) an LLM generates an explicit prompt from the source caption and instruction; and (iii) a target caption is combined with preservation text.

The direct instruction-rewriting baseline (i) performs worse than the baseline (Wu et al., 2025) (15.8% vs. 19.4%), indicating that instruction refinement without source-image context does not necessarily improve editing performance and can introduce unintended deviations. Conditioning the LLM additionally on the source caption (ii) improves Overall accuracy to 22.2%, but remains below R2E-Prompt. The baseline in (iii), which augments the target caption with generic preservation text, achieves 24.7% Overall accuracy, below the TSR+EPG variant in Table 2-(b) (27.5%). This suggests that generic preservation text is insufficient because a target caption specifies the desired output globally, without identifying which elements should be edited or preserved. Such constraints may therefore conflict with necessary changes to an object's attributes, pose, or parts.

In contrast, our scene-graph formulation and edit taxonomy yield a structured source-to-target representation by assigning explicit element-level operations for modification and preservation. As shown in Table 2, incorporating ESGT into the TSR+EPG configuration improves Overall accuracy from 27.5% in row (b) to 31.4% in row (d), demonstrating benefits beyond simply adding target-state or preservation text.

H Effect of LLM Scale

We analyze how the choice of LLM used to drive R2E-Prompt's reasoning stages affects editing performance. Table 9 reports RISEBench accuracy when the same R2E-Prompt pipeline is paired

Table 9: Effect of LLM scale within R2E-Prompt, evaluated on RISEBench (Zhao et al., 2026) with Qwen-Image-Edit as the frozen image editor. The same R2E-Prompt pipeline is run with three different LLM choices for the reasoning stages: Gemma3-4B, Qwen2.5-7B, and Qwen3-30B-A3B (our main setting). The image editor is held fixed across all rows.

Method	Temporal ↑	Causal ↑	Spatial ↑	Logical ↑	Overall ↑
Qwen-Image-Edit (Wu et al., 2025)	21.2	18.9	31.0	4.7	19.4
+ R2E-Prompt w/ Gemma3-4B	21.2	30.0	32.0	8.2	23.3
+ R2E-Prompt w/ Qwen2.5-7B	29.4	25.6	28.0	8.2	23.1
+ R2E-Prompt w/ Qwen3-30B-A3B (**Ours**)	**35.3**	**44.4**	**33.0**	**11.8**	**31.4**

Table 10: Per-category and overall scores on the ImgEdit-Bench (Ye et al., 2026), evaluated with the GPT-4.1. Within each backbone pair, the higher score per column is highlighted in **bold**.

Method	Add	Adjust	Extract	Replace	Remove	Background	Style	Hybrid	Action	Overall
FLUX.2-Klein-4B (Labs, 2026)	**4.48**	4.35	1.83	4.22	3.97	4.29	**4.93**	3.06	**4.76**	3.99
FLUX.2-Klein-4B + R2E-Prompt (**Ours**)	4.43	**4.41**	**3.31**	**4.46**	**4.27**	**4.42**	4.72	**3.07**	4.64	**4.19**
FLUX.2-Klein-9B-KV (Labs, 2026)	**4.51**	4.31	2.42	**4.54**	4.07	4.37	**4.95**	**3.09**	**4.75**	4.11
FLUX.2-Klein-9B-KV + R2E-Prompt (**Ours**)	**4.51**	**4.35**	**3.26**	4.47	**4.41**	**4.53**	4.81	2.80	4.72	**4.21**
FLUX.2-Dev (Labs, 2026)	4.45	**4.15**	2.68	**4.51**	4.01	**4.35**	**4.78**	**3.36**	4.68	4.11
FLUX.2-Dev + R2E-Prompt (**Ours**)	**4.48**	**4.15**	**3.44**	4.50	**4.59**	4.28	4.76	3.29	**4.69**	**4.24**
Qwen-Image-Edit-2511 (Wu et al., 2025)	4.48	**4.40**	**4.00**	**4.57**	**4.35**	**4.28**	**4.86**	**3.72**	4.76	**4.38**
Qwen-Image-Edit-2511 + R2E-Prompt (**Ours**)	**4.60**	4.26	3.29	**4.57**	**4.35**	4.23	4.63	3.48	**4.84**	4.25

with three LLMs of increasing scale—Gemma3-4B, Qwen2.5-7B, and Qwen3-30B-A3B—while holding the image editor (Qwen-Image-Edit) fixed.

R2E-Prompt is beneficial at every LLM scale. All three LLMs lift the baseline Qwen-Image-Edit accuracy (19.4%) by a non-trivial margin: Gemma3-4B brings the editor to 23.3% (+3.9), Qwen2.5-7B to 23.1% (+3.7), and Qwen3-30B-A3B to 31.4% (+12.0). This indicates that the benefit of structuring reasoning upstream of the editor is not tied to a specific class of LLMs—even a 4B model is sufficient to obtain a measurable gain over a baseline editor that consumes the raw instruction directly. R2E-Prompt's framework therefore offers a useful operating point at multiple LLM scales, including the small-model regime relevant for low-cost deployment.

Gains scale with the LLM's reasoning capacity, not with parameter count alone. The 4B and 7B models, despite differing in parameter count by nearly $2\times$, produce nearly identical overall accuracy (23.3% vs. 23.1%), with their per-category numbers fluctuating but converging on Overall. The substantial jump to 31.4% appears only with Qwen3-30B-A3B, which differs from the smaller models not only in scale but also in its stronger instruction-following and reasoning ability. The gain pattern is most visible on the categories that demand unpacking implicit visual consequences: *Causal* accuracy rises from 30.0% (4B) and 25.6% (7B) to 44.4% (30B-A3B), and *Temporal* from 21.2% / 29.4% to 35.3%. *Spatial* and *Logical*, where success depends more on the editor's intrinsic capacity than on the reasoning quality, see smaller and more uniform gains. This pattern suggests that R2E-Prompt's headroom is bounded by the LLM's reasoning capacity rather than its raw size, consistent with the main paper's framing of structured language-space reasoning as the primary lever.

Table 11: User study results of R2E-Prompt (percentage).

Method	Instruction Following ↑	Appearance Consistency ↑	Visual Plausibility ↑
Gemini-2.5-Flash-Image (Nano Banana)	**49.6**	<u>47.6</u>	**41.6**
EditThinker	1.6	0.0	19.6
BAGEL (w/ CoT)	3.6	2.4	8.0
R2E-Prompt (**Ours**)	<u>45.2</u>	**50.0**	<u>30.8</u>

I User Study

We conduct a user study involving 50 participants to further evaluate the effectiveness of R2E-Prompt. The generated results are evaluated from three aspects: (1) **Instruction Following**, assessing whether the model correctly interprets and executes the editing instruction; (2) **Appearance Consistency**, evaluating whether the model preserves regions that should remain unaltered while applying edits only to the intended regions; and (3) **Visual Plausibility**, evaluating the realism and overall visual coherence of the edited output. For the evaluation of **Visual Plausibility**, only the edited result images are shown to participants to enable evaluation independent of the source image and instruction context. For comparison, Gemini-2.5-Flash-Image (Nano Banana) (Google DeepMind, 2025)—the strongest closed-source proprietary model in Table 1—and the two best-performing open-source models with publicly available weights, EditThinker (Li et al., 2025) and BAGEL (w/ CoT) (Deng et al., 2025), are included as baselines.

The results of the user study are presented in Table 11. Notably, in terms of **Visual Plausibility**, which primarily evaluates the quality of the output image itself, EditThinker also achieves relatively strong performance by utilizing a powerful off-the-shelf image editor. However, in **Instruction Following** and **Appearance Consistency**, where reasoning capability and accurate localization of editable regions are crucial, R2E-Prompt demonstrates performance competitive with Gemini-2.5-Flash-Image (Nano Banana), while the other open-source baselines exhibit substantially lower scores.

Despite these improvements, a gap remains between R2E-Prompt and Nano Banana in **Visual Plausibility** (30.8% vs. 41.6%). Since this criterion is assessed solely from the final edited output, it is also affected by the rendering capability of the frozen downstream editor, particularly for fine-grained physical interactions and realistic visual details. Overall, the user study indicates that R2E-Prompt provides the strongest instruction following and appearance consistency over the evaluated open-source approaches, bridging their performance considerably closer to that of closed-source proprietary models. An example of the user study interface is illustrated in Figure 13.

J Compatibility with General Editing Tasks

Although R2E-Prompt is primarily designed for implicit-instruction image editing, its structured upstream reasoning pipeline can also be applied to standard explicit editing instructions. In this section, we analyze the behavior of R2E-Prompt on explicit editing instructions through quantitative and qualitative evaluations on ImgEdit-Bench (Ye et al., 2026).

Quantitative analysis on ImgEdit-Bench. Table 10 presents results on ImgEdit-Bench (Ye et al., 2026), a general-purpose instruction-following benchmark for image editing that covers nine edit types: Add, Adjust, Extract, Replace, Remove, Background, Style, Hybrid, and Action. We adopt the LMM-as-a-Judge protocol with GPT-4.1 as the evaluator, which rates each edited image on a 1–5 scale along three axes: instruction adherence, image-editing quality, and detail preservation. The Overall score is the average over the nine per-type scores. To assess transferability, we apply R2E-Prompt to the four editor backbones (FLUX.2-Klein-4B, FLUX.2-Klein-9B-KV, FLUX.2-Dev, and Qwen-Image-Edit-2511) and compare each against its baseline.

Since ImgEdit-Bench primarily evaluates explicit instruction following rather than implicit visual reasoning, the gains from upstream reasoning are limited, and performance trends across edit categories remain comparable. Nevertheless, R2E-Prompt remains broadly compatible with general editing tasks across different backbones, with overall scores showing only moderate variations relative to their corresponding baselines. These results suggest that the structured reasoning pipeline of R2E-Prompt can also be applied to explicit-instruction scenarios while remaining compatible with off-the-shelf editors in a training-free manner.

Qualitative examples on general editing. Figure 14 and Figure 15 present qualitative examples covering all nine editing categories of ImgEdit-Bench. These include object-level modifications such as Add, Adjust, Replace, and Remove; broader scene-level edits such as Background and Style; as well as relatively complex editing categories including Hybrid and Extract. In each case, the editing command p^{edit} generated by Executable Prompt Generation successfully captures both

the requested modification and the source elements to be preserved. These qualitative examples suggest that the structured upstream pipeline of R2E-Prompt remains applicable beyond implicit-instruction scenarios and can still produce coherent editing commands for general natural-language editing instructions.

K LIMITATIONS

While R2E-Prompt separates reasoning from image generation to remain training-free and editor-agnostic, its overall performance depends on the capabilities of both the upstream reasoning model and the downstream editor. The bound is most evident in the *Logical* category of RISEBench, where our method reaches 11.8%—a 7.1-point gain over the Qwen-Image-Edit baseline (4.7%)—yet still remains substantially below the other reasoning categories. Similar trends are observed in strong proprietary editors, suggesting that logically constrained image editing remains challenging for current image editing systems. At the same time, the performance variation across upstream LLMs in Appendix H indicates that the quality of target-state inference and element-level reasoning also affects the final result. Thus, errors from the upstream LLM may propagate through the framework. However, R2E-Prompt separates target-state inference from edit specification and uses an edit-aware scene graph to explicitly resolve source-to-target edit and preservation decisions at the element level, rather than jointly reasoning over them in a single generation. Furthermore, the modular, editor-agnostic design of R2E-Prompt allows stronger reasoning models and image editors to be incorporated without modifying the framework. From an efficiency perspective, R2E-Prompt introduces additional inference overhead from its upstream VLM/LLM reasoning process beyond the editor itself. Although R2E-Prompt incurs this reasoning overhead only once per edit and its explicit intermediate representations make each stage inspectable, the current pipeline does not include an explicit verification mechanism. Self-refinement or auxiliary judges could improve robustness, but would require additional model calls and latency. Reducing reasoning latency through more efficient reasoning modules or distilled pipelines remains an important direction for future work.



Source Image I^{src}

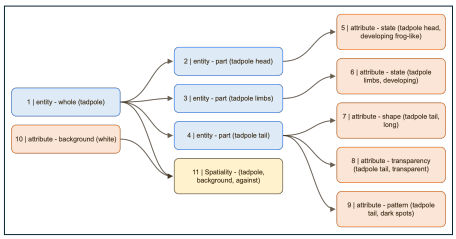
Source Caption c^{src}

A tadpole with a developing frog-like head and limbs, and a long, transparent tail with dark spots, set against a white background.

Instruction Prompt p^{src}

"Draw what it will look like three months later."

Source graph G^{src}





Edited Image I^{edit}

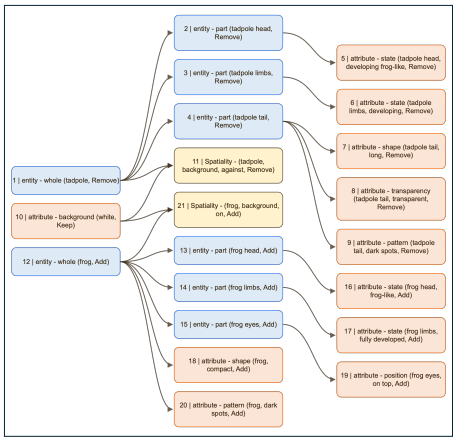
Target Caption c^{tgt}

A young frog with four fully developed limbs, a compact body, a frog-like head with eyes on top, and dark spots, standing on a white background, with no tail remaining.

Editing command p^{edit}

"Remove the tadpole, its developing head, limbs, and long transparent tail with dark spots. Replace it with a young frog that has a compact body, four fully developed limbs, a frog-like head with eyes on top, and dark spots. Position the frog on the white background. Keep the white background."

Edit graph G^{edit}





Source Image I^{src}

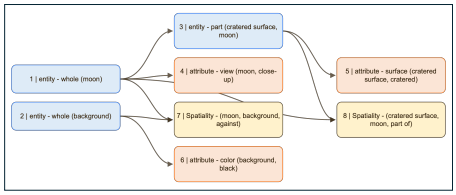
Source Caption c^{src}

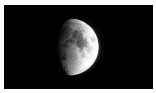
A close-up view of the moon showing its cratered surface against a black background.

Instruction Prompt p^{src}

"This picture is observed from the Northern Hemisphere. Draw what it will look like 7 days later."

Source graph G^{src}





Edited Image I^{edit}

Target Caption c^{tgt}

A close-up view of the moon showing a more illuminated cratered surface with the terminator shifted toward the right, against a black background.

Editing command p^{edit}

"Increase the illumination of the cratered surface of the moon. Shift the terminator on the cratered surface toward the right. Keep the moon, the black background, the close-up view of the moon, the cratered surface, the cratered surface being part of the moon, the moon against the background, and the background being black."

Edit graph G^{edit}

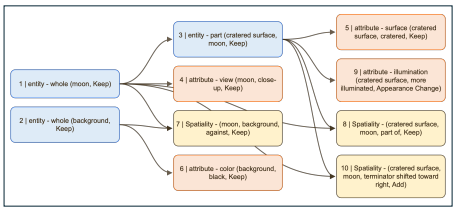


Figure 11: Pipeline traces of R2E-Prompt on two reasoning examples. For each instance, we show I^{src}, p^{src}, the captions $(c^{\text{src}}, c^{\text{tgt}})$, the scene graphs ($G^{\text{src}}, G^{\text{edit}}$, with edit operation labels on G^{edit} nodes), the editing command p^{edit}, and the edited image I^{edit}. **Top**: tadpole-to-frog metamorphosis. **Bottom**: lunar phase progression.

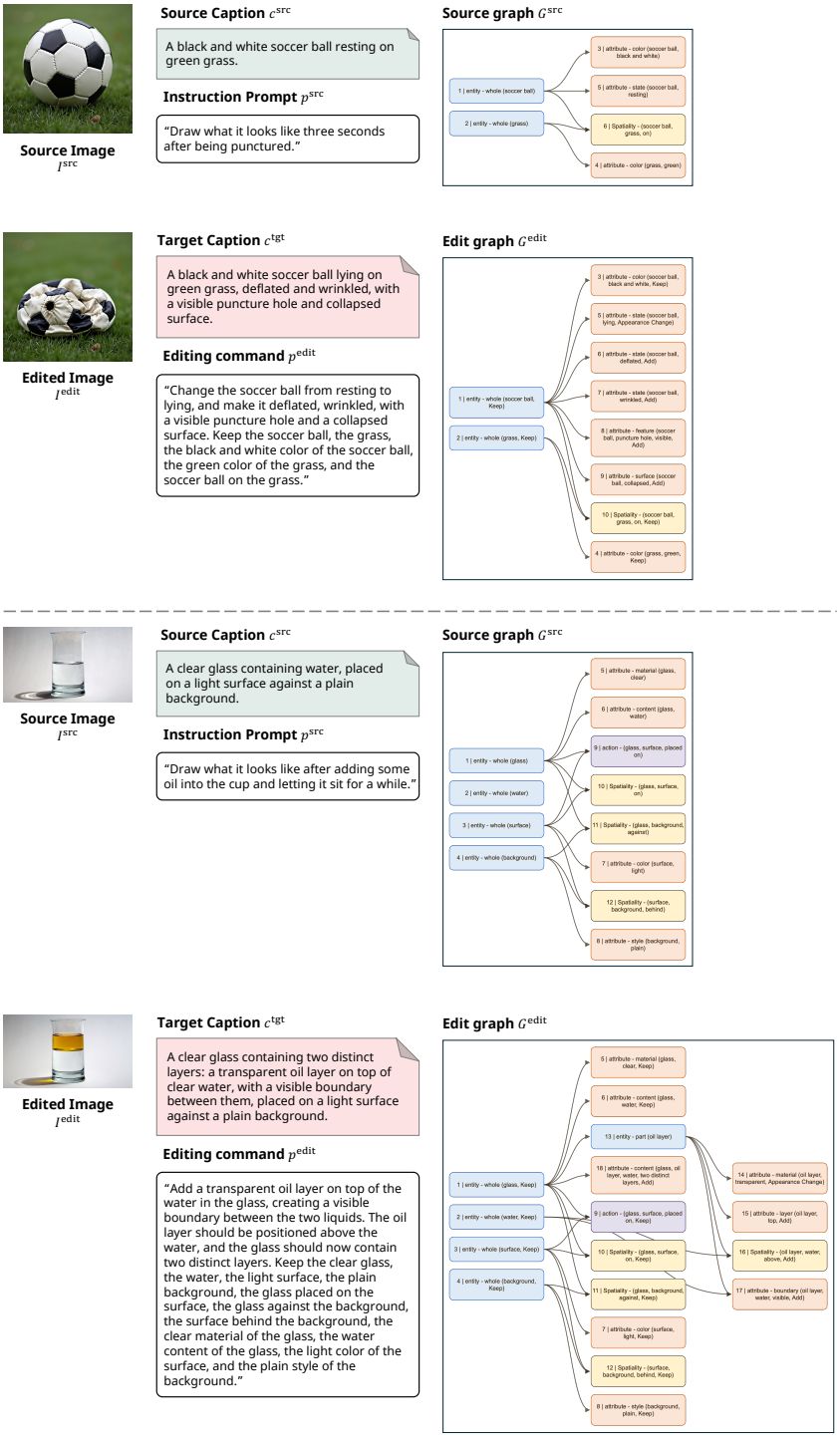


Figure 12: Pipeline traces of R2E-Prompt on two additional reasoning examples, with the same layout as Figure 11. **Top**: soccer ball deflation after puncture. **Bottom**: oil-on-water layer formation.

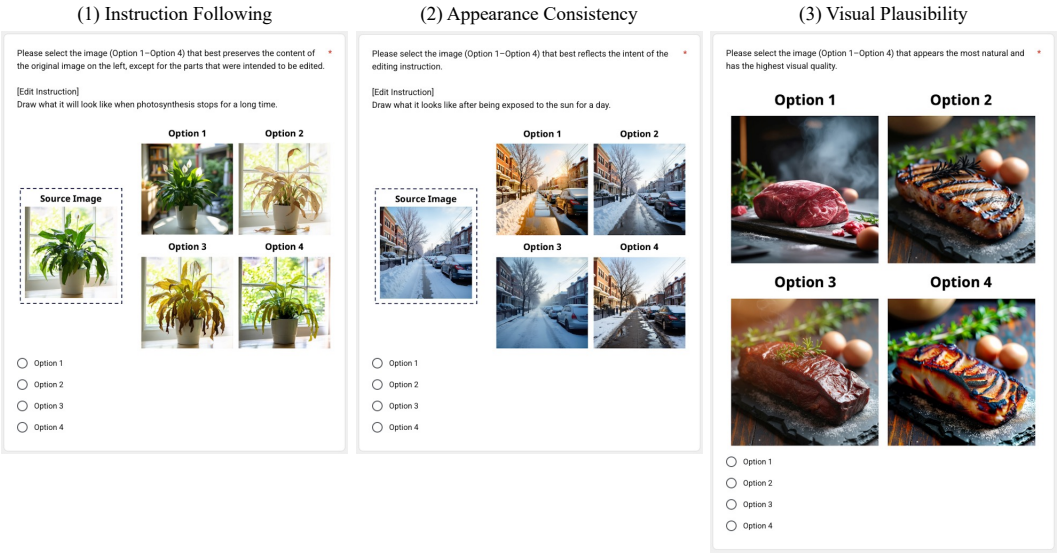


Figure 13: An example of the user study interface.

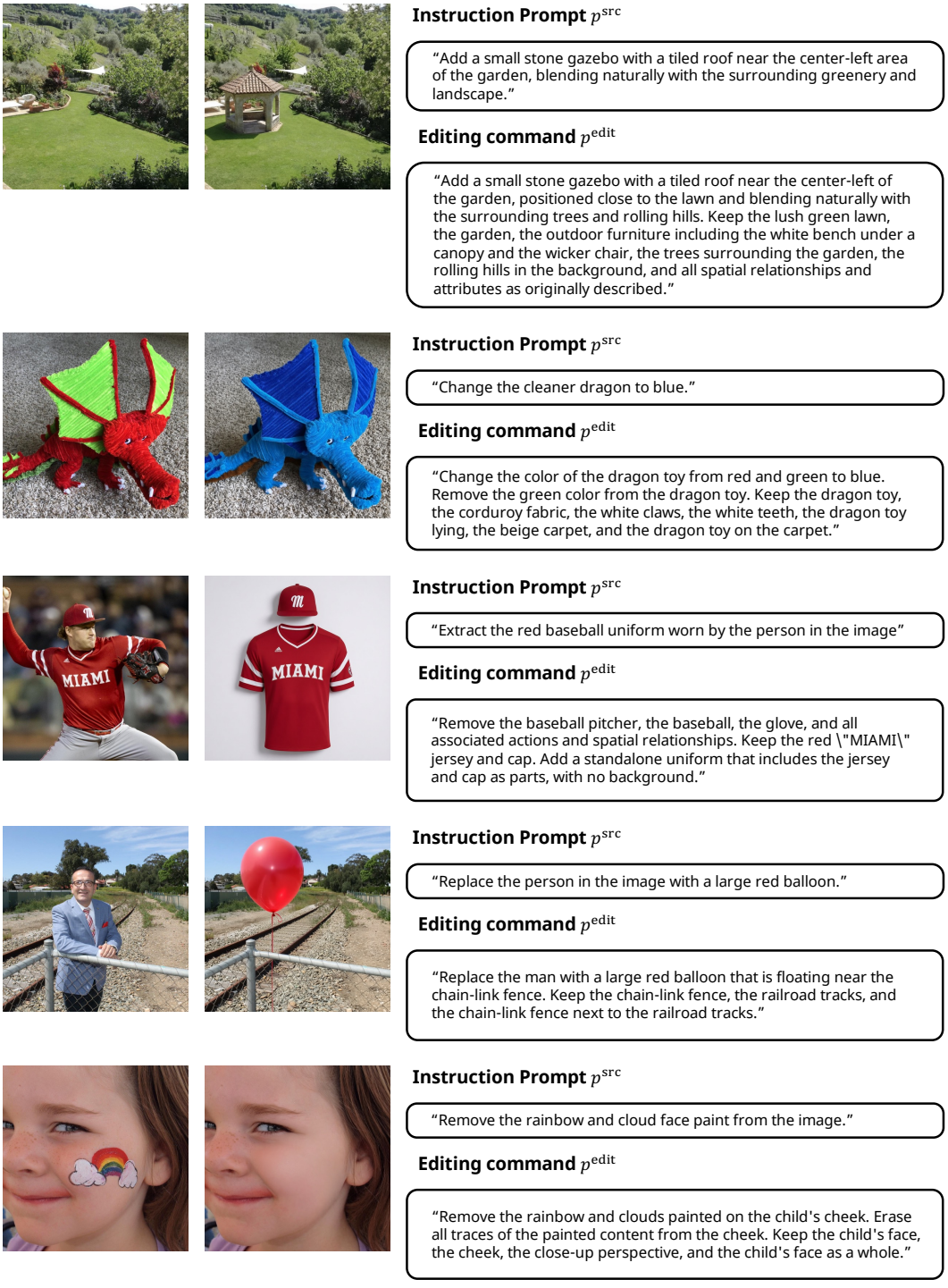


Figure 14: Qualitative examples of Qwen-Image-Edit + R2E-Prompt (Ours) on the Add, Adjust, Extract, Replace, and Remove edit categories of ImgEdit-Bench. For each instance, we show the source image I^{src} and the edited image I^{edit}, along with the user's instruction p^{src} and the editing command p^{edit} produced by R2E-Prompt's Executable Prompt Generation stage.



Figure 15: Qualitative examples of Qwen-Image-Edit + R2E-Prompt (Ours) on the Background, Style, Hybrid, and Action edit categories of ImgEdit-Bench. For each instance, we show the source image I^{src} and the edited image I^{edit}, along with the user's instruction p^{src} and the editing command p^{edit} produced by R2E-Prompt's Executable Prompt Generation stage.